Vision Language Agents as Text-to-Scene Generators

Zhicheng Zheng
Princeton University
zz9706@princeton.edu

1 Motivation and Introduction

Looking back at the short history of the modern film and video game industry, 3D modelers take the labor modeling the 3D scene. The truly innovative part of this workflow is to design the high-level scene (e.g. text description), create more expressive models, and adjust the models to fit the scene better. However, 3D modelers might be burdened with the heavy, repetitive, and tedious work of laying out the scene, which is less creative given a fixed text description. What if we hire AI text-to-scene generators to take this burden and leave the innovation to humans? Therefore, automatic scene generation could significantly reduce the hands-on workload devoted to modeling massive 3D scenes in films and games. Furthermore, this technique has the potential to provide massive scene-level synthetic data for video diffusion training. Without precise control, existing video generation models like Sora(Brooks et al., 2024) can struggle with generating accurate video scenes that align with the prompt, especially for highly complicated yet abstract descriptions. Similarly, robot learning also suffers from the lack of various physical scenes for reinforcement learning, which could benefit from this automatic scene generation system. To this end, we plan to propose a new text-to-scene generation pipeline that speeds up scene modeling and empowers model training with infinite supervision data.

Basically, we aim to build a workflow where the users customize what scene to synthesize via elaborate text descriptions. In the pipeline, Large Language Models (LLMs) lay out a scene and determine what objects to import given a text input, while Vision Language Models (VLMs) serve as the critics who provide layout revisions in the loop concerning the object correctness, spatial relationship, and physical compliance. These constitute our text-to-scene system. We also carefully design a comprehensive evaluation suite to assess our system.

2 Related works

Text-to-scene generation using LLMs/VLMs is an emerging field that saw minimal exploration prior to 2024. Here are some recent related works.

SceneCraft(Hu et al., 2024). One pioneering method SceneCraft(Hu et al., 2024) is based on the dual-loop controlled generation. One loop is for scene generating and correctness checking; the other is for detecting and caching frequently used Blender functions. However, the method only uses a fixed set of human-made models, posing the limitation of object diversity. Thanks to the power of text-to-3D diffusion models, we introduce more diverse objects for scene generation. The SceneCraft method also suffers from a limited evaluation suite, where it only evaluates the scene generation system under the CLIP score and a list of human-written score functions. Designing the score functions involves human expertise and labor, hard to generalize as a general text-to-scene system's quality metric. Therefore, here we introduce two new evaluation methods, one adapted from a former benchmark ConceptMix(Wu et al., 2024) to assess the quality of a text-to-scene system in a more general manner, one collecting human evaluators' feedback. Additionally, SceneCraft does not release the source code to public, so we partially reproduce the work in this project.

GALA3D(Zhou et al., 2024). The other work GALA3D(Zhou et al., 2024) generates the layout by LLMs and further refines it by a Diffusion model(Ho et al., 2020). Nonetheless, the overall system is compute-consuming, since it calls several Diffusion models locally to

generate images and 3D objects and further refines the layout with the diffusion prior. In this project, we only use the API of large generative models, so the system and experiments can be directly run on a home PC.

Others. Other works about scene generation include BlenderGPT[1](@gd3kr & @flipphillips, 2023), Infinigen(Raistrick et al., 2023; 2024). BlenderGPT introduces a GPT wrapper into Blender workspace and does not fully automate the scene generation process, while Infinigen is a procedure-based scene generation framework with no generative model participated in it. The most recent works, such as SemanticSDS(Yang et al., 2024), the Scene Language(Zhang et al., 2024), and SceneMotifCoder(Tam et al., 2024), significantly improve scene generation quality and are more or less following the path of GALA3D.

3 Design & Implementation

3.1 Agent System Design

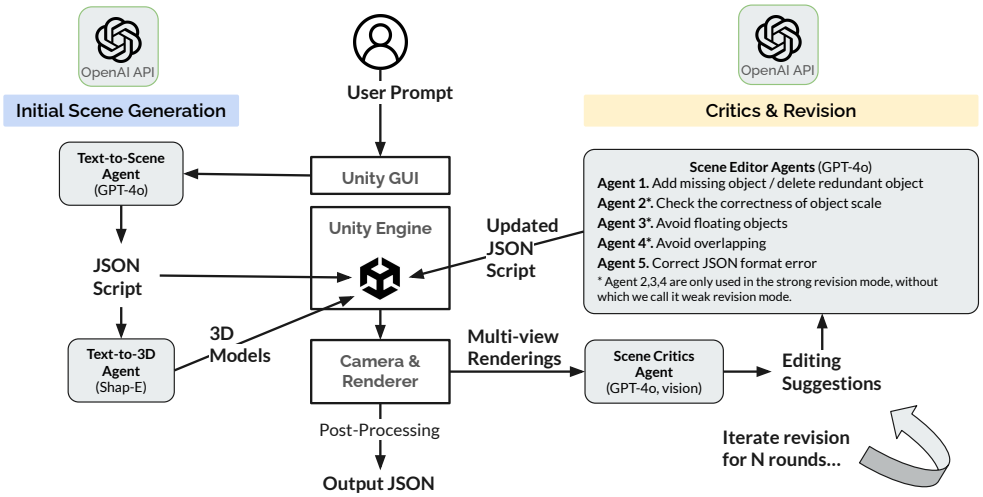

Figure 1: The design of the text-to-scene agent system. Upon the input of the user prompt, Unity requests the LLM agent to generate a JSON file following a certain format. Based on the description of objects in the JSON file, we generate the 3D object models through text-to-3D Diffusion models. Unity Engine further loads the scene from the JSON file in the Unity environment and operates cameras to render multiple views of images. The critic agent receives images along with the user prompt and tries to propose helpful suggestions. A group of editor agents revise the JSON layout file one after another based on these suggestions. The revision can be multiple steps or just two steps and can go through a one-round loop or multiple rounds of loops.

As shown in Fig. 1, our text-to-scene system utilizes multi-agent collaboration via structured files, natural language, and vision as information media. We further

For the initial scene layout generation, to utilize the powerful in-context learning ability(Brown et al., 2020) of Large Language Models (LLMs), we input the user prompt to an LLM and ask it to generate a layout matching the user prompt formatted in a fixed JSON structure.

To further introduce an in-the-loop reflection mechanism to this pipeline, we design a critics and revision iteration framework in this system, which is inspired by the work from Constitutional AI(Bai et al., 2022) and also the previous work SceneCraft(Hu et al., 2024).

[1]https://github.com/gd3kr/BlenderGPT

Ideally, the loop aims to improve the scene generation quality iteratively. To be specific, based on the guidance of the initial layout, we generate the related object models through text-to-3D Diffusion APIs, place all the objects in a room, and render multi-view images of the scene. Then, we ask Vision Language Models (VLMs) to look at the multi-view images, read the user request, and provide criticisms or suggestions for improvement, mostly on aligning the object existence with the user prompt. When the program receives the suggestions, it asks another LLM agent to revise and edit the JSON layout file according to them, including adding, deleting, and modifying objects. The LLM agent can also refer to the user prompt in the revision process. The elaborately engineered prompt also consists of rule-following requirements which will be introduced in the Implementation part.

We examine a nested revision agent illustrated in Fig. 1, which has five sub-agents taking different parts of the revision. The first sub-agent is responsible for matching the object's existence with the user prompt or the suggestions. For example, if the prompt is "the bed with a nightstand", then the scene should only show a bed and a nightstand aside. Any missing objects (e.g., bed), redundant objects (e.g., lamp), or wrong objects (e.g., nightstand $\rightarrow$ table) would be fixed by sub-agent 1. The sub-agent 2's task is to check whether the scales of the objects are correct or not to avoid relatively too large or too small objects regarding the scales of other objects, while sub-agent 3 and 4 work on adjusting the positions of floating objects and overlapping objects. The sub-agent 5 finally focuses on correcting the format errors in the JSON file.

From the experimental observations, we found the LLM and VLM's visual reasoning ability on spatial relation and physical compliance is far from effective. Thus, we incorporate a handcrafted post-processing algorithm to adjust object positions, ensuring better compliance with physical laws (e.g., avoiding overlap or objects floating in midair). The approach of this post-processing is entirely procedural and does not rely on neural networks.

3.2 System Implementation

3.2.1 Base Models

In this project, we choose OpenAI GPT-4o and GPT-4o-mini (OpenAI, 2024) as the base model for the LLM and VLM agents. We conduct experiments relying on both base models respectively. At the time, these models are the strongest multimodal LLMs and have won the leading place on many benchmarks. For the text-to-3D task, we choose to use Shap-E (Jun & Nichol, 2023), which is a popular conditional 3D generative model with a free and easy-to-use user interface. Users can input text or images to customize a 3D model represented in mesh or NeRF format.

3.2.2 Dataset

Training or fine-tuning either a multimodal LLM or a 3D diffusion model is undoubtedly compute-costly. Moreover, high-quality 3D data is largely proprietary and expensive to purchase. Therefore, in this project, we directly use the pre-trained base models as mentioned above without curating a dataset to train them. Instead, we choose to engineer the prompt using handmade few-shot examples for each agent to activate their in-context learning ability, because we believe the base models have already been strong enough to be qualified for this task. As for the evaluation prompt dataset, we developed an LLM evaluator that helps generate a list of random test prompts for evaluation use.

3.2.3 Process Control

As shown in Fig. 2, due to the cross-language and cross-platform nature of this system, we designed an effective and efficient framework to control the processes of message transfer, agent API handling, local file management, as well as text-to-scene workflow.

First, we ask the human user to input the prompt describing the desired scene, for example,

```
prompt = "An armchair beside a cabinet with a small fan on top."
```

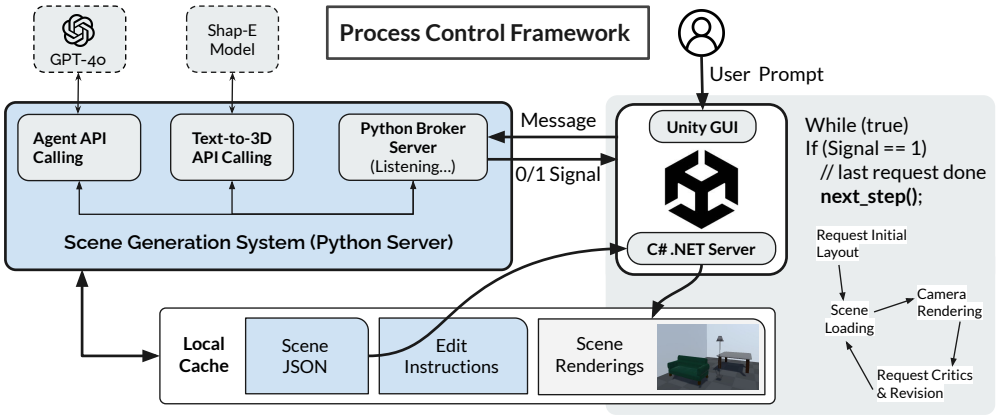

Figure 2: The process control framework of our text-to-3D system. Unity GUI collects user prompts and sends a message requesting the agent for initial scene generation. This communication is connected by a Python broker server using Flask web service, which constantly listens to any messages from the Unity server. Then Unity end starts to listen to the response and the Python end starts to complete the request. Upon task completion, the Python server returns a signal to the Unity server, so that the latter process could move forward. The Unity end goes through the loop of requesting suggestions and JSON revision files, loading scenes, rendering images, and requesting again for N iterations. The intermediate and final JSON files, criticisms, and renderings are all cached in the local file system.

Once the user clicks the start button, the Unity server receives the prompt and sends a generation request to the other process - this is where the Python broker server comes into play, functioning as the bridge between the Python process and the Unity C# process. The broker server uses the Flask Python package, a popular web application toolbox. The Python process listens to the messages from the Unity process and starts to call the text-to-scene agent API. When the initial scene JSON file is generated and saved to the local cache, the Python process then generates 3D individual objects through Shap-E API. After finishing these, the Python process returns a signal to the Unity process. When the Unity process receives the signal, it does the next step - loading the scene into the Unity environment and placing cameras to render multi-view images. After this, the Unity Engine requests the Python process again. Python process calls agents to criticize and edit the scene JSON based on the rendered images and user prompt. After this is completed, the Unity end again loads and renders the updated scene, saving it to the local cache. Depending on the user setting, the loop of requesting critics and revision, loading the scene, and rendering the multi-view images, may iterate for a number of rounds.

3.2.4 Prompt Engineering

Since this is a multi-agent system, prompt engineering plays a vital role in terms of the generation performance. Readers can check out the exact prompt examples which are carefully designed for each agent (or sub-agent) in Appendix 7.2.

Generally, the system prompts consist of three main parts: the introduction of the task, a bullet list of guidelines, and few-shot examples. The user prompts comprise scene-specific intermediate information, e.g., the user prompt, the latest JSON file, scene images, and critics' suggestions. For the guidelines, we generally ask the model to generate a target scene where the objects are reasonably selected, placed, and scaled. We also ask it to generate a JSON file that follows a certain format as below. We also add the few-shot example at the end of the prompt to best trigger the in-context learning ability of current large language models, an ability found in the former work Brown et al. (2020).

```
[JSON format]
        {
            "scene": "<scene name>",
            "scene_description": "<scene description>",
            "objects": [
                {
                    "name": "<object 1>",
                    "description": "<description of it>",
                    "scalar": <the scale relative to the original model>,
                    "bounding_box": {
                        "x_min": <minimal x of its bounding box>,
                        "x_max": <maximal x of its bounding box>,
                        "y_min": <minimal y of its bounding box>,
                        "y_max": <maximal y of its bounding box>,
                        "z_min": <minimal z of its bounding box>,
                        "z_max": <maximal z of its bounding box>,
                        "rotation": <rotation angle>
                    }
                },
                ...
                ... # <more objects>
                ...
            ]
        }
```

4 Evaluation & Results

4.1 Evaluation Metrics

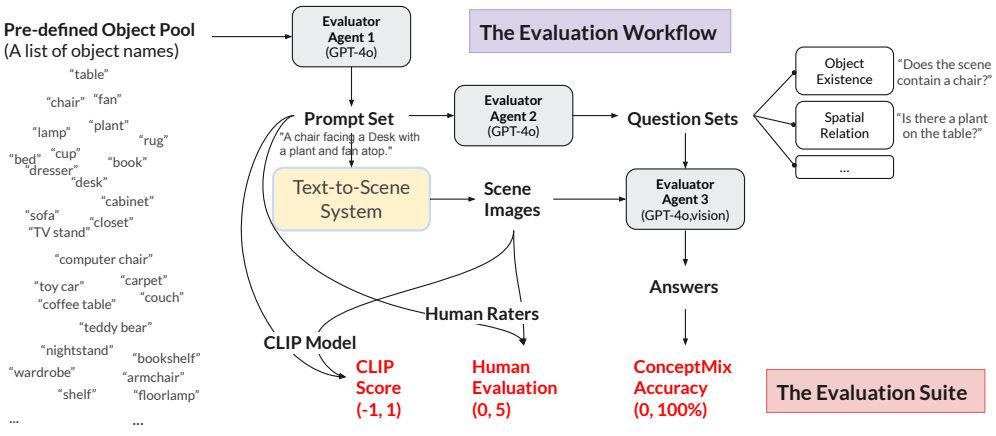

Figure 3: The proposed evaluation metrics and workflow for the text-to-3D system. We mainly adopt three evaluation metrics, i.e., the CLIP score, Human ratings, and ConceptMix accuracy (Wu et al., 2024). The last method is adapted from the ConceptMix benchmark (Wu et al., 2024) which benchmarks the text-to-image system's ability to generate compositional content following the text prompt. These metrics individually stress on different aspects of scene generation quality, thus together forming a comprehensive text-to-scene evaluation system.

In Fig. 3, we show the overview of the evaluation suite for the text-to-scene task. In the experiments, the CLIP score, human ratings, and ConceptMix accuracy (Wu et al., 2024) are

used. We employ these metrics because they assess the scene generation quality through different views. The following is the detailed explanation of evaluation system design.

First, we curate a pre-defined objects pool, which contains the names of about 25 common-place indoor objects, e.g., "table", "chair", etc., as shown on the left side of Fig. 3. Here comes the first evaluation agent, who refers to the object pool and generates a set of prompts (around 20). Iteratively, the prompts are fed to the target text-to-scene system to generate the scene and output rendered images. Using the scene prompt together with images we got, CLIP model and human evaluators can rate the (text, images) pairs from (-1, 1) and (0, 5) respectively.

4.1.1 CLIP Score for General Object Existence

For CLIP model (Radford et al., 2021), it is a cross-modal (vision-language) alignment algorithm. People tend to compute the cosine similarity value between the visual latent vector and the language latent vector, both from the CLIP model, and call it the "CLIP score". The CLIP score indicates that the higher the CLIP score, the more similar the input text and image are. Since there are multiple views, for example, in this project we take 4 multi-view images, so we have 4 CLIP scores in total. Choose the maximal value among these 4 values to return. The reason is that there might be occlusions in some views caused by large objects, so simply selecting the highest CLIP score may automatically solve the trouble of occlusion. In general, the CLIP score focuses on the general existence of objects and general text-to-scene satisfaction.

4.1.2 ConceptMix Accuracy for Certain Object Existence and Spatial Relationship

The ConceptMix (Wu et al., 2024) approach goes through more steps. As in Fig. 3, based on the prompts, the second evaluator agent creates a list of questions (n =5) for each prompt checking the existence of a certain object or the fact of spatial relation presented in the prompt. Next, the third evaluator with vision ability watches the scene and gives answers to these questions, expected to be simply "yes" or "no" after the Chain-of-Thought (CoT) reasoning (Wei et al., 2023). The answers are compared with the ground truth to compute the prediction accuracy. This is a pipeline adapted from the ConceptMix benchmark (mostly the second stage), so we call this "ConceptMix accuracy".

This metric mainly examines the existence of certain objects and their spatial relationship. Here is an example of a generated prompt with questions and ground-truth answers prepared for evaluation.

```
[Example: Evaluation Questions with Ground Truth Answers]
{
    "prompt": "A computer chair facing a Desk with a plant and fan atop.",
    "questions":[
        "Is there a computer chair?",
        "Is the chair facing a desk?",
        "Is there a plant on the desk?",
        "Is there a fan on the desk?",
        "Does the scene contain any other objects except the desk,
            plant, fan, and computer chair?"
    ],
    "ground_truth":[
        "yes",
        "yes",
        "yes",
        "yes",
        "no"
    ]
}
```

4.1.3 Human Evaluation for Physical Compliance

We assembled human participants (n=2) to join the user study. They are asked to follow these grading guidelines, which implies that the human evaluation metric focuses broadly on almost every part that makes a scene. What makes this metric unique is its stress on physical compliance, while the other metrics do not.

```
[Grading Guidelines]

    1. Give 5 points if the scene exactly matches the prompt.
        a. No missing or redundant objects;
        b. correct spatial relationship;
        c. proper scaling;
        d. compliance with physical laws.

    2. Deduct 1 point for:
        a. Each case of object wrong, missing, or redundant
        b. Each case of wrong spatial relationship
            (e.g. not in the corner, not on top of the other object,
            not nearby, etc.)
        c. Each case of object overlapping or flying.
        d. Obvious scale mismatching (e.g. too big or too small objects)
```

4.2 Experiment Setups

In this project, we take a look at how the following variables affect the performance of this system both quantitatively and qualitatively.

a. Base model selection. We choose GPT-4o and GPT-4o-mini as two base models for comparison. The former is a strong model while the latter is relatively weaker.

b. Revision methods. As depicted in Fig. 1, we design a strong revision agent and a weak revision agent. a weak revision agent only adds, deletes, or changes the objects, while the strong revision agent additionally takes the responsibility of rescaling too large or too small objects and adjusting overlapping or floating objects.

c. Revision rounds. As illustrated in Fig. 1, the revision mechanism supports multiple rounds of iteration. We try iteration numbers N=1 and N=2 respectively.

d. Baseline comparison. Since text-to-scene generation is a new research direction, the public code source for recent text-to-scene work, such as SceneCraft and GALA3D, has neither been well released nor well maintained. Therefore, for qualitative analysis, we choose one of the state-of-the-art text-to-image models, DALL·E 3 (Betker et al.), as a baseline for comparison. The evaluation metrics are compatible with images as well. To examine how these evaluation metrics work, we also add a fully blind model for reference, which only generates blank scenes and renders blank images.

For fairness, all the above experiments start from the same prompt set and question set generated by language models.

4.3 Result Analysis

4.3.1 Quantitative Analysis

Table 4.3.1 shows the quantitative results, where we have some interesting findings.

a. The revision mechanism is more beneficial to a weaker base model than to a stronger one. When we use a weaker base model, like GPT-4o-mini, the initial generation quality is relatively lower. In this case, the revision mechanism greatly improves the layout quality, e.g. raising CLIP score from 0.255 to 0.271 and Human Score from 3.5 to 4.8. However, for a stronger base model, like GPT-4o, the initial layout performance can be already satisfying enough, with a ConceptMix Accuracy as high as around 90% and human scores as high as

Metrics	Model	GPT-4o	GPT-4o	GPT-4o	GPT-4o-mini	DALLE3	Blind
	Revision	weak	weak	**strong**	weak		
	Iteration	**2**	1	1	1		
ConceptM	Initial	0.861	0.899	0.899	0.848	0.683	0.538
↑	Revised	0.949	0.924	0.772	0.886		
CLIP Scor	Initial	0.253	0.254	0.26	0.255	0.257	0.137
↑	Revised	0.257	0.262	0.265	0.271		
Human E	Initial	4.4	4.8	4	3.5	3.6	0
↑	Revised	4.9	4.6	3.9	4.8		

Table 1: Quantitative evaluation results.

4.8 out of 5. Therefore, the revision process only contribute moderately to the layout quality improvement.

b. Too much steps of revision within one iteration hurt the layout quality greatly. When we use the "strong" revision mode, which involves up to 5 agents (as shown in Fig. 1) additionally editing the mis-scaled, overlapped and floating objects one after another, the revised version is proved to have lower quality than the initial layout. For example, the ConceptMix value drops from 0.899 to 0.772, compared with the "weak" revision result which raises this value from 0.899 to 0.924. As to the reasons of this phenomena, we guess it is because current language models are not able to handle the physical and mathematical relationship between bounding boxes, such that it just follows the instruction of editing the JSON file recklessly without realizing it messes up the original layout.

c. Under weak revision mode, two times of critics & revisions outperform one-time revision. Two-time revisions uplift the ConceptMix score from 0.861 to 0.949, while the one-time loop raises the score from 0.899 to 0.924. Similarly, we note that the human raters also give higher scores to the second-revision results compared with the first revision. However, the CLIP scores do not differ much for both experiments.

d. Our method works better than the text-to-image model DALL-E 3. The system's ConceptMix accuracy is higher than that of DALL-E 3 by a huge percentage gap, greater than 16%. Although the CLIP scores of the initial layout are mostly the same as DALL-E 3 (around 0.257), the revised version can lift the score up to 0.271. As for the human rating, DALL-E's performance is close to GPT-4o-mini's initial layout performance, not comparable with GPT-4o.

e. The metrics are effective and 3.6 , beatennot dominated by noise. Comparing the text-to-scene system and blind model's performance, we can preliminarily tell that the metrics in this project are effectively implemented and are robust against visual noise.

4.3.2 Qualitative Analysis

As displayed in the comparative figures from Fig. 4 to Fig. 7, the DALL-E 3 model can generate correct objects mentioned in the prompt, but it fails to exclude the not-mentioned objects, leading to many redundant objects presented in the image. In contrast, due to the reflection process and carefully designed prompts, our system can generate more precise object-level layouts.

Nonetheless, a shortcoming can be found in our system that, as shown in Fig. 5, the Teddy bear is floating in the air above the bed, while it seamlessly sits on the bed in DALL-E's result, which is more natural and physically compliant. We argue that it is because we use the bounding box representation for all the generation steps. Therefore, language agents are not able to reason about the bed mattress's height since there is no such information provided. This shortage points out a future direction for improvement.

Figure 4: An example. (Left) Generation from our text-to-scene system. (Right) Generation from DALL-E 3. Prompt: A coffee table with a toy car and a cup on top, next to a sofa.

Figure 5: An example. (Left) Generation from our text-to-scene system. (Right) Generation from DALL-E 3. Prompt: A teddybear sitting on a bed beside a nightstand.

Figure 6: An example. (Left) Generation from our text-to-scene system. (Right) Generation from DALL-E 3. Prompt: A rug under a table with a cup on it and a chair.

Figure 7: An example. (Left) Generation from our text-to-scene system. (Right) Generation from DALL-E 3. Prompt: A computer chair facing a Desk with a plant and fan atop.

5 Discussion

5.1 Strength of the system

In the context of text-to-scene generation,

1. Generally, our system can generate effective scene layouts and render multi-view images from text descriptions effectively at a reasonable speed.

2. The layout from our system precisely matches the text description with correct object selection and spatial relation.

3. The system is inference-time scalable. The more critics & revision iterations, the better scene generation quality it reaches.

4. The system is especially suitable for small and light-weight base models which can greatly benefit from the reflection mechanism. Therefore, the generation process can be accelerated by using light models.

5. We propose a suite of evaluation metrics that comprehensively assesses the text-to-scene performance w.r.t. the object existence, spatial relation, and physical compliance.

5.2 Weakness of the system and future work

The work starts off from the assumption that objects can be represented by bounding boxes. This causes inaccurate spatial relations and floating issues. Therefore, future work can focus on introducing a more generalizable representation considering the objects' diverse geometry.

Additionally, it is universally known that language models lack physical reasoning and spatial reasoning abilities, while this system entirely stems from the response of vision language models. It has to go through manually designed post-processing procedures, which highly restricts its generalization to new domains such as out-door scenes and scenes with deformable objects. In the future, we hope such LLM/VLM abilities could be strengthened.

6 Acknowledgments

Aside from serving as the COS 429 project, this project partially overlaps with my final project in the course "COS 597R (Fall 2024): Deep Dive into Large Language Models". In this course, I submit it as my individual project, while in COS 597R course, I have a partner Cyrus Vachha working on this topic with me. Even though I wrote up this report myself

focusing mostly on my part, it was inevitable to also mention his contribution for soundness. A majority of the work in this report is done by myself. To clarify, in the project for this course, Cyrus Vachha mainly contributes to the message transfer between C# and Python server, pre-generating object models, Unity scene loading (not rendering), and initial layout requests to the LLM.

I would like to sincerely thank Prof. Olga Russakovsky, Prof. Vikram Ramaswamy, and TA Amaya Kelegedara for the feedback on my milestone report and this project. The TAs of COS 597R, Adithya Bhaskar, and Tyler Zhu also gave suggestions and recommended the paper ConceptMix, which finally became a crucial part of our evaluation suite. Special thanks here.

References

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. Constitutional ai: Harmlessness from ai feedback, 2022. URL https://arxiv.org/abs/2212.08073.

James Betker, Gabriel Goh, Li Jing, † TimBrooks, Jianfeng Wang, Linjie Li, † LongOuyang, † JuntangZhuang, † JoyceLee, † YufeiGuo, † WesamManassra, † PrafullaDhariwal, † CaseyChu, † YunxinJiao, and Aditya Ramesh. Improving image generation with better captions. URL https://api.semanticscholar.org/CorpusID:264403242.

Tim Brooks, Bill Peebles, Connor Holmes, Will DePue, Yufei Guo, Li Jing, David Schnurr, Joe Taylor, Troy Luhman, Eric Luhman, Clarence Ng, Ricky Wang, and Aditya Ramesh. Video generation models as world simulators. 2024. URL https://openai.com/research/video-generation-models-as-world-simulators.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Github User @gd3kr and @flipphillips. Blendergpt, 2023. URL https://github.com/gd3kr/BlenderGPT. Accessed: 2024-12-10.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *arXiv preprint arxiv:2006.11239*, 2020.

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 Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 19252–19282. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/hu24g.html.

Heewoo Jun and Alex Nichol. Shap-e: Generating conditional 3d implicit functions, 2023. URL https://arxiv.org/abs/2305.02463.

OpenAI. Gpt-4o system card, 2024. URL https://arxiv.org/abs/2410.21276.

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

Alexander Raistrick, Lahav Lipson, Zeyu Ma, Lingjie Mei, Mingzhe Wang, Yiming Zuo, Karhan Kayan, Hongyu Wen, Beining Han, Yihan Wang, Alejandro Newell, Hei Law, Ankit Goyal, Kaiyu Yang, and Jia Deng. Infinite photorealistic worlds using procedural generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12630–12641, 2023.

Alexander Raistrick, Lingjie Mei, Karhan Kayan, David Yan, Yiming Zuo, Beining Han, Hongyu Wen, Meenal Parakh, Stamatis Alexandropoulos, Lahav Lipson, Zeyu Ma, and Jia Deng. Infinigen indoors: Photorealistic indoor scenes using procedural generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21783–21794, June 2024.

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. 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Xindi Wu, Dingli Yu, Yangsibo Huang, Olga Russakovsky, and Sanjeev Arora. Conceptmix: A compositional image generation benchmark with controllable difficulty. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=MU2s9wwWLo.

Ling Yang, Zixiang Zhang, Junlin Han, Bohan Zeng, Runjia Li, Philip Torr, and Wentao Zhang. Semantic score distillation sampling for compositional text-to-3d generation. *arXiv preprint arXiv:2410.09009*, 2024.

Yunzhi Zhang, Zizhang Li, Matt Zhou, Shangzhe Wu, and Jiajun Wu. The scene language: Representing scenes with programs, words, and embeddings. *arXiv preprint arXiv:2410.16770*, 2024.

Xiaoyu Zhou, Xingjian Ran, Yajiao Xiong, Jinlin He, Zhiwei Lin, Yongtao Wang, Deqing Sun, and Ming-Hsuan Yang. Gala3d: Towards text-to-3d complex scene generation via layout-guided generative gaussian splatting. *arXiv preprint arXiv:2402.07207*, 2024.

7 Appendix

7.1 JSON file Example

Listing 1 is an example JSON file in a format we defined as the scene language in this project. The prompt for this layout is **"An armchair beside a cabinet with a small fan on top."**

Listing 1: An example of the JSON format (Prompt: "An armchair beside a cabinet with a small fan on top.")

```
{
    "scene": "a cozy reading nook",
    "scene_description": "This reading nook is designed for comfort and
        relaxation, featuring warm color palettes with soft textures.
        Ambient lighting enhances the inviting atmosphere, while natural
        light filters through nearby windows, creating gentle contrasts
        on the furnishings. The walls are adorned with subtle patterns,
        shared by an oversized armchair that invites you to sink into its
         deep cushions. A functional yet stylish cabinet stands nearby,
        exhibiting an organized and decluttered aesthetic, and a small
```

```
        fan rests neatly atop it, providing comfort during warmer days.
        The overall layout encourages one to unwind with a good book or
        simply enjoy the tranquility of the space.",
    "objects": [
        {
            "name": "armchair",
            "description": "An oversized armchair upholstered in rich,
                textured fabric that adds to its plush comfort. It
                features wide armrests and a high back, providing
                ergonomic support. The fabric is in a warm tone that
                complements the cozy theme, while the legs are made of
                smooth, polished wood, adding an elegant contrast.",
            "scalar": 1.0,
            "bounding_box": {
                "x_min": 1.0,
                "x_max": 2.83,
                "y_min": 0,
                "y_max": 1.98,
                "z_min": 1.5,
                "z_max": 3.28,
                "rotation": 0
            }
        },
        {
            "name": "cabinet",
            "description": "A sleek cabinet made of dark, rich wood,
                showcasing clean lines and a minimalist design. The
                cabinet offers several compartments for storage and
                features subtle hardware that adds a touch of
                sophistication. Its surface is smooth and spacious,
                perfect for displaying decorative items.",
            "scalar": 1.0,
            "bounding_box": {
                "x_min": 3.5,
                "x_max": 4.6,
                "y_min": 0,
                "y_max": 1.98,
                "z_min": 1.5,
                "z_max": 2.0,
                "rotation": 0
            }
        },
        {
            "name": "fan",
            "description": "A compact table fan designed for portability
                and efficient cooling. It features a durable base with
                adjustable height and a sleek, modern design. Its blades
                are encased in a protective grill, and the fan is
                finished in a neutral color that integrates well into the
                 cabinet setting.",
            "scalar": 1.0,
            "bounding_box": {
                "x_min": 3.76,
                "x_max": 4.3,
                "y_min": 1.58,
                "y_max": 2.5,
                "z_min": 1.65,
                "z_max": 1.89,
                "rotation": 0
            }
        }
    ]
}
```

7.2 Prompt Examples

Listing 2 is the prompt for the initial text-to-scene agent.

Listing 2: prompt for the initial text-to-scene agent

```
############### Prompt for the initial text-to-scene agent ###############

system_prompt = (
    "Act as a 3D scene creation tool helper and I need to create a 3D
        view layout for a scene as a json given a text prompt description
        . When I give a prompt, write a detailed description of how the
        room would look without referencing specific objects but describe
         the overall theme or visual texture and style of the room. Then
        create a list of up to 5 objects in that scene, a description of
        each object in visual and textual detail so if I used a text-to-3
        D object model it would generate it, and generate a room layout
        with coordinate bounding boxes for each object. Make this into a
        JSON format and only return the JSON. "
    "\n\nEnsure these guidelines are followed: "
    "(1) from the perspective of the camera, x axis is rightward, y axis
        is upward, z axis is the further direction to the camera. "
    "(2) We will provide you with a model zoo, from which you can select
        the objects and look up their 3D sizes. Please only select the
        object from the model zoo. Rescale each model uniformly with a
        scalar value applied to x/y/z sizes to make the size of bounding
        box match with the size in the real world. In the JSON, the size
        of bounding box should be scalar times the size in the model zoo
        provided, just like the example below. "
    "(3) Give a rotation direction of the bounding box for which way it
        is facing in degrees where 0 degrees is facing forward and 90
        degrees is facing left. "
    "(4) Make sure the name of the object is from the model zoo we
        provide you as below. Do not change the name of objects (keep
        lowercase and one word). "
    f"(5) Make sure the bounding boxes are in the room. The room itself
        is within the range of [0, {ROOM_SCALE}] x [0, {ROOM_SCALE}] x
        [0, {ROOM_SCALE}]."
    "(6) Try to make sure the layout of the scene is very realistic and
        objects should not be placed in a strange way. "
    "(7) Make sure the layout is aesthetically pleasing. "
    "(8) Do not generate objects that would be underneath other objects
        or on walls such as rugs or windows. "
    "(9) Object bounding boxes should not be scaled inaccurately and
        should not be unrealistically far or close from each other. "
    "(10) Make sure no bounding boxes overlap each other."
    "(11) Make sure the spatial relationship between objects is
        reasonable, like lamp on the table, instead of on the floor,
        nightstand close to one side of the bed, not at a strange
        isolated place. Do not place objects above a sofa, bed, chair,
        couch, and plant."
    "\n\nYour reward is based on the quality of the generated JSON. You
        will be given $20 for each reasonable and natural layout."
    "\n\nThe model zoo: " + f"{object_zoo}"
    "\n\nFor example: you may output your response in the following
        format: \n"
    "{ \"scene\": \"a cozy living room\", \"scene_description\": "
        "\"The living room embodies a warm and welcoming atmosphere,
            focusing on comfort and relaxation. The room showcases a
            harmonious blend of earthy tones and soft textures, with
            wooden accents and plush fabric elements. Natural light
            filters in softly, casting gentle shadows on the flooring,
            enhanced by rich, inviting colors that create a sense of
            tranquility. The layout is open and airy, encouraging
```

```
            movement and interaction while maintaining an intimate feel
            .\", "
    "\"objects\": [ "
    "{ \"name\": \"nightstand\", "
    "\"description\": "
        "\"A stylish nightstand made of warm, dark wood with clean lines
            and a smooth finish. A single drawer provides discreet
            storage, while the top surface is spacious enough to
            accommodate decor items or a lamp. Its design complements the
             cozy theme of the room.\", "
    "\"scalar\": 2, "
    "\"bounding_box\": { \"x_min\": 2.00, \"x_max\": 3.54, \"y_min\": 0,
        \"y_max\": 1.68, \"z_min\": 1.84, \"z_max\": 3.00, \"rotation\":
        0 } "
    "}, "
    "{ \"name\": \"lamp\", "
    "\"description\": "
        "\"A tall, elegant floor lamp with a slim base and a fabric shade
             that diffuses light softly. The base is a brushed brass
            finish, offering a touch of modernity, while the shade is in
            a light linen color, harmonizing beautifully with the
            surrounding decor.\", "
    "\"scalar\": 2, "
    "\"bounding_box\": { \"x_min\": 2.12, \"x_max\": 2.74, \"y_min\":
        1.68, \"y_max\": 3.58, \"z_min\": 2.40, \"z_max\": 2.88, \"
        rotation\": 0 } "
    "}, ...] }"
)

user_prompt = f"{User's Prompt}"
```

Listing 3 is the prompt for the scene criticism agent.

Listing 3: prompt for the scene criticism agent

```
################ Prompt for the scene criticism agent ################

system_prompt = (
    "Act as a scene layout critic. Given the user's prompt and the
        generated photos, which are the multi-view appearance of the
        generated scene. Do you have any suggestions for making the
        layout match the user prompt? Are any objects in the scene
        missing or not in need according to the prompt?"
    "\n\nThe suggestions are expected to be operations including adding
        and deleting objects. "
    "\n\nFor example, "
    "\n- add <X object>"
    "\nand/or"
    "\n- delete <Y object>"
    "\nor"
    "\n- no issues"
    "\n\nNote: "
    "\n(1) Please do not add objects that are not mentioned in the prompt
        . Make sure after revised according to your suggestion, the scene
         has all the objects mentioned in the prompt and do not have
        objects that is not mentioned in the prompt."
    "\n(2) It is very possible that you do not need to give any
        suggestions. If you think the layout is perfect, you can just say
         \"No issues\"."
    "\n(3) Your reward is based on the correctness of your suggestions.
        You will be given $20 for each correct suggestions."
)

user_prompt = [
    {
```

```
        "type": "text",
        "text": f"User's prompt: {User's Prompt}",
    }] + [
    {
        "type": "image_url",
        "image_url": {
            "url":  f"{encode_image(image_path)}",
            "detail": QLT,
        },
    } for image_path in image_paths
],
```

Listing 4 is the prompt for the scene revision sub-agent 1.

Listing 4: prompt for the scene revision sub-agent 1

```
############## Prompt for the scene revision sub-agent 1 ###############

system_prompt = (
    "Act as a indoor scene layout editor. You help to revise the "
    "scene layout based on the suggestions from the critic. The "
    "critic has given suggestions on how to improve the generated "
    "scene to better fit the user's requirements (user's prompt). "
    "The user will provide you with the scene prompt, suggestions"
    ", and the original JSON script which includes the description "
    "and the bounding boxes of all the objects, etc. Your task is to "
    "edit the JSON file based on the suggestions from the critic. "
    "\n\nNotes: "
    "(1) In the 3D world, x axis is rightward, y axis is upward, z axis
        is the further direction to the camera. "
    "(2) If you are suggested to add objects, choose objects from the
        object zoo below. The object name should be exactly the same! You
         can give it a scalar. The size of its bounding box should be
        scalar times the size in the model zoo provided."
    "(3) Do not add or remove objects that are not mentioned in the
        suggestion. "
    "(4) Do NOT move or rescale original objects."
    "(5) Try to avoid collision or overlap of the bounding boxes when you
         add the object. "
    f"(6) The room itself ranges within [0, {ROOM_SCALE}] x [0, {
        ROOM_SCALE}] x [0, {ROOM_SCALE}]. Make sure the bounding boxes
        are in the room."
    "(7) Only output the new JSON, which is the edited version of the
        original JSON. "
    "(8) If there is no issue from critics and you also think the json
        matches the prompt, it is ok to return the JSON with no revision.
        "
    "\n\nSome other minor issues:"
    "(1) Make sure the spatial relationship of the added object is
        reasonable. Do not place objects above a sofa, bed, chair, couch,
         or plant."
    "(2) Try to make sure the layout of the scene is very realistic and
        objects should not be placed in a strange way. Try to make the
        layout aesthetically pleasing. "
    "(3) New object's bounding boxes should not be scaled inaccurately
        and should not be unrealistically far or close from each other. "
    "(4) Do not place new objects that would be underneath other objects
        or on walls such as rugs or windows. "
    "\n\nThe model zoo for your reference: " + f"{object_zoo}"
    "\n\nNote that your reward is based on the correctness of your
        revisions according to prompts and suggestions. You will be given
         $20 for each correct revision."
)

user_prompt = f"User's prompt: {User's Prompt}"
```

```
+ "\n\nCritics' Suggestion: {suggestions}"
+ "\n\nOriginal JSON: {json_data}"
```

Following are the prompts for the scene revision sub-agents 2 to 5.

Listing 5: prompts for the scene revision sub-agents 2 to 5

```
############ Prompts for the scene revision sub-agent 2-5 #############

criteria = []
# use sub-agent 2,3,4 if enabled strong revision
if enable_strong_revision:
    criteria += [(
            f"check and rescale the bounding boxes of objects to match "
            "the mesh size in the model zoo according to the object's "
            "scalar. This means, the size of each bounding box (i.e., "
            "max - min) should be the object's scalar value times its "
            "size in the model zoo. For example, if an object's size "
            "is 1.5 times the mesh size (scalar=1.5), and its y_size "
            "in the model zoo is 0.8, then for this object, (y_max - "
            "y_min) in the original json file should be 1.5 * 0.8 = "
            "1.2. If not match, please rescale the bounding box to "
            "match the size in the model zoo in terms of the scalar. "
            f"For your reference, here is the model zoo: {object_zoo}"
        ),
        (
            "Avoid any object bounding box flying in the air. Objects "
            "might fall on the ground or pile up."
        ),
        (
            "edit the location of objects so that the bounding boxes "
            "NOT overlap with each other. If they do, move bounding "
            "boxes along X and/or Z axis to avoid overlap. No change "
            "on y axis. Note that the object's bounding box should be "
            f"in the room. The room ranges within [0, {ROOM_SCALE}] x "
            f"[0, {ROOM_SCALE}] x [0, {ROOM_SCALE}].\n\nPlease ONLY "
            "move the whole bounding box and keep the size unchanged."
        )
    ]
# use sub-agent 5 under both strong and weak conditions
criteria += [
    (
        "focus only on object names. Make sure the name string "
        "exactly match the object's name found in this list "
        f"{list(object_zoo.keys())}. Otherwise we cannot match "
        "with the file name)"
    )
]
for criterion in criteria:
    contents = [
                (
                     "Act as a indoor scene layout editor. "
                     "You help to correct the errors in a JSON formatted
                         scene layout. "
                     "The user will provide you with the original JSON
                         script which includes the description and the
                         bounding boxes of all the 3D objects. "
                     "Help user to correct the data in the JSON. Only
                         output the new JSON, which is the edited version
                          of the original JSON. "
                     "\n\nPlease only focus on the following task: Your "
                     f"task is to {criterion}. Do NOT edit other parts "
                     "unrelated to the task! "
                ),
                (
```

```
                    "Although you are expected to focus only on the "
                    "above task, it would be good to keep in mind that "
                    "the final scene should match this user's prompt: "
                    f"{user_prompt}."
                ) if enable_strong_revision else "",
                (
                    "\n\nNote that your reward is based on the
                       correctness of your revisions according to the
                       specific task above. You will be given $20 for
                       each correct revision."
                )
    ]
    contents = "".join(contents)

user_prompt = f"Original JSON: {Latest JSON File}",
```

Listing 6 are the prompts for the evaluator agent to generate a list of random test prompts.

Listing 6: prompts for the evaluator agent to generate a list of random test prompts

```
system_prompt = (
    "Act as a scene generation evaluator. Help the user generate a list
       of prompts for a scene layout generation task. In the user prompt
       , you will be given the number of prompts you need to generate
       and the object zoo from which you can select the objects to make
       up you prompts. Please select no more than 5 objects for each
       prompt. And make sure the prompts are clear, concise and natural.
        "
    "\nFor example, if you choose \"table\", \"chair\", \"sofa\" from the
        object zoo, an example prompt might be \"A table with a chair in
        front of it (or behind it) and a sofa beside it.\" Please
       generate the prompts in json format as a list of strings. Please
       ONLY return the json content. "
    "\nExample: [\"A table with a chair in front of it (or behind it) and
        a sofa beside it.\", \"A book and a lamp are on a table.\", ...]
       "
)

user_prompt = (
    f"Generate {num} prompts from the object zoo of " + f"{object_zoo}" +
        "."
)
```

Listing 7 are the prompts for the evaluator agent to generate a list of question sets following the ConceptMix approach.

Listing 7: prompts for the evaluator agent to generate a list of question sets

```
system_prompt = (
    "Act as a scene generation evaluator. Help the user generate a list
       of short questions for a scene layout evaluation task. You will
       be given a list of user prompts, for each prompt, please generate
        no more than 5 questions asking about whether the scene matches
       this text prompt. You can consider to ask the existence of a
       certain object in the scene or the spatial relationship between
       objects. The answers to these questions should be simply \"yes\"
       or \"no\". If the scene does exactly match the prompt, the
       question should be expected to have the \"yes\" answer. "
    "For example, if the prompt is \"a table with a plant on it and a
       chair nearby.\", example questions might be \n\"Does the scene
       contain a chair?\", \"Is there a plant on the table?\", etc.
       Please generate the prompts in the following json format. Please
       ONLY return the json content. "
    "Format example: \n{\"questions\": [[\"<question 1 for prompt 1>\",
       \"<question 2 for prompt 1>\", ...], [\"<question 1 for prompt
       2>\", \"<question 2 for prompt 2>\", ...], ...] } "
```

```
)

user_prompt = f"Scene prompts: {prompts[i:i+split_len]}."
```

Listing 8 are the prompts for the evaluator agent to answer the generated questions looking at the rendered scene images.

Listing 8: prompts for the evaluator agent to answer the generated questions

```
system_prompt = (
    "Act as a scene generation evaluator. Help me answer a list of short
        questions for a scene layout evaluation task. You will be given a
         list of questions and some multiple-view images for the same
        scene, according to the image content, please answer each
        question with \"yes\" or \"no\". Please answer each question in
        the order they are asked. Please pack your response in the
        following json format. Please ONLY return the json content. "
    "Format example: \n{\"reasons\": [\"<reason for question 1>\", \"<
        reason for question 2>\", ...], \"answers\": [\"yes\", \"no\", \"
        yes\", ...]} "
)

user_prompt = [
    {
        "type": "text",
        "text": f"Evaluation questions:\n-" + "\n-".join(questions),
    }] + [
    {
        "type": "image_url",
        "image_url": {
            "url":  f"{encode_image(image_path)}",
            "detail": QLT,
        },
    } for image_path in image_paths
]
```