SceneAgent: A Multi-Agent Framework for Portable Game Scene Generation

Xingyi Li[1*] Jiahao Yang[2*] Yiqin Zhu[1] Jingwen Ye[1†] Yang Sun[1] Weidong Zhang[1†]
[1]Tencent [2]Fudan University
{xingyili,jingwenye,wadewdzhang}@tencent.com
https://xingyi-li.github.io/sceneagent/

Abstract

Moving a scene generator to a new game engine requires more than exporting its final output: planning, construction tools, and stored experience may also depend on the original backend. We present SCENEAGENT*, a multi-agent framework for reusing the construction process and its experience across backends. A canonical SceneContext maintains the evolving world, while backend adapters isolate coordinate conventions, asset bindings, and export. Structured handoffs let fresh Builder sessions query this persistent state without inheriting the full construction transcript. Across generations, human-approved traces are distilled into asset-ID-free Layout skills and semantic Content skills that can be retrieved and recombined. We introduce GameSceneBench, comprising 69 indoor and outdoor prompts. Under a common asset and evaluation protocol,* SCENEAGENT *obtains human tie-adjusted preference scores of* 78.2–89.4% *against five baselines. Ablations show that reusable skills and the multi-agent protocol primarily improve prompt alignment and scene completeness, with additional runtime. Deployment in a second engine preserves the shared construction protocol and skill library while replacing backend integration components. Together, these results support separating persistent scene state, task-local reasoning, and reusable experience for portable scene construction.*

1. Introduction

Constructing a 3D game environment requires coordinated design decisions: buildings must follow roads, landmarks must face useful viewpoints, and vegetation must respect occupied space. These decisions should remain useful when the engine or asset library changes. Existing LLM-based systems can construct complex scenes [3, 16], but construction decisions are often expressed through backend-specific coordinates, asset identifiers, APIs, and execution traces. An interoperable output file alone does not make these decisions reusable.

We study *construction portability*: retaining the planning protocol, scene-state interface, and reusable design knowledge while localizing backend-specific implementation. This requires more than a common export schema. During generation, agents need to coordinate against the same evolving world without inheriting every preceding tool call. Across generations, experience must retain useful spatial and semantic patterns without carrying over incidental asset bindings. Both world state within a run and design knowledge across runs must remain accessible through the shared interface.

We introduce SCENEAGENT, organized around three complementary mechanisms.

Canonical construction state. A shared *SceneContext* records actor transforms, bounding boxes, occupied regions, and subtask provenance in canonical coordinates. Builders query and update this state through construction tools. Adapters handle coordinate conversion, asset bindings, and export. Backend-specific capability wrappers expose native operations through the construction interface. This boundary preserves the shared protocol while making the required backend integration explicit.

Structured handoffs. An Orchestrator launches a fresh Builder session for each planned subtask. The Builder receives its objective, shared conventions, relevant skills, and compact completion reports and then queries SceneContext for additional world information. Thus, later subtasks can respect earlier construction without replaying its full transcript. Geometry checks and bounded refinement provide control points before the next handoff.

Factored experience. Human-approved traces are distilled into asset-ID-free *Layout* skills for spatial organization and semantic *Content* skills for object roles, style, and substitution. Thin recipes compose these factors with prompt-specific requirements. Backend databases supply concrete assets. The same route layout can therefore guide different scene themes, and the same theme can be realized through different layouts. Rejected traces are retained separately as failure evidence.

We evaluate SCENEAGENT on GameSceneBench, a benchmark of 69 indoor and outdoor prompts. Under a

*Equal contribution.
†Corresponding authors.

shared asset and rendering protocol, a blinded human study yields tie-adjusted preferences of 78.2–89.4% over five baselines. VLM comparisons favor the method in alignment, spatial plausibility, and completeness. Skill-library and single-agent ablations show the clearest gains in alignment and completeness, while quantifying the runtime cost. A second-engine deployment tests reuse of the shared protocol and skills, and cross-LLM studies examine model dependence.

Our contributions are:

- A construction-portability design that couples canonical scene state with a shared tool interface and an explicit backend integration boundary.
- A Planner–Builder protocol that coordinates fresh task-local sessions through persistent scene state and structured completion reports.
- A factored Layout–Content experience library for compositional reuse, with evaluation on GameSceneBench through baseline comparisons, ablations, and cross-engine and cross-model studies.

2. Related Work

Agentic Scene Generation and Portable Representations. Language models have expanded text-to-scene generation from layout prediction [4] to assembling and refining complete 3D scenes [1]. Holodeck [22] populates scenes with retrieved assets, while Holodeck 2.0 [2] adds generated assets and interactive editing. Other agentic systems combine specialized roles, tool use, and visual feedback [11, 12, 23], while SAGE [20] further introduces visual and physics critics for simulator-validated generation. The Procedural Scene Programs approach [7] instead represents scene layouts as imperative programs, placing objects relative to previously constructed elements and improving layout validity through LLM-free program search. Many existing systems realize these capabilities through specific authoring environments: SceneCraft [9] and CityX [26] generate Blender programs or procedural structures, whereas UnrealLLM [16], AutoUE [24], and LatticeWorld [3] rely on Unreal Engine facilities and tooling.

A complementary line of work develops structured scene representations that remain editable and renderable across graphics systems. The Scene Language [27] represents structure, semantics, and visual identity through programs, words, and embeddings, supporting traditional, neural, and hybrid renderers. GenUSD [10] hierarchically plans scene layouts, generates 3D assets on demand, assembles them in Omniverse, and exports the resulting scenes in USD, thereby providing an interoperable scene output. SCENEAGENT focuses on portability of the construction process: agents query and update canonical world state through a shared interface, while adapters and capability wrappers handle backend realization. This complements interoperable output representations by also retaining the planning protocol and semantic experience across backend deployments.

Long-Horizon State and Reusable Experience. Long-horizon agents increasingly externalize task state instead of relying on the full interaction history. Magentic-One [5] coordinates specialized agents through structured task and progress ledgers, while InfiAgent [25] reconstructs bounded reasoning contexts from persistent workspace state and a fixed window of recent actions. Another line of work distills prior trajectories into reusable knowledge: Voyager [18] stores successful behaviors as executable skills, ExpeL [28] extracts transferable insights from trajectories, and Agent Workflow Memory [19] induces reusable workflows. In 3D scene synthesis, SceneMotifCoder [15] similarly distills example arrangements into reusable visual meta-programs.

SCENEAGENT brings these two forms of persistence into one construction protocol. SceneContext carries the evolving world between Builder sessions, while approved traces supply spatial and semantic guidance for later scenes. Layout and Content are stored and retrieved separately, with concrete asset mappings left to backend databases. The distinction is between reusable scene-design guidance and the backend-bound details of an individual construction trace.

3. System Overview

Figure 1 shows the generation and experience-update paths. Given a prompt or reference image, the Orchestrator retrieves Layout and Content skills and supplies them as read-only guidance to the Planner and relevant Builders. The Planner produces an ordered queue of semantic subtasks. For each subtask, a fresh Builder reads its task record and compact completion reports, queries SceneContext, and modifies the world through construction tools. Geometry checks diagnose the updated state and support bounded refinement. After construction, the backend adapter exports the canonical state to the target engine.

Two forms of information persist beyond individual Builder sessions. SceneContext stores the current world within a generation, while the skill library stores reusable guidance across generations. Human review determines which completed traces enter the experience-update path. Automatic checks provide supporting diagnostics. Planning, handoffs, canonical state, and semantic skills are shared across backends. Asset configuration, export adapters, and native-capability wrappers supply backend-specific realization. The Orchestrator coordinates these stages without placing actors directly.

4. Method

SCENEAGENT separates three responsibilities: storing the constructed world, reasoning about the current subtask, and retaining experience for future scenes.

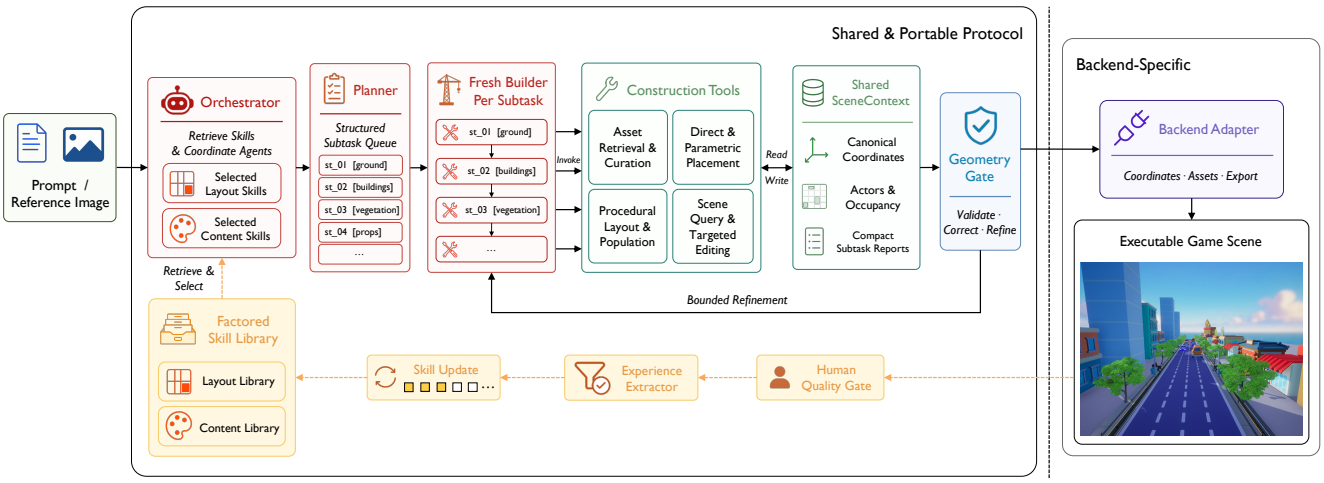


Figure 1. SCENEAGENT separates within-run construction from cross-run experience reuse. Fresh Builders coordinate through canonical scene state. Backend adapters realize that state in the target engine. The optional human-gated update path retains approved traces as composable Layout and Content skills.

SceneContext records what has been built, handoffs summarize task outcomes and unresolved issues, and skills provide reusable design guidance. A fresh Builder combines these sources without inheriting the full construction transcript. This division lets task-local reasoning use persistent world state and cross-scene experience, while adapters and asset databases handle backend-specific realization.

4.1. Canonical Scene State and Backend Boundary

The Planner expresses scene intent through semantic categories, roles, spatial relations, constraints, and gameplay affordances. Builders realize this intent in a shared SceneContext, rather than by invoking an engine directly. SceneContext stores each concrete actor in a builder coordinate system (X: left–right, Y: up, Z: forward–back), together with its asset reference, transform, bounding box, source tag, and subtask identifier. It also maintains scene bounds and an occupancy index for spatial queries. SceneContext serves as both an exportable world representation and the live coordination state: later Builders query what earlier subtasks constructed, including occupied space and provenance, without replaying their actions.

The exported payload is actor-centric: each actor carries an asset reference, a transform, and optional appearance or collision metadata. A backend adapter converts SceneContext into the target backend's coordinate, unit, scale, and rotation conventions, resolves backend-specific asset bindings when required, and serializes the result. For example, one adapter swaps the builder Y and Z axes, converts game units to meters, and emits quaternion rotations. The same construction state can therefore be exported through another adapter while preserving how Builders place actors and query the scene.

Table 1 defines the portability boundary. Retargeting requires a compatible asset catalog, an adapter, and wrappers for backend-specific capabilities. The canonical coordinates, SceneContext schema, core tool interface, Planner–Builder protocol, and semantic skill libraries remain shared. Portability therefore concerns reuse of the construction process with explicit backend integration.

From semantic design to backend assets. Retargeting preserves semantic roles and spatial rules while grounding them in the active backend. For a street scene, a Layout skill can specify a central route with buildings along its sides, while Content skills specify building families and supporting props. Builders retrieve candidates from the local catalog, inspect their metadata and dimensions, and place selected assets relative to SceneContext. Content skills can guide substitution when a requested object is unavailable. Placement and scene queries use the shared coordinate conventions. The adapter then converts transforms and resolves backend-specific bindings for export. This process reuses the construction strategy while allowing concrete assets and scene geometry to reflect the target catalog.

4.2. Planning and Tool-Mediated Construction

The Planner transforms the prompt and an optional reference image into an ordered queue of subtasks, such as ground, building, vegetation, prop, and refinement tasks. Each record specifies a semantic objective, target region, candidate content, construction guide, and shared conventions including the main axis, orientation rules, scale unit, and ground height.

For each subtask, the Orchestrator launches a fresh Builder session. Its initial context includes the current task

Table 1. Shared and backend-local components in SCENEAGENT.

Component	Responsibility
Shared across backends	
Agent protocol	Task decomposition and handoffs.
SceneContext	Canonical state and spatial queries.
Reusable skills	Layout and Content guidance.
Backend-specific integration	
Asset configuration	Asset catalogs and semantic mappings.
Backend adapter	Coordinate conversion, asset binding, and export.
Tool wrappers	Engine operations under shared conventions.

record, construction guide, shared conventions, selected Layout and Content skills, and compact reports from completed subtasks, while excluding raw placement and tool-call histories. When additional information is required, the Builder queries the live SceneContext as needed instead of reconstructing the world from a transcript.

The shared construction tools cover asset retrieval and inspection, direct and procedural placement, scene queries, and targeted editing. Geometric operations use builder coordinates and read from or write to SceneContext. For example, Builders can inspect candidate assets, populate a region, query per-subtask footprints, or remove actors from an occupied area. Additional backend-local wrappers encapsulate native APIs and normalize inputs and outputs to shared conventions where possible. The supplementary material lists the construction and agent-control interfaces.

Scene-changing operations trigger geometry checks for collisions, unsupported or out-of-bounds actors, extreme transforms, invalid assets, and density anomalies. High-confidence violations can receive conservative corrections, such as removal, clamping, or re-seating. Unresolved cases trigger a bounded number of refinement passes or return control to the user. At the end of a subtask, the Builder returns a compact report containing its completion status, summary, residual issues, and actor-count change. SceneContext, rather than the Builder conversation, carries the constructed world into the next subtask.

For a night market, the street Builder first establishes the road. The market-stall Builder receives its completion report and queries SceneContext for road geometry and occupied regions before placing vendors along the sides. A later prop Builder can inspect the resulting footprints before adding furniture. Each handoff preserves task outcomes and access to the world while discarding the preceding Builder's raw tool-call history.

4.3. Factored Experience Representation

A stored scene trace can entangle spatial organization, semantic content, and concrete asset choices. To make these decisions independently reusable, SCENEAGENT extracts three complementary objects:

- **Layout skill**: reusable spatial organization and geometric constraints, such as grouping, support, symmetry, focal placement, path structure, density gradients, and terrain constraints. It contains no concrete asset identifiers.
- **Content skill**: semantic roles, stylistic attributes, and substitution rules, such as object families, theme tags, and material styles. It contains no backend-specific bindings.
- **Thin recipe**: a scene-specific composition record that references Layout and Content skills and retains only residual details unique to the current prompt.

Factoring allows the same spatial organization to be paired with different content families and the same content family to be realized through different layouts, without storing a separate fused memory for every combination. A campsite layout, for example, may be paired with medieval, science-fiction, or cartoon content, while a village theme may use grid, radial, or terrain-following organization. This factorization need not perfectly decompose every scene, but keeps concrete asset IDs and backend directives separate from transferable experience.

Example of skill composition. Consider a night-market scene organized around a pedestrian street. A linear Layout skill specifies a continuous route between an entrance and a terminal focus, places major regions along its sides, and preserves an unobstructed circulation corridor. A night-market Content skill supplies semantic roles such as vendors, lighting, furniture, and supporting props, together with stylistic guidance and substitution rules. For example, a missing food stall can be replaced by a canopy-and-counter assembly. A thin recipe combines these skills and adds scene-specific requirements, such as staggered vendor rows and small dining pockets. Concrete assets are subsequently resolved through the backend-specific asset database. Thus, reusable spatial rules, semantic content, and prompt-specific requirements remain separately represented.

4.4. Quality-Gated Experience Update

Experience is retained through a human-led quality gate, with automatic checks serving as supporting diagnostic evidence. After generation, SCENEAGENT packages the prompt, construction record, final SceneContext, multi-view renderings, automatic validation results, and checkpoint feedback into an experience candidate. The automatic checks expose collisions, unsupported actors, invalid transforms, and density anomalies, but do not alone determine whether to retain the experience.

A human reviewer makes the final decision by considering prompt adherence, spatial organization, visual coherence, richness, and potential reusability. The reviewer may approve or reject the trace and optionally provide a natural-language critique or correction. Only approved traces are passed to the experience extractor, which assigns transfer-

able spatial rules to Layout skills, semantic and stylistic knowledge to Content skills, and prompt-specific residuals to a thin recipe. Candidate skills then undergo schema validation and deduplication before insertion into the library for future reuse. Rejected traces and reviewer feedback are retained separately as failure evidence.

At generation time, the Orchestrator retrieves a compact set of Layout and Content skills from their indexes and injects them as structured, read-only sections of the system prompts for the Planner and relevant Builder sessions. Separate indexes enable independent retrieval of spatial and semantic experience, while suppressing duplicate loads.

Visual experience acquisition. Reference images or screenshots provide an additional source of reusable experience. A vision-language model interprets their semantic content, spatial relations, and construction cues, which then condition the standard Planner–Builder pipeline. The resulting scene and construction trace undergo the same human-led quality gate as text-prompted generations. Only approved traces are factorized into Layout and Content skills and inserted into the experience library after schema validation and deduplication. This route can incrementally expand the library from visual examples without changing the skill representation, retrieval mechanism, or update protocol.

5. GameSceneBench and Evaluation Metrics

GameSceneBench. We introduce GameSceneBench, a text-driven benchmark containing 69 prompts: 56 outdoor and 13 indoor scenes spanning diverse natural, built, historical, stylized, and functional environments. Prompts range from open-ended requests to detailed descriptions with required objects and spatial relations. Every method uses the same asset catalog, rendering pipeline, and camera protocol and is run independently three times per prompt. Each scene is rendered from four perspective views and one top-down view. Because multiple realizations may satisfy a prompt, we evaluate alignment, efficiency, spatial plausibility, and completeness rather than reconstruction error. The prompts, category breakdown, and rendering details are provided in the supplementary material.

CLIPScore and runtime. We compute CLIPScore [8] with CLIP ViT-L/14 [14] for each prompt–view pair and average over the five views. Runtime measures wall-clock time from prompt submission to scene completion, including planning and construction but excluding offline rendering, VLM evaluation, and human evaluation. Both metrics are averaged over the three runs.

VLM evaluation. We evaluate *text–scene alignment* (prompt satisfaction), *spatial plausibility* (placement, support, scale, accessibility, and overlap), and *scene completeness* (sufficient, nonredundant supporting content). For each method pair, prompt, and run, the two sets of five views form an anonymous 2×5 composite with randomized top–bottom order. The VLM selects top, bottom, or tie separately for each dimension. We repeat the comparison with independently randomized ordering and report tie-adjusted preference, counting a tie as half a win. For method A, the preference score is

$$P_A = \frac{W_A + 0.5T}{W_A + L_A + T} \times 100\%, \tag{1}$$

where W_A, L_A, and T count wins, losses, and ties. 50% indicates parity. Human preferences use the same definition. These judgments assess visible scene quality from the rendered views. They do not directly test physical stability or successful navigation.

Human evaluation. We recruit 65 volunteers and exclude 14 who do not complete the required number of comparisons, leaving 51 participants and 1,860 judgments. Given the prompt and the same anonymous, randomized 2×5 composite, participants select the scene with better overall plausibility and consistency with the input text, or indicate no preference. Method identities are hidden, and no preference counts as half a win. Study oversight, consent, and protocol details are provided in the supplementary material.

6. Experiments

6.1. Experimental Setup

Unless otherwise stated, all experiments use Engine A as the target backend and follow the GameSceneBench protocol. All methods use DeepSeek-V4-Flash [21] with a context window of up to 204,800 tokens. Pairwise VLM evaluation uses `gpt-4o-2024-11-20`. For SCENEAGENT, the Orchestrator is limited to 30 interaction rounds, the Planner to 50 rounds and 40 tool calls, and each Builder session to 40 rounds. The Planner produces 5–10 outdoor or 3–6 indoor subtasks, and the Orchestrator allows up to three automatic refinement rounds.

For fairness, all baselines query the same Engine A retrieval service and asset catalog while retaining their own search prompts. We otherwise preserve their original planning, prompting, and construction procedures wherever possible. We evaluate AutoUE [24] through its scene-layout stage in the common Engine A pipeline. Full adaptation details are provided in the supplementary material. The experience library is frozen before evaluation and is not updated on GameSceneBench.

6.2. Main Comparison on GameSceneBench

We compare SCENEAGENT with AutoUE [24], Holodeck [22], ImperativeScene [7], Reason-3D [1],



Figure 2. Qualitative comparison on representative GameSceneBench prompts. Rows share an input prompt, and columns show different methods under the common Engine A asset and rendering protocol.

Table 2. GameSceneBench comparison. Panel (a) reports mean ± standard deviation. Panel (b) reports the tie-adjusted preference for SCENEAGENT against each comparator, with 50% indicating parity. † denotes AutoUE adapted to the common backend using its original scene-layout stage. Detailed results are provided in the supplementary material.

(a) Absolute metrics

Method	CLIPScore ↑	Runtime (s) ↓
AutoUE†	0.188 ± 0.030	$\mathbf{16.3 \pm 2.9}$
Holodeck	0.185 ± 0.030	60.7 ± 19.1
ImperativeScene	0.214 ± 0.030	21.4 ± 29.8
Reason-3D	0.196 ± 0.033	23.2 ± 11.4
SceneWeaver	0.193 ± 0.030	226.4 ± 140.0
SCENEAGENT	$\mathbf{0.216 \pm 0.034}$	203.2 ± 97.8

(b) SCENEAGENT preference vs. comparator (%)

Comparator	Align. ↑	Plaus. ↑	Comp. ↑	Human ↑
AutoUE†	94.2	74.4	91.3	89.4
Holodeck	81.5	74.1	77.1	85.2
ImperativeScene	77.3	65.3	74.9	78.2
Reason-3D	78.3	61.6	82.6	81.2
SceneWeaver	83.5	64.4	76.7	82.7

and SceneWeaver [23]. Table 2 reports the results.

Human evaluation consistently favors SCENEAGENT. Across 1,860 judgments, SCENEAGENT wins 76.9%, ties in 12.7%, and loses 10.3%, yielding an overall tie-adjusted preference of 83.3%. It is preferred to each baseline, with scores of 78.2–89.4%. Pairwise VLM evaluation agrees in direction: SCENEAGENT is favored over every comparator in all three dimensions, with aggregate preferences of 72.5–86.7%.

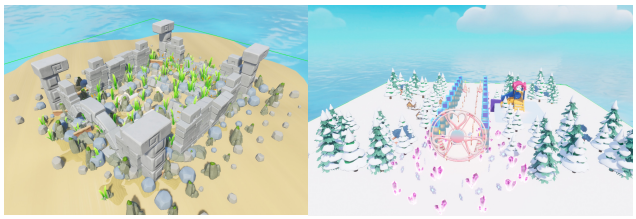


Figure 3. Representative generations in Engine B.

CLIPScore is also highest at 0.216, although the margin over ImperativeScene (0.214) is small. The preference results therefore provide complementary evidence about scene organization and content beyond global text–image similarity. These gains incur additional computation: average generation time is 203.2 seconds, lower than SceneWeaver but higher than the other baselines. Figure 2 provides representative outputs under the same asset and rendering protocol.

6.3. Cross-Engine and Cross-Model Deployment

Cross-engine. To demonstrate cross-backend portability, we deploy SCENEAGENT in Engine B and run all 69 GameSceneBench prompts while keeping the language model, agent protocol, SceneContext, core tools, skills, and

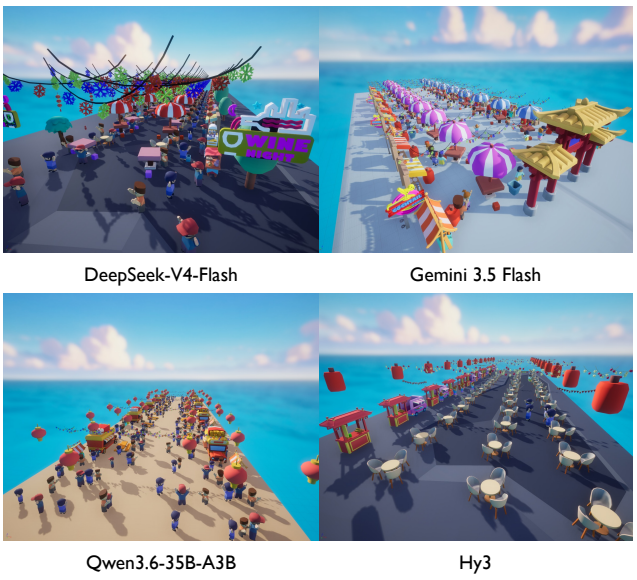


Figure 4. Cross-LLM comparison on the night-market prompt.

generation budget fixed. Only the adapter, asset catalog and database, and capability-tool wrappers change. Figure 3 shows representative Engine B generations. Additional outputs and export schemas appear in the supplement. This study tests whether the shared protocol can be retargeted through the stated backend boundary. Because the asset catalogs differ, the examples provide qualitative evidence of retargeting rather than a matched quality comparison.

Cross-LLM. We replace the main DeepSeek-V4-Flash backbone with Gemini 3.5 Flash [6], Hy3 [17], and Qwen3.6-35B-A3B [13], running each on all 69 GameSceneBench prompts with all other settings fixed.

Gemini achieves a slightly higher CLIPScore than DeepSeek (0.220 vs. 0.216) but is slower (374.8 vs. 203.2 seconds). Their aggregate VLM preferences are near parity (51.2% vs. 48.9%): Gemini is favored for alignment (52.4%) and plausibility (53.0%), while DeepSeek is favored for completeness (51.9%). Figure 4 presents a qualitative comparison across all four language models for the same prompt. These deployments show that the shared protocol accommodates multiple model families. The near-parity comparison between Gemini and DeepSeek illustrates that model choice can change runtime substantially without a corresponding overall preference gain.

6.4. Reusable Experience

Library-size ablation. We construct a frozen library of 50 human-approved Layout and Content skills from development prompts disjoint from GameSceneBench. We compare no skills, a random subset of 20 skills, and the full library,

Table 3. Effect of experience-library size.

(a) Absolute metrics by library size

Setting	CLIPScore ↑	Runtime (s) ↓
SceneAgent (0 skills)	0.211 ± 0.031	$\mathbf{173.3 \pm 56.4}$
SceneAgent (20 skills)	0.214 ± 0.032	190.8 ± 67.9
SceneAgent (50 skills)	$\mathbf{0.216 \pm 0.034}$	203.2 ± 97.8

(b) Full-library preference vs. reduced library (%)

Comparator	Align. ↑	Plaus. ↑	Comp. ↑
0 skills	59.7	46.2	63.8
20 skills	55.6	50.5	50.0

with all other settings unchanged. Table 3 reports the results.

The full library improves alignment and completeness over the no-skill variant, with preference scores of 59.7% and 63.8%, respectively. Spatial plausibility is 46.2%, giving an aggregate preference of 56.6% across the three dimensions. Thus, the clearest benefit is selecting and organizing prompt-relevant content. Additional guidance does not uniformly improve geometry. CLIPScore increases from 0.211 to 0.214 to 0.216 as the library grows from 0 to 20 to 50 skills. The full library's aggregate preference over 20 skills is 52.1%, with similar plausibility and completeness, indicating modest additional gains in this comparison. Runtime increases from 173.3 to 203.2 seconds. This ablation measures access to reusable experience. It does not isolate factorization from alternative memory representations.

Layout–Content composition. Figure 5 crosses representative Layout skills (rows) with Content skills (columns). Layout controls spatial organization, whereas Content controls semantic and stylistic realization. The examples illustrate recombination at inference time without storing each pairing separately. They illustrate the representation's compositional use.

Visual experience acquisition. Figure 6 illustrates how visual examples extend the same experience library. Gemini 3.5 Flash [6] extracts semantic, spatial, and construction cues from a reference image for the Planner–Builder pipeline. After human approval, the trace is factorized into Layout and Content skills through the standard update procedure. Later scenes reuse these skills without the source image.

6.5. Single- versus Multi-Agent Construction

We compare the full multi-agent SceneAgent with a monolithic single-agent variant on all 69 GameSceneBench prompts. The single-agent variant receives the original prompt and the complete construction-tool interface, and follows a plan-then-construct workflow specified in its system prompt. Unlike the full system, however, planning and construction are performed by the same agent within a single



Figure 5. Composition of Layout and Content skills. Rows fix the Layout skill, while columns fix the Content skill.

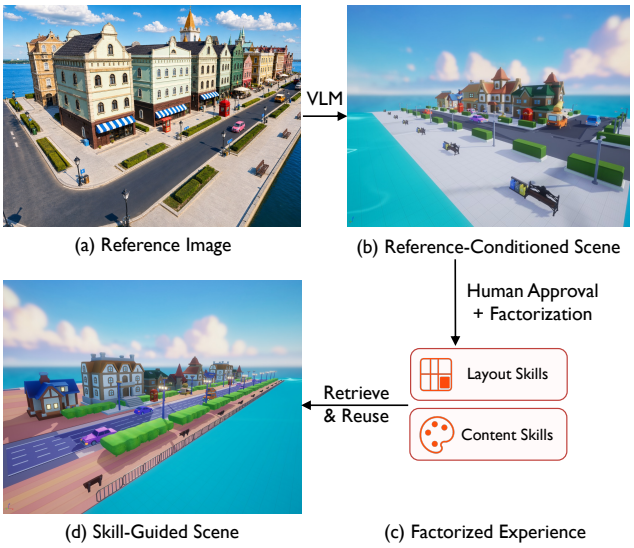


Figure 6. Visual experience acquisition and reuse. A reference image guides scene construction, and the human-approved trace is factorized into Layout and Content skills for reuse.

persistent interaction history. It does not instantiate separate Orchestrator, Planner, or Builder agents, nor does it use structured inter-agent handoffs or fresh task-local contexts.

Both variants use the same language model backbone, Engine A asset catalog, retrieval service, construction tools, rendering protocol, and evaluation procedure. We do not enforce an identical number of model or tool calls, because these calls are determined by the respective execution protocols. Runtime therefore reflects the end-to-end cost of each design. Each variant is run three times on every benchmark prompt. Table 4 reports the results.

The multi-agent system improves CLIPScore from 0.210 to 0.216. In the three VLM dimensions, it obtains tie-adjusted preference scores of 57.5% for alignment, 51.0% for spatial plausibility, and 62.4% for scene completeness. Aggregated across dimensions, it wins 54.6% of cases, ties in 4.8%, and loses 40.6%, corresponding to an overall preference score of 57.0%. The largest preference gain is in completeness, while plausibility is close to parity. Average runtime increases from 39.3 to 203.2 seconds with the additional planning, construction, and coordination stages. The result supports the integrated protocol for more complete construction, with a substantial quality–efficiency trade-off.

Table 4. Single- versus multi-agent construction on GameSceneBench. Results report mean ± standard deviation and tie-adjusted VLM preferences.

Metric	Single agent	Multi-agent
CLIPScore ↑	0.210 ± 0.032	$\mathbf{0.216 \pm 0.034}$
Runtime (s) ↓	$\mathbf{39.3 \pm 24.2}$	203.2 ± 97.8
VLM-Align. ↑	42.5	**57.5**
VLM-Plaus. ↑	49.0	**51.0**
VLM-Comp. ↑	37.7	**62.4**

The ablations consistently favor coverage of requested content, while spatial plausibility remains a challenge. The single-agent comparison tests decomposition, context management, and handoffs jointly. Attributing gains to individual mechanisms requires separate controls.

7. Conclusion

SceneAgent extends portability from scene output to the construction process and its reusable experience. Canonical SceneContext state connects fresh Builder sessions through structured handoffs, while factored Layout and Content skills preserve design guidance across generations. Backend adapters, asset configuration, and capability wrappers localize engine-specific implementation. On GameSceneBench, human and VLM comparisons favor SceneAgent over five baselines under a shared evaluation protocol. Ablations show the clearest benefits in alignment and completeness, and second-engine deployment demonstrates reuse of the shared protocol and skills. Persistent world state and composable experience provide a practical basis for portable scene construction.

Limitations. Porting still requires backend integration and compatible assets. The cross-engine study uses different catalogs and establishes qualitative retargeting rather than matched quantitative quality. Fresh Builder sessions avoid replaying raw histories, but do not guarantee constant context size as scenes grow. Multi-agent orchestration adds runtime, and human approval limits update scalability. The ablations test the integrated protocol and experience avail-

ability, leaving individual mechanisms and alternative memory representations for further study. Finally, rendered-view preferences and geometry checks do not establish physical stability, navigability, or gameplay success. These limits motivate more efficient orchestration, geometry-aware refinement, and broader validation of experience transfer.

LLM Usage Statement

The research uses DeepSeek-V4-Flash for scene construction, with Gemini 3.5 Flash, Hy3, and Qwen3.6-35B-A3B in cross-model experiments; Gemini 3.5 Flash also supports visual experience acquisition. GPT-4o (`gpt-4o-2024-11-20`) is used for the pairwise scene evaluation described in the paper. OpenAI GPT and Codex assisted with editing and refining author-produced manuscript content. The authors remain responsible for all scientific claims, experimental results, citations, and the final manuscript.

References

[1] Frédéric Berdoz, Luca A. Lanzendörfer, Nick Tuninga, and Roger Wattenhofer. Text-to-scene with large reasoning models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 2435–2443, 2026. 2, 5, 15

[2] Zixuan Bian, Ruohan Ren, Yue Yang, and Chris Callison-Burch. HOLODECK 2.0: Vision-language-guided 3D world generation with editing. arXiv preprint arXiv:2508.05899, 2025. 2

[3] Yinglin Duan, Zhengxia Zou, Tongwei Gu, Wei Jia, Zhan Zhao, Luyi Xu, Xinzhu Liu, Yenan Lin, Hao Jiang, Kang Chen, et al. LatticeWorld: A multimodal large language model-empowered framework for interactive complex world generation. arXiv preprint arXiv:2509.05263, 2025. 1, 2

[4] Weixi Feng, Wanrong Zhu, Tsu-Jui Fu, Varun Jampani, Arjun Akula, Xuehai He, Sugato Basu, Xin Eric Wang, and William Yang Wang. LayoutGPT: Compositional visual planning and generation with large language models. In *Advances in Neural Information Processing Systems*, pages 18225–18250, 2023. 2

[5] Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang Zhu, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, et al. Magentic-One: A generalist multi-agent system for solving complex tasks. arXiv preprint arXiv:2411.04468, 2024. 2

[6] Google. Gemini 3.5 Flash. `https://ai.google.dev/gemini-api/docs/models/gemini-3.5-flash`, 2026. Accessed: 2026-07-21. 7

[7] Maxim Gumin, Do Heon Han, Seung Jean Yoo, Aditya Ganeshan, R. Kenny Jones, Kailiang Fu, Rio Aguina-Kang, Stewart Morris, and Daniel Ritchie. Procedural scene programs for open-universe scene generation: LLM-free error correction via program search. In *ACM SIGGRAPH Asia 2025 Conference Papers*, pages 1–14. Association for Computing Machinery, 2025. 2, 5, 15

[8] Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. CLIPScore: A reference-free evaluation metric for image captioning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7514–7528. Association for Computational Linguistics, 2021. 5, 13

[9] 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 *Proceedings of the 41st International Conference on Machine Learning*, pages 19252–19282. PMLR, 2024. 2

[10] Tsung-Yi Lin, Chen-Hsuan Lin, Yin Cui, Yunhao Ge, Seungjun Nah, Arun Mallya, Zekun Hao, Yifan Ding, Hanzi Mao, Zhaoshuo Li, Yen-Chen Lin, Xiaohui Zeng, Qinsheng Zhang, Donglai Xiang, Qianli Ma, J. P. Lewis, Jingyi Jin, Pooya Jannaty, and Ming-Yu Liu. GenUSD: 3D scene generation made easy. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Real-Time Live!*, pages 1–2. Association for Computing Machinery, 2024. 2

[11] Lu Ling, Chen-Hsuan Lin, Tsung-Yi Lin, Yifan Ding, Yu Zeng, Yichen Sheng, Yunhao Ge, Ming-Yu Liu, Aniket Bera, and Zhaoshuo Li. SceneThesis: A language and vision agentic framework for 3D scene generation. arXiv preprint arXiv:2505.02836, 2025. 2

[12] Jun Luo, Jiaxiang Tang, Ruijie Lu, and Gang Zeng. SceneAssistant: A visual feedback agent for open-vocabulary 3D scene generation. arXiv preprint arXiv:2603.12238, 2026. 2

[13] Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all. `https://qwen.ai/blog?id=qwen3.6-35b-a3b`, 2026. Accessed: 2026-07-21. 7

[14] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021. 5, 13

[15] Hou In Ivan Tam, Hou In Derek Pun, Austin T. Wang, Angel X. Chang, and Manolis Savva. SceneMotifCoder: Example-driven visual program learning for generating 3D object arrangements. In *Proceedings of the 2025 International Conference on 3D Vision*, pages 179–188, 2025. 2

[16] Song Tang, Kaiyong Zhao, Lei Wang, Yuliang Li, Xuebo Liu, Junyi Zou, Qiang Wang, and Xiaowen Chu. UnrealLLM: Towards highly controllable and interactable 3D scene generation by LLM-powered procedural content generation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 19417–19435. Association for Computational Linguistics, 2025. 1, 2

[17] Tencent Hy Team. Hy3. `https://hy.tencent.com/research/hy3`, 2026. Accessed: 2026-07-21. 7

[18] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. 2

[19] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the*

42nd International Conference on Machine Learning, pages 63897–63911. PMLR, 2025. 2

[20] Hongchi Xia, Xuan Li, Zhaoshuo Li, Qianli Ma, Jiashu Xu, Ming-Yu Liu, Yin Cui, Tsung-Yi Lin, Wei-Chiu Ma, Shenlong Wang, Shuran Song, and Fangyin Wei. SAGE: Scalable agentic 3D scene generation for embodied AI. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22358–22368, 2026. 2

[21] Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. DeepSeek-V4: Towards highly efficient million-token context intelligence. arXiv preprint arXiv:2606.19348, 2026. 5, 15

[22] Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, et al. Holodeck: Language-guided generation of 3D embodied AI environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16227–16237, 2024. 2, 5, 15

[23] Yandan Yang, Baoxiong Jia, Shujie Zhang, and Siyuan Huang. SceneWeaver: All-in-one 3D scene synthesis with an extensible and self-reflective agent. In *Advances in Neural Information Processing Systems*, pages 140319–140351, 2025. 2, 6, 15

[24] Lei Yin, Wentao Cheng, Zhida Qin, Tianyu Huang, Yidong Li, and Gangyi Ding. AutoUE: Automated generation of 3D games in Unreal Engine via multi-agent systems. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 2341–2364. Association for Computational Linguistics, 2026. 2, 5, 15

[25] Chenglin Yu, Yuchen Wang, Songmiao Wang, Hongxia Yang, and Li Ming. InfiAgent: An infinite-horizon framework for general-purpose autonomous agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 35884–35894. Association for Computational Linguistics, 2026. 2

[26] Shougao Zhang, Mengqi Zhou, Yuxi Wang, Chuanchen Luo, Rongyu Wang, Yiwei Li, Zhaoxiang Zhang, and Junran Peng. CityX: Controllable procedural content generation for unbounded 3D cities. arXiv preprint arXiv:2407.17572, 2024. 2

[27] Yunzhi Zhang, Zizhang Li, Matt Zhou, Shangzhe Wu, and Jiajun Wu. The scene language: Representing scenes with programs, words, and embeddings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 24625–24634, 2025. 2

[28] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 19632–19642, 2024. 2

SceneAgent: A Multi-Agent Framework for Portable Game Scene Generation

Supplementary Material

This supplementary material provides the complete benchmark specification, implementation and evaluation details, complete outcome distributions, cross-engine export examples, representative factored skills, and additional qualitative results. It contains the following:

1. The composition, category coverage, and complete prompt list of GameSceneBench.
2. The construction and agent-control interfaces, including the implementation of entity-first asset retrieval.
3. The rendering, runtime, and automated-evaluation protocols, including the pairwise VLM evaluation protocol.
4. The human-evaluation protocol and study interface.
5. The common-backend adaptation and runtime accounting for the evaluated baselines.
6. Complete win, loss, and tie distributions for the main comparison.
7. Cross-engine serialization examples and representative Layout, Content, and thin-recipe records.
8. Additional qualitative results from Engines A and B.

A. GameSceneBench Specification

A.1. Benchmark Composition

GameSceneBench is a text-driven benchmark for complete 3D game scene generation. It contains 69 user prompts, comprising 56 outdoor scenes and 13 indoor scenes. Each entry contains a unique scene identifier, a display name, an indoor–outdoor subset label, and the prompt supplied to every evaluated method. The prompts range from short, open-ended requests to detailed descriptions that specify required objects and spatial relations. The benchmark therefore tests both broad scene completion and instruction following under compositional constraints.

The outdoor and indoor subsets cover the scene families summarized in Table 5. These categories summarize the types of scenes covered by the benchmark. They do not prescribe a unique layout or visual appearance for each prompt.

Each method receives the same input prompt and produces a complete 3D scene using the common asset catalog, rendering pipeline, and standardized camera protocol. Every method is run independently three times for each prompt to account for generation variability. Because the benchmark admits multiple valid scene realizations and does not define a unique reference scene, the evaluation measures text–scene alignment, generation efficiency, spatial plausibility, and scene completeness rather than reconstruction error against a single fixed ground truth.

Table 5. Category coverage of GameSceneBench.

Subset	Scene families
Outdoor (56)	Natural landscapes, settlements, sports and recreational spaces, historical and stylized environments, and urban infrastructure.
Indoor (13)	Residential, workplace, commercial, hospitality, entertainment, and utility spaces.

A.2. Complete Benchmark Prompts

The following list provides the complete set of 69 prompts used in GameSceneBench. These prompts cover a diverse range of scene types, visual themes, spatial organizations, and content requirements, and are used consistently across all methods and evaluation settings. Stable scene identifiers are shown in monospace to make each prompt directly traceable and comparable with generation outputs, rendering results, and evaluation records. A machine-readable transcription of this list is included as `gamescenebench.json` in the supplementary archive.

Outdoor scenes (56).

1. `park_square`: Generate an urban park square scene.
2. `mongolia_grassland`: Generate an Inner Mongolian grassland scene.
3. `rural_village`: Generate a rural village scene.
4. `winter_town`: Generate a winter town scene.
5. `ice_wonderland`: Generate an ice-and-snow wonderland with a scenic promenade, a giant ice slide, a snowflake-shaped Ferris wheel, and lively interactive winter attractions that convey the distinctive atmosphere of an ice city.
6. `primitive_tribe`: Generate a primitive tribal settlement scene.
7. `egypt_adventure`: Generate an ancient Egyptian expedition scene with a wooden rail track and mine cart leading to a cave, as well as expedition tents and wooden crates.
8. `chinese_garden`: Generate a classical Chinese garden with traditional buildings, a pond, a small pavilion, and covered corridors.
9. `epic_valley`: Generate a valley flanked by steep mountain ranges covered with fir forests. A winding river should cross the valley floor and form pebble banks and shallows, with dense vegetation and rocks in the foreground.
10. `summer_field`: Generate a summer field scene.
11. `venice`: Generate a Venetian canal-city scene.
12. `school_track`: Generate a school athletic field surrounded by classroom buildings and lush trees.
13. `mt_fuji`: Generate a scene at the foot of snow-covered Mount Fuji, where a vintage tram travels along a quiet Japanese street lined with blooming pink cherry trees, creating a beautiful and soothing atmosphere.
14. `flower_garden`: Generate a small flower-garden scene.
15. `beach_resort`: Generate a resort beach scene.
16. `volleyball_court`: Generate a beach volleyball scene.
17. `japanese_onsen`: Generate an outdoor Japanese hot-spring scene.
18. `pine_forest`: Generate a pine forest scene.
19. `zen_garden`: Generate a dry-landscape Zen garden.

20. baseball_field: Generate a baseball field scene.
21. sakura_grove: Generate a cherry-blossom grove.
22. european_garden: Generate a European-themed garden.
23. pond_stream: Generate a pond-and-stream scene.
24. desert_gobi: Generate a desert and Gobi landscape.
25. desert_oasis: Generate a desert oasis surrounded by sparse cacti of different species and several dead trees.
26. desert_camp: Generate a desert-oasis campsite surrounded by scattered dead trees. Place a clear circular pool at the center, with dense flowers, shrubs, and several tall palm trees around it. Include a stone-built water point and several tents beside the pool.
27. swimming_pool: Generate an outdoor swimming pool with white lounge chairs and sun umbrellas along the poolside, conveying a relaxed vacation atmosphere.
28. creek_canyon: Generate a stream-canyon scene.
29. tropical_beach: Generate a tropical palm beach.
30. tropical_resort: Generate a tropical seaside resort.
31. tomb_raider: Generate a tomb-exploration adventure scene.
32. bamboo_forest: Generate a bamboo forest scene.
33. basketball_court: Generate a regulation basketball court.
34. tennis_court: Generate a tennis court scene.
35. soccer_field: Generate a regulation soccer field.
36. snow_camp: Generate a snowy campsite enclosed by dense snow-covered coniferous forest. The central clearing should have level snow, several tents, and a wind-sheltered fire pit built from stones. Place several wooden stakes around the campsite as boundary markers.
37. chinese_temple: Generate a traditional Chinese temple scene.
38. military_base: Generate a military base scene.
39. military_march: Generate a scene of a Chinese military unit marching through the countryside.
40. villa_town: Generate a villa town with several neatly arranged modern villas along level streets, each with a small garden. Include a circular fountain plaza at the center, and enclose the scene with dense trees and orderly hedges.
41. ancient_city: Generate a scene depicting an assault on a fortified city in ancient China.
42. ancient_ruins: Generate ancient architectural ruins with broken walls, drifting sand, and overgrown weeds.
43. subway_station: Generate a subway station scene.
44. city_street: Generate an urban street scene with a roadway and ordinary buildings.
45. night_market: Generate a night-market food street.
46. wasteland_defense: Generate a post-apocalyptic wasteland defense position centered on a circular base enclosed by abandoned cars, barbed wire, and sandbags. Include a campfire and several worn tents inside the base.
47. airport: Generate an international airport scene.
48. forest_town: Generate a forest town surrounded by dense woodland. At its center, stone-paved streets connect rows of wooden and brick houses. Include an open-air market in the central square with stalls displaying furs, herbs, and handicrafts, and place a fenced stable and a blacksmith shop at the edge of the settlement.
49. amusement_park: Generate an amusement park scene.
50. banana_factory: Generate a banana-processing factory.
51. tang_dynasty: Generate a prosperous Tang-dynasty Chang'an scene with a lively food street, an ornate palace, and lanterns filling the sky.
52. western_cowboy: Generate an American Western scene in which a train departs along a railroad bordered by sparse cacti of different species.
53. western_town: Build an American frontier-era cowboy town with wooden buildings lining both sides of a main street.
54. supermarket_parking: Generate a Sam's Club parking-lot scene.
55. marketplace: Generate a marketplace scene.
56. graveyard: Generate a graveyard scene.

Indoor scenes (13).

1. indoor_cozy_bedroom_tv: Generate a cozy double bedroom with the head of the bed against a wall, one nightstand and table lamp on each side, and a wall-mounted television directly opposite the foot of the bed.
2. indoor_living_room: Generate a modern apartment living room.
3. indoor_dining_room: Generate a family dining room with a rectangular dining table at the center, four chairs facing the table, a sideboard against a wall, and a pendant light directly above the table.
4. indoor_home_office: Generate a home office with a desk against a wall, an office chair facing the desk, a bookcase against a wall, and a reading corner composed of a floor lamp and a small plant.
5. indoor_kitchen: Generate a kitchen.
6. indoor_meeting_room: Generate a large meeting room.
7. indoor_gym: Generate the interior of a gym.
8. indoor_fruit_store: Generate the interior of a fruit store.
9. indoor_hotel_lobby: Generate a hotel lobby.
10. indoor_laundry_room: Generate a laundry room.
11. indoor_nursery: Generate a nursery.
12. indoor_gaming_room: Generate a gaming room.
13. indoor_garage: Generate a garage interior.

B. Agent Tool Interfaces

B.1. Builder Construction Interface

Builder agents operate on the canonical SceneContext through the shared tool interface summarized in Table 6. The interface covers the complete construction loop: grounding assets, placing individual objects, synthesizing parametric structures, populating regions, inspecting the evolving scene, and making targeted corrections. It also provides a consistent execution boundary through which Builders can query existing scene state, validate intermediate results, and incrementally refine previously constructed content without directly accessing backend-specific APIs. Tools consume and return builder-coordinate quantities, while backend adapters handle engine-specific coordinates, units, asset bindings, command execution, and scene serialization. This separation allows the same Builder policy and construction protocol to remain unchanged across supported backends.

Entity-First Asset Retrieval with `search_modules`. `search_modules` maps a natural-language asset request to a ranked list of candidates from the active backend's asset catalog. Textual asset cards are encoded offline and stored in an approximate nearest-neighbor index, while each incoming query is encoded online and ranked by cosine similarity. The retriever follows an entity-first objective: candidates should match the requested object category before satisfying secondary attributes such as style, material, condition, or scene context. Its training data combine asset-grounded query synthesis with semantically similar hard negatives, including deceptive negatives that match the requested attributes but represent the wrong entity. A shared encoder is trained with a standard InfoNCE objective. The returned top-K candidates remain proposals rather than final bindings. Agents may inspect their metadata and thumbnails before selecting an asset.

Table 6. Builder-visible construction tools. The interface is shared across backends. Backend-specific configuration supplies compatible asset categories and export conventions.

Tool family	Tools
Asset discovery and grounding	`search_modules`, `get_module_info`, `web_search`, `view_thumbnails`, `plan_assets`, `get_planned_assets`
Direct and relation-aware placement	`place_module`, `place_facing`, `place_against_wall`, `place_on_ceiling`
Parametric geometry and structures	`create_floor`, `build_room`, `create_line`, `create_rect`, `create_circle`, `create_arc`, `build_column`, `build_board`, `build_stairs`, `create_shape`
Procedural layout and population	`region_define`, `get_landscape_data`, `surface_sampler`, `transform_points`, `remove_points_by_region`, `batch_place`, `pcg_scatter`
Scene-state queries and handoff	`get_scene_summary`, `update_conventions`, `report_subtask`
Targeted editing and rollback	`undo_last`, `clear_scene`, `remove_by_type`, `remove_by_region`, `undo_subtask`
Geometry verification	`check_bbox_collision`

Table 7. Role-specific control interfaces. Area-budget estimation is enabled for area-aware planning, and room-size estimation is enabled for indoor planning.

Agent	Control tools
Orchestrator	`load_skill`, `plan_scene`, `revise_plan`, `build_scene`, `refine_scene`, `export_scene`
Planner	`search_modules`, `get_module_info`, `web_search`, `view_thumbnails`, `estimate_area_budget`, `estimate_room_size`, `commit_subtasks`
Builder	`update_conventions`, `report_subtask`, plus the construction tools in Table 6

B.2. Agent-Control Interface

The role-specific tools in Table 7 delimit the responsibilities of the Orchestrator, Planner, and Builders. The Orchestrator does not place actors directly, and the Planner commits semantic subtasks rather than modifying SceneContext. Final serialization is kept outside individual Builder sessions.

The low-level `export_json` operation is hidden from Builder sessions and invoked through the Orchestrator's `export_scene` interface. This keeps construction backend-neutral and localizes final serialization to the active backend adapter.

C. Rendering and Automated Evaluation

C.1. Five-View Rendering Protocol

For each generated scene, the camera rig is centered on the bounding box that encloses all valid actors. Its distance is adjusted automatically according to the scene extent. We render four perspective views distributed around the scene at an elevation angle of approximately 30°, together with one top-down view. The perspective views capture the scene from complementary directions, whereas the top-down view exposes the global layout, object distribution, and spatial relations. All views use a 60° field of view and a resolution of 1920×1080.

The same camera-placement and rendering procedure is used for all methods. This controlled setup reduces variation from presentation and helps isolate differences in planning and scene construction.

C.2. CLIPScore and Runtime

We compute CLIPScore [8] using CLIP ViT-L/14 [14]. For each generated scene, we compute the score between the input prompt and each of the five rendered views and average the five per-view scores. We then average over the three independent runs for each prompt when aggregating benchmark results.

Runtime is wall-clock time from prompt submission to completion of the scene file. It includes planning and construction and excludes offline rendering, VLM evaluation, and human evaluation. Runtime is reported in seconds and averaged over the three independent runs.

C.3. Pairwise VLM Evaluation

The VLM evaluation considers three complementary dimensions:

- **Text–scene alignment**: whether the scene satisfies the objects, style, spatial relations, and other constraints in the prompt.
- **Spatial plausibility**: object placement, support, scale, accessibility, and unreasonable overlap.
- **Scene completeness**: whether the scene contains sufficient, nonredundant supporting content to form a coherent environment.

For each method pair, prompt, and run, the five views of both scenes are letterboxed into an anonymous 2×5 composite. The assignment of methods to the top and bottom rows is randomized. The VLM answers a separate pairwise question for each dimension with three possible outcomes: top, bottom, or tie. We repeat each comparison with independently randomized positions. The main experiments use `gpt-4o-2024-11-20` for this evaluation.

For a method A, the reported tie-adjusted preference is

$$P_A = \frac{W_A + 0.5T}{W_A + L_A + T} \times 100\%, \tag{2}$$

where W_A, L_A, and T denote the numbers of wins, losses, and ties for A, respectively. Thus, 50% indicates parity.

D. Human Evaluation Protocol

Study oversight and consent. The human evaluation consisted of preference judgments between rendered synthetic scenes. No formal institutional ethics review was conducted because the study was limited to scene-preference evaluation. All participants were informed about the study and provided informed consent before taking part.

We conducted a pairwise human preference study using the same anonymous 2×5 composites as in the VLM evaluation. Participants saw the input prompt and the two sets of rendered views, with method identities hidden and the top–bottom order randomized. For each comparison, they selected the scene with better overall plausibility and consistency with the input text, or indicated no preference. No preference is counted as half a win when computing the tie-adjusted score.

The study interface offered 50 trials and required at least 30 completed comparisons, as shown in Figure 7. We recruited 65 volunteers. 14 participants were excluded because they did not complete the required number of comparisons, leaving 51 participants and 1,860 retained judgments. The judgments were approximately balanced across comparisons between SCENEAGENT and the five baselines.

The instruction screen shown before evaluation is reproduced in Figure 7. It explains the blind comparison setting, response options, three evaluation dimensions, keyboard shortcuts, and completion requirement. Figure 8 shows the trial interface. The prompt and anonymous 2×5 composite occupy the main panel, while the response controls, optional notes, trial navigation, and progress indicator remain visible in a fixed side panel.

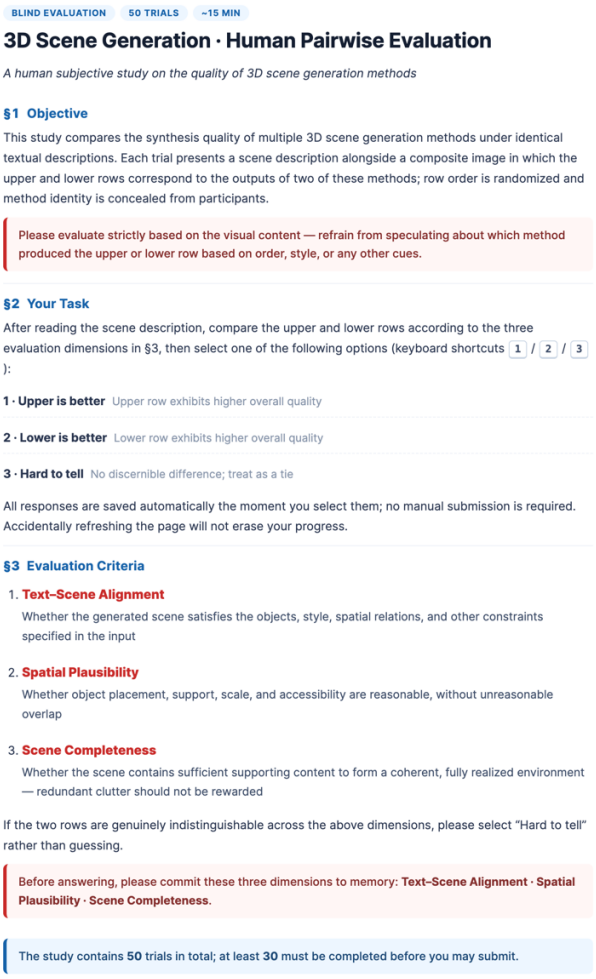


Figure 7. Instructions presented to participants before the human pairwise evaluation. The page describes the blind protocol, response options, evaluation dimensions, and minimum completion requirement.

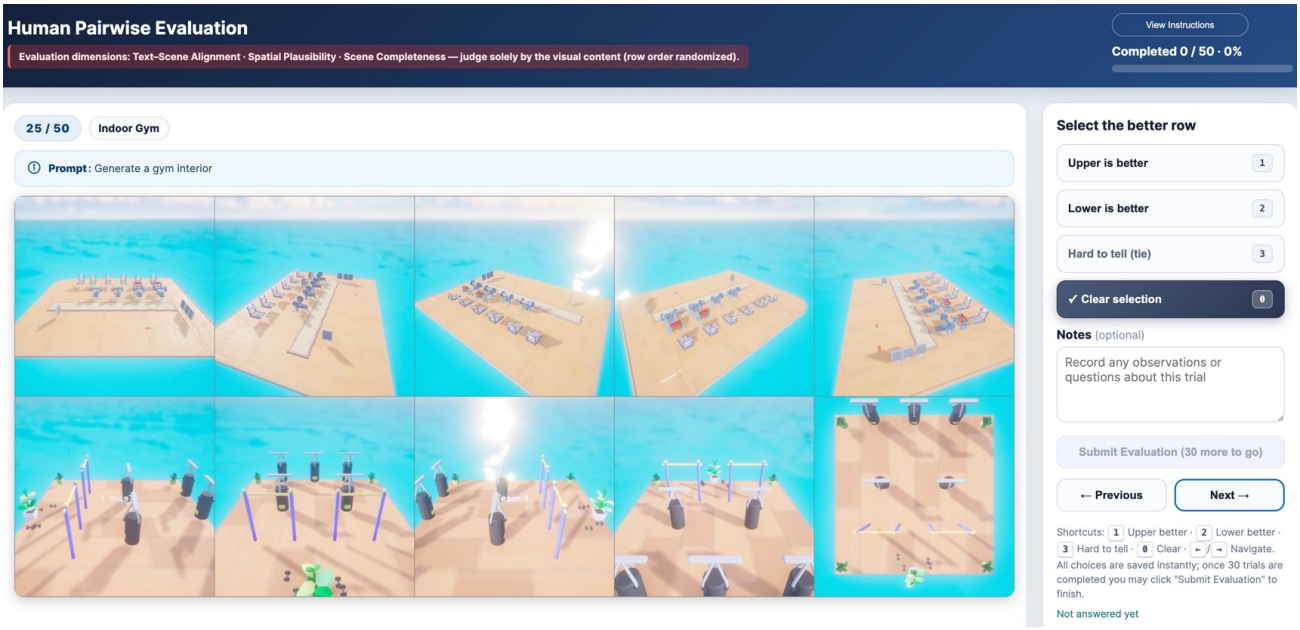


Figure 8. Human pairwise-evaluation interface. Each trial presents the scene prompt and an anonymous, randomly ordered 2×5 composite containing five views per method. Participants select the better row or a tie, may record optional notes, and navigate through the assigned trials.

E. Baseline Adaptation & Runtime Accounting

E.1. Baseline-Specific Adaptations

We compare SCENEAGENT against five representative baselines spanning distinct paradigms of automatic 3D scene generation. AutoUE [24] prompts an LLM to author a procedural scene graph executed by a game-engine runtime. Holodeck [22] synthesizes indoor layouts through retrieval-guided constraint solving. ImperativeScene [7] expresses scene layouts as LLM-generated imperative programs. Reason-3D [1] pairs LLM-driven spatial reasoning with embedding-based asset retrieval. SceneWeaver [23] integrates LLM-based scene planning with a procedural asset-generation engine.

These methods were originally developed around mutually incompatible asset ecosystems and rendering backends, many of which rely on proprietary, online-only, or platform-specific components. Their original pipelines therefore cannot be deployed uniformly and reproducibly in a single evaluation environment. To enable a fair comparison, we standardize three infrastructure layers across all five baselines:

- **Asset retrieval**: retrieval is redirected to a shared 3D asset repository accessed through a common text-based retrieval service. Each method retains its own search prompt or query formulation.
- **Execution and rendering**: layout execution and rendering are routed through the shared Engine A pipeline and camera protocol.
- **Language model**: every method uses DeepSeek-V4-Flash [21] with a context window of up to 204,800 tokens.

The adaptations retain each method's core scene-generation procedure, including its graph structure, constraint formulation, program templates, or reasoning workflow, within the common execution environment. All five baselines and SCENEAGENT use the same asset repository, retrieval service, rendering pipeline, and language model. This controls major sources of infrastructure variation and compares the adapted generation procedures. It does not reproduce each original system's full native capabilities. Results should be interpreted within the common-backend setting.

E.2. AutoUE Adaptation

AutoUE [24] is designed for end-to-end game generation in Unreal Engine. For the common-backend comparison, we evaluate its scene-layout stage. The adaptation retains layout generation, asset grounding, and scene instantiation, while excluding Unreal-Engine-specific gameplay, interaction, and testing stages. This condition is denoted AutoUE† in the result tables.

E.3. Runtime Boundary

The same runtime boundary is applied to every evaluated method: measurement begins when the prompt is submitted and ends when scene generation completes. Planning, model interaction, retrieval, tool invocation, and construction

are included when performed by the method. Offline rendering, VLM scoring, and human evaluation are excluded. This measures end-to-end generation cost while avoiding evaluation-time overhead.

For SCENEAGENT, the Orchestrator is limited to 30 interaction rounds, the Planner to 50 rounds and 40 tool calls, and each Builder to 40 rounds. The Planner produces 5–10 subtasks for outdoor scenes and 3–6 subtasks for indoor scenes. The Orchestrator may invoke at most three automatic refinement rounds. The experience library is frozen before evaluation and is not updated using any GameSceneBench generation or feedback.

F. Complete Outcome Distributions and Cross-Engine Details

F.1. Main Comparison on GameSceneBench

The main paper reports the aggregate CLIPScore, runtime, and tie-adjusted preferences. Table 8 instead provides the underlying three-way outcome distributions, preserving how frequently SCENEAGENT wins outright, the baseline wins outright, or a comparison is judged a tie. The human-study panel also gives the exact number of retained judgments for each baseline comparison. These counts sum to 1,860.

F.2. Cross-Engine Portability

The cross-engine study runs all 69 prompts in Engine B with the same language model, agent protocol, SceneContext, core construction interface, skill library, and generation budget. The adapter, asset catalog and database, and capability wrappers are backend-specific. Because the catalogs differ, the study demonstrates deployment through this boundary rather than a matched quantitative comparison of engine quality. Here, abridged export payloads illustrate how canonical state is serialized for each backend.

Backend export payloads. The canonical SceneContext is converted into the native import schema of each target engine. Listings 1 and 2 show abridged examples of the final files delivered to Engines A and B, respectively. Only one representative actor is shown in each listing. Thus, the listings illustrate the serialization boundary rather than reproducing the full scenes.

Engine A stores the actor array directly. Asset identifiers are strings, positions use the engine's centimeter convention, and each actor carries a display name, an instance name, collision and mode flags, an instance index, a quaternion rotation, scale, and optional color slots. For compactness, floating-point values are rounded in the excerpt.

Listing 1. Abridged Engine A import payload.

```
[
  {"actors": [{
    "typeId": "20009245",
    "name": "Simple Residential Stone Floor 01",
    "gname": "asset0", "c": 1,
    "pos": [0.0, 0.0, -5.0],
    "rot": [0.0, 0.0, 0.0, 1.0],
    "sca": [27.550, 26.732, 0.080],
    "col": [], "id": 0, "m": 1
  }]}
]
```

Engine B wraps scene generation in a `PCGGenerate` command. Its `Param.Info` field contains a serialized JSON object with the actor, group, and relation records. Asset identifiers are numeric, positions are converted to meters, and transform components are serialized as fixed-precision strings. The per-export `AIID` is generated at runtime. The listing separates the decoded `Info` payload solely to make the nested schema readable.

Listing 2. Abridged Engine B import payload.

```
[
  {"Type": "PCGGenerate",
   "Param": {
     "AIID": "<generated-id>",
     "bNeedGroup": false,
     "Info": "<serialized payload below>"
   }}
]

Decoded Param.Info:
{
  "actors": [{
    "typeId": 1101002001004002,
    "pos": ["0.000", "0.000", "-0.025"],
    "sca": ["40.000", "40.000", "0.017"],
    "rot": ["0.000", "0.000", "0.000", "1.000"],
    "c": 0, "col": [[235, 240, 255]]
  }],
  "group": {},
  "relation": []
}
```

G. Reusable Experience Update

G.1. Human-Led Quality Gate

The main paper describes the human-led quality gate and the roles of automatic diagnostics, schema validation, and deduplication. Algorithm 1 provides the corresponding procedural specification.

G.2. Representative Factored-Skill Examples

Listings 3–5 show an abridged, mutually compatible triplet from the experience library. The Layout skill contains only reusable spatial organization and geometric constraints. The Content skill specifies semantic roles, stylistic attributes, and substitution rules without backend-specific asset bindings. The thin recipe references both skills while retaining only requirements specific to the target scene.

Table 8. Complete pairwise outcome distributions for the main comparison. VLM entries report mean ± standard deviation. "Overall" aggregates the three VLM dimensions. Human percentages are computed over the retained judgments for each comparator and may not sum to exactly 100% after rounding.

(a) Raw VLM outcomes by comparator and dimension (%)

Comparator	Dimension	SCENEAGENT win	Baseline win	Tie
AutoUE†	Overall	85.4 ± 1.6	12.1 ± 2.1	2.6 ± 1.5
	Alignment	94.2 ± 2.9	5.8 ± 2.9	0.0 ± 0.0
	Plausibility	70.5 ± 2.2	21.7 ± 3.8	7.7 ± 4.4
	Completeness	91.3 ± 0.0	8.7 ± 0.0	0.0 ± 0.0
Holodeck	Overall	76.2 ± 3.6	21.1 ± 3.4	2.7 ± 0.8
	Alignment	81.2 ± 2.9	18.4 ± 2.2	0.5 ± 0.8
	Plausibility	70.5 ± 10.2	22.2 ± 9.3	7.2 ± 2.9
	Completeness	76.8 ± 3.8	22.7 ± 4.2	0.5 ± 0.8
ImperativeScene	Overall	71.0 ± 2.7	26.1 ± 2.2	2.9 ± 0.5
	Alignment	77.3 ± 3.3	22.7 ± 3.3	0.0 ± 0.0
	Plausibility	60.9 ± 1.4	30.4 ± 0.0	8.7 ± 1.4
	Completeness	74.9 ± 4.7	25.1 ± 4.7	0.0 ± 0.0
Reason-3D	Overall	72.8 ± 1.6	24.5 ± 0.2	2.7 ± 1.4
	Alignment	77.8 ± 3.0	21.3 ± 3.0	1.0 ± 1.7
	Plausibility	58.0 ± 5.0	34.8 ± 2.5	7.2 ± 2.5
	Completeness	82.6 ± 2.5	17.4 ± 2.5	0.0 ± 0.0
SceneWeaver	Overall	72.3 ± 2.3	22.7 ± 1.3	5.0 ± 2.2
	Alignment	82.5 ± 1.3	15.5 ± 1.6	1.9 ± 0.8
	Plausibility	58.3 ± 4.8	29.6 ± 0.7	12.1 ± 4.4
	Completeness	76.2 ± 2.1	22.8 ± 3.0	1.0 ± 1.7

(b) Raw human-study outcomes by comparator (%)

Comparator	SCENEAGENT win	Baseline win	Tie	Judgments	Qualified sessions
AutoUE†	83.4	4.8	11.9	379	51
Holodeck	78.5	8.2	13.3	368	51
ImperativeScene	71.8	15.4	12.8	383	51
Reason-3D	75.0	12.6	12.4	364	51
SceneWeaver	76.0	10.7	13.4	366	51

Algorithm 1 Human-Led Quality-Gated Experience Update

Initialize Layout library $\mathcal{L}$, Content library $\mathcal{C}$, and recipe library $\mathcal{R}$
for each generation trace τ **do**
  Collect SceneContext, renderings, and automatic diagnostics
  Present τ and its diagnostics for human review
  if the reviewer approves τ **then**
    Extract Layout candidates $\hat{L}$, Content candidates $\hat{C}$, and recipe r
    Reject candidates that violate the transferable-skill schemas
    Deduplicate and update $\mathcal{L}$ and $\mathcal{C}$; insert r into $\mathcal{R}$
  else
    Store τ and reviewer feedback as failure evidence
  end if
end for

Listing 3. Representative Layout skill containing backend- and asset-independent linear organization.

```
type: linear_guide
category: layout
name: Linear Guided Layout
when_to_use: >
  Scenes organized around one dominant route with short
  branches that quickly reconnect to the main path.

topology:
  structure: linear
  anchors: [entrance, terminal_focus]
  main_axis: shared_by_all_route_segments

spatial_rules:
- Connect the entrance and terminal focus with a
    continuous route.
- Arrange major regions along both sides of the route.
- Preserve an unobstructed circulation corridor.
- Orient flanking structures toward the route.
- Frame the scene without blocking traversal.

constraints:
- Derive all route segments from one consistent axis.
- Validate footprints before placing large structures.
- Align directional elements with the route.
```

Listing 4. Representative Content skill containing semantic and stylistic guidance without concrete asset identifiers.

```
type: night_market_chinese_palette
category: content
```

```
name: East Asian Night-Market Content
when_to_use: >
  Lively night markets and pedestrian food streets with
  warm illumination and dense everyday details.

semantic_roles:
  vendors: [food stall, snack booth, market cart,
     canopy]
  lighting: [red lantern, string lights, neon sign]
  furniture: [small table, chair, bench, stool]
  supporting_props: [crate, signboard, waste bin,
                     utility box]

style:
  palette:
    base: [dark gray, asphalt black, warm wood]
    lighting: [red, orange, yellow]
    accents: [saturated red, blue, green]
  appearance: stylized or low-poly
  atmosphere: lively, warm, dense, and informal

substitution_rules:
- Replace a missing food stall with a
     canopy-and-counter assembly.
- Replace neon signage with illuminated hanging signs.
- Vary vendor and furniture types to avoid repetition.
```

Listing 5. Representative thin recipe containing scene-specific residuals for the selected Layout and Content skills.

```
type: night_market
category: recipe
name: Lively Night-Market Street

composed_of:
  layout: linear_guide
  content: night_market_chinese_palette

residual_requirements:
- Place two staggered rows of vendors along a
     pedestrian street.
- Preserve a continuous central corridor from entrance
     to endpoint.
- Insert small dining pockets between selected vendors.
- Suspend lighting across the street at irregular
     intervals.
- Use a gate or prominent sign as the terminal focus.

validation_notes:
- Check hanging-prop pivots before applying height
     offsets.
- Ensure stalls, furniture, and decorations do not
     block circulation.
```

H. Additional Qualitative Results

Figures 9–16 show additional generations in Engine A, and Figures 17–18 show additional generations in Engine B.



Figure 9. Additional qualitative results in Engine A (set 1). From top to bottom: Amusement Park, Ancient Siege, Ancient Ruins, Basketball Court, and Beach Resort. Each row shows four views of the same generated scene.



Figure 10. Additional qualitative results in Engine A (set 2). From top to bottom: Chinese Garden, Chinese Temple, City Street, Desert Camp, and Egypt Adventure. Each row shows four views of the same generated scene.



Figure 11. Additional qualitative results in Engine A (set 3). From top to bottom: Japanese Onsen, Ice Wonderland, European Garden, Forest Town, and Graveyard. Each row shows four views of the same generated scene.



Figure 12. Additional qualitative results in Engine A (set 4). From top to bottom: Pond Stream, Pine Forest, Military Base, Military March, and Night Market. Each row shows four views of the same generated scene.



Figure 13. Additional qualitative results in Engine A (set 5). From top to bottom: Tomb Raider, Tang Dynasty, Primitive Tribe, Sakura Grove, and Parking Lot. Each row shows four views of the same generated scene.



Figure 14. Additional qualitative results in Engine A (set 6). From top to bottom: Winter Town, Western Town, Snow Camp, Subway Station, and Villa Town. Each row shows four views of the same generated scene.



Figure 15. Additional qualitative results in Engine A (set 7). From top to bottom: Venice, Hotel Lobby, Laundry Room, Bedroom, and Kitchen. Each row shows four views of the same generated scene.



Figure 16. Additional qualitative results in Engine A (set 8). From top to bottom: Fruit Store, Meeting Room, Nursery Room, and Dining Room. Each row shows four views of the same generated scene.



Figure 17. Additional qualitative results in Engine B (set 1). From top to bottom: Ancient Ruins, Bamboo Forest, Creek Canyon, European Garden, and Ice Wonderland. Each row shows four views of the same generated scene.

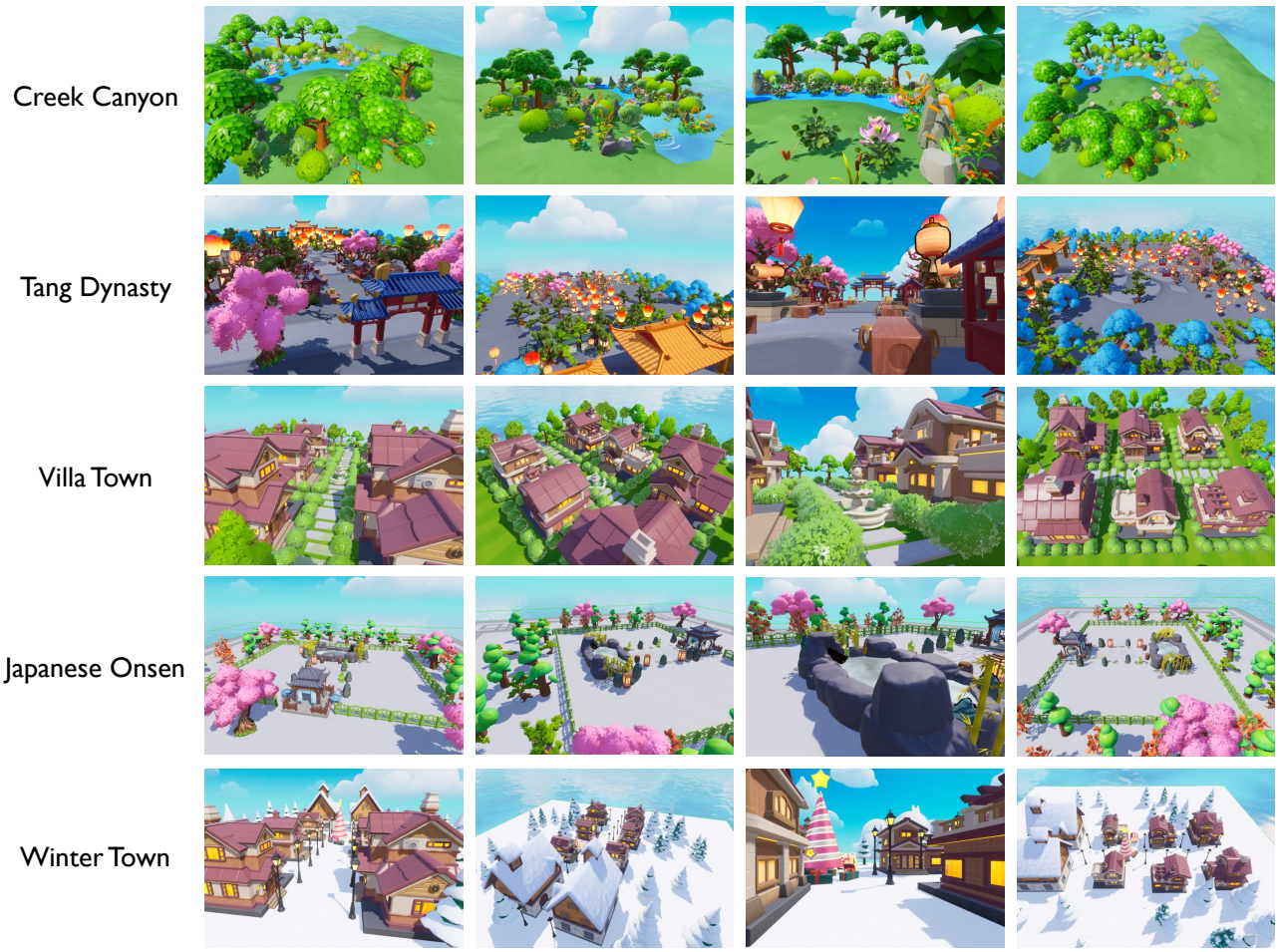


Figure 18. Additional qualitative results in Engine B (set 2). From top to bottom: Creek Canyon, Tang Dynasty, Villa Town, Japanese Onsen, and Winter Town. Each row shows four views of the same generated scene.