SceneOrchestra: Efficient Agentic 3D Scene Synthesis via Full Tool-Call Trajectory Generation

Yun He, Kelin Yu, and Matthias Zwicker*

University of Maryland, College Park, USA

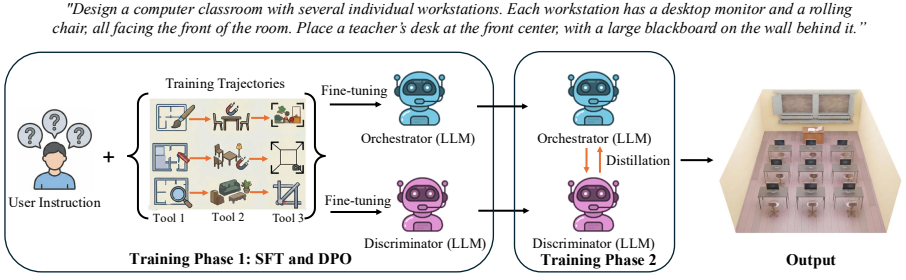

Fig. 1: SceneOrchestra is a trainable orchestration framework that outputs full tool-call trajectories to generate 3D scenes, optimized for efficiency and quality. Building on initial tool-call trajectories generated using an existing method, we train SceneOrchestra in two phases leveraging supervised fine tuning (SFT), direct preference optimization (DPO), and model distillation techniques. At inference time, SceneOrchestra outputs efficient tool call trajectories that generate high quality scenes in one shot.

Abstract. Recent agentic frameworks for 3D scene synthesis have advanced realism and diversity by integrating heterogeneous generation and editing tools. These tools are organized into workflows orchestrated by an off-the-shelf LLM. Current approaches typically adopt an execute–review–reflect loop: at each step, the orchestrator executes a tool, renders intermediate results for review, and then decides on the tool and its parameters for the next step. However, this design has two key limitations. First, next-step tool selection and parameter configuration are driven by heuristic rules, which can lead to suboptimal execution flows, unnecessary tool invocations, degraded output quality, and increased runtime. Second, rendering and reviewing intermediate results after each step introduces additional latency. To address these issues, we propose SceneOrchestra, a trainable orchestration framework that optimizes the tool-call execution flow and eliminates the step-by-step review loop, improving both efficiency and output quality. SceneOrchestra consists of an orchestrator and a discriminator, which we fine-tune with a two-phase training strategy. In the first phase, the orchestrator learns context-aware tool selection and complete tool-call trajectory generation, while the discriminator is trained to assess the quality of full trajectories,

* Corresponding author.

enabling it to select the best trajectory from multiple candidates. In the second phase, we perform interleaved training, where the discriminator adapts to the orchestrator's evolving trajectory distribution and distills its discriminative capability back into the orchestrator. At inference, we only use the orchestrator to generate and execute full tool-call trajectories from instructions, without requiring the discriminator. Extensive experiments show that our method achieves state-of-the-art scene quality compared to previous work.

Keywords: Scene Synthesis · Tool Orchestration · Agentic Framework

1 Introduction

3D scene synthesis plays a crucial role in advancing virtual reality, embodied AI and robotic simulation [37, 46]. As these fields evolve, natural language has emerged as a powerful interface for scene specification, offering intuitive control over complex generation processes [3–6, 10, 13–15, 23, 31, 35, 40, 42]. The ability to efficiently synthesize high-quality 3D scenes from textual descriptions not only democratizes content creation, but also opens new possibilities for personalized virtual experiences and adaptive simulation environments.

Previous 3D scene synthesis approaches can be roughly grouped into three categories: 1) rule-based methods [4, 31] that rely on hand-crafted constraints for layout and object placement, 2) data-driven methods [15, 28, 36, 41] that learn generative priors from curated scene datasets, and 3) LLM/VLM-powered approaches [3,5,6,10,23,35,42] that leverage foundation models for semantic understanding. Each paradigm demonstrates specific strengths: rule-based methods ensure physical validity, data-driven approaches capture realistic scene distributions, and LLM/VLM-based methods offer flexible controllability. However, they remain fundamentally monolithic (only a single approach), unable to simultaneously achieve realism, fine-grained controllability, and physical reliability [40].

This limitation has motivated the development of SceneWeaver [40], an agentic framework that integrates multiple heterogeneous tools within a feedback-driven paradigm. SceneWeaver iteratively invokes different tools to refine the scene and renders intermediate results after each step to guide subsequent tool selection. While this iterative approach produces high-quality outputs compared to earlier techniques, it suffers from two critical issues. 1) Next-step tool selection is guided by heuristic rules. Specifically, an off-the-shelf LLM [1] acts as the orchestrator, selecting the next-step tool and its parameters based on manually summarized use cases. This heuristic-driven strategy often leads to suboptimal execution flows, unnecessary tool invocations, degraded generation quality and increased runtime. 2) The feedback loop requires rendering and reviewing intermediate results after each step, which introduces additional overhead. On average, SceneWeaver requires 1.5 hours per scene on a single A6000 GPU, severely limiting its practical scalability.

To address these limitations, we propose SceneOrchestra, a trainable tool orchestration framework that directly predicts complete tool-call trajectories from

user instructions. This design eliminates heuristic, step-by-step tool selection and avoids the costly execute–review–reflect loop with repeated rendering and review. Specifically, SceneOrchestra comprises two fine-tuned LLMs [39], an orchestrator and a discriminator. And we train the orchestrator and discriminator in two phases, 1) independent training and 2) interleaved training.

For independent training, we first collect tool-call trajectories as training data by running SceneWeaver [40] multiple times per training instruction, producing diverse execution sequences. We train the orchestrator with a curriculum that combines stepwise and full trajectory-level supervision. Stepwise training enables the orchestrator to perform local context-aware tool selection, while trajectory-level training guides the model to generate full trajectories for global planning. In addition, we use supervised fine-tuning (SFT) [27] to learn basic prediction capabilities, and direct preference optimization (DPO) [30] to enable the orchestrator to self-evolve via preference learning. Meanwhile, we train the discriminator to rank trajectories according to generation quality and efficiency.

After the first phase, the trajectory distribution generated by the fine-tuned orchestrator deviates from the original distribution, since DPO pushes it towards higher quality trajectories. Hence, we employ interleaved training to adapt the discriminator to this updated distribution, and then distill its improved ranking ability back into the orchestrator. This procedure alternates between two steps. First, the orchestrator samples multiple trajectories for each user instruction in a training set. We then execute and rank the trajectories to fine-tune the discriminator. Second, the orchestrator samples multiple trajectories for a new set of training instructions, and the updated discriminator ranks them without execution. We then use these rankings to construct preference pairs for trajectory-level DPO fine-tuning of the orchestrator, further improving its trajectory generation. Repeating this cycle progressively distills the discriminator's ranking capability back into the orchestrator. Finally, for inference we only invoke the orchestrator to generate complete tool-call trajectories from user instructions and execute them end to end.

In summary, our contributions are as follows:

- We propose a trainable tool orchestration framework for efficient agentic 3D scene synthesis. It consists of an orchestrator that predicts complete tool-call trajectories from user instructions and a discriminator that selects the best trajectory among multiple candidates.
- We introduce a two-phase training strategy. We first train the orchestrator and discriminator independently to obtain initial trajectory generation and ranking capabilities. We then perform interleaved training to adapt the discriminator to the orchestrator's evolving trajectory distribution and distill its assessment capability back into the orchestrator.
- Extensive experiments show that our approach achieves state-of-the-art generation quality.

2 Related Work

3D Indoor Scene Synthesis. Early works in 3D indoor scene synthesis mainly rely on procedural rules and handcrafted priors. Systems such as ProcTHOR [4] and Infinigen Indoors [31] construct large-scale indoor scenes through rule-based procedural pipelines, which enable scalable scene creation. However, these approaches are highly dependent on manually designed heuristics, which offer limited flexibility for open-ended scene generation. With the availability of large 3D datasets, data-driven models have been proposed to learn scene layouts directly from data. Autoregressive and diffusion-based models, such as ATISS [28], DiffuScene [36], and Text2Room [15], generate object arrangements by modeling spatial relationships between scene elements. Moreover, PhysScene [41] extends this paradigm to more realistic and interactive environments. Despite improved realism, these approaches typically still lack semantic controllability.

Recently, large language models (LLMs) and vision-language models (VLMs) have been explored for structured scene generation. Methods such as LayoutGPT [5], I-Design [3], AnyHome [6], and Holodeck [42] leverage LLMs to generate structured scene layouts from open-world textual instructions. Other works integrate language with visual feedback or optimization, including LayoutVLM [35], ArticScene [10], and SceneThesis [23]. While these methods improve semantic alignment and controllability, generating physically consistent indoor scenes remains challenging. To this end, recent work starts using agentic pipelines to ensure both semantic alignment and physical grounding.

Agentic Scene Synthesis. Scene synthesis relying on a single generative technique often struggles to simultaneously achieve realism, physical validity, and fine-grained controllability [40]. Hence, recent work [17, 40, 44] is building on agentic workflows that integrate multiple tools to improve generation quality and control. Specifically, SceneWeaver [40] introduces an execute–review–reflect loop: at each step, it selects and executes a tool, renders the intermediate result for review, and then selects the next tool based on heuristics to iteratively refine the current output. However, there are two limitations: 1) heuristic-based next-tool selection cannot guarantee an optimal execution flow, which may degrade quality and increase runtime; and 2) rendering and reviewing intermediate results at each step introduces additional latency. To address these issues, we propose a trainable orchestration framework that optimizes the execution flow and eliminates the step-by-step review loop, achieving higher quality in less time.

In addition, SceneSmith [29] and SAGE [38] are concurrent works that also adopt agentic workflows. Compared to SceneWeaver [40], they integrate more powerful tools, but their orchestration still relies on off-the-shelf LLMs. In contrast, we explicitly fine-tune an orchestrator for improved tool calling and global planning, distinguishing our approach from these techniques.

Agentic Reinforcement Learning. Recent work has explored reinforcement learning (RL) techniques for training agentic systems that interact with environments through tool use [12, 18, 22, 43]. Offline RL methods learn from precollected datasets without requiring additional environment interaction [7, 19–

21], enabling safe learning from logged data. In contrast, online RL approaches continuously interact with environments to collect new data [11,32–34], allowing policies to adapt based on real-time feedback. In the context of agentic systems [45], offline training often relies on demonstrations and preference data. Supervised fine-tuning (SFT) [24, 27] initializes the policy by imitating expert demonstrations, while Direct Preference Optimization (DPO) [30] further refines it using pre-collected preference data. More recent online methods such as GRPO [34] perform real-time policy optimization by continuously collecting trajectories and updating policies based on immediate environment feedback. However, such online methods are impractical in our setting, where executing a single rollout takes over one hour, making real-time policy updates infeasible. Therefore, we adopt SFT and DPO in our approach.

3 Method

SceneOrchestra consists of two fine-tuned LLM components based on Qwen3 [39]: an orchestrator and a discriminator. The orchestrator learns to generate complete tool-call trajectories from instructions, while the discriminator learns to select the optimal trajectory from multiple candidates. Our training proceeds in two phases, as shown in Fig. 2. In the first phase, we train both components independently to develop initial trajectory generation and assessment capabilities. In the second phase, we employ interleaved training that adapts the discriminator to the orchestrator's evolving trajectory distribution and distills its assessment capability back into the orchestrator. During inference, the orchestrator directly generates a full tool-call trajectory and executes it end-to-end within SceneWeaver's [40] framework, without requiring the discriminator.

3.1 Preliminaries

We employ two complementary techniques to fine-tune our models in the first phase: supervised fine-tuning (SFT) and direct preference optimization (DPO).

Supervised Fine-Tuning (SFT). SFT [24, 27] adapts a pretrained language model (we use Qwen3 [39]) to a target task by learning from demonstrations. Given training data $\mathcal{D}_{\text{SFT}} = \{(x_i, y_i)\}_{i=1}^{N}$ containing input-output pairs, SFT maximizes the likelihood of generating correct outputs:

$$\mathcal{L}_{\text{SFT}}(\theta) = -\mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{SFT}}} \left[\log \pi_\theta(y|x)\right] \tag{1}$$

where π_θ denotes the model parameterized by θ. This establishes basic task-solving capability and output format alignment.

Direct Preference Optimization (DPO). DPO [30] improves upon SFT by learning from comparative feedback. Given a preference dataset $\mathcal{D}_{\text{DPO}}$ containing triplets (x_i, y_i^w, y_i^l), where output y_i^w is preferred over y_i^l, DPO optimizes:

$$\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}_{(x,y^w,y^l)\sim\mathcal{D}_{\text{DPO}}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(y^w|x)}{\pi_{\text{ref}}(y^w|x)} - \beta \log \frac{\pi_\theta(y^l|x)}{\pi_{\text{ref}}(y^l|x)}\right)\right] \tag{2}$$

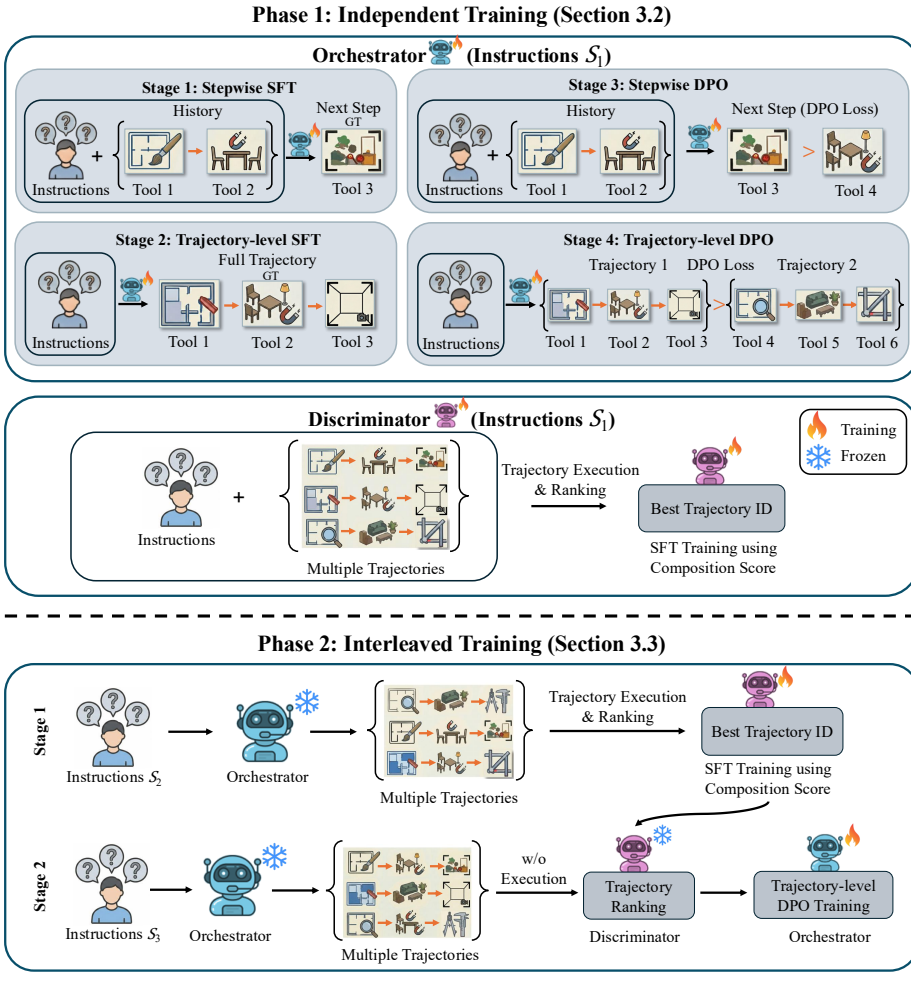

Fig. 2: Training Paradigm. Our training consists of two phases. In the independent training phase, we train the orchestrator and discriminator separately to develop their initial capabilities. The orchestrator undergoes four stages: 1) stepwise SFT, 2) trajectory-level SFT, 3) stepwise DPO, and 4) trajectory-level DPO, learning context-aware tool selection and complete trajectory generation. Meanwhile, the discriminator learns to select the optimal trajectory from multiple candidates. In the interleaved training phase, we adapt the discriminator to the orchestrator's evolving trajectory distribution, then distill this discriminative capability back into the orchestrator. This enables inference with only the orchestrator, eliminating the need for the discriminator.

where π_{ref} is the reference model, β controls the sharpness of the preference signal, and σ is the sigmoid function. This encourages the model to favor higher-quality outputs y_i^w while staying close to the reference model.

3.2 Independent Training

To obtain complete tool-call trajectories for training, we first construct a training instruction set $\mathcal{S}_1$. We then run SceneWeaver [40] multiple times for each instruction to generate diverse tool trajectory rollouts. For step t in each rollout i, we record its quality score $Q^{(i,t)}$ and cumulative runtime $T^{(i,t)}$.

Quality Score. For quality assessment, we adopt the evaluation metrics from SceneWeaver [40], which consist of physical and visual-semantic components. We use GPT-4.1 [25] as the evaluator to compute these scores. Physical metrics include the number of objects in the scene ($N_{\text{obj}}^{(i,t)}$), number of out-of-boundary objects ($N_{\text{ob}}^{(i,t)}$), and number of collided object pairs ($N_{\text{col}}^{(i,t)}$). The physical score is computed as $Q_{\text{phy}}^{(i,t)} = N_{\text{obj}}^{(i,t)} - \alpha(N_{\text{ob}}^{(i,t)} + N_{\text{col}}^{(i,t)})$, where α is a penalty coefficient. Visual metrics evaluate visual realism ($S_{\text{real}}^{(i,t)}$), functionality ($S_{\text{func}}^{(i,t)}$), layout correctness ($S_{\text{lay}}^{(i,t)}$), and scene completeness ($S_{\text{comp}}^{(i,t)}$). The visual score is $Q_{\text{vis}}^{(i,t)} = (S_{\text{real}}^{(i,t)} + S_{\text{func}}^{(i,t)} + S_{\text{lay}}^{(i,t)} + S_{\text{comp}}^{(i,t)})/4$. The overall quality score combines both components:

$$Q^{(i,t)} = \lambda Q_{\text{phy}}^{(i,t)} + Q_{\text{vis}}^{(i,t)} \tag{3}$$

where λ balances physical and visual quality.

Composition Score. We use a composition score to jointly consider generation quality and efficiency:

$$C^{(i,t)} = Q^{(i,t)} - \gamma T^{(i,t)} \tag{4}$$

where γ is a coefficient that balances quality and time cost.

Orchestrator. In this phase, the orchestrator is trained in four stages: 1) stepwise SFT, 2) trajectory-level SFT, 3) stepwise DPO, and 4) trajectory-level DPO. Each stage initializes from the checkpoint of the previous stage, which also serves as the reference model for DPO training. Stepwise training predicts the next tool call given the instruction and the execution history, teaching the model context-aware tool selection. Trajectory-level training, in contrast, requires the model to generate a complete trajectory conditioned only on the instruction. Since language models generate outputs token-by-token, stepwise training naturally provides a foundation for trajectory-level training. In addition, we first use SFT to establish the desired output format and basic prediction ability, and then apply DPO to further improve trajectory generation through preference learning. All SFT and DPO stages consistently employ their respective optimization objectives discussed above (Equations 1 and 2).

In the following, we detail the training data construction process for each stage, with the goal of curating a high-quality data set from the initial tool call trajectories obtained using SceneWeaver [40].

Stepwise SFT. In this stage, we focus on teaching the orchestrator basic context-aware tool selection ability. For each rollout i, we identify steps where the composition score increases significantly. Specifically, consecutive steps where $C^{(i,t)} - C^{(i,t-1)} > \tau_1$, with τ_1 being a threshold. We then construct stepwise SFT training pairs $\{x_s, y_s\}$, where x_s consists of the instruction, available tool list, and execution history, and y_s is the next tool call.

Trajectory-level SFT. In this stage, we aim to teach the orchestrator complete trajectory prediction ability and align its output format for inference. For each rollout i, we randomly truncate the trajectory at step t and evaluate its quality using the composition score $C^{(i,t)}$ at that step. If $C^{(i,t)} > \tau_2$, where τ_2 is a predefined threshold, we consider it a high-quality trajectory and keep it. We then construct trajectory-level SFT training pairs $\{x_t, y_t\}$, where x_t contains the instruction and available tool list, and y_t is the complete trajectory up to step t.

Stepwise DPO. For this stage, we aim to enhance the orchestrator's next tool selection ability by comparing alternative tool choices under the same execution history. For each rollout i, we first identify steps with significant composition score changes—consecutive steps where $|C^{(i,t)} - C^{(i,t-1)}| > \tau_1$. Since the orchestrator has acquired basic prediction capability from the previous two stages, we feed the execution history of these selected steps to the fine-tuned orchestrator to predict alternative tools and their parameters. We then execute the predicted tool and obtain its composition score $C^{(i,t)}_{\text{pred}}$, which we compare with the original tool's score $C^{(i,t)}_{\text{orig}}$. If the score difference exceeds threshold τ_3 ($|C^{(i,t)}_{\text{pred}} - C^{(i,t)}_{\text{orig}}| > \tau_3$), we construct a DPO training triplet (x_s, y^w_s, y^l_s), where x_s contains the instruction, available tool list, and execution history; y^w_s is the chosen tool (with higher score); and y^l_s is the rejected tool (with lower score).

Trajectory-level DPO. To improve the orchestrator's ability to generate complete trajectories, we leverage pairwise comparisons between different execution sequences derived from the same instruction. Specifically, for each instruction, we obtain two rollouts i and j through random sampling, and truncate each at independently chosen steps t_i and t_j. The resulting trajectories are evaluated using their composition scores $C^{(i,t_i)}$ and $C^{(j,t_j)}$. When the score gap is larger than threshold τ_4 ($|C^{(i,t_i)} - C^{(j,t_j)}| > \tau_4$), we form a DPO training triplet (x_t, y^w_t, y^l_t), where x_t specifies the instruction and tool list, y^w_t represents the superior trajectory (higher score), and y^l_t represents the inferior one (lower score).

Discriminator. The orchestrator learns a conditional distribution over high-quality trajectories and samples from it during inference. However, this does not guarantee optimal trajectory selection, as the orchestrator lacks explicit trajectory classification or ranking capability. To address this limitation, inspired by GAN [8], we introduce a discriminator that we train to perform trajectory-level assessment through SFT. Specifically, for each instruction, we sample multiple truncated trajectories from its rollouts, and compute their composition scores as in Eq. 4. The trajectory with the highest score is designated as the best. We

then construct discriminator SFT training pairs $\{x_d, y_d\}$, where x_d contains the instruction, available tool list and a set of candidate trajectories, and y_d is the index of the best trajectory.

3.3 Interleaved Training

After independent training, the orchestrator and discriminator have developed basic trajectory generation and assessment capabilities. However, the trajectory distribution generated by the orchestrator evolved since we employed DPO to improve over the original distribution. Specifically, the fine-tuned orchestrator induces a trajectory distribution that differs from the original SceneWeaver [40]. We address this distribution shift through interleaved training, which adapts the discriminator to the orchestrator's evolving generation distribution, and then distills its updated discriminative capability back into the orchestrator to further improve trajectory quality.

Interleaved training alternates between two stages shown in Fig. 2. At first, the orchestrator samples and executes multiple trajectories for each training instruction, denoted $\mathcal{S}_2$ in Fig. 2. We then select the best trajectory based on composition scores (Eq. 4), construct discriminator training data $\{x_d, y_d\}$ as in the first phase, and fine-tune the discriminator. Second, the orchestrator generates multiple candidates for each new instruction, denoted $\mathcal{S}_3$ in Fig. 2. Then the updated discriminator directly identifies the best trajectory without time-consuming execution. This provides preference signals (x_t, y_t^w, y_t^l) that we use for trajectory-level DPO training of the orchestrator. While these two stages could be repeated in principle, we perform one interleaved training phase in practice.

Finally, at inference time the discriminator is no longer needed and the orchestrator generates complete trajectories and executes them end to end.

4 Experiments

4.1 Experiment Setup

Settings. Following SceneWeaver [40], we use the template-based prompt "Design me a <room_type>" as test instructions. We evaluate on 10 room types: 5 unseen types (bedroom, living room, kitchen, gym, restaurant) that never appear in training, and 5 seen types (bathroom, children room, meeting room, office, waiting room) that appear in training but with different instructions. Specifically, training instructions provide detailed specifications of object counts, positions, layouts, etc. While test instructions use only the generic template. For each instruction, we generate 5 scenes and report the average metrics. In addition, we also use complex instructions to test model's fine-grained control ability.

Baselines. Following SceneWeaver [40], we compare against state-of-the-art LLM-powered 3D scene synthesis methods that support open-vocabulary queries: LayoutGPT [5], Holodeck [42], and I-Design [3]. We also include SceneWeaver itself as a baseline. Note that the difference between our method and SceneWeaver

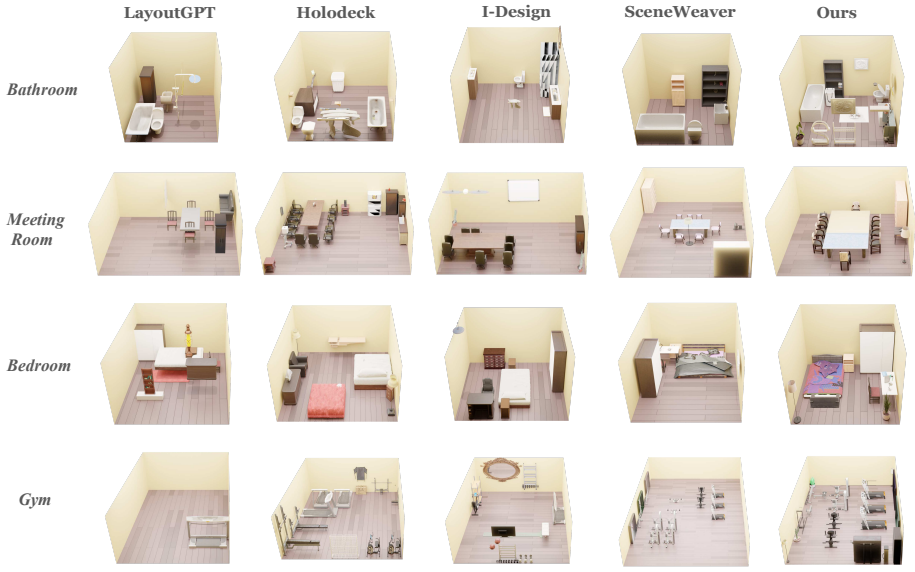

Fig. 3: Qualitative comparison with baselines on both seen (bathroom, meeting room) and unseen (bedroom, gym) room types. Our method achieves higher visual realism, more plausible layouts, and richer details.

lies solely in the tool-call trajectory generation—we use the same tool implementations from SceneWeaver's framework.

Metrics. For a fair comparison, we strictly follow the same evaluation protocol as SceneWeaver [40]. The metrics are organized into two categories: physical metrics and visual-semantic metrics. Physical metrics are directly computed from the generated scene geometry, including the average object count (#Obj), number of out-of-boundary objects (#OB), and collided object pairs (#CN). For visual-semantic metrics, each scene is evaluated by providing GPT-4.1 [25] with the instruction and a top-down rendered view. These metrics assess perceptual quality across four dimensions: visual realism (Real.), functionality (Func.), layout correctness (Lay.), and scene completeness (Comp.).

Implementation Details. We use Qwen3-4B-Instruct-2507 [39] as the base model for both the orchestrator and the discriminator. All SFT and DPO training is conducted using the LLaMAFactory framework [47], and we adopt LoRA [16] for parameter-efficient fine-tuning. We use GPT-4 [1] to generate the training instructions, which are then manually checked and filtered. These instructions provide detailed descriptions of indoor scenes, including object counts, object positions, layouts, and other scene-level constraints. We further apply instruction rephrasing as a form of data augmentation during training.

Table 1: Quantitative comparison with baselines on seen and unseen room types. We use the template-based instruction "Design me a <room_type>". For each room type, all methods are executed five times and the results are averaged. Overall, our method achieves the best performance on nearly all metrics.

Method	Bedroom (Unseen)							Living Room (Unseen)						
	#Obj↑	#OB↓	#CN↓	Real.↑	Func.↑	Lay.↑	Comp.↑	#Obj↑	#OB↓	#CN↓	Real.↑	Func.↑	Lay.↑	Comp.↑
LayoutGPT [5]	6.8	1.2	1.6	7.2	8.4	6.2	4.2	8.4	0.8	2.0	6.4	6.6	5.4	3.8
Holodeck [42]	**28.8**	0.0	34.4	8.0	8.6	6.4	6.0	**32.0**	0.0	10.2	8.8	9.2	7.4	8.0
I-Design [3]	8.8	0.0	0.0	7.8	9.0	7.2	5.2	9.0	0.0	0.0	7.4	8.2	6.6	4.8
SceneWeaver [40]	13.6	0.0	0.0	8.8	9.0	7.4	6.6	16.0	0.0	0.0	8.8	8.6	7.8	7.4
Ours	15.4	**0.0**	**0.0**	**9.0**	**9.2**	**8.0**	**8.2**	18.6	**0.0**	**0.0**	**9.2**	**9.4**	**8.0**	**8.6**

Method	Kitchen (Unseen)							Gym (Unseen)							Restaurant (Unseen)						
	#Obj↑	#OB↓	#CN↓	Real.↑	Func.↑	Lay.↑	Comp.↑	#Obj↑	#OB↓	#CN↓	Real.↑	Func.↑	Lay.↑	Comp.↑	#Obj↑	#OB↓	#CN↓	Real.↑	Func.↑	Lay.↑	Comp.↑
LayoutGPT [5]	8.6	1.6	2.0	5.8	6.4	5.0	3.8	4.4	0.8	0.0	5.4	4.4	5.8	2.4	9.4	1.4	2.2	3.6	3.4	4.8	2.4
Holodeck [42]	19.6	0.0	3.2	6.6	5.8	6.0	4.0	24.2	0.0	4.8	9.2	**9.4**	7.2	6.6	31.0	0.0	7.8	4.8	3.6	4.2	3.4
I-Design [3]	10.4	0.0	0.0	6.2	6.6	5.6	3.4	11.8	0.0	0.8	8.4	8.8	7.2	6.0	27.2	0.0	0.0	4.8	4.6	5.0	3.2
SceneWeaver [40]	32.4	0.0	0.0	9.0	8.8	7.4	7.8	21.2	0.0	0.0	8.8	9.2	7.2	7.6	71.4	0.0	1.4	8.2	7.4	6.8	6.0
Ours	**34.8**	**0.0**	**0.0**	**9.2**	**9.2**	**7.6**	**8.0**	**27.6**	**0.0**	**0.0**	**9.2**	9.2	**8.0**	**7.8**	**78.2**	**0.0**	**0.0**	**8.6**	**8.0**	**7.8**	**7.4**

Method	Bathroom (Seen)							Children Room (Seen)							Meeting Room (Seen)						
	#Obj↑	#OB↓	#CN↓	Real.↑	Func.↑	Lay.↑	Comp.↑	#Obj↑	#OB↓	#CN↓	Real.↑	Func.↑	Lay.↑	Comp.↑	#Obj↑	#OB↓	#CN↓	Real.↑	Func.↑	Lay.↑	Comp.↑
LayoutGPT [5]	7.4	1.2	0.8	8.2	9.0	6.8	5.4	8.6	1.4	1.0	4.8	5.0	5.2	3.0	8.4	1.2	1.6	5.4	5.2	5.0	3.0
Holodeck [42]	12.4	0.0	1.4	8.0	7.4	6.6	5.4	17.8	0.0	3.6	7.8	8.4	6.6	6.0	29.2	0.0	3.4	8.8	9.0	7.4	7.2
I-Design [3]	8.4	0.0	0.0	7.6	8.4	7.0	4.8	8.8	0.0	0.0	6.8	7.8	6.0	4.8	17.2	2.6	0.4	6.6	6.8	5.8	4.4
SceneWeaver [40]	12.0	0.0	0.0	9.0	9.2	7.4	**8.2**	16.0	0.0	1.0	9.2	9.2	7.2	8.4	26.4	0.0	0.0	8.6	8.2	7.0	7.6
Ours	**14.2**	**0.0**	**0.0**	**9.2**	**9.4**	**8.0**	8.0	**19.6**	**0.0**	**0.0**	**9.4**	**9.2**	**7.8**	**8.6**	**31.2**	**0.0**	**0.0**	**9.0**	**9.0**	**8.2**	**8.4**

Method	Office (Seen)							Waiting Room (Seen)							Average						
	#Obj↑	#OB↓	#CN↓	Real.↑	Func.↑	Lay.↑	Comp.↑	#Obj↑	#OB↓	#CN↓	Real.↑	Func.↑	Lay.↑	Comp.↑	#Obj↑	#OB↓	#CN↓	Real.↑	Func.↑	Lay.↑	Comp.↑
LayoutGPT [5]	8.4	0.6	0.6	6.6	7.2	5.6	3.8	7.4	1.0	0.6	6.8	6.8	5.6	3.8	7.8	1.1	1.2	6.0	6.2	5.5	3.6
Holodeck [42]	36.2	0.0	10.8	7.8	7.2	5.6	5.6	30.8	0.0	6.6	8.4	9.0	6.6	6.0	26.2	0.0	8.6	7.8	7.8	6.4	5.8
I-Design [3]	9.8	0.0	0.0	7.2	8.4	6.6	4.6	11.4	0.0	0.0	6.6	7.2	5.8	4.2	12.3	0.3	0.1	6.9	7.6	6.3	4.5
SceneWeaver [40]	35.2	0.0	0.0	8.8	8.0	6.8	7.4	26.2	0.0	0.0	8.8	9.0	7.4	7.6	27.0	0.0	0.2	8.8	8.7	7.2	7.5
Ours	**38.6**	**0.0**	**0.0**	**9.2**	**8.8**	**7.6**	**8.2**	**32.6**	**0.0**	**0.0**	**9.2**	**9.2**	**8.0**	**8.0**	**31.1**	**0.0**	**0.0**	**9.1**	**9.1**	**7.9**	**8.1**

4.2 Experimental Results

We first report quantitative comparisons with baselines in Tab. 1. The evaluated room types fall into two categories: unseen room types that never appear in the training instructions, and seen room types that appear in training, but with different instructions. Overall, our method achieves the best performance on almost all metrics for both seen and unseen room types. Specifically, although Holodeck [42] sometimes generates scenes with more objects than ours, it often suffers from collisions, as its physical and geometric constraints are weak and the placement is closer to random. Our method also significantly outperforms SceneWeaver, despite using the same set of tools. This demonstrates that our fine-tuned orchestrator learns more effective tool selection and complete trajectory generation, thereby improving the overall tool-call execution flow.

We further provide qualitative comparisons in Fig. 3. LayoutGPT [5] produces poor physical layouts, such as tables and chairs intersecting each other, while Holodeck [42] and I-Design [3] often yield unrealistic arrangements. SceneWeaver [40] produces generally plausible results but still exhibits subtle inconsistencies, such as a toilet placed next to a bathtub in a bathroom or a floor lamp positioned awkwardly beside a chair in a meeting room. In contrast, our method generates layouts that are consistently coherent and realistic.

Beyond generation quality, we also report the runtime of different methods in Tab. 2 (Detailed generation times for each room type are provided in the supplementary). LayoutGPT [5], Holodeck [42], and I-Design [3] are relatively fast because they are monolithic approaches. However, this efficiency comes at the cost of lower generation quality. SceneWeaver [40] achieves higher-quality results by integrating multiple tools, but it is much slower. In contrast, our method

Table 2: Average generation time comparison (minutes). We report the average generation time across all room types. All methods run on a single A6000 GPU. Monolithic methods are much faster than agentic approaches, since agentic methods always invoke multiple tools. By optimizing the execution flow and removing the step-by-step review process, our method substantially reduces the runtime of agentic generation.

	LayoutGPT [5]	Holodeck [42]	I-Design [3]	SceneWeaver [40]	Ours
Time (min) ↓	2.4	12.4	6.1	91.0	39.7

Fig. 4: Qualitative comparison on complex instructions. Our method follows instructions more faithfully and produces more realistic layouts. Please zoom in for better visualization.

optimizes the execution flow and removes the step-by-step review process, significantly reducing runtime while further improving generation quality.

Comparison on Complex Instructions. In addition to template-based instructions, we further evaluate our method and SceneWeaver on 70 complex instructions, each involving an unseen room type. These complex instructions specify object counts, positions, spatial relationships, and overall layout, providing a strong test of the model's fine-grained control capability during generation. We report the quantitative results in Tab. 3, where each instruction is executed three times and the results are averaged over all instructions. We observe that even on the more challenging complex instructions, our method consistently outperforms SceneWeaver on both physical and visual metrics. We also provide qualitative comparisons in Fig. 4, showing that our method follows complex instructions more faithfully and produces more plausible, realistic layouts.

4.3 Ablation Studies

Ablation Study on Training Strategy. To demonstrate the effectiveness of our two-phase training strategy, we evaluate the following variants:

Table 3: Quantitative comparison on complex instructions. Our method consistently outperforms SceneWeaver, even on more challenging instructions.

Method	#Obj ↑	#OB ↓	#CN ↓	Real. ↑	Func. ↑	Lay. ↑	Comp. ↑	Time (min) ↓
SceneWeaver [40]	24.3	0.2	0.4	8.2	7.6	6.7	7.3	82.6
Ours	**27.6**	**0.1**	**0.2**	**8.6**	**8.3**	**7.2**	**7.7**	**46.9**

- *Orchestrator without DPO training*: the orchestrator is trained only with stepwise SFT and trajectory-level SFT (first row in Tab. 4).
- *Orchestrator without Stepwise Training*: the orchestrator is trained only with trajectory-level SFT and trajectory-level DPO (second row in Tab. 4).
- *Orchestrator without Discriminator*: we train the orchestrator with stepwise SFT, trajectory-level SFT, stepwise DPO, and trajectory-level DPO, but remove the discriminator (third row in Tab. 4).
- *Independent Training Only*: we perform the full independent training but skip interleaved training. At test time, the orchestrator samples multiple trajectories for each test instruction, and the discriminator selects the best one for execution (fourth row in Tab. 4).

As shown in Tab. 4, DPO training, stepwise training, the discriminator, and interleaved training all contribute positively to the final results. Specifically, DPO training enhances the orchestrator's ability to predict full trajectories through preference learning; stepwise training equips the orchestrator with context-aware tool selection capabilities; the discriminator possesses explicit ranking ability to select the optimal trajectory from multiple candidates; and interleaved training allows the discriminator to adapt to the orchestrator's evolving trajectory distribution while distilling its discrimination capabilities back to the orchestrator. Note that we do not evaluate an "Orchestrator without trajectory-level training" variant (i.e., S-SFT + S-DPO) because, in this setting, the orchestrator can only learn next-step tool selection without acquiring the ability to predict complete trajectories, which is inconsistent with our inference process.

Table 4: Ablation study on training strategy. S-SFT: stepwise SFT; T-SFT: trajectory-level SFT; S-DPO: stepwise DPO; T-DPO: trajectory-level DPO; Disc.: discriminator; Inter.: interleaved training. All results are based on generic template-based prompts. Each component contributes positively to the final results.

Setting	S-SFT	T-SFT	S-DPO	T-DPO	Disc.	Inter.	#Obj ↑	#OB ↓	#CN ↓	Real. ↑	Func. ↑	Lay. ↑	Comp. ↑
w/o DPO	✓	✓					24.9	0.0	0.2	8.0	7.5	7.0	7.0
w/o Stepwise		✓		✓			26.2	0.0	0.1	8.2	8.0	7.3	7.2
w/o Discriminator	✓	✓	✓	✓			27.6	0.0	0.0	8.4	8.4	7.5	7.4
Indep. only	✓	✓	✓	✓	✓		29.7	0.0	0.0	8.8	8.7	7.7	7.8
Full	✓	✓	✓	✓	✓	✓	**31.1**	**0.0**	**0.0**	**9.1**	**9.1**	**7.9**	**8.1**

Ablation Study on Orchestrator. To further validate the necessity of orchestrator fine-tuning and demonstrate the efficiency of our training strategy, we compare against off-the-shelf LLM orchestrators. Specifically, we evaluate GPT-5.5 [26], Gemini-3.1-Pro-Preview [9] and Qwen3-4B-Instruct-2507 [39] in an *in-context learning* setting [2]. By providing these models with extensive instructions paired with high-quality, complete trajectories (sourced from our trajectory-level SFT training data), we enable the models to learn the map-

ping between instructions and high-quality trajectories. We then feed each test instruction and ask the model to predict the full tool-call trajectory.

Table 5: Ablation study on orchestrator. We use off-the-shelf LLMs as orchestrators in an in-context learning setting, but they perform far worse than our fine-tuned orchestrator. All results are based on generic template-based prompts.

Method	#Obj ↑	#OB ↓	#CN ↓	Real. ↑	Func. ↑	Lay. ↑	Comp. ↑	Time (min)↓
GPT-5.5 [26]	24.2	0.0	0.3	8.1	7.7	7.1	7.4	45.3
Gemini-3.1 [9]	20.7	0.0	0.2	8.1	7.6	7.0	7.3	40.9
Qwen3 (w/o Finetune) [39]	18.2	0.3	0.3	7.7	7.1	6.9	7.0	43.6
Ours	**31.1**	**0.0**	**0.0**	**9.1**	**9.1**	**7.9**	**8.1**	**39.7**

The results in Tab. 5 show that, even when provided with a large number of example pairs in the context, these models still struggle to learn how to generate high-quality trajectories. In contrast, our fine-tuned models acquire strong trajectory generation and discrimination capabilities, leading to a substantial improvement in generation quality.

4.4 User Study

To further validate our evaluation metrics with human judgments, we conduct a user study on generated scene quality. We randomly collect 30 generated scenes from our method and the baselines, and ask 16 participants to rate them on a 1–10 scale across four dimensions: realism, functionality, layout, and completeness. The scoring criteria are the same as those used by the judge model (GPT-4.1 [1]). As shown in Tab. 6, our method achieves higher human scores than all baselines across all dimensions. We also conduct an additional pairwise user study to evaluate human preferences in terms of generation quality and diversity. Our user study received an IRB exemption from our university, and all participants provided informed consent. Please refer to the supplementary for details.

Table 6: Human rating results. Human participants rate generated scenes on a 1–10 scale across realism, functionality, layout, and completeness. Our method achieves the highest scores across all dimensions.

Method	Real. (Human) ↑	Func. (Human) ↑	Lay. (Human) ↑	Comp. (Human) ↑
LayoutGPT [5]	3.01	2.97	2.84	3.25
Holodeck [42]	5.57	5.78	5.29	5.88
I-Design [3]	4.73	4.98	4.33	4.64
SceneWeaver [40]	6.72	6.83	6.41	6.57
Ours	**7.60**	**7.99**	**7.46**	**7.76**

5 Conclusion

We propose a trainable orchestration framework for efficient agentic scene synthesis. It consists of an orchestrator and a discriminator. The orchestrator performs context-aware tool selection and predicts complete tool-call trajectories, while the discriminator ranks trajectories and selects the best one from multiple candidates. We further introduce a two-phase training strategy. In the first phase, we train the orchestrator and discriminator separately to align their output formats and build initial generation and discrimination abilities. Specifically, the orchestrator is trained in four stages: stepwise SFT, trajectory-level SFT, stepwise

DPO, and trajectory-level DPO. And the discriminator learns trajectory-level assessment via SFT. In the second phase, we perform interleaved training: the discriminator first adapts to the orchestrator's evolving trajectory distribution and then distills its discrimination capability back to the orchestrator, eliminating the need for the discriminator at inference time. Extensive experiments show that, compared to the state-of-the-art [40], our method produces higher-quality results while significantly reducing execution time.

Acknowledgements. This research is based upon work supported by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via IARPA R&D Contract No. 140D0423C0076. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

References

1. Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al.: Gpt-4 technical report. arXiv preprint arXiv:2303.08774 (2023)
2. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. Advances in neural information processing systems **33**, 1877–1901 (2020)
3. Çelen, A., Han, G., Schindler, K., Van Gool, L., Armeni, I., Obukhov, A., Wang, X.: I-design: Personalized llm interior designer. In: European Conference on Computer Vision. pp. 217–234. Springer (2024)
4. Deitke, M., VanderBilt, E., Herrasti, A., Weihs, L., Ehsani, K., Salvador, J., Han, W., Kolve, E., Kembhavi, A., Mottaghi, R.: Procthor: Large-scale embodied ai using procedural generation. Advances in Neural Information Processing Systems **35**, 5982–5994 (2022)
5. Feng, W., Zhu, W., Fu, T.j., Jampani, V., Akula, A., He, X., Basu, S., Wang, X.E., Wang, W.Y.: Layoutgpt: Compositional visual planning and generation with large language models. Advances in Neural Information Processing Systems **36**, 18225–18250 (2023)
6. Fu, R., Wen, Z., Liu, Z., Sridhar, S.: Anyhome: Open-vocabulary generation of structured and textured 3d homes. In: European Conference on Computer Vision. pp. 52–70. Springer (2024)
7. Fujimoto, S., Meger, D., Precup, D.: Off-policy deep reinforcement learning without exploration. In: International conference on machine learning. pp. 2052–2062. PMLR (2019)
8. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial networks. Communications of the ACM **63**(11), 139–144 (2020)
9. Google DeepMind: Gemini 3.1 pro. `https://deepmind.google/models/gemini/pro/` (2026), accessed: 2026-06-22
10. Gu, Z., Cui, Y., Li, Z., Wei, F., Ge, Y., Gu, J., Liu, M.Y., Davis, A., Ding, Y.: Artiscene: Language-driven artistic 3d scene generation through image intermediary. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 2891–2901 (2025)
11. Hafner, D., Lillicrap, T., Norouzi, M., Ba, J.: Mastering atari with discrete world models. arXiv preprint arXiv:2010.02193 (2020)
12. He, Y., Pittaluga, F., Jiang, Z., Zwicker, M., Chandraker, M., Tasneem, Z.: Langdrivectrl: Natural language controllable driving scene editing with multi-modal agents. arXiv preprint arXiv:2512.17445 (2025)
13. He, Y., Ren, X., Tang, D., Zhang, Y., Xue, X., Fu, Y.: Density-preserving deep point cloud compression. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 2333–2342 (2022)
14. He, Y., Tang, D., Zhang, Y., Xue, X., Fu, Y.: Grad-pu: Arbitrary-scale point cloud upsampling via gradient descent with learned distance functions. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 5354–5363 (2023)
15. Höllein, L., Cao, A., Owens, A., Johnson, J., Nießner, M.: Text2room: Extracting textured 3d meshes from 2d text-to-image models. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 7909–7920 (2023)

16. Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al.: Lora: Low-rank adaptation of large language models. Iclr **1**(2), 3 (2022)
17. Hu, Z., Iscen, A., Jain, A., Kipf, T., Yue, Y., Ross, D.A., Schmid, C., Fathi, A.: Scenecraft: An llm agent for synthesizing 3d scenes as blender code. In: Forty-first International Conference on Machine Learning (2024)
18. Jiang, D., Lu, Y., Li, Z., Lyu, Z., Nie, P., Wang, H., Su, A., Chen, H., Zou, K., Du, C., et al.: Verltool: Towards holistic agentic reinforcement learning with tool use. arXiv preprint arXiv:2509.01055 (2025)
19. Kostrikov, I., Nair, A., Levine, S.: Offline reinforcement learning with implicit q-learning. arXiv preprint arXiv:2110.06169 (2021)
20. Kumar, A., Zhou, A., Tucker, G., Levine, S.: Conservative q-learning for offline reinforcement learning. Advances in neural information processing systems **33**, 1179–1191 (2020)
21. Levine, S., Kumar, A., Tucker, G., Fu, J.: Offline reinforcement learning: Tutorial, review, and perspectives on open problems. arXiv preprint arXiv:2005.01643 (2020)
22. Li, Z., Yu, K., Cheng, S., Xu, D.: League++: Empowering continual robot learning through guided skill acquisition with large language models. In: ICLR 2024 Workshop on Large Language Model (LLM) Agents (2024)
23. Ling, L., Lin, C.H., Lin, T.Y., Ding, Y., Zeng, Y., Sheng, Y., Ge, Y., Liu, M.Y., Bera, A., Li, Z.: Scenethesis: A language and vision agentic framework for 3d scene generation. arXiv preprint arXiv:2505.02836 (2025)
24. Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., Gao, J.: Large language models: A survey. arXiv preprint arXiv:2402.06196 (2024)
25. OpenAI: Introducing gpt-4.1 in the api. `https://openai.com/index/gpt-4-1/` (2025), accessed: 2026-06-22
26. OpenAI: Introducing GPT-5.5. `https://openai.com/index/introducing-gpt-5-5/` (2026), accessed: 2026-06-22
27. Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al.: Training language models to follow instructions with human feedback. Advances in neural information processing systems **35**, 27730–27744 (2022)
28. Paschalidou, D., Kar, A., Shugrina, M., Kreis, K., Geiger, A., Fidler, S.: Atiss: Autoregressive transformers for indoor scene synthesis. Advances in neural information processing systems **34**, 12013–12026 (2021)
29. Pfaff, N., Cohn, T., Zakharov, S., Cory, R., Tedrake, R.: Scenesmith: Agentic generation of simulation-ready indoor scenes. arXiv preprint arXiv:2602.09153 (2026)
30. Rafailov, R., Sharma, A., Mitchell, E., Manning, C.D., Ermon, S., Finn, C.: Direct preference optimization: Your language model is secretly a reward model. Advances in neural information processing systems **36**, 53728–53741 (2023)
31. Raistrick, A., Mei, L., Kayan, K., Yan, D., Zuo, Y., Han, B., Wen, H., Parakh, M., Alexandropoulos, S., Lipson, L., et al.: Infinigen indoors: Photorealistic indoor scenes using procedural generation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 21783–21794 (2024)
32. Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al.: Mastering atari, go, chess and shogi by planning with a learned model. Nature **588**(7839), 604–609 (2020)
33. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
34. Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y., Wu, Y., et al.: Deepseekmath: Pushing the limits of mathematical reasoning in open language models. arXiv preprint arXiv:2402.03300 (2024)

35. Sun, F.Y., Liu, W., Gu, S., Lim, D., Bhat, G., Tombari, F., Li, M., Haber, N., Wu, J.: Layoutvlm: Differentiable optimization of 3d layout via vision-language models. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 29469–29478 (2025)
36. Tang, J., Nie, Y., Markhasin, L., Dai, A., Thies, J., Nießner, M.: Diffuscene: Denoising diffusion models for generative indoor scene synthesis. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 20507–20518 (2024)
37. Wen, B., Xie, H., Chen, Z., Hong, F., Liu, Z.: 3d scene generation: A survey. arXiv preprint arXiv:2505.05474 (2025)
38. Xia, H., Li, X., Li, Z., Ma, Q., Xu, J., Liu, M.Y., Cui, Y., Lin, T.Y., Ma, W.C., Wang, S., et al.: Sage: Scalable agentic 3d scene generation for embodied ai. arXiv preprint arXiv:2602.10116 (2026)
39. Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., et al.: Qwen3 technical report. arXiv preprint arXiv:2505.09388 (2025)
40. Yang, Y., Jia, B., Zhang, S., Huang, S.: Sceneweaver: All-in-one 3d scene synthesis with an extensible and self-reflective agent. arXiv preprint arXiv:2509.20414 (2025)
41. Yang, Y., Jia, B., Zhi, P., Huang, S.: Physcene: Physically interactable 3d scene synthesis for embodied ai. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 16262–16272 (2024)
42. Yang, Y., Sun, F.Y., Weihs, L., VanderBilt, E., Herrasti, A., Han, W., Wu, J., Haber, N., Krishna, R., Liu, L., et al.: Holodeck: Language guided generation of 3d embodied ai environments. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 16227–16237 (2024)
43. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y.: React: Synergizing reasoning and acting in language models. arXiv preprint arXiv:2210.03629 (2022)
44. Yin, S., Ge, J., Wang, Z.Z., Li, X., Black, M.J., Darrell, T., Kanazawa, A., Feng, H.: Vision-as-inverse-graphics agent via interleaved multimodal reasoning. arXiv preprint arXiv:2601.11109 (2026)
45. Zhang, G., Geng, H., Yu, X., Yin, Z., Zhang, Z., Tan, Z., Zhou, H., Li, Z., Xue, X., Li, Y., et al.: The landscape of agentic reinforcement learning for llms: A survey. arXiv preprint arXiv:2509.02547 (2025)
46. Zhang, S.H., Zhang, S.K., Liang, Y., Hall, P.: A survey of 3d indoor scene synthesis. Journal of Computer Science and Technology **34**(3), 594–608 (2019)
47. Zheng, Y., Zhang, R., Zhang, J., Ye, Y., Luo, Z.: Llamafactory: Unified efficient fine-tuning of 100+ language models. In: Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 3: system demonstrations). pp. 400–410 (2024)