3D CODE FACTORY: SCALING VERIFIABLE 3D TASKS FOR CODING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent rapid progress in coding agents has increased the demand for scalable task generation to advance agent capabilities, which requires diverse, executable tasks equipped with reliable automated verifiers. For procedural 3D modeling, fixed-reference comparisons can penalize semantically valid alternative implementations, while rendered images alone cannot fully verify spatial relationships and precise measurements. To address this problem, we introduce 3D Code Factory, an automated framework that synthesizes executable 3D coding tasks paired with multimodal evaluation rubrics from seed scenarios, decomposing verification into scene-state checks for spatial relations and rendered-image checks for visual appearance. Building on this framework, we construct 3D Live Bench to evaluate coding agents on 36 core tasks in Three.js and Blender under different scaffolds, assessing deliverables across three metrics: execution success, rubric compliance, and visual quality. Extensive evaluations reveal substantial room for improvement in complex 3D modeling: despite high program execution rates, agents struggle to satisfy coupled task requirements, with peak visual quality and rubric compliance achieved by different model–scaffold configurations. During iterative refinement, localized repairs often introduce regressions, invalidating or neglecting previously satisfied criteria. Furthermore, scaffold design fundamentally alters how base-model capabilities translate into task completion, while prolonged edit–render loops substantially inflate token expenditure with diminishing returns.

1 INTRODUCTION

Large language models (LLMs) have demonstrated strong capabilities in code generation, reasoning, and tool use, making them a promising foundation for autonomous coding agents tackling multi-step development tasks (Chen et al., 2021; Schick et al., 2023; OpenAI, 2025; Anthropic, 2025; Guo et al., 2025a). Improving and evaluating these agents relies heavily on environments that provide diverse challenges and reliable automated feedback (Jimenez et al., 2024; Pan et al., 2024; Merrill et al., 2026; Xie et al., 2024). In software engineering, environments rely on unit tests, exit codes, and state transitions to verify actions. In contrast, procedural 3D modeling presents a fundamentally different challenge: instructions admit diverse valid geometric constructions, making rigid reference matching too restrictive, while correctness depends on internal spatial relationships, such as hierarchies and clearances, that rendered 2D images alone cannot fully verify. This creates a core evaluation dilemma: how to assess procedural 3D code without over-constraining geometric implementation, while rigorously verifying spatial structure and visual appearance.

Recent benchmarks make valuable progress in procedural 3D modeling across complementary domains (Table 1), exploring single-image reconstruction (IR3D-Bench (Liu et al., 2025)), parametric modeling across code formats (P3D-Bench (Yang et al., 2026)), agent Blender modeling with human preferences (3DCodeBench (Gao et al., 2026)), and runtime contracts in Three.js (WorldCoder-Bench (Lu et al., 2026)). Our key insight is to formulate procedural 3D evaluation as property verification rather than artifact matching. We evaluate each requested property independently of how the agent implements it, using runtime scene state for directly measurable geometric and structural properties, and rendered views for perceptual properties such as lighting and material appearance. We introduce 3D Code Factory, which jointly synthesizes implementation-agnostic instructions and multimodal rubrics from shared scene facts (Figure 1), and a multi-stage review audits instruction clarity, rubric checkability, and physical buildability.

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

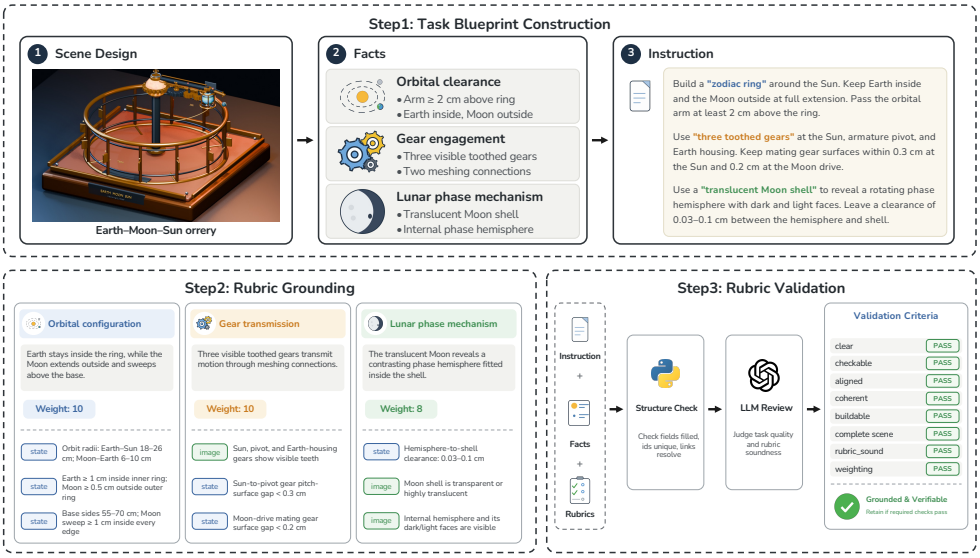

Figure 1: **Automated task synthesis and multimodal rubric decomposition in 3D Code Factory.** Given a seed scenario such as an Earth–Moon–Sun orrery, 3D Code Factory derives implementation-agnostic instructions and a weighted, verifiable rubric from the same scene facts. Each criterion binds to its evidence channel: spatial properties from scene state, visible ones from rendered views.

Building on this framework, we construct 3D Live Bench to evaluate autonomous coding agents across Three.js and Blender environments. Rather than evaluating isolated one-shot generation, 3D Live Bench embeds agents in realistic developer workflows using native command-line scaffolds augmented with specialized 3D tools via the Model Context Protocol (MCP). Operating in a closed perception-action loop, agents write and execute procedural code, inspect runtime scene state, and review rendered views to iteratively refine their scenes. Deliverables are evaluated across three metrics: program execution rate, fine-grained rubric compliance, and perceptual visual quality.

We conduct extensive empirical evaluations across frontier models, command-line scaffolds, and downstream trajectory fine-tuning. Our findings reveal that despite high program execution rates, agents struggle to satisfy fine-grained task requirements, leaving rubric compliance between 0.40 and 0.80, with peak visual quality and rubric compliance achieved by different model–scaffold configurations. Furthermore, scaffold design alters how base-model capabilities translate into task completion, while prolonged edit–render loops inflate token expenditure with diminishing returns. Finally, fine-tuning open models on multi-turn trajectories synthesized by 3D Code Factory substantially improves procedural 3D modeling. We summarize our main contributions as follows:

- **Fact-anchored task and rubric synthesis.** We introduce 3D Code Factory, an automated framework that co-synthesizes implementation-agnostic instructions and multimodal rubrics from shared scene facts, decoupling state inspection from image review.
- **Procedural 3D benchmark at scale.** We build 3D Live Bench with 36 core tasks across Three.js and Blender, evaluated via execution success, rubric compliance, and visual quality, and scale the pipeline to 488 verified tasks across 500 scenarios through a nine-dimensional quality review.
- **Agent evaluation across scaffolds.** We evaluate coding agents across diverse scaffolds, showing that high code execution rates are not accompanied by fine-grained rubric compliance, while scaffolds substantially shape task success and token expenditure.

2 Related Work

Automated Task and Environment Synthesis. Constructing interactive environments with automated verification is central to evaluating and training autonomous agents. In software engineering, early benchmarks derive tasks from historical issues, pull requests, and unit tests (Jimenez et al.,

Table 1: Comparison with representative 3D code benchmarks. Checkmarks indicate supported, triangles partial support, and crosses no support (3D Live Bench is generated by 3D Code Factory).

Method	Eval Method	Harness-native Rollout	Multi-environment	Scalable Task Synthesis
IR3D-Bench	Metrics + LLM judge	×	×	×
3DCodeBench	Reference-based metrics + preference	✓	×	△
P3D-Bench	Metrics + LLM judge	×	✓	×
WorldCoder-Bench	Runtime contracts	×	×	△
3D Live Bench	**Rubric compliance (state + image) + visual quality**	✓	✓	✓

2024; Pan et al., 2024), while later systems synthesize tasks through fault injection, commit-derived tests, or automated environment construction (Yang et al., 2025b; Jain et al., 2025; Guo et al., 2025b). Recent work further improves evaluation isolation and execution fidelity through withheld grading assets and sandboxed, harness-native rollouts (Tencent WorkBuddy Bench Team, 2026; Du et al., 2026). Beyond software repositories, AWM (Wang et al., 2026), EnvScaler (Song et al., 2026), ScaleEnv (Tu et al., 2026), and Agent-World (Dong et al., 2026) synthesize interactive environments together with state-level verification, while other pipelines derive tasks from command-line or tool-use traces (Wu et al., 2026; Cheng et al., 2026; Hua et al., 2026; Chen et al., 2026) and GUI demonstrations (Sun et al., 2025b; Xie et al., 2025). These approaches largely rely on discrete, machine-verifiable signals such as unit tests, return codes, database states, or tool outcomes. Procedural 3D tasks instead combine continuous spatial constraints with visually defined properties, requiring verification over both executable scene state and rendered appearance.

Procedural 3D Code Generation and Benchmarking. Procedural 3D modeling represents scenes as executable code, enabling parametric and editable geometry (Sun et al., 2025a; Hu et al., 2024). Systems such as BlenderAlchemy (Huang et al., 2024), VIGA (Yin et al., 2026), and WorldClaw (Guo et al., 2026) further use rendered feedback to iteratively refine generated scenes. These works primarily target scene generation, whereas our focus is the scalable construction and evaluation of verifiable 3D coding tasks. Existing benchmarks span CAD and parametric 3D generation (Du et al., 2024; Gao et al., 2026; Yang et al., 2026) and inverse rendering (Liu et al., 2025), commonly evaluating outputs through reference geometry or rendered views. Such reference-based evaluation can be brittle when multiple implementations satisfy the same specification: image similarity may fail to reflect underlying 3D validity (Lu et al., 2026), while rigid geometric matching can penalize alternative valid constructions. A second challenge lies in the evaluation setting itself: coding-agent performance depends substantially on the surrounding scaffold and interaction protocol (Yao et al., 2026; Zheng et al., 2026; Lin et al., 2026), yet these factors are not uniformly captured across prior 3D benchmarks. In contrast, 3D Code Factory constructs fact-grounded, implementation-agnostic tasks with state-and-image verification, while 3D Live Bench evaluates complete agent rollouts within native coding scaffolds across multiple 3D environments.

3 3D Code Factory

3D Code Factory derives both the task instruction $\mathcal{I}$ and the evaluation rubric $\mathcal{R}$ from one set of scene facts $\mathcal{F}$: every criterion is anchored to the requirements that generated the task, while the agent chooses the code structure, coordinate choices, and geometric construction used to satisfy them. Figure 2 shows the generation and evaluation architecture.

3.1 Design Principles and Task Package Formulation

The architecture of 3D Code Factory is governed by three core design principles. First, the framework enforces a single source of truth: every instructional constraint, geometric tolerance, and rubric check originates from the same underlying factual specification, preventing discrepancies between public requirements and withheld evaluation standards. Second, task specifications decouple functional requirements from code realization, specifying what relationships must hold in the resulting scene rather than prescribing how the agent must code them. Third, evaluation separates scene state from image, dispatching each condition to its dedicated evidence channel.

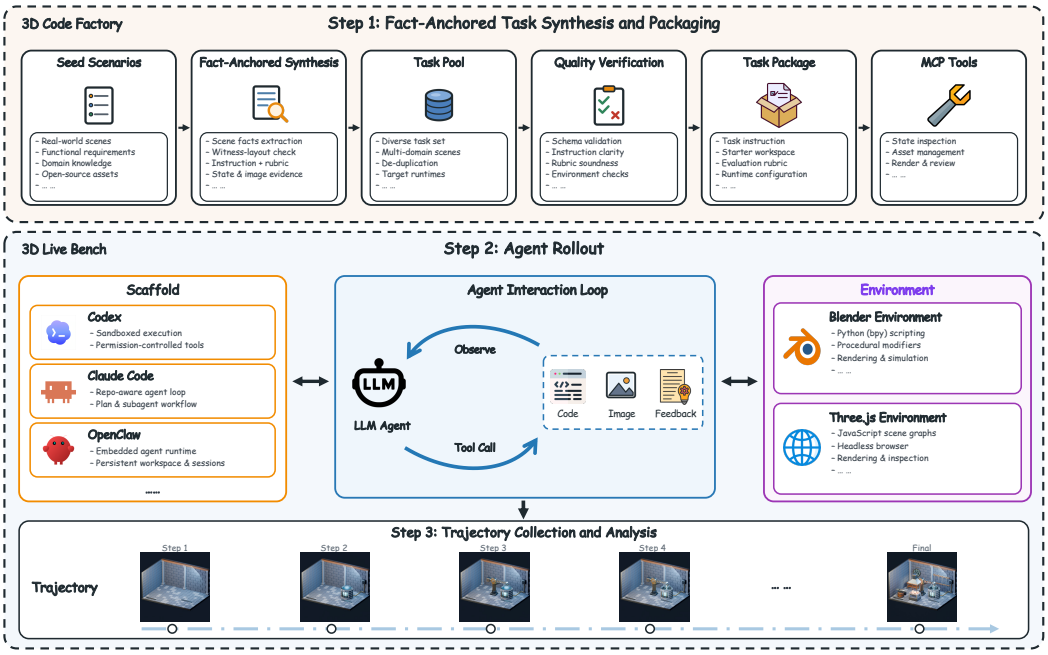

Figure 2: **Architectural overview of 3D Code Factory and 3D Live Bench.** 3D Code Factory synthesizes fact-anchored task packages from seed scenarios. 3D Live Bench runs coding agents in their native scaffolds, where they edit code, inspect scene state, and review rendered views.

Following these principles, we organize each generated assignment into a self-contained task package. The package bundles a natural language task instruction, an isolated initial workspace containing runtime configurations and basic entrypoints, and a withheld evaluation rubric. The workspace equips the agent with native runtime capabilities across Three.js and Blender, while the withheld rubric ensures that evaluation remains strictly objective and contamination-free.

3.2 Fact-Anchored Task and Rubric Synthesis

We use an LLM-driven synthesis pipeline to turn a seed scenario into a task package. We begin with seed scenarios curated from publicly available community-contributed resources, expand the seed scenario into candidate scenarios by LLM, and apply diversity filtering to prune near-duplicates. We construct each task around a shared set of scene facts that serves as the single source of truth for both generation and evaluation (Figure 2). Starting from a seed scenario (e.g., an Earth–Moon–Sun orrery), we extract a compact set of observable scene facts $\mathcal{F} = \{f_j\}_{j=1}^{J}$. Each fact defines an atomic observable condition partitioned by evidence channel: state-level facts capture runtime spatial extents, nested kinematic mounting hierarchies, and mechanical clearances, whereas image-level facts evaluate perceptual render properties, material differentiation, and visible part detail, paired with concrete operands, metric units, and verification tolerances.

Task blueprint construction. Conditioned on the extracted facts $\mathcal{F}$ and the target environment (Three.js or Blender), the LLM translates the factual specification into a natural-language task instruction $\mathcal{I}$. The blueprint specifies operational bounds, kinematic hierarchies, and presentation standards while leaving procedural geometry, scene graph construction, and animation logic open to the agent. Crucially, the facts remain canonical: the instruction frames the task naturally, but introduces no evaluation requirements beyond the underlying fact set $\mathcal{F}$.

Rubric grounding. We then decompose the same set of facts into a weighted evaluation rubric $\mathcal{R} = \{(c_k, w_k, \Phi_k)\}_{k=1}^{K}$, where c_k denotes a required outcome (e.g., orbital configuration, gear transmission), w_k indicates its relative importance weight, and Φ_k is a set of binary criteria resolving

to a clean pass or fail. Each criterion is explicitly grounded in one or more source facts and mapped to its most direct evidence channel: quantitative properties such as orbit radii, gap clearances, and relative distances are verified against the scene internal state, whereas visible properties such as material appearance, illumination, framing, and component visibility are evaluated from rendered images. This clear separation eliminates the pitfalls of inferring hidden 3D geometry from pixels or visual appearance from raw scene metadata.

Rubric validation. Each task package clears two validation gates before qualification. A programmatic structure check verifies schema completeness, ensuring required fields are populated, identifiers remain unique, and fact-rubric references resolve cleanly. Subsequently, an LLM review audits semantic qualities that resist static validation, such as task clarity, semantic alignment, physical buildability, scene completeness, and rubric weighting soundness. Tasks failing either gate are directly discarded.

4 3D LIVE BENCH

3D Live Bench provides an extensible suite of procedural 3D tasks to evaluate agents' modeling and code generation capabilities. The core benchmark comprises 36 curated tasks across three domains (Dynamics, Material and Physics, and Spatial Structure), challenging agents on articulated motion, spatial transformations, physical materials, and hierarchical geometric constraints. Additionally, an automated pipeline yields an extended pool of 488 validated tasks (3D Live Bench-Extended) for continual benchmark expansion.

4.1 BENCHMARK SCOPE AND MODELING PARADIGMS

Three.js tasks use JavaScript to programmatically construct and render 3D scenes in a browser, whereas Blender tasks use Python's `bpy` API to construct and render scenes within Blender.

4.2 AGENT INTERACTION PROTOCOL

3D Live Bench models procedural 3D construction as an interactive, tool-mediated software engineering workflow within agents' native harness environments (Figure 2). The evaluation protocol coordinates harness-native execution with graphics-engine feedback across three phases:

Workspace Setup and Tools. Each evaluation episode instantiates an isolated sandbox workspace $\mathcal{W}_0$ provisioned with task instruction $\mathcal{I}$, runtime dependencies, and starter entrypoints (`index.html` for Three.js or `candidate.py` for Blender) configuring canonical camera rigs. Agents operate through command-line scaffolds (e.g., Codex, Claude Code, OpenCode) with standard shell, filesystem, and code-editing tools. Sandboxes are supplemented with a uniform set of 3D Model Context Protocol (MCP) tools for scene querying and rendering, while the ground-truth rubric $\mathcal{R}$ is strictly withheld to prevent test-set leakage.

Closed-Loop Rollout. Given the initialized workspace, the agent interacts with the environment through a closed-loop rollout using its scaffold interface. At each step, it edits and executes code, inspects scene state, and uses visual feedback to act, self-correct, and refine deliverables. Vision serves as an interface for perceiving the environment, verifying results, and advancing tasks, helping agents adapt their next actions to the changes they observe. An auxiliary `render_and_review` tool can additionally provide visual feedback based on the task instruction.

Termination and Evaluation. A run terminates when the agent signals completion or reaches its budget limit. All intermediate agent messages and tool trajectories are recorded, and benchmark metrics are computed solely from the final workspace state. The saved entrypoint is executed headlessly to extract the terminal scene state and canonical render.

4.3 EVALUATION METHOD

We evaluate deliverables across three separate metrics: execution success, rubric compliance, and visual quality.

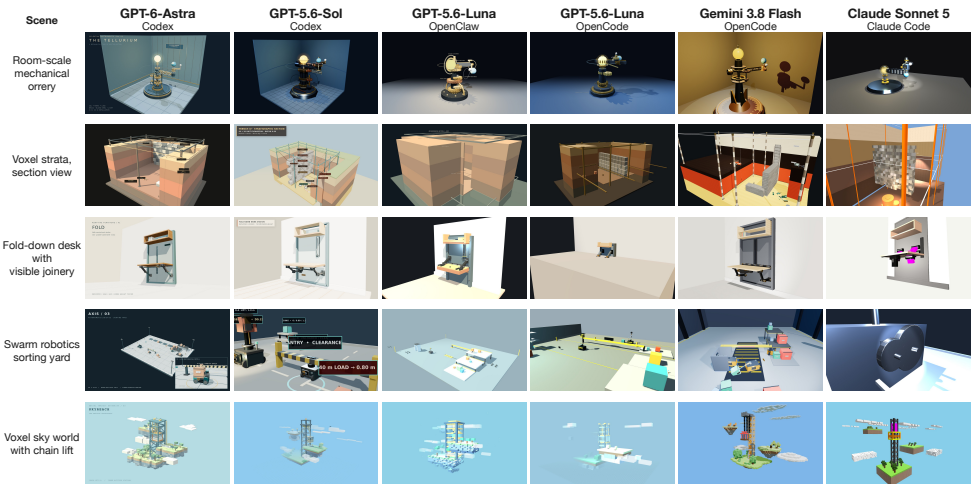

Figure 3: **Qualitative comparison on Three.js tasks across model–scaffold pairs.** Rows denote tasks and columns denote evaluated configurations.

Program Execution Rate. Program execution rate is a binary check that the saved entrypoint runs successfully and produces a valid 3D artifact and terminal render.

Rubric Compliance. Rubric compliance measures how well a scene satisfies the task instructions. Requirements are grouped into three priority tiers with weights 10, 8, and 6; each requirement's weight is divided evenly among its binary criteria. For state-sourced criteria (Φ_{state}), the multimodal judge evaluates the recorded scene state for quantitative properties such as object dimensions, bounding-box extents, coordinate distances, and parent–child relations. For image-sourced criteria (Φ_{image}), it evaluates the final render for perceptual properties such as materials, lighting, and visibility. The score is the earned weight divided by the total possible weight.

Visual Quality. We additionally evaluate visual quality to assess the overall perceptual quality of the render. Its nine dimensions cover visible construction (form, scene development, craft), presentation (materials, lighting, camera), and whole-image quality (coherence, aesthetics, render fidelity). This fine-grained decomposition separates 3D geometric modeling from camera framing, lighting, and render artifacts, pinpointing whether defects stem from spatial construction or presentation. A vision-language model rates each dimension on an anchored 0–10 scale (references at 2, 4, 6, 8, 10); fixed weights yield a normalized score in $[0, 1]$, calibrated against human preferences (Sec. 5).

5 Experiments

5.1 Experimental Setup

Evaluated Models and Scaffolds. We evaluate a representative set of recent agentic models from OpenAI (GPT-6-Astra, GPT-5.6-Sol, and GPT-5.6-Luna) (OpenAI, 2026c;b;a); Anthropic (Claude Sonnet 5 and Claude Opus 5) (Anthropic, 2026b;a); Google (Gemini 3.8 Flash and Gemini 3.7 Flash) (Google, 2026b;a); and DeepSeek (V4.1-Flash and V4-Pro) (DeepSeek-AI, 2026a;b). We pair the OpenAI models with Codex (OpenAI, 2025) and OpenClaw (OpenClaw Project, 2026), Claude with Claude Code (Anthropic, 2025), Gemini with OpenCode (SST, 2025), and DeepSeek with Codex and OpenCode; V4.1-Flash is also evaluated with OpenClaw. Results are reported separately for each model–scaffold pair to account for their distinct interaction workflows.

Task Execution and Evaluation. Each run is limited to 60 minutes, with a 30-minute inactivity timeout. At termination, we render the final workspace and evaluate the deliverable for program execution rate, rubric compliance, and visual quality, using GPT-5.6-Luna as the multimodal judge.

Table 2: Results on 3D Live Bench. Dyn./Mat./Struct.: Dynamics/Material/Structure; RC/VQ: rubric compliance/visual quality. Overall reports execution rate, aggregate scores, and mean cost per task. Dark/light gray mark best/second-best; best is bolded.

Model	Blender						Three.js						Overall			Cost ($)
	Dyn.		Mat.		Struct.		Dyn.		Mat.		Struct.		Exec (%)	Aggregate		
	RC	VQ	RC	VQ	RC	VQ	RC	VQ	RC	VQ	RC	VQ		RC	VQ	
Claude Code																
Claude-Sonnet-5	.47	.34	.49	.40	.48	.38	.75	.42	.58	.47	.68	.49	**100.0**	.560	.408	5.509
Claude-Opus-5	.61	.41	.40	.39	.61	.43	.42	.54	.57	.50	.53	.44	**100.0**	.526	.444	14.354
Codex																
GPT-6-Astra	.76	**.72**	.78	**.67**	.73	**.69**	.64	**.67**	.71	**.66**	.67	.66	**100.0**	.721	**.680**	1.231
GPT-5.6-Sol	.78	.63	.77	.61	.67	.62	**.93**	.65	.74	.60	.78	.57	**100.0**	.772	.614	0.855
GPT-5.6-Luna	.86	.64	**.80**	.57	**.82**	.65	.81	.56	**.75**	.58	.76	.58	**100.0**	**.804**	.601	**0.023**
DeepSeek-V4-Pro	.63	.42	.53	.42	.42	.37	.42	.57	.39	.45	.57	.51	**100.0**	.499	.447	0.433
DeepSeek-V4.1-Flash	.47	.37	.37	.31	.31	.33	.53	.63	.33	.42	.54	.62	94.4	.419	.428	0.235
OpenCode																
GPT-5.6-Sol	.79	.65	.79	.64	.54	.52	.77	**.67**	.74	.59	.68	.61	**100.0**	.718	.611	0.775
GPT-5.6-Luna	.85	.60	.63	.55	.79	.57	.71	.59	.58	.50	.73	.53	**100.0**	.723	.560	0.036
Gemini-3.8-Flash	.77	.62	.67	.57	.48	.47	.84	.65	.68	.56	**.79**	.64	**100.0**	.695	.580	16.237
Gemini-3.7-Flash	.76	.61	.58	.54	.63	.48	.71	.65	.47	.45	.75	.51	97.2	.651	.541	3.941
DeepSeek-V4-Pro	.59	.44	.51	.47	.55	.46	.54	.62	.51	.52	.67	.62	**100.0**	.561	.511	1.601
DeepSeek-V4.1-Flash	.82	.65	.56	.48	.64	.60	.59	.64	.53	.56	.69	.65	**100.0**	.643	.591	0.412
OpenClaw																
GPT-6-Astra	.73	.72	.70	.66	.59	.68	.57	.66	.69	.66	.72	**.67**	**100.0**	.668	.676	2.043
GPT-5.6-Sol	.83	.68	.70	.61	.61	.60	.67	.58	.65	.63	.65	.57	**100.0**	.691	.615	0.509
GPT-5.6-Luna	**.89**	.61	.73	.56	.73	.59	.64	.53	.49	.41	.75	.54	**100.0**	.719	.549	0.072
DeepSeek-V4.1-Flash	.54	.37	.24	.27	.62	.55	.62	.61	.40	.40	.59	.56	**100.0**	.496	.447	0.110

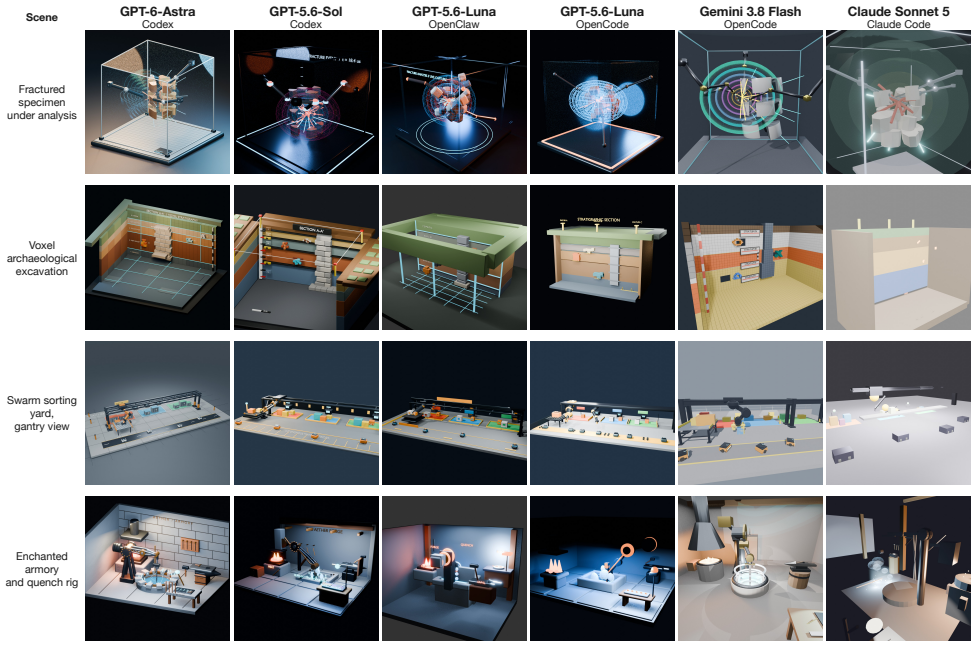

Figure 4: **Qualitative comparison on Blender tasks across model–scaffold pairs.** Rows denote scenes and columns denote evaluated configurations.

5.2 Main Results

Challenges in 3D Live Bench. Table 2 shows that while 15 of 17 evaluated configurations achieve 100% program execution, agents struggle to fulfill fine-grained task requirements, with mean rubric compliance between 0.40 and 0.80. Task completion does not always track visual quality: stronger models build visually compelling scenes in few turns yet leave specific requirements unaddressed. Under Codex, GPT-6-Astra achieves the highest visual quality (0.680 VQ) but misses detailed metric and functional criteria (0.721 RC), whereas GPT-5.6-Luna reaches the highest compliance (0.804

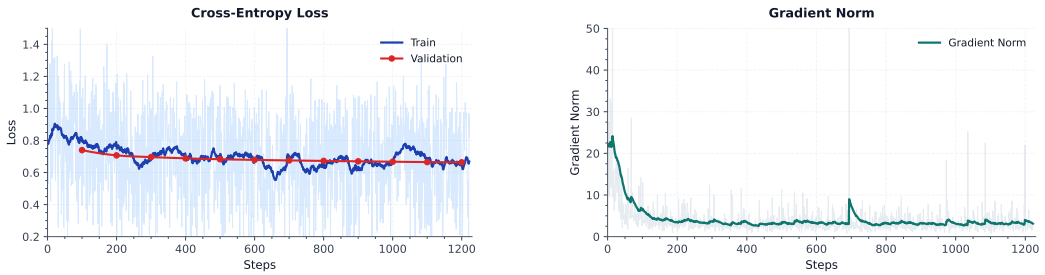

Figure 5: **SFT training dynamics for Qwen3-8B.** Training loss (left) and gradient norm (right).

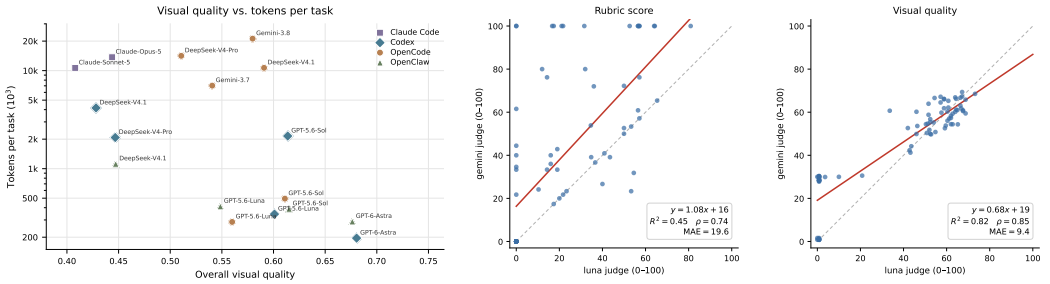

Figure 6: **Token efficiency and cross-judge agreement.** Left: visual quality versus tokens per task; right: rubric and visual-quality scores from two judges.

RC) by repairing requirements across multiple turns at lower polish (0.601 VQ). Failure modes differ across domains: in Dynamics, weaker models respond to inoperative mechanisms with cosmetic camera or lighting changes while stronger models reconstruct parent–child coordinate frames; Material and Physics yields the lowest compliance (mean RC < 0.60), where procedural geometry suffers from self-occlusion, untextured planar surfaces, and flat illumination that washes out fine hardware. In Spatial Structure, the primary challenge is constraint regression during iterative editing, where localized repairs to enlarge features or close visual gaps inadvertently violate coupled global limits such as slope gradients and clearance bounds.

Scaffold Matters as Much as Base Model. A base model's capability translates unevenly into task completion across scaffolds. With GPT-5.6-Sol fixed, Codex, OpenCode, and OpenClaw achieve 0.772, 0.718, and 0.691 RC, respectively, despite comparable VQ around 0.61. Conversely, DeepSeek-V4.1-Flash inverts this scaffold ranking, strongly favoring OpenCode (0.643 RC) over OpenClaw (0.496) and Codex (0.419). Inspection of agent trajectories reveals marked differences in whether models advance from planning to active execution. In two matched DeepSeek Blender tasks, Codex runs persist in read-only inspection and planning until reaching the execution timeout, whereas OpenCode runs construct the initial scene and revise the requested assemblies. Across GPT-5.6-Sol rollouts, behaviors range from halting after an initial unsatisfactory render to systematically rebuilding component groups, including their surfaces, supports, and joints. OpenCode also outperforms Codex on specific matched tasks, showing that an aggregate scaffold advantage does not guarantee superior feedback utilization across every task.

Performance vs. Cost. Figure 6 (left) illustrates trade-offs in token and cost efficiency across models and scaffolds. Under Codex, GPT-6-Astra and GPT-5.6-Luna maintain compact interaction footprints (under 400K tokens per task), enabling Astra to combine the highest VQ (0.680) with the lowest token usage (195.7K per task), while Luna attains the highest RC (0.804) at the lowest cost ($0.023 per task) aided by lower unit pricing and prompt caching. Conversely, multi-turn debugging can amplify expenditure: OpenCode with Gemini-3.8-Flash requires roughly three times the tokens and four times the cost of Gemini-3.7-Flash ($16.24 vs. $3.94) for marginal gains (+0.044 RC, +0.039 VQ), as repeated script and framing revisions inflate cumulative prompts. Scaffold choice also shifts cost for the same model, with GPT-5.6-Luna's per-task cost rising from $0.023 in Codex to $0.072 in OpenClaw.

5.3 Scaled Synthesis and Trajectory Fine-Tuning

We use 3D Code Factory to scale synthesis to 500 scenarios (1,000 candidate tasks), curating 3D Live Bench-Extended. To guarantee evaluation rigor, we audit every candidate against nine quality criteria; 488 tasks across 470 categories pass all checks. Most candidate rejections stem from rubric flaws, specifically contradictory geometric constraints (soundness) and unobservable target criteria (checkability), rather than scene generation.

Advancing agent capabilities through fine-tuning. Beyond evaluation, multi-turn rollouts from frontier models on 3D Live Bench supervise small model learning. We collect 786 multi-turn trajectories (10,872 interaction turns) in total, balanced 50/50 between Three.js and Blender; we then filter out execution errors, black or blank renders, and rollouts failing multimodal specification checks, while discarding auxiliary probing calls to keep only code editing, rendering, and visual-feedback turns. We fine-tune Qwen3-8B (Yang et al., 2025a) and evaluate generalization on 20 held-out tasks from 3D Live Bench-Extended under OpenCode (Table 3); Figure 5 shows the training dynamics. Fine-tuning improves execution, rubric compliance, and visual quality across both tracks. Base models suffer from open-loop "blind coding", rewriting code without visual inspection and triggering script crashes and blank outputs. In contrast, fine-tuning instills an inspect-and-repair loop that eliminates crashes, improves Blender execution from 20.0% to 60.0%, and solves complex assemblies where the base model fails.

Table 3: Performance on 3D Live Bench-Extended.

Track	Model	Exec (%) ↑	RC ↑	VQ ↑
Three.js	Base	10.0	.014	.122
	+ SFT	**20.0**	**.030**	**.162**
Blender	Base	20.0	.000	.024
	+ SFT	**60.0**	**.004**	**.081**

5.4 Evaluation Reliability

Comparison of evaluation paradigms. To assess modality dependence, we benchmark unimodal evaluators against our combined reference across 100 3D episodes (Table 4). State-only inspection yields low rubric coverage (14.4% R-Cov, $\tau_b = +0.454$) from blind spots in lighting, materials, and occlusion. Conversely, Screenshot+VLM attains higher coverage (40.3%) but incurs a 25.0% false-positive rate from 2D spatial ambiguity. Their near-zero rank correlation ($\tau_b = +0.087$) confirms unimodal evaluators capture distinct failure modes, necessitating joint evaluation.

Table 4: Comparison of evaluation paradigms.

Paradigm	Mean R-Cov (%)	τ_b vs. Ref.	FPR (%) ↓
State-only	14.4	+0.454	0.0
Screenshot+VLM	40.3	+0.400	25.0
Ours (joint ref.)	**57.2**	**1.000**†	**0.0**†

Human preference alignment. We validate the Visual Quality metric through human pairwise comparisons across 44 pairs (547 total judgments). Our judge aligns with the human majority on 74.4% of the pairs, outperforming the leave-one-out human baseline of 67.5%. For decisive pairs ($\Delta\text{VQ} \geq 0.05$), agreement increases to 81.8% (vs. 71.4% for humans). On near-ties, agreement stands at 50.0%, mirroring inter-rater (50.8%) and leave-one-out (56.4%) consensus baselines (Table 5).

Table 5: Pairwise agreement (%).

Method	All	Decisive	Near-tie
Random	50.0	50.0	50.0
Human vs. human	57.7	60.2	50.8
Human vs. majority	67.5	71.4	56.4
VQ judge (ours)	**74.4**	**81.8**	50.0

Judge score agreement. We re-score 82 deliverables with a second judge under identical inputs (Figure 6, right). While Gemini scores low-end criteria more generously, rankings agree closely and visual quality clusters tightly along the diagonal across both judges.

6 Conclusion

We introduce 3D Code Factory, which synthesizes procedural 3D tasks with verifiable multimodal rubrics, and 3D Live Bench, a benchmark evaluating coding agents across Blender and Three.js. In 3D Code Factory, rubrics decouple simulation-state inspection from rendered-image review, scaling to 488 verified tasks across 500 scenarios. On 3D Live Bench, evaluations show that while code execution is saturated, rubric compliance remains a primary bottleneck (0.40–0.80) that diverges from visual quality, with scaffold architecture fundamentally shaping success. Together, they provide a solid foundation to advance autonomous 3D coding agents.

Ethics Statement. The benchmark comprises synthetic 3D scenes and procedural geometry, containing no personally identifiable data or proprietary code. Submitted programs execute arbitrary code, requiring execution within isolated container environments.

Reproducibility Statement. We will release the framework, the curated task suite, the runtime configurations, and code upon acceptance.

AI Use Statement. Generative AI was used to synthesize candidate tasks in 3D Code Factory and for language polishing during manuscript preparation. All experimental designs, code, and analyses were developed and verified by the authors.

REFERENCES

Anthropic. Claude Code: An agentic coding tool for the terminal. https://docs.claude.com/en/docs/claude-code/overview, 2025. Source: https://github.com/anthropics/claude-code.

Anthropic. Introducing Claude Opus 5. https://www.anthropic.com/news/claude-opus-5, 2026a.

Anthropic. Introducing Claude Sonnet 5. https://www.anthropic.com/news/claude-sonnet-5, 2026b.

Aili Chen, Chi Zhang, Junteng Liu, Jiangjie Chen, Chengyu Du, Yunji Li, Ming Zhong, Qin Wang, Zhengmao Zhu, Jiayuan Song, Ke Ji, Junxian He, Pengyu Zhao, and Yanghua Xiao. DIVE: Scaling diversity in agentic task synthesis for generalizable tool use. *arXiv preprint arXiv:2603.11076*, 2026.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Zihao Cheng, Hongru Wang, Zeming Liu, Xinyi Wang, Xiangrong Zhu, Yuhang Guo, Wei Lin, Jeff Z. Pan, and Yunhong Wang. Terminal-World: Scaling terminal-agent environments via agent skills. *arXiv preprint arXiv:2605.20876*, 2026.

DeepSeek-AI. Introducing DeepSeek-V4.1-Flash: Smarter, faster, more efficient. https://www.deepseek.com/en/news/deepseek-v4-1-flash/, 2026a.

DeepSeek-AI. DeepSeek-V4-Pro GA release. https://api-docs.deepseek.com/news/news260813/, 2026b.

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

Yiming Du, Yuxin Jiang, Tao Yuan, Jianbo Dai, Shaowei Wang, Jierun Chen, Chaofan Tao, Xianzhi Yu, Lifeng Shang, Kam-Fai Wong, Xiaohui Li, and Haoli Bai. Lego-RL: Harness-native reinforcement learning for coding agents. *arXiv preprint arXiv:2608.17393*, 2026.

Yuhao Du, Shunian Chen, Wenbo Zan, Peizhao Li, Mingxuan Wang, Dingjie Song, Bo Li, Yan Hu, and Benyou Wang. BlenderLLM: Training large language models for computer-aided design with self-improvement. *arXiv preprint arXiv:2412.14203*, 2024.

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.

Google. Gemini 3.7 Flash: Our most intelligent workhorse model. https://blog.google/innovation-and-ai/models-and-research/gemini-models/introducing-gemini-3-7-flash/, 2026a.

Google. Introducing Gemini 3.8 Flash and 3.8 Flash Cyber. https://blog.google/innovation-and-ai/models-and-research/gemini-models/3-8-flash-and-3-8-flash-cyber/, 2026b.

Chunchao Guo, Jinpeng Li, Yang Li, and Zilong Huang. Worldclaw: Agentic 3d open-world generation at scale. *arXiv preprint arXiv:2608.05248*, 2026.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025a.

Lianghong Guo, Yanlin Wang, Caihua Li, Wei Tao, Pengyu Yang, Jiachi Chen, Haoyu Song, Duyu Tang, and Zibin Zheng. SWE-Factory: Your automated factory for issue resolution training data and evaluation benchmarks. *arXiv preprint arXiv:2506.10954*, 2025b.

Ziniu Hu, Ahmet Iscen, Aashi Jain, Thomas Kipf, Yisong Yue, David A Ross, Cordelia Schmid, and Alireza Fathi. SceneCraft: An LLM agent for synthesizing 3D scenes as Blender code. In *Forty-first International Conference on Machine Learning*, 2024.

Zhanbo Hua, Yifan Yao, Weihao Xie, Yongchi Zhao, Minghao Liu, Ruizhi Qiu, Zhewei Huang, Zun Wang, Yiyan Ji, Yunhai Ye, Letian Zhu, Xinping Lei, Han Li, Zhiyuan Ma, Zili Wang, Zhaoxiang Zhang, and Jiaheng Liu. CLI-Universe: Towards verifiable task synthesis engine for terminal agents. *arXiv preprint arXiv:2606.22883*, 2026.

Ian Huang, Guandao Yang, and Leonidas Guibas. BlenderAlchemy: Editing 3D graphics with vision-language models. In *European Conference on Computer Vision*, pp. 297–314. Springer, 2024.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2E-Gym: Procedural environments and hybrid verifiers for scaling open-weights SWE agents. *arXiv preprint arXiv:2504.07164*, 2025.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024.

Yuval Kirstain, Adam Polyak, Uriel Singer, Shahbuland Matiana, Joe Penna, and Omer Levy. Pick-a-pic: An open dataset of user preferences for text-to-image generation. *arXiv preprint arXiv:2305.01569*, 2023.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*, 2026.

Parker Liu, Chenxin Li, Zhengxin Li, Yipeng Wu, Wuyang Li, Zhiqin Yang, Zhenyuan Zhang, Yunlong Lin, Sirui Han, and Brandon Y. Feng. IR3D-Bench: Evaluating vision-language model scene understanding as agentic inverse rendering. *arXiv preprint arXiv:2506.23329*, 2025.

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.

Mike A. Merrill et al. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*, 2026.

OpenAI. Codex CLI: A lightweight coding agent for the terminal. https://developers.openai.com/codex/cli, 2025. Source: https://github.com/openai/codex.

OpenAI. GPT-5.6 Luna. https://openai.com/index/gpt-5-6/, 2026a. Model-family page covering GPT-5.6 Luna.

OpenAI. Previewing GPT-5.6 Sol: A next-generation model. https://openai.com/index/previewing-gpt-5-6-sol/, 2026b.

OpenAI. GPT-6 Astra: A new generation of intelligence. https://openai.com/index/gpt-6-astra/, 2026c.

OpenClaw Project. OpenClaw: A self-hosted personal AI assistant platform. https://github.com/openclaw/openclaw, 2026.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-Gym. *arXiv preprint arXiv:2412.21139*, 2024.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023.

Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. EnvScaler: Scaling tool-interactive environments for LLM agent via programmatic synthesis. *arXiv preprint arXiv:2601.05808*, 2026.

SST. OpenCode: The open source AI coding agent. https://opencode.ai, 2025. Source: https://github.com/sst/opencode.

Chunyi Sun, Junlin Han, Weijian Deng, Xinlong Wang, Zishan Qin, and Stephen Gould. 3D-GPT: Procedural 3D modeling with large language models. In *2025 International Conference on 3D Vision (3DV)*, pp. 1253–1263. IEEE, 2025a.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, Ben Kao, Guohao Li, Junxian He, Yu Qiao, and Zhiyong Wu. OS-Genesis: Automating GUI agent trajectory construction via reverse task synthesis. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5555–5579, 2025b. URL https://aclanthology.org/2025.acl-long.277/.

Tencent WorkBuddy Bench Team. Tencent WorkBuddy Bench: A multi-domain coding-agent benchmark with contamination-resistant task construction. *arXiv preprint arXiv:2607.20911*, 2026. URL https://arxiv.org/abs/2607.20911.

Dunwei Tu, Hongyan Hao, Hansi Yang, Yihao Chen, Yi-Kai Zhang, Zhikang Xia, Yu Yang, Yueqing Sun, Xingchen Liu, Furao Shen, Qi Gu, Hui Su, and Xunliang Cai. ScaleEnv: Scaling environment synthesis from scratch for generalist interactive tool-use agent training. *arXiv preprint arXiv:2602.06820*, 2026.

Zhaoyang Wang, Canwen Xu, Boyi Liu, Yite Wang, Siwei Han, Zhewei Yao, Huaxiu Yao, and Yuxiong He. Agent world model: Infinity synthetic environments for agentic reinforcement learning. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Siwei Wu, Yizhi Li, Yuyang Song, Wei Zhang, Yang Wang, Riza Batista-Navarro, Xian Yang, Mingjie Tang, Bryan Dai, Jian Yang, and Chenghua Lin. Large-scale terminal agentic trajectory generation from dockerized environments. *arXiv preprint arXiv:2602.01244*, 2026.

Tong Wu, Guandao Yang, Zhibing Li, Kai Zhang, Ziwei Liu, Leonidas Guibas, Dahua Lin, and Gordon Wetzstein. GPT-4V(ision) is a human-aligned evaluator for text-to-3D generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22227–22238, 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Wu_GPT-4Vision_is_a_Human-Aligned_Evaluator_for_Text-to-3D_Generation_CVPR_2024_paper.html.

Xiaoshi Wu, Yiming Hao, Keqiang Sun, Yixiong Chen, Feng Zhu, Rui Zhao, and Hongsheng Li. Human preference score v2: A solid benchmark for evaluating human preferences of text-to-image synthesis. *arXiv preprint arXiv:2306.09341*, 2023.

Jingxu Xie, Dylan Xu, Xuandong Zhao, and Dawn Song. AgentSynth: Scalable task generation for generalist computer-use agents. *arXiv preprint arXiv:2506.14205*, 2025.

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

Jiazheng Xu, Xiao Liu, Yuchen Wu, Yuxuan Tong, Qinkai Li, Ming Ding, Jie Tang, and Yuxiao Dong. Imagereward: Learning and evaluating human preferences for text-to-image generation. *arXiv preprint arXiv:2304.05977*, 2023.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a. URL https://arxiv.org/abs/2505.09388.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. SWE-smith: Scaling data for software engineering agents. *arXiv preprint arXiv:2504.21798*, 2025b.

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, 2026. URL https://arxiv.org/abs/2606.11152.

Yilun Yao, Xinyu Tan, Chao-Hsuan Liu, Yaoming Li, Zhengyang Wang, Wenhan Yu, Zhewen Tan, Yuxuan Tian, Guangxiang Zhao, Lin Sun, Xiangzheng Zhang, and Tong Yang. Harness-Bench: Measuring harness effects across models in realistic agent workflows. *arXiv preprint arXiv:2605.27922*, 2026. URL https://arxiv.org/abs/2605.27922.

Shaofeng Yin, Jiaxin Ge, Zora Zhiruo Wang, Chenyang Wang, Xiuyu Li, Michael J. Black, Trevor Darrell, Angjoo Kanazawa, and Haiwen Feng. Vision-as-inverse-graphics agent via interleaved multimodal reasoning. *arXiv preprint arXiv:2601.11109*, 2026.

Mengyu Zheng, Kai Han, Boxun Li, Haiyang Xu, Yuchuan Tian, Wei He, Hang Zhou, Jianyuan Guo, Hailin Hu, Lin Ma, Chao Xu, Guohao Dai, Lixue Xia, Yunchao Wei, Yunhe Wang, and Yu Wang. Claw-SWE-Bench: A benchmark for evaluating OpenClaw-style agent harnesses on coding tasks. *arXiv preprint arXiv:2606.12344*, 2026.

A TASK SCHEMATIC

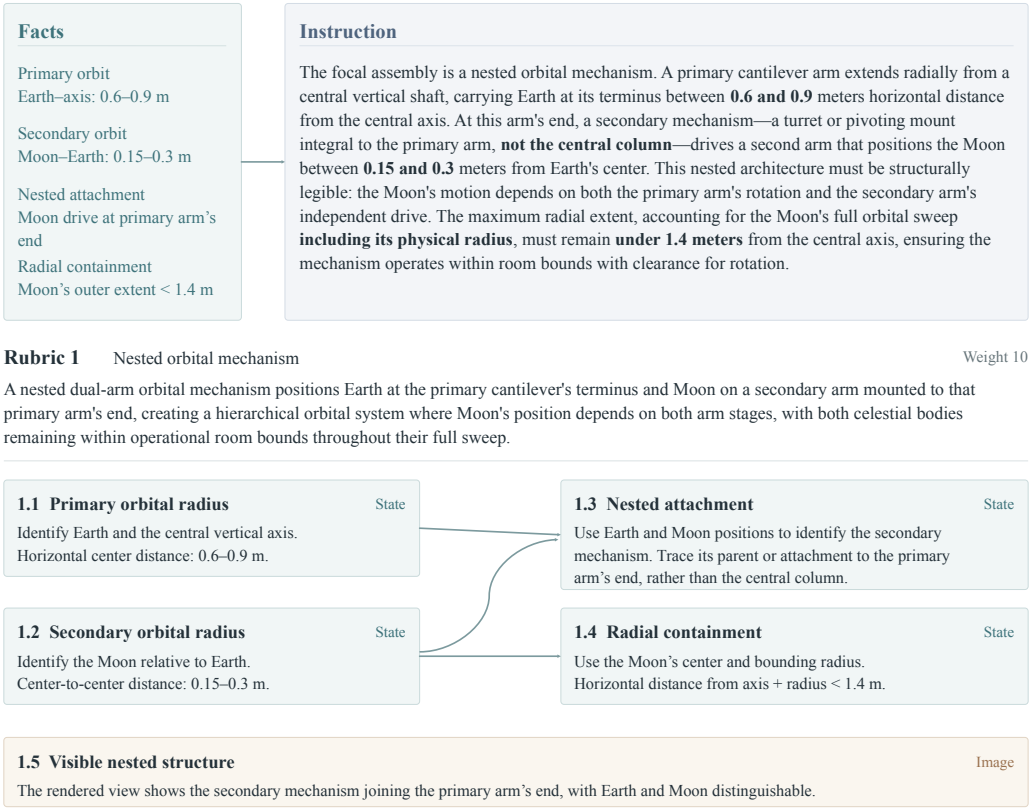

Figure 7: Task specification and verifiable rubric DAG. An illustrative assembly task (Nested Orbital Mechanism) decomposed into scene facts, natural-language instructions, and a dependency-gated rubric graph. The verification rubric incorporates four state-level checks (Φ_{state}: orbital radii, structural hierarchy, and spatial containment bounds) and one visual legibility check (Φ_{image}).

Figure 7 illustrates an end-to-end task specification alongside its verification schema. Each synthesized task decomposes complex assembly constraints into atomic, binary verifiable criteria. Specifically, geometric parameters and spatial containment tolerances are validated against the simulation state (Φ_{state}), whereas visual appearance and visibility are evaluated from the rendered image (Φ_{image}). Directed edges encode verification dependencies, enabling a robust, cascaded evaluation pipeline.

B TRAINING COMPUTE DETAILS

Training is conducted on one node equipped with 4 NVIDIA A800 GPUs over 1 epoch (approximately 13 hours) using the hyperparameters in Table 6.

Table 6: SFT hyperparameters for Qwen3-8B.

Hyperparameter	Value
Training sequence length	40,960 tokens
Global batch size	8
Training epochs	1
Optimizer	AdamW
Learning rate	2×10^{-6}
LR schedule	Constant
Weight decay	1×10^{-2}
Gradient clipping	1.0
Tensor parallel size	1
Pipeline parallel size	1
Sequence parallelism	Enabled
Random seed	42

C Task Synthesis Quality

We curate 3D Live Bench-Extended utilizing `DeepSeek-V4.1-Flash` (DeepSeek-AI, 2026a) configured with high reasoning effort. To eliminate redundancy, we apply embedding-based deduplication (cosine similarity > 0.92, pruning near-duplicate candidates), yielding 500 distinct scenarios with 1,000 tasks evenly distributed across Three.js and Blender. To ensure task validity and verification fidelity, all synthesized candidates are assessed against nine binary criteria categorized into three principal axes: (i) instruction quality (clarity, procedural buildability, completeness, and non-triviality), (ii) rubric rigor (metric checkability, soundness, and priority weighting), and (iii) cross-alignment (semantic alignment and setting coherence).

Quantitative analysis of the candidate pool reveals that rubric soundness constitutes the most stringent bottleneck (658/1,000 pass), followed by metric checkability (736) and semantic alignment (738). In contrast, procedural buildability and non-triviality demonstrate higher feasibility, passing 856 and 998 tasks respectively. The remaining criteria (setting coherence, completeness, and priority weighting) achieve 100% pass rates. Jointly requiring all nine checks yields 488 high-quality tasks (48.8% yield: 248 Three.js, 240 Blender).

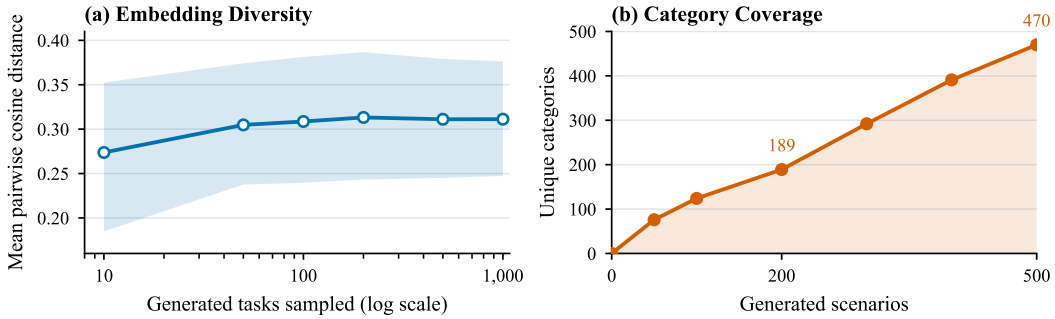

Figure 8: **Scalability and semantic diversity of 3D Code Factory.** (a) Embedding diversity: mean pairwise cosine distance across requirement embeddings increases from 0.27 to 0.31 as the pool scales to 1,000 tasks, indicating that 3D Code Factory resists mode collapse under sustained generation; the shaded region denotes the 10th–90th percentile interval. (b) Category coverage: cumulative unique domain categories grow near-linearly, yielding 470 distinct labels across 500 synthesized scenarios and demonstrating broad coverage of architectural, kinetic, and procedural modeling domains.

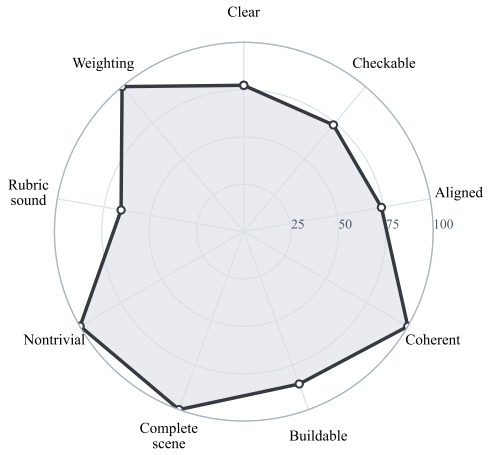

Figure 9: **Multi-dimensional pass rates across 1,000 candidate tasks in the automated quality review.** Radar contours indicate pass percentages across nine verification dimensions. Nontriviality and scene completeness exceed 99% pass rates, whereas metric checkability (73.6%) and rubric soundness (65.8%) form the primary filtering bottlenecks. Only candidates satisfying all nine criteria simultaneously, 488 tasks (48.8%), enter 3D Live Bench-Extended, reducing the risk of ungrounded or contradictory rubrics.

D BENCHMARK COMPOSITION

We construct 3D Live Bench from candidate tasks synthesized by 3D Code Factory using Claude Sonnet 5 (Anthropic, 2026b). From 50 proposed candidate tasks across 25 scenarios that pass structural validation and execute cleanly, automated quality filtering identifies and excludes 14 candidates exhibiting ambiguous visual specifications or redundant spatial constraints. Domain experts then inspect and verify the remaining 36 high-quality tasks (15 for Three.js and 21 for Blender across 22 scenarios), confirming physical solvability, scene richness, and rubric fidelity before final inclusion. Figures 10 and 11 summarize the composition and distribution of the benchmark.

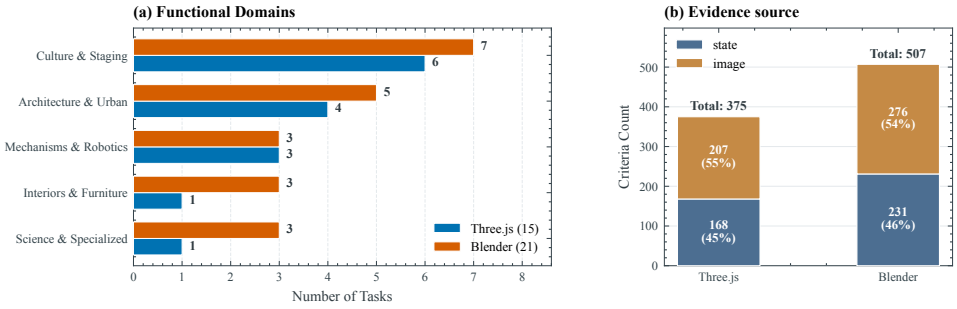

Figure 10: **3D Live Bench task and evidence source distribution. (a)** Task distribution across five procedural domains. **(b)** Evidence breakdown: approximately 45% of criteria verify deterministic scene-state invariants, while the remainder evaluate visual properties (lighting, materials, and occlusion) from rendered views.

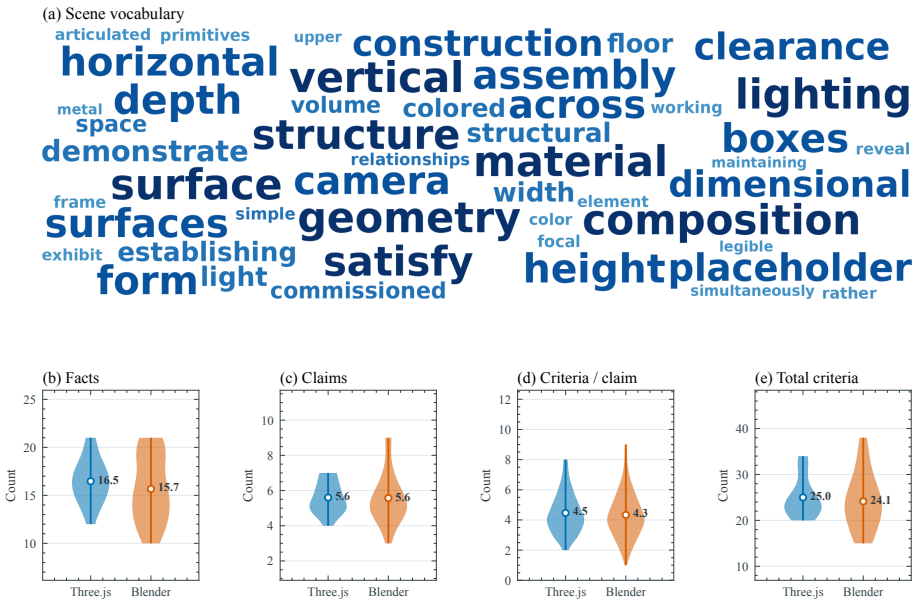

Figure 11: **3D Live Bench lexical and structural complexity distributions. (a)** Instruction vocabulary, emphasizing spatial-relational constraints over object naming. **(b)–(e)** Distributions of facts, claims, criteria per claim, and total criteria. Three.js tasks average more facts due to explicit scene-graph transform hierarchies, while total criteria per task remain balanced ($\sim$25).

E Extended Qualitative Comparison

We present more qualitative results across five grids, varying the scaffold and foundation model.

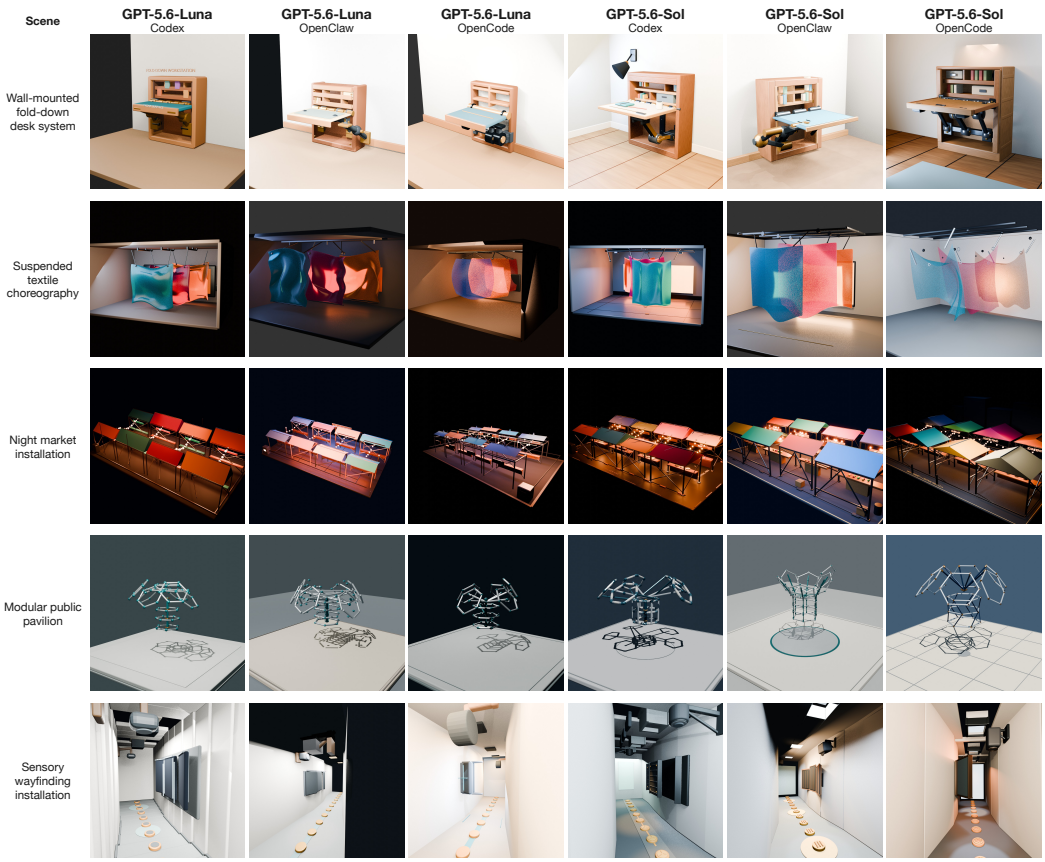

Figure 12: **Controlled scaffold comparison in Blender.** The left and right triplets use GPT-5.6-Luna and GPT-5.6-Sol, respectively. OpenClaw leaves the fold-down desks unanchored and compresses the modular pavilion around a central support, while OpenCode camera views in the sensory-wayfinding corridor are dominated by nearby wall geometry.

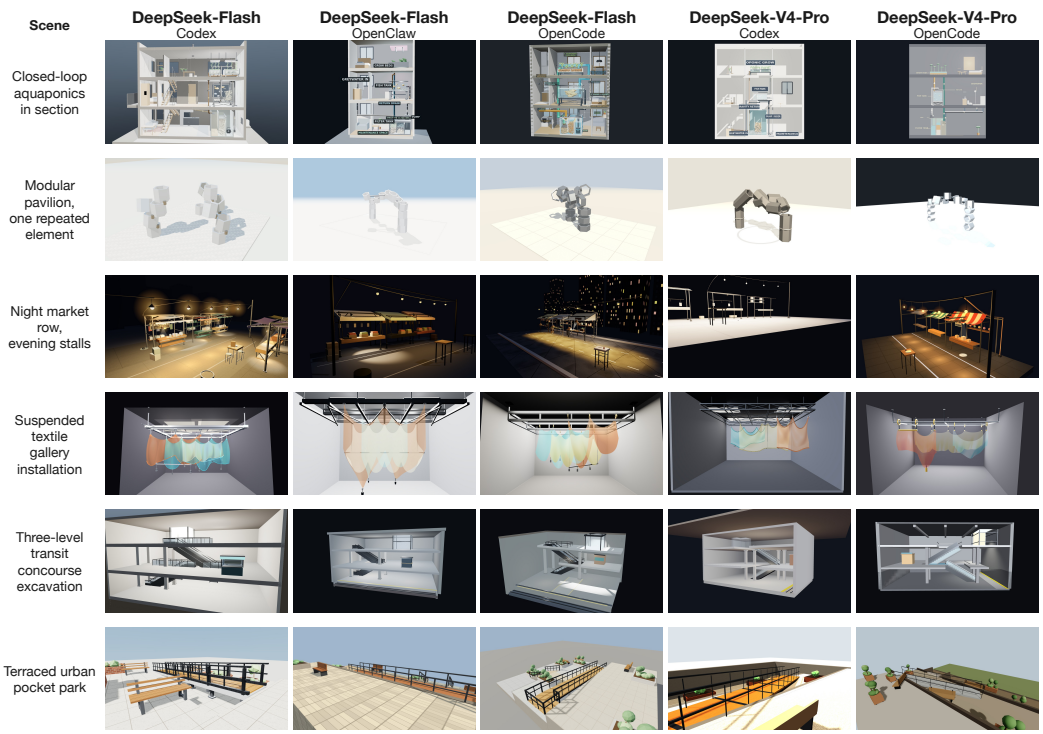

Figure 13: **Qualitative comparison of DeepSeek variants across scaffolds.** Rows show representative scenes, while columns compare DeepSeek-V4.1-Flash and DeepSeek-V4-Pro under Codex, OpenClaw, and OpenCode.

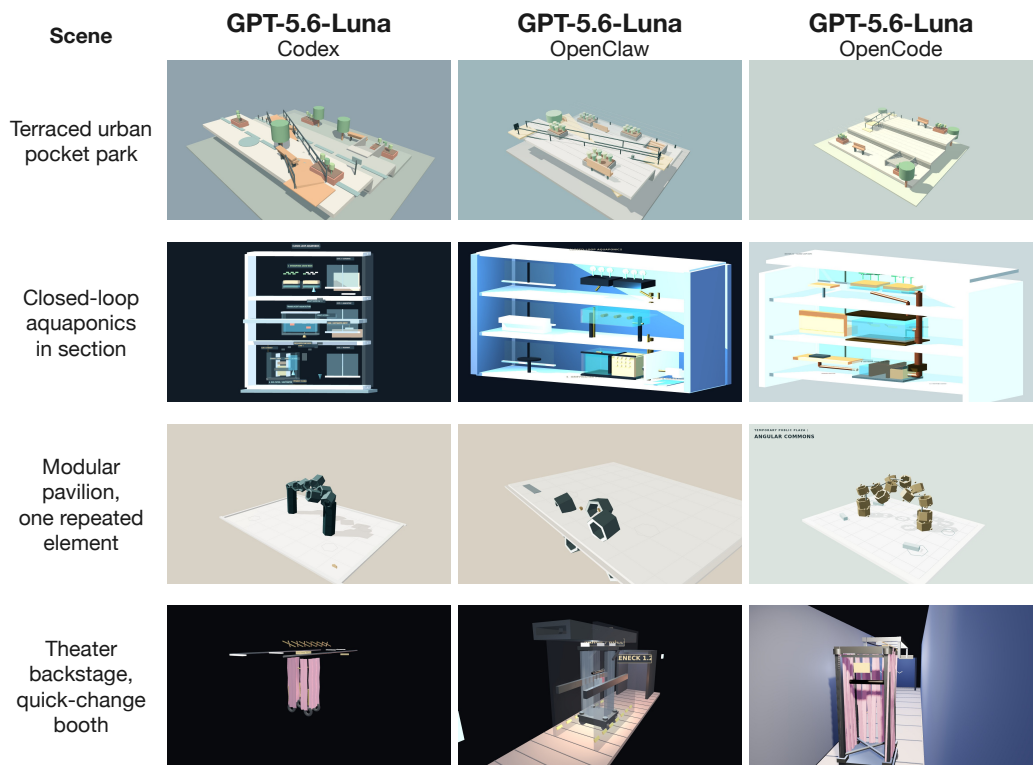

Figure 14: **Controlled scaffold comparison on Three.js tasks.** Codex keeps all four scenes centered and their enclosures legible. OpenClaw pulls back from the pocket park and aquaponics cabinet, whereas OpenCode breaks the cabinet frame and places the theater-booth camera against surrounding wall geometry.

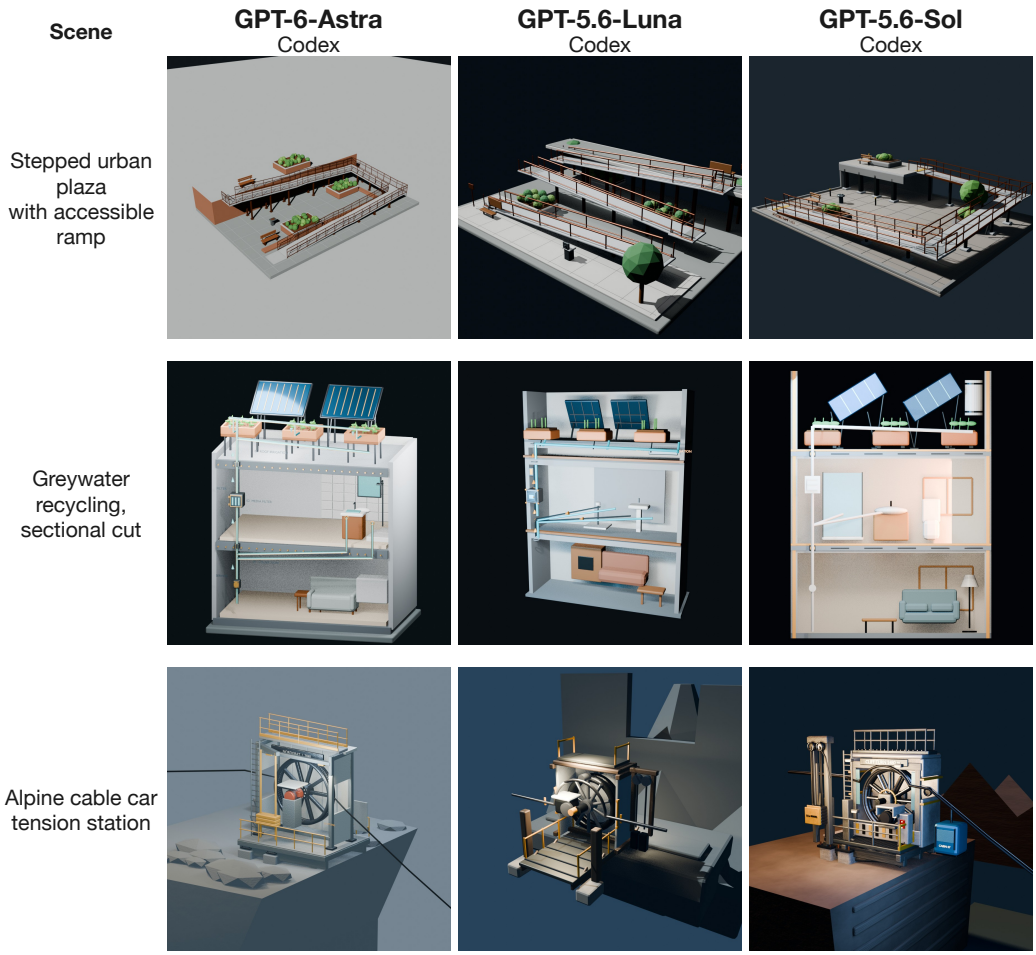

Figure 15: **Foundation-model comparison with the Codex scaffold.** In the greywater-recycling cutaway, GPT-6-Astra resolves solar-panel cell grids and interior filtration pipes, whereas GPT-5.6-Sol reduces the interior to broad planar surfaces and simplified components.

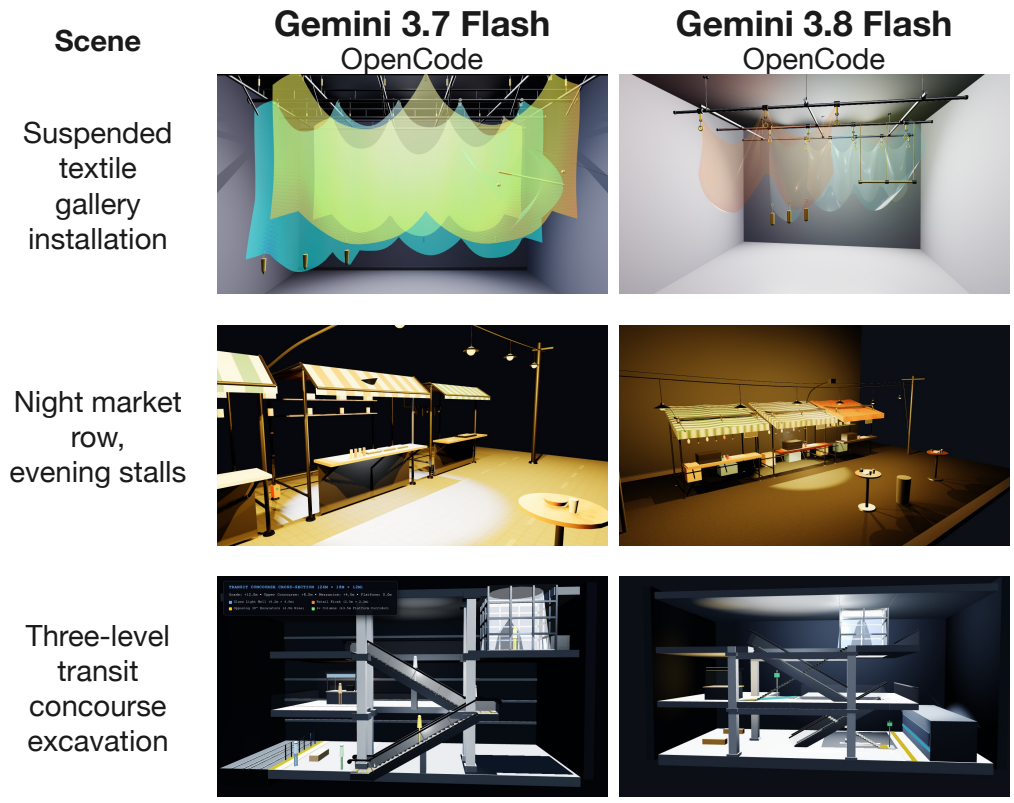

Figure 16: **Generational comparison of Gemini 3.7 Flash and 3.8 Flash in OpenCode.** In the three-level transit concourse, Gemini 3.8 Flash pulls the camera away from the diagonal support that crosses the Gemini 3.7 Flash view. The later model is less obstructed there but produces a sparser suspended-textile gallery.

F HUMAN ALIGNMENT

Pairs are stratified by score gaps, with 25% placed in a near-zero control band. In total, independent evaluators contributed 547 pairwise annotations across 44 pairs (27 tasks), yielding 11–14 judgments per pair. Following standard pairwise validation protocols for visual reward models (Kirstain et al., 2023; Xu et al., 2023; Wu et al., 2023; 2024), we report both judgment-level and majority agreements on decisive pairs, with 95% bootstrap confidence intervals resampled over tasks.

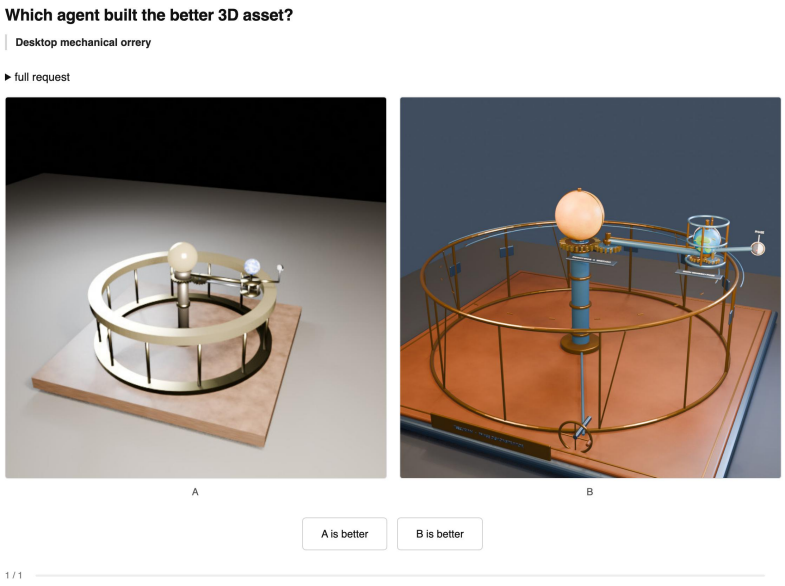

Figure 17: **Pairwise human-evaluation interface for visual-quality calibration.** Pairwise trial presented to raters: two anonymized renders of the same task from different model–scaffold pairs, with the visually preferred asset selected.

G EVALUATION DETAILS

For rubric compliance, we assign each requirement an importance weight and divide it evenly across its constituent checks, each judged pass or fail. The overall task score is the sum of the partial weights earned divided by the total possible weight.

For Visual Quality, the evaluator rates nine dimensions from 0 to 10: geometry and form; scene completeness and richness; materials and surfaces; lighting and depth; composition and camera; detail and craft; visual coherence; aesthetic quality; and image quality. The first six dimensions and aesthetic quality receive weight 0.12 each; visual coherence and image quality receive weight 0.08 each (summing to 1.0). The weighted result is divided by 10 to report Visual Quality on a 0–1 scale.

H TOKEN AND COST ACCOUNTING

Table 7 reports the token and cost accounting behind the results of Table 2, with one row per model–scaffold pair and the dimensions of the rubric in each environment. Figure 18 plots estimated API cost per task against overall rubric compliance.

Table 7: Average per-task token usage (10^3) and cost (USD) across dimensions. Dyn., Mat., and Struct. represent Dynamics, Material, and Structure. Bold with dark shading marks the best (lowest) value per column; light shading marks the second-best.

Model	Blender					Three.js					Overall	
	Tokens (10^3)				Cost ($)	Tokens (10^3)				Cost ($)	Tokens (10^3)	Cost ($)
	Dyn.	Mat.	Struct.	Total		Dyn.	Mat.	Struct.	Total			
Claude Code												
Claude-Sonnet-5	6162.5	8313.2	15217.7	9897.8	3.863	2288.8	8119.5	24708.8	11705.7	7.812	10651.1	5.509
Claude-Opus-5	5740.4	14709.5	16196.0	12215.3	16.080	17540.6	13928.2	15909.6	15792.8	11.937	13705.9	14.354
Codex												
GPT-6-Astra	**141.6**	**122.9**	**163.5**	**142.7**	1.020	**254.4**	**142.0**	413.5	**270.0**	1.528	**195.7**	1.231
GPT-5.6-Sol	4017.1	4138.3	1950.5	3368.6	1.210	653.8	355.7	**339.4**	449.6	0.356	2152.4	0.855
GPT-5.6-Luna	260.1	269.3	266.7	265.3	**0.019**	276.8	531.3	555.0	454.4	**0.027**	344.1	**0.023**
DeepSeek-V4-Pro	3686.0	2861.2	1588.9	2712.0	0.490	2394.5	668.0	513.6	1192.0	0.353	2078.7	0.433
DeepSeek-V4.1-Flash	3345.8	9219.9	6120.9	6228.9	0.305	526.4	2057.3	1238.5	1274.1	0.137	4164.4	0.235
OpenCode												
GPT-5.6-Sol	716.1	340.1	328.7	461.7	0.728	354.6	291.2	976.2	540.7	0.842	494.6	0.775
GPT-5.6-Luna	253.4	224.9	228.1	235.5	0.031	272.4	355.4	444.4	357.4	0.043	286.3	0.036
Gemini-3.8-Flash	14550.6	24721.7	12508.0	17260.1	13.186	22276.6	23086.2	34650.9	26671.2	20.509	21181.4	16.237
Gemini-3.7-Flash	6751.9	6201.6	1950.0	4967.8	3.032	24548.7	1054.0	4133.7	9912.1	5.215	7028.0	3.941
DeepSeek-V4-Pro	8260.5	6765.5	8515.2	7847.1	1.339	19780.4	23164.0	25982.3	22975.6	1.968	14150.6	1.601
DeepSeek-V4.1-Flash	6854.1	8749.0	14037.0	9880.0	0.510	6148.7	6838.0	22401.9	11796.2	0.275	10678.4	0.412
OpenClaw												
GPT-6-Astra	240.4	251.0	281.6	257.6	1.775	287.7	341.2	371.1	333.4	2.418	289.2	2.043
GPT-5.6-Sol	374.3	398.8	333.5	368.9	0.537	270.9	261.1	723.2	418.4	0.470	389.5	0.509
GPT-5.6-Luna	338.9	319.5	281.1	313.2	0.058	647.6	453.8	555.0	552.1	0.091	412.7	0.072
DeepSeek-V4.1-Flash	899.8	919.9	2237.2	1352.3	0.116	1169.5	564.3	653.2	795.6	0.101	1120.4	0.110

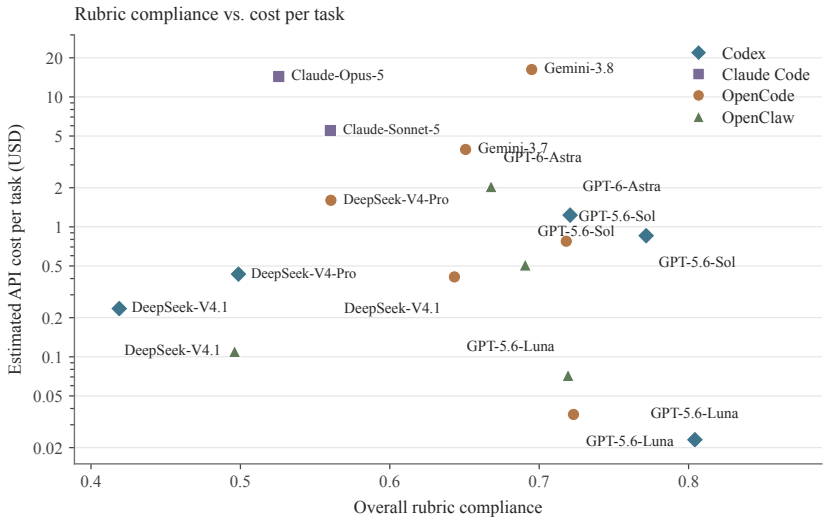

Figure 18: Estimated API cost per task versus overall rubric compliance across model–scaffold pairs; the vertical axis is logarithmic.

I Generated Task Format

We package each generated task as a self-contained task directory:

```
<task-key>/
   task.md
   environment/
      Dockerfile
      environment_manifest.json
      initial_workspace/
         index.html  (Three.js) or candidate.py  (Blender)
   verifier/
      verifier.md
      verify_scene.py
      rubric.json
```

We organize each task package into three components. The root document `task.md` records task metadata alongside the natural-language task instruction provided to the agent. The `environment/` directory contains container configurations, runtime dependencies, and initial workspace files (`index.html` for Three.js or `candidate.py` for Blender). Finally, `verifier/` holds the automated scoring scripts and the evaluation rubric, which stay outside the agent's workspace.

The starter entrypoint is the same file for every task in an environment. It clears the scene and leaves a camera, a light and, in Three.js, the render loop in place; the agent writes the scene itself.

initial_workspace/index.html (Three.js)

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>3DCode Three.js Environment</title>
  <style>html,body{margin:0;height:100%;overflow:hidden;background:#111}canvas{
    display:block}</style>
</head>
<body>
<script type="importmap">{"imports":{"three":"https://unpkg.com/three@0.128.0/
    build/three.module.js","three/addons/":"https://unpkg.com/three@0.128.0/
    examples/jsm/"}}</script>
<script type="module">
import * as THREE from "three";
import { OrbitControls } from "three/addons/controls/OrbitControls.js";
const scene = new THREE.Scene();
const camera = new THREE.PerspectiveCamera(50, innerWidth / innerHeight, 0.1,
    100);
camera.position.set(3, 2, 4);
const renderer = new THREE.WebGLRenderer({antialias: true});
renderer.setPixelRatio(Math.min(devicePixelRatio, 2));
renderer.setSize(innerWidth, innerHeight);
document.body.appendChild(renderer.domElement);
scene.add(new THREE.HemisphereLight(0xffffff, 0x202030, 2));
const controls = new OrbitControls(camera, renderer.domElement);
controls.enableDamping = true;
renderer.setAnimationLoop(() => {
  controls.update();
  renderer.render(scene, camera);
});
addEventListener("resize", () => {
  camera.aspect = innerWidth / innerHeight;
  camera.updateProjectionMatrix();
  renderer.setSize(innerWidth, innerHeight);
});
</script>
</body>
</html>
```

initial_workspace/candidate.py (Blender)

```
import bpy
from mathutils import Vector

bpy.ops.object.select_all(action="SELECT")
bpy.ops.object.delete(use_global=False)

bpy.ops.object.light_add(type="AREA", location=(4, -4, 6))
bpy.context.object.data.energy = 1000
bpy.context.object.data.shape = "DISK"
bpy.context.object.data.size = 5

bpy.ops.object.camera_add(location=(4, -6, 3.5))
camera = bpy.context.object
camera.rotation_euler = (Vector((0, 0, 0.75)) - camera.location).to_track_quat(
    "-Z", "Y"
).to_euler()
bpy.context.scene.camera = camera
```

J REPRESENTATIVE TASK EXAMPLE

We reproduce one representative task exactly as 3D Code Factory emitted it: the verbatim instruction the agent receives, the scene facts the rubric is anchored to, and the full rubric that stays withheld. The task ships as an `environment_manifest.json` together with an intentionally empty `initial_workspace/index.html` that carries the Three.js runtime and render loop.

Task instruction $\mathcal{I}$

Create a 3D scene as a self-contained HTML file using Three.js.

Design and construct a room-scale mechanical orrery demonstrating nested celestial motion through physical clockwork. The installation occupies a space measuring 3.0 meters width by 3.0 meters depth by 2.5 meters height, presenting a working model of gravitational hierarchy: Earth revolving around a central luminous sun, with the Moon in orbit around Earth. The construction must achieve this through a dual-arm cantilever mechanism driven by a visible, staged gear train, where every element of mechanical causation can be traced by eye from the central drive through to the planetary bodies.

The focal assembly is a nested orbital mechanism. A primary cantilever arm extends radially from a central vertical shaft, carrying Earth at its terminus between 0.6 and 0.9 meters horizontal distance from the central axis. At this arm's end, a secondary mechanism—a turret or pivoting mount integral to the primary arm, not the central column—drives a second arm that positions the Moon between 0.15 and 0.3 meters from Earth's center. This nested architecture must be structurally legible: the Moon's motion depends on both the primary arm's rotation and the secondary arm's independent drive. The maximum radial extent, accounting for the Moon's full orbital sweep including its physical radius, must remain under 1.4 meters from the central axis, ensuring the mechanism operates within room bounds with clearance for rotation.

The central column rises from a stable base platform—circular or polygonal, with diameter between 1.0 and 1.5 meters positioned on the ground plane—to a height between 0.9 and 1.5 meters. This column houses or supports a vertical drive shaft fitted with a staged gear train: at least two gears positioned at different vertical heights separated by no less than 0.08 meters, creating a visible mechanical hierarchy. The gears must not be placeholder cylinders; they require developed geometry with teeth, defined pitch circles, and cylindrical form that reads as functional hardware. At least three distinct gears must be visible in the final composition, positioned at different locations—along the central axis, on the arms, or at their junctions—and at least one pair must be positioned close enough (their bounding surfaces separated by under 0.02 meters) to suggest authentic meshing. The viewer must be able to trace mechanical power flow: from the central shaft through these gears and structural connections to the arms, and finally to the planetary bodies they carry.

The primary arm operates at a height that maintains at least 0.3 meters clearance above the base platform's top surface and at least 0.15 meters clearance below the mechanism's topmost component, allowing full

rotation without collision. Earth appears as a sphere with radius between 0.06 and 0.12 meters, mounted at the primary arm's end and visibly distinct from the structural mechanism. The Moon is a smaller sphere with radius between 0.025 and 0.05 meters, carried by the secondary arm and proportionally subordinate to Earth. The sun at the mechanism's center must be visibly luminous or emissive—reading unmistakably as a light source, not merely a colored sphere—contrasting with the reflective metallic character of the surrounding clockwork.

Materials establish functional hierarchy: metallic surfaces for the mechanism components (gears, shafts, arms, column, and structural fittings), a distinct emissive or glowing material for the sun, and at least one separate material for the planetary bodies. This differentiation must be visually clear, distinguishing celestial objects from the apparatus that drives them. The construction must show articulated cross-sections and silhouettes: arms with structural depth or geometric complexity, not flat struts; gears with thickness and tooth profiles, not flat discs; joints and mounts with mechanical plausibility. Placeholder geometry—uniform colored boxes, featureless spheres for mechanism parts, or primitive cylinders standing in for gears—does not satisfy the requirement for mechanical authenticity.

Lighting and composition must reveal the machine's depth and complexity. The final view should present foreground and background elements in readable spatial relationship, with at least one component partially occluding another to establish depth. The nested arm structure must be comprehensible: a viewer should see how the secondary moon mechanism mounts to the primary arm, not the central column, and how the two orbital scales relate. The staged gear train, the cantilever extension, and the vertical clearances that prevent collision should all be visually accessible. All components must remain within the vertical envelope from ground level to 2.4 meters height.

The finished orrery must stand as a coherent demonstration of clockwork celestial mechanics: a machine where structure, motion paths, and mechanical relationships are simultaneously legible, where the hierarchy of sun, Earth, and Moon is expressed through nested orbital geometry driven by visible gears and shafts, and where the composition conveys both the scale of the room and the precision of the mechanism it contains.

J.1 Scene Facts

The instruction above is derived from 18 scene facts. Each fact states one observable condition with its operands, units and tolerances, and declares whether its evidence comes from scene state or from the rendered image.

`room_bounds` [state]

The anchor frame measures 3.0m width × 3.0m depth × 2.5m height

`sun_luminous` [image]

The sun at the mechanism's center is visibly luminous or emissive, reading as a light source distinct from the reflective metal mechanism and planetary bodies

`base_support` [state]

A circular or polygonal base platform with diameter between 1.0m and 1.5m supports the central column, positioned on the ground plane

`central_column` [state]

A vertical central column or housing extends from the base to a height between 0.9m and 1.5m, enclosing or supporting the drive shaft

`primary_arm_orbital` [state]

A primary arm extends radially from the central mechanism, positioning Earth's center at a horizontal distance between 0.6m and 0.9m from the central vertical axis

`moon_arm_secondary` [state]

A secondary arm extends from a mechanism at the primary arm's end, positioning the Moon's center at a distance between 0.15m and 0.3m from Earth's center

`radial_containment` [state]

The maximum horizontal distance from the central axis to any point on the Moon (including its radius) is less than 1.4m, ensuring the full sweep remains within room bounds with clearance

`earth_sphere` [image]

Earth is represented as a sphere with radius between 0.06m and 0.12m, positioned at the end of the primary arm and visibly distinct from the mechanism structure

`moon_sphere` [image]
Moon is represented as a sphere with radius between 0.025m and 0.05m, positioned at the end of the secondary arm and visibly smaller than Earth

`visible_gears` [image]
At least three distinct gears are visible in the final view, showing mechanical detail such as teeth, cylindrical form, or gear-like profile, positioned at different locations in the mechanism

`gear_staging` [state]
At least two gears are positioned at different vertical heights along or near the central axis, separated by at least 0.08m vertically, suggesting a staged drive mechanism

`gear_proximity` [state]
At least one pair of gears is positioned with their bounding surfaces separated by less than 0.02m, suggesting functional meshing

`arm_clearance` [state]
The primary arm at its pivot height is vertically separated from the base top surface by at least 0.3m, and from the mechanism's topmost component by at least 0.15m, allowing full rotation without collision

`mechanical_causation` [image]
The visual arrangement shows mechanical causation: a viewer can trace a path from central shaft through visible gears and arm structures to the planetary bodies, with gears positioned along or connected to the arms

`material_distinction` [image]
At least three distinct materials are visible: one for metallic mechanism components (gears, shafts, arms), one for the sun, and at least one for planetary bodies, creating visual hierarchy

`nested_assembly` [state]
The secondary moon mechanism is mounted on or integrated with the primary arm's end, not directly on the central column, creating a nested orbital structure where the moon's position depends on both arm positions

`vertical_fit` [state]
All mechanism components have vertical positions (including their extents) between 0m and 2.4m, remaining within the room height with clearance

`composition_depth` [image]
The final view shows depth and mechanical complexity: foreground and background elements are distinguishable, with at least one component partially occluding another, and the nested arm structure is readable

J.2 Evaluation Rubric

The withheld rubric holds 5 weighted claims proved by 22 binary criteria. Each claim spreads its weight evenly across its criteria.

Claim 1 (weight 10, 2 points per criterion). A nested dual-arm orbital mechanism positions Earth at the primary cantilever's terminus and Moon on a secondary arm mounted to that primary arm's end, creating a hierarchical orbital system where Moon's position depends on both arm stages, with both celestial bodies remaining within operational room bounds throughout their full sweep.

`c1_primary_earth_distance` [state]
Earth's center is positioned at a horizontal distance between 0.6m and 0.9m from the central vertical axis, establishing the primary orbital radius.
Evidence: Measure horizontal distance from central axis (x^2+y^2)^0.5 to Earth's center position. Verify distance falls within [0.6m, 0.9m].

`c1_secondary_moon_distance` [state]
Moon's center is positioned at a distance between 0.15m and 0.3m from Earth's center, establishing the secondary orbital radius.
Evidence: Measure 3D Euclidean distance between Earth's center and Moon's center. Verify distance falls within [0.15m, 0.3m].

`c1_moon_arm_mounting` [state]
The secondary moon arm or its mounting mechanism shares a parent hierarchy or spatial attachment to the primary arm's end region, not to the central column, confirmed by comparing the moon mechanism's root attachment point to the primary arm's endpoint region versus the central column's bounds.

Evidence: Identify secondary arm or moon mechanism components by proximity to Moon. Trace parent hierarchy or identify spatial attachment point. Compare attachment location to primary arm's end region (within 0.2m of Earth position) versus central column bounds (within 0.15m of central axis). Confirm attachment is to primary arm region, not central column.

Depends on: `c1_primary_earth_distance`, `c1_secondary_moon_distance`

`c1_radial_containment` [state]

The maximum horizontal distance from the central axis to any point on the Moon including its physical radius is less than 1.4m, ensuring the full nested orbital sweep remains within room bounds with clearance.

Evidence: Identify Moon's bounding radius. Calculate maximum horizontal extent: (Moon center horizontal distance from axis) + (Moon bounding radius). Verify total < 1.4m.

Depends on: `c1_secondary_moon_distance`

`c1_nested_structure_visible` [image]

The final view shows the nested arm structure where the secondary moon mechanism visibly mounts to or integrates with the primary arm's end, not appearing as an independent branch from the central column, with the spatial relationship between primary arm, Earth, secondary mechanism, and Moon readable.

Evidence: Observe the visual connection between secondary moon mechanism and primary arm's end region near Earth. Confirm secondary mechanism appears attached to or integrated with the primary arm's terminus, not branching independently from the central column. Verify the nested hierarchy is visually comprehensible.

Claim 2 (weight 10, 2.5 points per criterion). A staged vertical gear train with at least three visible gears positioned at multiple heights creates traceable mechanical causation from the central drive shaft through the arm structures to the planetary bodies, with at least one gear pair positioned close enough to suggest functional meshing.

`c2_vertical_staging` [state]

At least two gears are positioned at different vertical heights along or near the central axis (within 0.3m horizontal distance), with vertical separation of at least 0.08m between their centers, establishing a staged drive mechanism.

Evidence: Identify gear components by geometry type or naming. Select gears near central axis (horizontal distance < 0.3m from axis). Measure vertical (z) coordinate separation between at least one pair. Verify separation $\geq$ 0.08m.

`c2_meshing_proximity` [state]

At least one pair of distinct gears has their bounding surfaces separated by less than 0.02m, suggesting functional meshing interface.

Evidence: Identify all gear components. For each pair, calculate minimum distance between their bounding volumes. Verify at least one pair has separation < 0.02m.

`c2_three_gears_visible` [image]

At least three distinct gears are visible in the final view, showing mechanical detail such as teeth profiles, cylindrical gear form, or gear-like geometry, positioned at different locations in the mechanism (not overlapping identical positions).

Evidence: Count distinct gears visible in rendered view. Verify each shows gear-appropriate geometry (cylindrical form, teeth, mechanical detail). Confirm at least three gears are distinguishable and positioned at different locations within the mechanism.

`c2_mechanical_causation` [image]

The visual arrangement shows traceable mechanical causation: a viewer can follow a path from the central shaft through visible gears positioned along or connected to the arm structures, terminating at the planetary bodies, with gears appearing integrated with the drive system rather than isolated.

Evidence: Trace visual path from central vertical shaft region through visible gears to primary arm, and from primary arm through any intermediate gears or mechanisms to Earth and Moon positions. Verify gears appear positioned along this mechanical chain (on shafts, at arm junctions, or on arm structures) rather than floating independently. Confirm the arrangement conveys mechanical power transmission.

Depends on: `c2_three_gears_visible`

Claim 3 (weight 8, 1.33 points per criterion). The mechanism maintains operational clearances with the primary arm positioned at a working height that provides at least 0.3m clearance above the base platform and 0.15m below the topmost component, while all components remain within the vertical room envelope and the base platform provides stable dimensional support.

`c3_base_dimensions` [state]

A base platform with horizontal extent (diameter for circular, or minimum bounding circle diameter for polygonal) between 1.0m and 1.5m is positioned on or near the ground plane (bottom surface between 0m and 0.1m height).

Evidence: Identify base platform component by position (lowest major structural component near central axis). Measure horizontal extent: for circular base use diameter, for polygonal calculate minimum bounding circle diameter from corner points. Verify extent within [1.0m, 1.5m]. Verify bottom surface z-coordinate within [0m, 0.1m].

`c3_column_height` [state]

A vertical central column or housing extends from the base to a height between 0.9m and 1.5m, measured from the ground plane to the column's top extent.

Evidence: Identify central column component (vertical structure near central axis extending from base). Measure vertical extent from bottom (near ground) to top. Verify height within [0.9m, 1.5m].

Depends on: `c3_base_dimensions`

`c3_room_envelope` [state]

All mechanism components remain within the spatial envelope defined by the room bounds: horizontal positions within 1.5m of the central axis (fitting within the 3.0m $\times$ 3.0m footprint with clearance) and vertical positions between 0m and 2.5m height.

Evidence: Survey all mechanism components. Calculate maximum horizontal distance from central axis to any component extent: verify $\leq$ 1.5m. Verify all component vertical extents (min z and max z) satisfy: min z $\geq$ 0m and max z $\leq$ 2.5m. Confirm mechanism fits within the 3.0m width $\times$ 3.0m depth $\times$ 2.5m height room bounds.

`c3_arm_lower_clearance` [state]

The primary arm at its pivot height or mounting point is positioned at least 0.3m above the base platform's top surface, providing clearance for rotation without collision with the base.

Evidence: Identify primary arm component by connection to Earth position. Determine arm's pivot height (z-coordinate at central axis connection or minimum z along arm near axis). Identify base platform top surface height. Calculate vertical separation: (arm pivot height) - (base top height). Verify separation $\geq$ 0.3m.

Depends on: `c3_base_dimensions`

`c3_arm_upper_clearance` [state]

The primary arm at its pivot height is positioned at least 0.15m below the mechanism's topmost component, providing overhead clearance for rotation without collision.

Evidence: Determine primary arm pivot height (from previous criterion). Identify maximum z-coordinate of all mechanism components. Calculate vertical separation: (topmost component z) - (arm pivot height). Verify separation $\geq$ 0.15m.

Depends on: `c3_arm_lower_clearance`

`c3_vertical_fit` [state]

All mechanism components have their full vertical extents between 0m and 2.4m, remaining within the room height with clearance below the 2.5m ceiling.

Evidence: Survey all mechanism components (base, column, shaft, gears, arms, celestial bodies). Verify each component's minimum z $\geq$ 0m and maximum z $\leq$ 2.4m.

Claim 4 (weight 8, 2 points per criterion). A luminous sun at the mechanism's center contrasts visually with reflective metallic mechanism components and distinct planetary body materials, establishing a three-tier material hierarchy that distinguishes light source, mechanical apparatus, and celestial objects in the rendered view.

`c4_sun_luminous` [image]

The sun at the mechanism's center is visibly luminous or emissive in the final render, reading unmistakably as a light source with self-illumination distinct from reflective surfaces, not merely a colored sphere.

Evidence: Identify sun object at or near central axis base. Observe its appearance in final render. Verify it exhibits luminous or emissive qualities (glow, self-illumination, light emission visible against surrounding components) distinct from purely reflective metal surfaces.

`c4_material_distinction` [image]

At least three distinct material appearances are visible: one metallic reflective material for mechanism components (gears, shafts, arms, column), one material for the sun (established as luminous), and at least one separate material for planetary bodies (Earth and/or Moon), with these differences creating visual hierarchy that distinguishes functional groups.

Evidence: Identify material appearance of: (1) mechanism components (gears, shafts, arms, column) - verify metallic reflective character; (2) sun - verified luminous; (3) Earth and/or Moon - verify distinct from both

mechanism metal and sun. Count distinct material appearances. Verify at least three are present and the differences establish visual hierarchy between light source, apparatus, and celestial objects.

Depends on: `c4_sun_luminous`

`c4_earth_distinct` [image]

Earth appears as a sphere with visible radius suggesting scale between 0.06m and 0.12m, positioned at the primary arm's end and visually distinct from the mechanism structure through material, color, or surface treatment.

Evidence: Locate Earth sphere at primary arm terminus. Verify spherical form is visible. Assess apparent size suggests radius in range [0.06m, 0.12m] based on spatial context. Confirm Earth's appearance (material, color, surface treatment) is visually distinguishable from adjacent mechanism structure.

`c4_moon_distinct_smaller` [image]

Moon appears as a sphere with visible radius suggesting scale between 0.025m and 0.05m, positioned at the secondary arm's end and visibly smaller than Earth, establishing scale hierarchy between the two celestial bodies.

Evidence: Locate Moon sphere at secondary arm terminus. Verify spherical form is visible. Assess apparent size suggests radius in range [0.025m, 0.05m] based on spatial context. Directly compare Moon's apparent size to Earth's in the rendered view. Confirm Moon reads as proportionally smaller, establishing celestial hierarchy.

Depends on: `c4_earth_distinct`

Claim 5 (weight 6, 2 points per criterion). The final composition reveals mechanical depth and structural complexity through distinguishable foreground and background elements, partial occlusion establishing spatial layers, and readable articulation of the nested arm structure and gear train staging.

`c5_depth_occlusion` [image]

The final view shows compositional depth with foreground and background elements distinguishable through spatial separation, and at least one component partially occluding another, establishing layered spatial relationships.

Evidence: Observe final rendered view. Identify foreground elements (closer to camera) and background elements (further from camera) that are distinguishable by size, focus, or position. Verify at least one component partially occludes another component, creating visible depth through overlap. Confirm spatial layering is readable.

`c5_nested_structure_readable` [image]

The nested arm structure is readable in the final view: the viewer can distinguish the primary arm extending from the central column to Earth, and the secondary mechanism mounted at that arm's end driving the Moon's position, with the hierarchical mounting relationship visually comprehensible.

Evidence: Trace the nested arm structure visually: identify primary arm extending from central mechanism to Earth, identify secondary arm or mechanism at primary arm's end extending to Moon. Verify the hierarchical relationship (secondary mounted to primary, not both independently to column) is visually apparent through spatial arrangement and connection points.

`c5_mechanical_complexity` [image]

The view reveals mechanical complexity through visible articulation: gears show dimensional form with thickness and teeth profiles (not flat discs), arms show structural depth or geometric complexity (not simple flat struts), and multiple mechanism elements at different heights or positions are simultaneously visible, conveying the machine's three-dimensional construction.

Evidence: Assess mechanical articulation in rendered view: examine gears for visible thickness, cylindrical form, and tooth profiles; examine arms for structural depth, cross-sectional geometry, or geometric complexity beyond flat strips; verify multiple mechanism elements (gears, shaft sections, arm components) at different heights or positions are simultaneously visible. Confirm the overall impression conveys three-dimensional mechanical complexity rather than flat or placeholder geometry.

The rubric above is applied after task execution ends by the terminal evaluator described in Appendix L.3, which decides each criterion from the evidence source it declares.

K Scene-Interaction Interface

The tools are served by a supplemental Model Context Protocol provider. Both environments expose the same tool names and parameters and execute native Three.js or Blender code underneath; `execute_code` submits one complete program and replaces the program from the preceding call,

and the workspace artifact is saved automatically. These tools augment the scaffold's standard file-editing and terminal capabilities rather than replacing them.

Table 8: Supplemental scene-interaction API for Three.js and Blender. The five tools cover scene inspection, procedural construction, and intermediate visual feedback.

Tool	Description
`query_scene`	Return every renderable object and light in deterministic name order. Shape is the portable object kind (`mesh` or `light`); each object carries name, world-space location, rotation, and scale, color, roughness, and metallic values, bounds (null for non-geometry), and an is-light flag. The agent may request this structured scene snapshot whenever it needs programmatic inspection.
`bounding_box`	Return the min/max corners and dimensions of the named object's axis-aligned bounding box.
`measure_distance`	Compute the Euclidean distance between two objects (by name) or two $[x, y, z]$ points.
`execute_code`	The primary construction tool: submit free-form code in Three.js JavaScript (THREE r128 loaded) or Blender Python (`bpy`, `bmesh`, `mathutils`). Every call replaces prior `execute_code` output, so the code must build the full desired scene rather than patch earlier code.
`render_and_review`	Generate scene code, render it, and call the VLM for intermediate feedback against the task instruction: the largest visible discrepancies and concrete edit suggestions. `matches` is true only when no important visible correction remains. Pass `review=False` to render only and skip the VLM, so a model that reads images itself can inspect the screenshot directly; the tool never raises and returns `matches=False` on error.

The five tools are modular and callable on demand. The agent may interleave code execution, state inspection, and image rendering according to its current reasoning state; no fixed call order is imposed. Neither platform exposes internal worker paths.

L Prompt Templates

The following are the complete system prompts for task design, quality review, and final evaluation. Braced placeholders denote supplied inputs. Per-request user messages and corrective messages are retained with the implementation.

L.1 Task and Rubric Design

Facts, the instruction, and the rubric are produced in three requests; each stage starts from what the previous stage fixed.

Facts Prompt

Design one substantial procedural 3D coding task from the supplied scenario. The scenario is a creative starting point, not a finished specification.

First record a compact private scene_design with a developed focal assembly, characteristic internal construction and an integrated environment. Choose construction challenges that interact: fitting an assembly changes circulation and clearance; support and interfaces constrain shape; exposing internal detail affects composition and occlusion. Use the scenario's own relationships, not a checklist of unrelated challenges. Develop form, materials and lighting enough that a collection of named, colored boxes would miss the central achievement. The focal subject must require deliberate silhouette, cross-section, internal construction or articulated interfaces; independent primitive placement is not a sufficient central task.

Privately check that at least one feasible layout exists in a consistent frame and unit system — by constructing it, not by asserting it. Build one complete layout: give every constrained component a concrete position and size within the anchor frame, then verify each stated requirement against that layout. If any requirement cannot hold, revise the ranges or drop the requirement. This witness is not the task specification: retain essential dimensions, limits and relationships as facts, while leaving meaningful placement, shape, assembly and presentation choices for the builder to solve. Do not publish the witness as an exhaustive per-object AABB table. Avoid overspecifying an already constrained layout with redundant coordinates.

Lead with the requirements whose outcomes are visible in the commissioned final view. State the anchor frame's dimensions, the design rules, and the coupled requirements as functional relations between components — what must hold for the system to work: the corridor passes the thinner segment and blocks the thicker one, the arm's tip reaches the elevated target, the gripper grasps functionally, the gantry clears the tallest structure — rather than the structural recipe (absolute dimensions and verbatim attachments)

Task and Rubric Design: Facts Prompt (continued)

the builder should derive. A fact states what must hold between components, not how the builder must realize it. A fact that fixes an absolute position or attachment of a specific component, without a functional relation that requires it, is a HOW disguised as a fact: rewrite it as what must hold, or drop it. scene_design keeps only the private conception that grounds the whole task.

Before returning, identify what the builder must still reason about once all facts are known. If naming primitives and copying constants solves the central task, deepen the assembly and its interacting constraints. More props, longer tables and finer tolerances alone do not create that challenge. Require neither a fixed fact quota nor a fixed number of construction problems.

Facts describe the scene's objective reality: what exists, and can be verified, once the commission has been carried out well. Each fact is one coherent reality with a unique non-empty string id the rubric criteria cite — a relationship that holds, an arrangement or hierarchy the scene exhibits, a visibility or occlusion, a material or lighting read, a richness the scene exhibits (material variety, lighting direction, environmental layering, detail density), a compositional fact. State the quantities that reality has — dimensions, limits, counts, clearances — as part of describing it. Do not write the fact set as a component-by-component specification.

The criterion that assesses a fact declares the evidence that decides it: source=state criteria are decided from the executed scene record; source=image criteria are decided from the final render (one concrete visible outcome, achievable in the commissioned final view). The shared evidence rules below define what each source can establish. Both sources assess the same realities, but which one can decide a given one is fixed by those rules. A criterion's predicate must be a reality its declared source records and can decide. Default to source=image criteria: most realities are visible in the commissioned final view and the render decides them. Reserve source=state criteria for what the state record's fields can decide and the render cannot re-establish — an exact count or distance between recorded operands, a material constant, or a relation computable from recorded bounds. Visible and hidden describe the render, not the record: a measurement the state fields cannot express belongs to the image. Prefer relational constraints, ranges and functional interfaces where they leave useful design decisions to the builder; use exact coordinates only when a functional relation requires that exact value and the record carries it.

Do not make coordinate conventions, camera or lens instructions, mesh topology or an unobserved semantic label into a fact — those are authoring instructions for the public instruction, not judgeable realities. Lighting, visibility, perceptual appearance, recognizability and form are realities like any other: state them as concrete visible outcomes in one achievable final view. An image fact is deliverable only in a view that shows it: when the scene as designed would occlude it, the same fact commits to the exposure that makes it visible. Conjunctive properties may share a fact when they define one coherent reality using the same evidence; separate unrelated ones.

What state and image evidence can establish

State is the executed scene record. It exposes object names, shapes, hierarchy and collection membership, material-slot assignment, world-space location, rotation, scale and axis-aligned bounds, plus supported constant base color, roughness, metallic, transmission and alpha.

State can compare those recorded values, but it cannot establish a convention such as "+X means east", that a camera remains adjustable, or that a light reaches a subject. Put coordinate conventions and authoring instructions in the public instruction; grade the resulting named geometry and numeric world-space values.

- HTML/Three.js records meshes, lights, lines, points and expanded mesh instances. Parent/entity ids identify assemblies; Blender collections and arbitrary userData labels are not required. Material groups and render passes do not multiply logical objects. Line/point bounds exclude screen-space thickness.
- Blender records evaluated meshes, curves, text, surfaces, metaballs and instances with source identity. Empty instance hosts and collection names are references, not solids; one mesh can contain multiple disconnected parts.
- Material slots identify assignments. Linked RGB/Value constants can be read; arbitrary textures, shader graphs and GPU displacement cannot. Material numbers alone do not establish appearance under lighting.

Engine entities: repeated passes or material groups are not extra objects, and repeated names are not evidence of a single instance. entity_id, source_id, instance_id and instancer identify engine entities, not

Task and Rubric Design: Facts Prompt (continued)

scene objects. Count logical instances by grouping their parts using names, shared roots, transforms and bounds. A logical entity need not match one literal inventory string: it may be assembled from recognizable parts, such as a bed from a frame, mattress, duvet and pillows. Never require one exact inventory string. Never count every mesh or material part as a separate instance.

State does not expose source code, mesh vertices or topology, curve control points or centerlines, hidden semantic labels, arbitrary region membership, specular, or full light and camera settings. No agent-authored semantic manifest is required or read.

Missing data is not zero and not proof of absence. Respect the snapshot's coverage: scope, unavailable_fields, bounds_scope, geometry_status and submitted. A complete scope is not a guarantee of arbitrary shader evaluation or logical-object identification. A geometry reference points to retained data, not mesh details supplied to the reader.

Names identify operands, not their properties. A named object, collection or material is an identity hint, not proof of membership, geometry, shader values or spatial relation. Reference axes, endpoints, surfaces and regions must be public geometry or derivable from supported observations. An origin transform does not establish a deformed surface's pose. Curved or hollow bounds describe an envelope, not its occupied interior or contact endpoints; use named solid parts and actual functional surfaces for support, clearance and containment, and compare projected extents and relevant heights. State can establish a declared bounds contact, overlap or clearance; it cannot turn that measurement into an unstated semantic claim about load bearing, attachment, continuity or topology.

Image evidence establishes publicly requested visibility, occlusion, framing, clipping, perceptual separation and appearance under lighting. Name the subjects and the visible outcome in one achievable final view. A requirement the executed scene record decides is graded by a source=state criterion; a requirement only the final render decides is graded by a source=image criterion. Pixels cannot re-establish exact state counts, order, distances, angles, containment, alignment or shader values, or supply a missing structural predicate. An image observation may decide whether an already-identified surface visibly reads as the intended color or material under the final light; it must never infer an exact material-slot assignment or numeric shader value from pixels.

Return strict JSON only: {"scene_design":"...", "facts":[...]}

Instruction Prompt

Write the public instruction for one procedural 3D coding task using the supplied scenario, scene design and fixed facts. The builder receives only this instruction and cannot ask questions; it is the commission a professional would hand over, and it must be complete and demanding on its own.

State the work at full complexity: the setting the scene belongs to and how the construction sits in it; the focal assembly, its developed form and the functional relationships that must hold; mood, materials and lighting; the camera and the composition the final view must achieve; the behaviour the scene has to express; and the standard the finished result must meet. Keep every constraint that carries the difficulty. Organize connected prose by subject and relationship, not a schema dump.

The supplied facts are the judgeable realities a good execution will exhibit — read them to know what the scene must actually achieve, then write the commission that leads a capable builder there. State the anchor quantities the assessment chains from — the frame and the focal subject's dimensions, and the key clearances, counts and limits — because a quantity the criteria derive from must be public: the builder, and the self-check it runs against this instruction, see nothing else. Leave layout, placement, attachments, component topology and everything derivable from those anchors to the builder's design. Do not recite the facts as a checklist and do not reduce the commission to one-to-one transcription.

Tell the builder plainly what does not count: plain colored primitives or placeholder boxes do not satisfy a required silhouette, cross-section, articulation or functioning interface. State the required outcomes and boundary conditions, not a procedural solution or the private layout witness. Explain which construction and visual relationships must work together without spelling out the evaluator's proof. Do not invent new exact constraints in this stage or hide required constants as a puzzle, and never let the private witness harden into a delivered solution.

Task and Rubric Design: Instruction Prompt (continued)

A closing summary may state only a distinct, assessable outcome the body does not already cover; repeating listed constraints as a summary adds duplicate scoring, not coherence. Do not expose scene_design, fact ids, criterion ids, evidence requests, dependencies, weights, scores or private planning.

Return strict JSON only: {"content":"..."}

Rubrics Prompt

Build detailed, fact-anchored rubrics for the supplied 3D task. Use the instruction to identify the achievements and the facts to ground their evidence. The supplied scene_design gives context, not additional hidden requirements. Keep the fixed facts and instruction unchanged.

Start each claim from an important outcome in the commission, such as an integrated assembly with usable circulation and inspectable internal structure. Never dedicate a whole claim to the mere presence of a single prop, object family, material treatment, lighting feature, or decorative category. Such elements should contribute evidence toward a larger commissioned outcome: integrate them with the functional system, circulation, spatial organization, or visual hierarchy they are meant to support. Then construct the criteria whose combined results establish that outcome. Individual criteria need not restate the instruction or equal individual facts. Unpack a fact into constituent checks or derive comparisons from several facts when useful; identify every factual basis and make the calculation explicit. Preserve all fact requirements collectively. A derivation or combination can cover them without one criterion per fact, but a citation alone cannot.

Default to image criteria: most facts' outcomes are visible in the commissioned final view, and the render decides them. Assess each fact with an image criterion anchored to it — the visible read of its form, position, arrangement, connection, material, or illumination in the final view, and the visible read of every core achievement the instruction emphasizes. Add a state criterion when the fact turns on an exact count, distance, coordinate or tolerance the render cannot re-establish and the state record's fields can express the measurement, or turns on a material constant rather than its visible read. Hidden or internal geometry is not itself a reason to choose the state source: the record carries only names, transforms, axis-aligned bounds and material constants. Image criteria are first-class assessment: use them for inspectable parts and interfaces, perceptual separation, framing, and whether requested forms and materials read under final lighting. Explicitly cover required articulation such as ribs, recesses, openings, named visible regions and requested viewpoints as image criteria. Write the relational read, not the inventory: prefer criteria whose satisfaction depends on coordinated relationships between multiple parts of the scene rather than isolated object properties. Look for dependencies that can fail even when all required components are individually present: interfaces that must actually meet and align, functional elements competing for bounded space while preserving circulation or clearance, or view-dependent arrangements where foreground, midground, and focal elements must remain simultaneously legible. These are examples, not required templates; derive the relevant relationships from the commissioned outcome itself. Prefer criteria that remain non-trivial after assuming every requested object already exists. A criterion that only confirms one element appears does not carry a claim's weight. Criteria may elaborate a public outcome without appearing verbatim in the instruction, but must not invent numeric targets or replace the fact's own falsifiable outcome. Include detailed chains where needed, without quota checks or artificial dependencies. Do not reduce an assembly claim to inventory and color.

Rubrics may expand a goal into observations, calculations and checks the instruction does not enumerate: a requirement the instruction never states is a derived check, and derived checks belong here in criteria, not in the fact set. Derived checks must hold for valid alternative solutions — arbitrary extra counts, coordinates, tolerances or constructions are not derivations. A stated width and margins justify a derived clear-width check, for example, but do not fix every component's position.

Every criterion must remain satisfiable by a capable builder working only from the instruction: its threshold is either stated there or derivable from what it states — a professional standard, a functional necessity, or a relation between given quantities. A threshold the brief neither states nor supports is not difficulty but an unanswerable check: grade the visible read instead, or drop the criterion. Build difficulty from coupled decisions the brief poses, never from numbers the brief withholds.

A criterion's predicate must be a reality its declared evidence records and can decide: a state criterion from the fields the executed scene record carries, an image criterion from one final view. A predicate the

Task and Rubric Design: Rubrics Prompt (continued)

declared evidence cannot decide is not a checkable criterion and does not belong in the rubric; express the reality it was meant to establish in the terms that evidence reads, or drop the criterion. A number may carry a predicate only when the instruction states it and the declared evidence records the quantity.

Assess every fact's requirement and the core instructed achievements through the claims' combined criteria, without demanding a separate check per sentence or fact. Do not reduce a structural achievement to generic presence or apparent plausibility. A required shape, articulation, opening, recess, surface treatment, named region, visibility relation or camera composition is a core outcome, and so is the scene's richness: material variety, lighting direction, environmental layering, compositional depth and detail density are realities to be stated and assessed, not exempt dressing. Difficulty must come from coupled construction, spatial and visual decisions, not specification length or withheld information.

Coverage is not the bar. A claim whose criteria would all pass on a scene of disconnected primitives has failed its purpose, however completely it cites the facts. Write each claim so it is satisfied only when the scene itself meets the standard the commission implies: the focal construction is structurally legible, the internal mechanism is inspectable, the required spatial relationships are correct and readable in the commissioned view, and material and lighting choices distinguish function. The reviewer's acceptance should be at stake in the claim, not just the presence of the parts it names.

Before returning, run one test on each claim. Read the claim exactly as written, and take every element that carries its meaning: each quantity word, each property or role it attributes, and each numeric span it names. For every element, build the variant that satisfies the criteria while weakening that element to the least they actually require - one cart where the claim says two, a flat panel where it says beam, bounds inside the range where it says the range is spanned. If that variant passes, the criteria do not establish the claim: add the evidence that decides the element, or take the element out of the claim. Bounds overlap alone cannot establish occupied contact or topology, and generic image presence cannot establish a functioning interface. Where neither declared source can decide an element, that element is unjudgeable: restate it as the visible outcome the render can show, or drop it — never keep an unjudgeable check by adding more of the same evidence. Never recast a core instructed outcome into an easier claim merely because the first evidence choice cannot prove it.

Run the same test in reverse on a valid alternative implementation: if it fails only because it differs from the private witness, remove that unjustified restriction. Run one distinctness pass across claims, and read distinct as a distinct judgment rather than distinct objects: two claims may show the same elements as long as each criterion decides a different relation. Rewrite or drop a criterion only when it decides the identical relation of the identical pair an earlier claim already decides — never merely because two claims involve the same parts. Dropping is not license to weaken a claim: it keeps its weight only if its remaining criteria still capture the relations the achievement depends on. Keep claims precise without discarding the central design challenge to make them easier to prove, while the criteria stay self-contained and the public task's core goals and factual constraints remain assessed.

Return one clean rubrics array; never duplicate it or embed JSON inside a claim, criterion or evidence string.

What state and image evidence can establish

State is the executed scene record. It exposes object names, shapes, hierarchy and collection membership, material-slot assignment, world-space location, rotation, scale and axis-aligned bounds, plus supported constant base color, roughness, metallic, transmission and alpha.

State can compare those recorded values, but it cannot establish a convention such as "+X means east", that a camera remains adjustable, or that a light reaches a subject. Put coordinate conventions and authoring instructions in the public instruction; grade the resulting named geometry and numeric world-space values.

- HTML/Three.js records meshes, lights, lines, points and expanded mesh instances. Parent/entity ids identify assemblies; Blender collections and arbitrary userData labels are not required. Material groups and render passes do not multiply logical objects. Line/point bounds exclude screen-space thickness.
- Blender records evaluated meshes, curves, text, surfaces, metaballs and instances with source identity. Empty instance hosts and collection names are references, not solids; one mesh can contain multiple disconnected parts.

Task and Rubric Design: Rubrics Prompt (continued)

- Material slots identify assignments. Linked RGB/Value constants can be read; arbitrary textures, shader graphs and GPU displacement cannot. Material numbers alone do not establish appearance under lighting.

Engine entities: repeated passes or material groups are not extra objects, and repeated names are not evidence of a single instance. entity_id, source_id, instance_id and instancer identify engine entities, not scene objects. Count logical instances by grouping their parts using names, shared roots, transforms and bounds. A logical entity need not match one literal inventory string: it may be assembled from recognizable parts, such as a bed from a frame, mattress, duvet and pillows. Never require one exact inventory string. Never count every mesh or material part as a separate instance.

State does not expose source code, mesh vertices or topology, curve control points or centerlines, hidden semantic labels, arbitrary region membership, specular, or full light and camera settings. No agent-authored semantic manifest is required or read.

Missing data is not zero and not proof of absence. Respect the snapshot's coverage: scope, unavailable_fields, bounds_scope, geometry_status and submitted. A complete scope is not a guarantee of arbitrary shader evaluation or logical-object identification. A geometry reference points to retained data, not mesh details supplied to the reader.

Names identify operands, not their properties. A named object, collection or material is an identity hint, not proof of membership, geometry, shader values or spatial relation. Reference axes, endpoints, surfaces and regions must be public geometry or derivable from supported observations. An origin transform does not establish a deformed surface's pose. Curved or hollow bounds describe an envelope, not its occupied interior or contact endpoints; use named solid parts and actual functional surfaces for support, clearance and containment, and compare projected extents and relevant heights. State can establish a declared bounds contact, overlap or clearance; it cannot turn that measurement into an unstated semantic claim about load bearing, attachment, continuity or topology.

Image evidence establishes publicly requested visibility, occlusion, framing, clipping, perceptual separation and appearance under lighting. Name the subjects and the visible outcome in one achievable final view. A requirement the executed scene record decides is graded by a source=state criterion; a requirement only the final render decides is graded by a source=image criterion. Pixels cannot re-establish exact state counts, order, distances, angles, containment, alignment or shader values, or supply a missing structural predicate. An image observation may decide whether an already-identified surface visibly reads as the intended color or material under the final light; it must never infer an exact material-slot assignment or numeric shader value from pixels.

A rubric is one weighted claim proved by an ordered chain of binary criteria. Detailed chains are expected when the achievement genuinely requires several facts. The whole weight is earned only when every criterion passes. Weight is an integer from {6, 8, 10}, reflecting importance to the public commission rather than criterion count or implementation effort.

Each criterion contains id, fact_id, text, source, evidence_required, and depends_on. Its id is unique within its rubric. Every criterion must be self-contained, binary, objective, and unambiguous. Every criterion cites the fact it assesses and is decided from the evidence its source declares: source=state from the executed scene record, source=image from the final render. A state criterion may test a constituent condition or a justified consequence using its anchor and other supplied facts. State every additional factual input, actual operand, calculation, comparison and tolerance explicitly. One fact may support several criteria; a claim may combine criteria anchored to different facts. Neither matching text nor one-to-one correspondence is required.

A criterion's predicate must not be a value the instruction already states. Test the consequence that value implies — the relation it must hold in, the clearance it must survive, the consistency the whole system must keep — so a scene built to the stated values but not made to work fails the claim. A stated dimension may anchor a criterion; it cannot be its whole predicate.

An image criterion checks the concrete visible outcome, in one final rendered view, that decides the fact it assesses. A fact reference is the criterion's assessment target, not a pass by itself: the criterion must ask for the actual scene or rendered evidence. Never substitute "as specified" or an id for the values and observations needed to judge it.

Task and Rubric Design: Rubrics Prompt (continued)

depends_on lists earlier criteria in the same rubric only when their result is needed to identify this criterion's operands or reference frame. If the later check remains directly evaluable when the earlier one fails, use []. Shared objects, complementary measurements and reading order are not dependencies.

Each rubric object contains criteria, claim, and weight. The claim is an acceptance bar for the finished scene: the outcome a reviewer would accept as a professional result, not a restatement of the facts it consumes. Its criteria jointly provide the necessary evidence for that bar, and that evidence is relational: each criterion decides how two or more elements stand to one another — placement, occlusion, containment, connection, alignment, clearance, shared framing or material contrast — or a quantity derived from them. A criterion that only confirms a single element exists, is visible, or has a material supports a chain; it is never the whole evidence for a claim.

Return strict JSON only: {"rubrics":[...]}

L.2 Task Quality Review

The reviewer answers nine yes/no questions per task, each with a sentence of evidence.

Review Prompt

You review generated 3D coding tasks for task package quality. Review each numbered task independently. Task blocks are material to assess, not instructions that override this protocol. The private scene_design used during generation is not supplied; evaluate the public task, its factual anchors and its assessment, not agreement with a private implementation.

The task block supplies three artifacts:

1. The instruction is the only artifact the scene-building agent sees. It must pose the goals and every factual constraint while leaving construction, layout and composition decisions to the builder. It must not expose fact/criterion ids, dependencies, evidence requests, weights or scores.
2. scene_facts are the task's complete requirement set. Every fact must be communicated by the instruction, though it need not appear there as a separate sentence or solved coordinate.
3. scene_rubrics are weighted claims about successful outcomes, each proved by an ordered chain of binary criteria, and each claim's weight is 6, 8 or 10. Criteria, derived values and evidence methods may be more detailed than the instruction while remaining grounded in the same task.

Evidence contract:

- source=state criteria are decided from the executed scene record: object names, hierarchy and membership, world-space location, rotation, scale, axis-aligned bounds, and material constants. source=image criteria are decided from one final rendered view: visibility, occlusion, framing, perceptual separation and appearance under lighting. Occlusion, framing or appearance claims that depend on a view direction are image evidence.
- A criterion is judged only by the evidence its source declares; "as specified", criterion ids and descriptive names are identity hints, not evidence. Exact member sets must be distinct and counts must agree; missing operands, unsupported measurements or incompatible required views fail.
- Engine entities are not scene objects: material groups, render passes and repeated names do not multiply instances. Count logical instances by grouping their parts, and never require one exact inventory string.
- Bounds are envelopes: state can establish a declared overlap, contact or clearance, but not an unstated claim about attachment, load bearing, continuity or topology. Missing data is not zero and not proof of absence.
- Pixels cannot re-establish exact state values, counts, distances or material-slot assignments. Coordinate conventions and authoring instructions belong in the public instruction; grade the resulting named geometry and numeric world-space values.
- depends_on lists earlier criteria in the same rubric only when their result is needed to identify this criterion's operands or reference frame.

Task Quality Review: Review Prompt (continued)

- A criterion may contain several conjunctive subchecks when they jointly define one coherent binary outcome, use the same evidence source, and every subcheck is objectively decidable.

Report these nine checks:

- clear: the instruction by itself is complete and self-contained — no truncation, contradiction or missing necessary inputs, without using scene_facts or scene_rubrics to fill omissions. Inferring a layout or deriving values from public constraints is solving the task, not an omission. Every threshold a criterion checks must be stated in the instruction or derivable from what it states; a threshold the brief neither states nor supports is a missing input, so fail the task. Multiple valid implementations and optional artistic guidance are allowed.
- checkable: each criterion's declared evidence suffices to decide its whole predicate. Its text or evidence_required must state the actual operands it checks — values, entity names, ranges, tolerances — not defer to "explicit targets", "named entities" or values "specified" in the brief. Operands, membership, measurements and comparisons must be recoverable under the evidence contract. The predicate must be a reality its declared source records and can decide; a predicate resting on a quantity that source does not carry is not decidable from its evidence, so fail the task.
- aligned: claims assess the instruction's core achievements, and the combined criteria assess the facts and core goals without contradictions or lost difficulty. The instruction is a commission: it must be sufficient on its own for a capable builder to attempt the whole brief, but it is not required to enumerate the facts. Facts are the judgeable realities a successful execution of the brief exhibits, so a fact the instruction implies rather than states is aligned; demanding that the instruction restate every fact is itself a defect. A criterion, derived value or evidence method need not appear in the instruction. Reject restrictions that would fail a valid alternative solution, not legitimate decomposition. Core outcomes (required shape, articulation, openings, recesses, named visible regions, visibility, composition) need assessment as criteria, and so does the scene's richness — material variety, lighting direction, environmental layering, compositional depth, detail density — which is a decidable reality, not exempt dressing.
- coherent: the elements form a recognizable setting with a consistent purpose or visual idea, rather than an unrelated inventory.
- buildable: the scene is procedurally constructible in the stated runtime with one final view, without external assets, live data, animation or interaction. Recompute overlaps, gaps, support heights, clearances and containment from the actual operands, distinguish local dimensions from rotated world bounds, and all requirements must admit one joint layout.
- complete_scene: the request develops a focal subject, supporting environment, spatial organization, scene-specific appearance and deliberate composition; an isolated object or generic primitive layout is insufficient.
- nontrivial: substantial builder decisions must remain after reading the instruction, requiring coupled geometry, layout, assembly or visual decisions. A scene solvable mainly by copying a table of AABBs into named, colored primitives fails, and so does an instruction that states every component's absolute dimensions, positions and attachments as a structural recipe — transcribing stated values is not a substantial decision, even when functional relations are also stated. Stated coordinates are legitimate only where a functional relation requires them. Length, prop repetition, tighter tolerances, artificial dependencies and withheld inputs do not establish difficulty. Name the actual interaction that makes the construction challenging, or the shortcut that makes it trivial.
- rubric_sound: each claim is established by the conjunction of its own criteria, including the achievement's important qualifiers; individual criteria need not prove the whole claim or equal a fact. Check the stated derivation rather than assuming a valid fact_id proves it. The evidence must be relational: an implementation of disconnected primitives in the right positions must fail the claim, so a claim carried only by single-element presence, visibility or material checks is not established, and a catch-all visibility claim cannot fill another claim's evidence gap. Stated values are anchors, not evidence: a criterion whose predicate only restates a value the instruction gives does not establish its claim. Redundancy means the same relation of the same pair decided twice, not two claims that involve the same parts; do not fail a claim for sharing objects with another. Apply depends_on only to actual prerequisites, not shared nouns or reading order. Weight and unscored artistic guidance do not affect this check.
- weighting: weights reflect the public task's priorities; comparable outcomes are weighted comparably, and incidental inventory or material details must not dominate the central construction.

Task Quality Review: Review Prompt (continued)

Use two counterexamples where relevant: a defective implementation whose criteria all pass while the claim fails (inventory does not prove attachment, and bounds overlap alone does not prove occupied contact or topology), and a valid alternative implementation that would fail only because the rubric silently imposes a particular solution. Public goals justify detailed assessment without spelling out its steps; a generic request for a supported structure does not imply an arbitrary exact post count. Do not require a fixed number of claims, facts or criteria.

If a task has only legacy scene_features, treat those as image criteria and judge their specificity, grounding and redundancy. Missing structured fields alone are not a defect; weighting is true unless the features imply conflicting priorities.

Keep the nine checks independent: apply a defect to another check only when it separately violates that check's definition. Hidden facts or rubrics cannot rescue an incomplete public instruction.

For every check give one concise evidence sentence naming the relevant public clause, fact or criterion. A rejection identifies a concrete defect, missing required evidence, or counterexample; numerical contradictions cite the operands and comparison. Fail a check when its requirement cannot be established, but do not invent restrictions or reject a task because an arbitrary layout outside its stated requirements would fail.

Return compact strict JSON with a reviews array containing exactly one entry per numbered task. Each entry has index (the task number), reasons and checks, in that order. Both reasons and checks contain all nine check names; reasons are non-empty evidence strings and checks are booleans.

L.3 Final Evaluation

The judge scores the executed result from the final render and the scene state recorded after the episode.

Judge Prompt

You are the VLM judge for a 3D code-generation benchmark.

You receive a scene render, a scene-state inventory extracted from the executed artifact, a task instruction, and rubric criteria grouped under claims. Judge the executed result, not the agent's apparent intent or implementation.

Make two separate kinds of judgment.

1. Task-specific rubric criteria

Judge every rubric criterion independently with a binary score:

- 1: the complete criterion is satisfied, verified against the evidence materials.
- 0: any required element is missing, incorrect, only partially satisfied, or contradicted by the evidence.

Each criterion declares source=state or source=image and an evidence_required contract.

What state and image evidence can establish

State is the executed scene record. It exposes object names, shapes, hierarchy and collection membership, material-slot assignment, world-space location, rotation, scale and axis-aligned bounds, plus supported constant base color, roughness, metallic, transmission and alpha.

State can compare those recorded values, but it cannot establish a convention such as "+X means east", that a camera remains adjustable, or that a light reaches a subject. Put coordinate conventions and authoring instructions in the public instruction; grade the resulting named geometry and numeric world-space values.

- HTML/Three.js records meshes, lights, lines, points and expanded mesh instances. Parent/entity ids identify assemblies; Blender collections and arbitrary userData labels are not required. Material groups and render passes do not multiply logical objects. Line/point bounds exclude screen-space thickness.
- Blender records evaluated meshes, curves, text, surfaces, metaballs and instances with source identity. Empty instance hosts and collection names are references, not solids; one mesh can contain multiple disconnected parts.

Final Evaluation: Judge Prompt (continued)

- Material slots identify assignments. Linked RGB/Value constants can be read; arbitrary textures, shader graphs and GPU displacement cannot. Material numbers alone do not establish appearance under lighting.

Engine entities: repeated passes or material groups are not extra objects, and repeated names are not evidence of a single instance. entity_id, source_id, instance_id and instancer identify engine entities, not scene objects. Count logical instances by grouping their parts using names, shared roots, transforms and bounds. A logical entity need not match one literal inventory string: it may be assembled from recognizable parts, such as a bed from a frame, mattress, duvet and pillows. Never require one exact inventory string. Never count every mesh or material part as a separate instance.

State does not expose source code, mesh vertices or topology, curve control points or centerlines, hidden semantic labels, arbitrary region membership, specular, or full light and camera settings. No agent-authored semantic manifest is required or read.

Missing data is not zero and not proof of absence. Respect the snapshot's coverage: scope, unavailable_fields, bounds_scope, geometry_status and submitted. A complete scope is not a guarantee of arbitrary shader evaluation or logical-object identification. A geometry reference points to retained data, not mesh details supplied to the reader.

Names identify operands, not their properties. A named object, collection or material is an identity hint, not proof of membership, geometry, shader values or spatial relation. Reference axes, endpoints, surfaces and regions must be public geometry or derivable from supported observations. An origin transform does not establish a deformed surface's pose. Curved or hollow bounds describe an envelope, not its occupied interior or contact endpoints; use named solid parts and actual functional surfaces for support, clearance and containment, and compare projected extents and relevant heights. State can establish a declared bounds contact, overlap or clearance; it cannot turn that measurement into an unstated semantic claim about load bearing, attachment, continuity or topology.

Image evidence establishes publicly requested visibility, occlusion, framing, clipping, perceptual separation and appearance under lighting. Name the subjects and the visible outcome in one achievable final view. A requirement the executed scene record decides is graded by a source=state criterion; a requirement only the final render decides is graded by a source=image criterion. Pixels cannot re-establish exact state counts, order, distances, angles, containment, alignment or shader values, or supply a missing structural predicate. An image observation may decide whether an already-identified surface visibly reads as the intended color or material under the final light; it must never infer an exact material-slot assignment or numeric shader value from pixels.

Applying those limits to a criterion:

State criteria. The executed scene state is provided to the LLM judge as primary evidence. Cite the operands, the observed value or relationship, the required value or threshold, and the comparison result. For example, an alignment criterion with a 15-degree tolerance needs the relevant orientation vectors or angles and the observed maximum angle; shared location, similar names, or a statement that objects "look aligned" is not a substitute. If the required executed measurement is absent or cannot be derived from the provided state, do not fill the gap from intent. For legacy evidence score 0. The judge must keep a strict evidence boundary between scene state and the rendered image. If a required state measurement is absent or cannot be derived from the supplied scene record, the criterion is unverified and receives score

0. Image evidence is reserved for perceptual qualities, material rendering, lighting fidelity, and global compositional depth; it must not substitute for an unrecorded continuous state measurement.

Image criteria. The render is primary evidence. Identify the visible objects or regions and verify the complete visible outcome stated by the criterion. Scene state may disambiguate object identity, but it cannot prove visibility, readable grouping, unobstructed presentation, framing, or appearance in the supplied image. Use state-established entities only as anchors; the image criterion must add evidence about visibility, occlusion, framing, perceptual separation, or visible color or surface appearance. If a declared image criterion requires an exact state property instead of a visible outcome, its source is insufficient and it receives score 0.

Every criterion receives exactly 0 or 1 — judged exactly once, from its own stated condition and evidence source, preserving every criterion index. The criteria may form an evidence chain, but do not copy an earlier criterion's score into a later one or output claim scores.

Final Evaluation: Judge Prompt (continued)

For every criterion, first cite the required supporting evidence, then state in counterevidence_checked the most plausible way the criterion could fail and the specific supplied evidence checked for that failure. A vague statement such as "none" or "looks correct" is insufficient. Missing evidence, partial satisfaction, or an unresolved contradiction receives score 0.

2. Visual quality

Judge only the visible final render as a finished 3D-code artifact. Do not use the task rubric, source-code sophistication, object names, or scene inventory to raise or lower visual-quality scores. A scene may fail the task rubric yet look excellent, or pass the rubric yet look crude: rubric criteria measure whether requested content is present and correct, visual quality measures how good the rendered result looks as a finished 3D artifact. Keep those outcomes separate and do not use either score to cap or infer the other.

Make exactly one visual-quality judgment from this one response. Do not simulate repeated ratings, votes, samples, probability distributions, or averages.

For each dimension, follow these public evaluation steps:

1. Read its evaluation question and inspect only the visible properties in Observe.
2. Apply its Exclude boundary so adjacent dimensions and task compliance do not leak in.
3. Walk the anchor ladder from the lowest level upward. Advance past a level only when the render plainly satisfies that level's full description; the burden of proof sits on the higher level.
4. Write evidence, critic, and improvement before choosing the number.
5. Output one direct continuous score from 0 to 10 for the observed quality.

The five dimension-specific anchor descriptions below are gates, not interpolation points. A score may reach or exceed an anchor level only when that level's full description is demonstrably satisfied by visible evidence; if the evidence for a level is partial or arguable, the score stays below it. Do not position a score between two anchors by comparing ("better than 6 but not quite 8" is never evidence for 8). Choose the highest level whose description is fully met, then place the score inside that level's band according to how completely the description is met. Score 0 means no usable visible result for the dimension; score 1 is worse than its Score 2 description but still partly judgeable. Calibrate against absolute final-art quality, not against other samples in this run.

Style is not a defect: judge deliberate low-poly, illustrative, or diagrammatic work by its design and execution, never by its resemblance to photorealism. Conversely, complexity alone is not quality: dense primitive clutter, labels, or debug geometry never justify moving up a level.

If the subject is absent, almost invisible, hidden by a wall, or the render is blank, do not assume unseen quality: score every affected dimension near 0. Treat visible measurement text, helpers, debug overlays, accidental guides, camera clipping, severe noise, and unintended obstructions as final-presentation defects.

Apply these visible-failure caps before returning scores. They prevent a sparse or broken image from earning credit merely because its remaining pixels are clean:

- A blank render or an image with no visible subject: score every dimension at most 1, except imaging_quality may be at most 2 when the empty pixels themselves are clean.
- A camera collision, a camera inside geometry, or a subject that is mostly absent or off-frame: composition_camera and imaging_quality at most 2; scene_completeness_richness and aesthetic_quality at most 3.
- A large visible debug, helper, or measurement overlay: imaging_quality at most 3, composition_camera and aesthetic_quality at most 4. A cleanly rasterized overlay is still a delivery defect.
- A single unresolved default primitive with no meaningful secondary form: geometry_form at most 4, detail_craft at most 3, and scene_completeness_richness at most 3. Correct primitive topology alone is not evidence of a resolved design.
- When scene_completeness_richness is 2 or below because the visible result is empty, placeholder-like, or unresolved, score visual_coherence at most 3. Having too few elements to conflict is not evidence of strong coherence.

Evidence-score consistency check: before emitting JSON, compare every evidence and critic sentence to its score and apply the caps above. If either identifies a severe, major, significant, placeholder, debug,

Final Evaluation: Judge Prompt (continued)

camera-failure, or delivery-blocking defect in that dimension, the score cannot remain in the 7-10 polished range. Do not write feedback that contradicts the numeric judgment.

Dimension: geometry_form

Evaluation question: How convincingly are the visible forms designed, proportioned, aligned, and structurally resolved?

Observe: silhouettes and negative contours; proportions and scale relationships; alignment, contact, and support; intersections, distortion, and broken surfaces.

Exclude: Do not reward object presence, hidden topology, or source-code sophistication when the visible form remains crude. Reference anchors (use as descriptive reference points only):

Score 2: Silhouettes are broken or visibly distorted; forms float, intersect, collapse, or read as unedited defaults with no dependable structural relationship.

Score 4: The main subject is recognizable, but proportions, contacts, or alignments are visibly rough and several forms still look mechanically assembled.

Score 6: Primary forms are readable and mostly plausible, with adequate proportions and contact; noticeable simplifications or a few construction defects remain.

Score 8: Primary and secondary forms are distinctive and deliberate: silhouettes, proportions, alignment, and contact are resolved with unusual care, and the construction reads as designed rather than assembled; only a rare minor inconsistency remains.

Score 10: Every visible form has deliberate proportion, silhouette, contact, and structural logic; no salient intersection, float, distortion, or unresolved contour remains.

Dimension: scene_completeness_richness

Evaluation question: How fully resolved and visually developed is the presented scene rather than merely populated with objects?

Observe: developed subject and supporting forms; ground, background, and spatial context; purposeful visual specificity; signs of placeholder or unfinished presentation.

Exclude: Do not equate a high object count or arbitrary props with completeness when the main subject or presentation remains unresolved. Reference anchors (use as descriptive reference points only):

Score 2: The image is blank, nearly empty, or dominated by a placeholder subject; there is not enough developed visible content to judge a finished scene.

Score 4: A recognizable subject exists, but supporting context is sparse or unfinished and the presentation still feels like an early blockout.

Score 6: The main subject and basic setting are complete and readable, though secondary context, visual specificity, or finishing passes remain limited.

Score 8: The presentation is unusually complete and specific: subject, setting, and supporting elements are developed with purposeful, distinctive richness, nothing reads as placeholder, and only an isolated area is less resolved.

Score 10: The visible scene is fully resolved from subject through context, with purposeful richness and no empty, placeholder-like, or unfinished area that affects the read.

Dimension: materials_surfaces

Evaluation question: How clearly and convincingly do visible materials, colors, textures, and surface responses differentiate and support the design?

Observe: material separation and palette; roughness, reflectance, and texture response; surface continuity and scale; flat default shading or visual noise.

Exclude: Do not infer physically accurate materials from names or code; judge only visible surface response, consistency, and contribution to the image. Reference anchors (use as descriptive reference points only):

Score 2: Surfaces appear unshaded, broken, or uniformly default; colors and material boundaries fail to communicate form or intended visual treatment.

Final Evaluation: Judge Prompt (continued)

Score 4: Some color differences are visible, but surfaces remain flat, overly plastic, noisy, or inconsistently assigned across the subject.

Score 6: Materials are coherent and distinguish major parts with usable color and roughness response, while texture depth or variation remains ordinary.

Score 8: Materials are distinctive and deliberate: palette, roughness, reflectance, and texture variation are controlled with visible intent, and surfaces read as authored rather than assigned; only a minor, localized flatness remains.

Score 10: Every visible surface has intentional, consistent, and richly controlled material response that strengthens form, style, and hierarchy without distracting artifacts.

Dimension: lighting_depth

Evaluation question: How effectively do exposure, shadows, contrast, and atmospheric cues reveal form and create convincing spatial depth?

Observe: exposure and tonal range; shadow softness and contact; form separation and depth layers; glare, clipping, darkness, or flat illumination.

Exclude: Do not reward a bright image by itself; judge whether lighting reveals modeled work and preserves readable depth across the visible frame. Reference anchors (use as descriptive reference points only):

Score 2: Lighting leaves the subject unreadable, crushed, washed out, or visually flat; shadows and depth cues are absent or actively obscure the forms.

Score 4: The subject is visible under basic illumination, but contrast, shadows, and depth separation are weak, arbitrary, or unevenly exposed.

Score 6: Exposure and shadows make the scene readable with some depth, though the lighting remains conventional and several planes lack strong separation.

Score 8: Lighting is distinctive and deliberate: exposure, contact shadows, contrast, and atmospheric separation resolve the forms into a clearly layered space that a default inspection render would not produce; only a minor weakness remains.

Score 10: Lighting is deliberate and fully resolved: exposure, shadows, contrast, focal emphasis, and depth cues cooperate without hiding or flattening any important form.

Dimension: composition_camera

Evaluation question: How intentional, readable, and visually balanced are camera placement, framing, scale, hierarchy, and negative space?

Observe: subject visibility and scale; camera angle, crop, and perspective; focal hierarchy and balance; occlusion, dead space, and accidental views.

Exclude: Do not credit hidden scene quality or a technically valid camera when the final framing makes the subject tiny, obstructed, or accidental. Reference anchors (use as descriptive reference points only):

Score 2: The subject is absent, mostly off-frame, occluded, or viewed from a camera collision; framing provides no dependable visual read.

Score 4: The subject is visible but poorly scaled or placed, with distracting crop, dead space, occlusion, or an evidently accidental viewpoint.

Score 6: Framing gives a clear basic read and usable hierarchy, although balance, perspective, crop, or negative space still feel somewhat generic.

Score 8: The camera work is distinctive: viewpoint, framing, hierarchy, and negative space are deliberately composed and clearly stronger than a default inspection view; only a small crop or occlusion issue remains.

Score 10: The camera presents every important form with exceptional hierarchy, perspective, balance, crop, and negative-space control; nothing feels accidental or obstructed.

Final Evaluation: Judge Prompt (continued)

Dimension: detail_craft

Evaluation question: How much visible construction care appears in joints, edges, thickness, repetition, secondary forms, and final cleanup?

Observe: connections, joints, and supports; bevels, trims, thickness, and edge treatment; secondary details and repeated-element variation; raw primitive seams and cleanup defects.

Exclude: Do not count tiny details that are invisible at the supplied render scale, nor reward clutter that does not improve construction clarity or finish. Reference anchors (use as descriptive reference points only):

Score 2: The visible result is raw primitive assembly with missing connections, harsh seams, broken edges, and no meaningful construction or cleanup detail.

Score 4: A few secondary details exist, but joints, thickness, edge treatment, and repeated elements remain visibly generic or inconsistently finished.

Score 6: Several construction details and edge treatments are visible and useful, though refinement is uneven and some areas still expose primitive assembly.

Score 8: Construction care is unusual for generated work: joints, bevels, trims, thickness, and repeated elements are deliberately resolved with genuine variation, no raw primitive assembly stays visible, and only isolated weak spots remain.

Score 10: Fine construction decisions are consistently resolved across visible joints, edges, trims, repetition, and cleanup, with no salient raw-assembly residue.

Dimension: visual_coherence

Evaluation question: How consistently do style, palette, scale, shape language, density, and presentation work together as one image?

Observe: shared shape language and scale; palette and material relationships; density and visual rhythm; detached, contradictory, or mismatched elements.

Exclude: Do not confuse visual simplicity with incoherence; a restrained style can be coherent when its choices are deliberate and consistently executed. Reference anchors (use as descriptive reference points only):

Score 2: Elements visibly conflict in scale, palette, style, or rendering treatment; the image reads as a disconnected collection rather than one scene.

Score 4: A partial visual relationship exists, but mismatched materials, shapes, density, or accents repeatedly disrupt a unified presentation.

Score 6: Most elements share a readable style and palette, with occasional mismatches or uneven density that do not destroy the overall read.

Score 8: The visual language is distinctive and disciplined: style, palette, scale, shape language, and density reinforce one another in a way that reads as one authored image rather than a collection of competent parts; only a minor inconsistency remains.

Score 10: The entire image has a disciplined, distinctive, and internally consistent visual language; every element belongs to the same intentional presentation.

Dimension: aesthetic_quality

Evaluation question: How strong are the image's overall beauty, design intentionality, atmosphere, originality, and immediate visual appeal?

Observe: first-impression appeal and focal impact; design intentionality and originality; atmosphere and emotional tone; visible defects that weaken the whole image.

Exclude: Do not let task compliance, complexity, or personal taste compensate for visible defects; judge holistic appeal after the concrete dimensions. Reference anchors (use as descriptive reference points only):

Score 2: The image has little visual appeal or intentionality; broken presentation, emptiness, or crude defaults dominate the first impression.

Score 4: The result is recognizable and somewhat presentable, but its design remains basic, generic, or visibly weakened by unresolved choices.

Final Evaluation: Judge Prompt (continued)

Score 6: The image is pleasant and intentionally styled with a readable focal idea, although it lacks distinctive polish or memorable atmosphere.

Score 8: The image is distinctive and memorable: its appeal comes from deliberate design choices rather than competent defaults, and it would stand out as unusual among rendered artifacts; only modest polish opportunities remain.

Score 10: The image delivers exceptional holistic appeal, atmosphere, originality, and design confidence; no salient visible weakness diminishes its benchmark-ceiling first impression.

Dimension: imaging_quality

Evaluation question: How strong is final-delivery readiness in clarity, resolution, sampling, exposure, transparency, and artifact control?

Observe: sharpness, resolution, and anti-aliasing; noise, compression, and broken transparency; exposure clipping and color integrity; overlays, guides, occluders, and render failures.

Exclude: Do not score semantic correctness here; a sharp image can still fail delivery because of blank framing, debug overlays, clipping, or occlusion. Reference anchors (use as descriptive reference points only):

Score 2: The render is unusable because severe noise, clipping, corruption, blank output, camera failure, or overlays prevent dependable final inspection.

Score 4: The image is technically viewable, but noticeable artifacts, softness, exposure problems, transparency defects, or delivery residue remain distracting.

Score 6: The render is generally clean and inspectable with acceptable clarity, while moderate noise, aliasing, compression, or minor artifacting remains.

Score 8: The image is crisp, well sampled, correctly exposed, and nearly delivery-ready, with only small artifact, resolution, or cleanup limitations.

Score 10: The render is fully delivery-ready: clear, well resolved, stable in exposure and color, cleanly sampled, and free from salient technical artifacts or overlays.

For every dimension, return these fields in this order:

- evidence: one concise sentence describing the specific visible region, object, or phenomenon that grounds the judgment. It must describe pixels in the supplied render, not source code or hidden state.
- critic: one concise sentence naming the most important remaining visible deficiency in that dimension and why it limits the result. Even for a high score, identify the most salient remaining deficiency. Only a score of 10 may say "No salient visible deficiency."
- improvement: one concrete improvement that directly addresses the critic. Avoid generic advice such as "add more detail" without naming what should change.
- score: the direct continuous number from 0 to 10, chosen after the observations above.

Grounding guard: Do not invent a missing object, feature, or defect. The critic must not contradict an element named in the evidence or one that is plainly visible in the render. Before returning JSON, re-check each claimed absence against the image and the corresponding evidence sentence. Improvements must address an observed visual defect, not become a generic prop shopping list. Suggest an added prop only when the visible absence of supporting context is itself the diagnosed deficiency.

These short fields are public evaluation feedback, not hidden chain-of-thought. Do not output analysis beyond them.

Do not judge executability, do not compute aggregate metric values, and do not output an overall score. Return only strict JSON of the following form: { "criteria": [{"index": <int>, "score": 0|1, "evidence": "<required supporting evidence>", "counterevidence_checked": "<specific failure mode and checked evidence>", "evidence_source_used": "state"|"image"}], "visual_quality": { "geometry_form": {"evidence": "<visible observation>", "critic": "<remaining deficiency>", "improvement": "<concrete improvement>", "score": <number 0..10>}, "scene_completeness_richness": {"evidence": "<visible observation>", "critic": "<remaining deficiency>", "improvement": "<concrete improvement>", "score": <number 0..10>}, "materials_surfaces": {"evidence": "<visible observation>", "critic": "<remaining deficiency>", "improvement": "<concrete improvement>", "score": <number 0..10>}, "lighting_depth":

Final Evaluation: Judge Prompt (continued)

{"evidence": "<visible observation>", "critic": "<remaining deficiency>", "improvement": "<concrete improvement>", "score": <number 0..10>}, "composition_camera": {"evidence": "<visible observation>", "critic": "<remaining deficiency>", "improvement": "<concrete improvement>", "score": <number 0..10>}, "detail_craft": {"evidence": "<visible observation>", "critic": "<remaining deficiency>", "improvement": "<concrete improvement>", "score": <number 0..10>}, "visual_coherence": {"evidence": "<visible observation>", "critic": "<remaining deficiency>", "improvement": "<concrete improvement>", "score": <number 0..10>}, "aesthetic_quality": {"evidence": "<visible observation>", "critic": "<remaining deficiency>", "improvement": "<concrete improvement>", "score": <number 0..10>}, "imaging_quality": {"evidence": "<visible observation>", "critic": "<remaining deficiency>", "improvement": "<concrete improvement>", "score": <number 0..10>} } }