EditRoom: LLM-parameterized Graph Diffusion for Composable 3D Room Layout Editing

Anonymous ACL submission

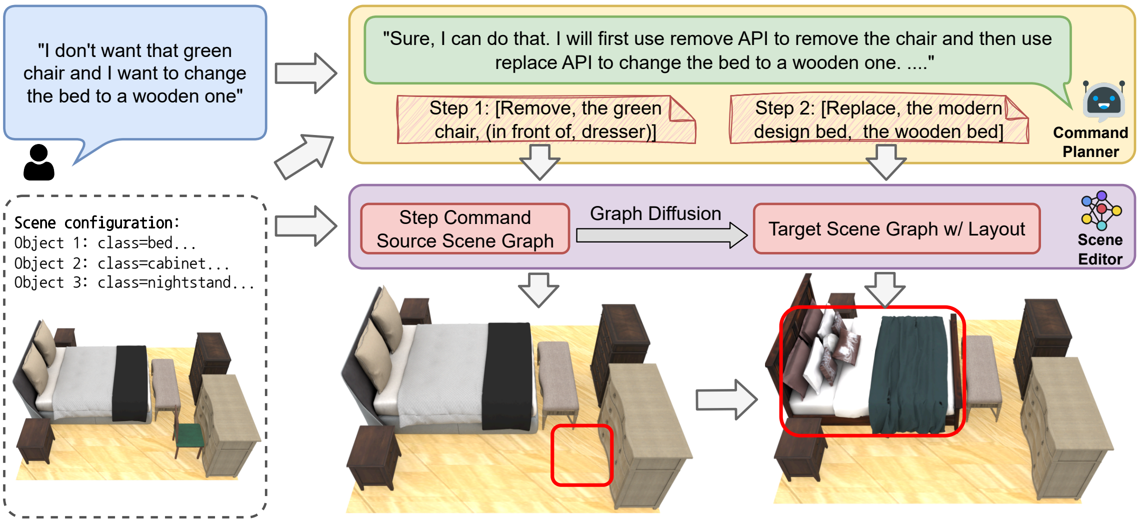

Figure 1: **Editing Pipeline with EditRoom.** EditRoom is a language-guided 3D scene editing method based on LLM planning and graph diffusion. It can accept natural language commands and source scenes, generating coherent and appropriate editing results.

Abstract

Language-guided 3D scene editing has emerged as a pivotal technology in fields such as virtual reality, augmented reality, gaming, architecture, and film production. Traditional methods of 3D scene editing require extensive expertise and time due to the complexity of 3D environments. Recent advancements in language-guided 3D scene editing offer promising solutions, but existing approaches either limit editing to generated scenes or focus on appearance modifications without supporting comprehensive scene layout changes. In this work, we propose **EditRoom**, a novel framework for language-guided 3D room layout editing that addresses these limitations. EditRoom leverages Large Language Models (LLMs) for command planning and a graph diffusion-based method for executing six editing types: rotate, translate, scale, replace, add, and remove. In addition, we introduce **EditRoom-DB**, a large-scale dataset with 83k editing pairs, for training and evaluation purposes. Our approach significantly improves the accuracy and coherence of scene editing, effectively handling complex commands with multiple operations. Experimental results demonstrate EditRoom's superior performance in both single and complex editing scenarios, highlighting its potential for practical applications.

1 Introduction

Language-guided 3D scene editing tasks, particularly in environments such as bedrooms, demand coherent and precise modifications based on verbal instructions. Traditionally, editing 3D scenes necessitates manual intervention via specialized software, requiring extensive expertise and considerable time. Consequently, an automated system capable of interpreting natural language and accurately manipulating these scenes holds substantial value. However, the complexity, diversity, and ambiguity of natural language pose significant challenges, especially when the commands involve comprehensive scene layout adjustments, such as "creating more movable space in my room" or "making my room more modern." These types of commands typically necessitate an understanding of the interplay between the verbal directive and the overall scene configuration, often involving multiple object manipulations. Furthermore, the relatively small size of available 3D scene datasets limits the development of large-scale pretrained models for fully automated, end-to-end language-guided scene editing.

Recently, several works have demonstrated capabilities in language-guided 3D scene editing (Haque et al., 2023; Zhuang et al., 2023; Bartrum et al., 2024; Chen et al., 2023; Ye et al., 2023; Vilesov et al., 2023; Zhou et al., 2024b; Lin and Yadong, 2023; Tang et al., 2023). However, some works (Vilesov et al., 2023; Zhou et al., 2024b) are limited to editing the scenes generated by the model itself, while other methods mainly focus on changing the appearance of a single object (Haque et al., 2023; Zhuang et al., 2023; Bartrum et al., 2024) or requiring manual intervention for any layout adjustments (Chen et al., 2023; Ye et al., 2023; Lin and Yadong, 2023; Tang et al., 2023), like adding a new object or changing the object pose.

To address these challenges, we propose **EditRoom**, which can accept complex natural language commands and coherently modify the 3D room layout for a provided scene. Intuitively, we find that every common natural language command can be converted into the compositions of six basic editing types on single objects: `adding`, `removing`, `replacing`, `translating`, `rotating`, and `scaling`. Therefore, we design a graph diffusion-based method to achieve every basic editing type in a unified framework and use LLM as a planner for high-level command comprehension. In order to provide accurate results on each editing type, we construct an automatic data generation pipeline and collect a synthetic scene editing dataset named **EditRoom-DB**.

EditRoom consists of two main modules: the *command planner* and the *scene editor*. In our *command planner*, we employ an LLM, specifically GPT-4o, to transform natural language commands into sequences of template commands for basic editing operations by providing the source scene information in text format. These template commands, along with the source scenes, are then fed sequentially into the *scene editor* for execution. The *scene editor* is dedicated to constructing single-operation editing results by conditioning on the template commands and input scenes. It encompasses two graph diffusion-based models: the first is designed to generate high-level target scene graphs, which define object shapes and their relative spatial relationships; the second model uses these generated target scene graphs, the source scene, and language commands to estimate the final target scene layout. All object meshes are sourced from a high-quality object dataset and adjusted according to the generated layout.

To enable the *scene editor* to estimate accurate conditional scene distributions for each basic editing type, we have compiled EditRoom-DB, which includes approximately 83,000 editing pairs featuring both template and natural language commands. We designed several pipelines to augment the existing 3D scene dataset, 3D FRONT (Fu et al., 2021a), which contains 16,000 indoor scene designs equipped with high-quality object models. We implement each basic editing operation on these scenes and generate corresponding language commands using predefined templates. Subsequently, we employ GPT-4o to transform these template commands into more natural language forms, serving both as training material for our baselines and as test cases for single-operation evaluations.

In our experimental framework, we assess the performance of EditRoom in scenarios involving both single-operation and complex multi-operation commands. The results indicate that EditRoom not only achieves higher precision in editing specific types of operations and room categories but also demonstrates robust generalization capabilities in handling complex natural language commands that encompass multiple operations, even in zero-shot settings.

Our contributions are summarized as follows:

- We propose a new framework, named EditRoom, consisting of the *command planner* and *scene editor*, which accepts scene inputs and can edit scenes using natural language commands by leveraging LLM for planning.
- We propose a unified graph diffusion-based module that serves as the *scene editor*, capable of executing every basic editing type, including `adding`, `removing`, `replacing`, `translating`, `rotating`, and `scaling`.
- To address the lack of 3D indoor scene editing data, we introduce an automatic data augmentation pipeline to generate edited pairs with corresponding language commands.
- From the experiments, we demonstrate that EditRoom outperforms other baselines across all editing types and room types on single operation commands, and it can generalize to complex operation commands without further training.

2 Related Work

Language-guided 3D Scene Editing Current language-guided scene editing works can be abstractly categorized into four main approaches. The first approach involves pretrained image editing models to edit the appearance of objects inside the scene (Haque et al., 2023; Zhuang et al., 2023; Bartrum et al., 2024; Karim et al., 2023). The second approach leverages neural field representation, like 3D Gaussian Splatting (Kerbl et al., 2023), to obtain individual object representation and apply layout change by manually selecting targets (Chen et al., 2023; Ye et al., 2023). The third approach is to learn conditional scene generation from scene description and manually mask the target attributions for editing (Haque et al., 2023; Tang et al., 2023). The fourth approach starts with generating new scenes and limits to editing these generated scenes (Vilesov et al., 2023; Zhou et al., 2024b). In contrast to these previous works, EditRoom can accept an existing scene as input and apply free-form editing commands for 3D scene layout without manual interventions.

LLM for 3D Scene Understanding Recent works demonstrate that existing LLMs can facilitate 3D spatial reasoning. These works usually leverage the pretrained caption models to convert 3D scenes into text descriptions and ask the LLM to generate navigation steps (Zhou et al., 2023, 2024a), provide room layout (Feng et al., 2024), or ground 3D objects (Yang et al., 2023; Hong et al., 2023; Huang et al., 2023). In our work, we leverage LLM (GPT-4o) to take source scenes in text format and break the natural language commands into basic editing operations.

3 The EditRoom Method

In this section, we introduce EditRoom, our proposed framework for language-guided 3D room layout editing, comprising two primary modules: the *Command Planner* and the *Scene Editor*. We denote $D := \{(S_1, T_1, C_1), \ldots, (S_N, T_N, C_N)\}$ as a collection of N editing pairs of indoor scenes, where S_i is the source scene, T_i is the target scene, C_i is the corresponding language command for the i-th pair, and N is the total number of editing pairs.

Given a natural language command C_i and source scene S_i, we aim to estimate the conditional target scene distribution $q(T_i|S_i, C_i)$. Our *command planner* takes the source scene S_i and natural command C_i to generate the template commands L_i. Then, the *scene editor* conditions on template commands L_i to obtain the final target scene T_i, where the whole pipeline can be written as $q(T_i|S_i, C_i) = q(L_i|S_i, C_i) \times q(T_i|S_i, L_i)$, shown in Figure 1.

3.1 LLM as Command Planner

In order to process open natural language commands, we use GPT-4o to convert natural language command C_i into a set of combinations of basic editing types with template commands $L_i := \{l^i_j\}_{j=1}^{N_L}$, where N_L is the number of template commands, shown in Figure 1. To cover the general manipulations on the scene, we design six basic editing operations:

- Rotate an object: [Rotate, Target Object Description, Angle]
- Translate an object: [Translate, Target Object Description, Direction, Distance]
- Scale an object: [Scale, Target Object Description, Scale Factor]
- Replace an object: [Replace, Source Object Description, Target Object Description]
- Add an object: [Add, Target Object Description, Target Object Location]
- Remove an object: [Remove, Target Object Description]

We instruct the LLM to use another unique object as a reference to describe the spatial relation if the target object is not unique. During the inference phase, we prompt the LLM with attributes of objects within the source scene along with the natural language command, tasking the model to analyze the scene and delineate basic editing operations through template commands in specified formats. The attributes include categories, locations, sizes, rotations, and object captions. Detailed descriptions of the full prompt and examples are provided in Figure 5 of the appendix.

3.2 Graph Diffusion as 3D Scene Editor

Given the template command l and source scene S, our objective is to determine the conditional target scene distribution $q(T|S, l)$. Drawing inspiration from recent advancements in language-guided 3D scene synthesis (Lin and Yadong, 2023), we transform scenes into semantic graphs and employ a graph transformer-based conditional diffusion model to learn the conditional target scene graph distribution, as depicted in Figure 2. Our approach

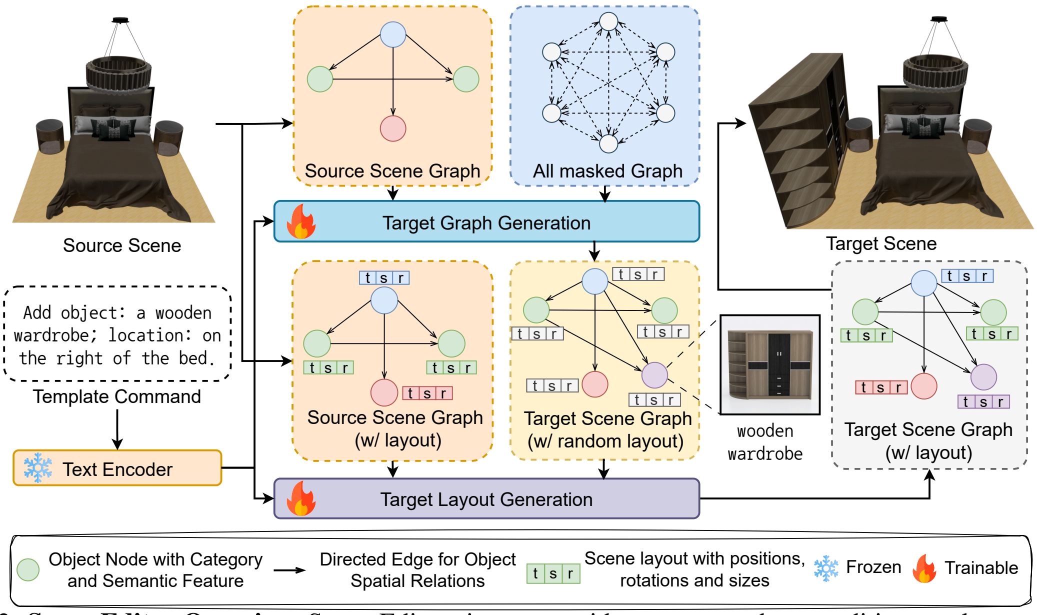

Figure 2: **Scene Editor Overview.** Scene Editor aims to provide accurate, coherent editing results according to the given source scene and language commands. It consists of two graph transformer-based conditional diffusion models. One diffusion model generates semantic target scene graphs. Another diffusion model can estimate accurate poses and size information for each object inside the generated target scene graphs. All diffusion processes are conditioned on Source Scene and Template Command.

involves two key graph transformer-based diffusion models: the *Target Graph Diffusion*, which generates object shapes and their spatial relations as graphs, and the *Target Layout Diffusion*, which computes the final layout of the target scene. To reduce the alignment challenges between the 3D scene distribution and language, all commands are encoded using the text encoder of CLIP-ViT-B-32.

Scene Graph Representation Each scene is represented as a combination of a layout B and a scene graph G (Lin and Yadong, 2023). The layout B encapsulates the position, size, and orientation of each object, while the scene graph G encodes additional high-level semantic information. Formally, a semantic scene graph $G := (V, E)$ comprises nodes $v_i \in V$, where each v_i corresponds to an object o_i with high-level attributes. Directed edges $e_{ij} \in E$ represent spatial relationships such as "left of", connecting the i-th object to the j-th object. Each node v_i is characterized by a discrete category c_i and continuous semantic features f_i, derived from a pretrained multimodal-aligned point cloud encoder, OpenShape (Liu et al., 2024c), which features a 1280-dimensional representation space.

Target Graph Diffusion In this stage, we aim to learn semantic scene graphs G_{tg} for target scenes by giving source scenes G_s and language commands l through a discrete diffusion model ε_g, where G_{tg} includes category C_{tg} and semantic features F_{tg} for each node and the edges E_{tg} for object relative relations. Since high-dimensional object semantic features ($d = 1280$) are too complicated to learn from limited data, we use a VQ-VAE model (Lin and Yadong, 2023; Wang et al., 2019) to compress them into low-dimensional features $z \in \mathbb{R}^{n_f \times d_Z}$, which consists of n_f vectors extracted from a learned codebook $Z \in \mathbb{R}^{K_f \times d_Z}$ by a sequence of feature indices $f_{idx} := \{1, ..., K_f\}^{n_f}$, where K_f and d_Z are the size and dimension of codebook. Then, we use the feature indices to replace the original object semantic features as targets for training, denoted as $\hat{F}$. Therefore, $G_{tg} = (C_{tg}, \hat{F}_{tg}, E_{tg})$ and $G_s = (C_s, \hat{F}_s, E_s)$, and our goal is to learn the conditional distribution $q(G_{tg}|G_s, l)$. During the training process, at timestep t, the noises are added to the G_{tg} to get G^t_{tg}, and the model ε_g aims to reconstruct G^0_{tg} by-conditioning on G_s and l. To add the conditions, we concatenate each element of source graphs into noisy target graphs as context and use cross-attention layers to incorporate language features. The loss function can be written as:

$$\begin{aligned} L_g :=& \mathbb{E}_{q(G^0_{tg})}[\sum_{t=2}^{T} L_{t-1} - \\ & \mathbb{E}_{q(G^1_{tg}|G^0_{tg})}[log p_{\varepsilon_g}(G^0_{tg}|G^1_{tg}, G_s, l)]] \end{aligned} \quad (1)$$

$$\begin{aligned} L_{t-1} :=& D_{KL}[q(G^{t-1}_{tg}|G^t_{tg}, G^0_{tg})|| \\ & p_{\varepsilon_g}(G^{t-1}_{tg}|G^t_{tg}, G_s, l)] \end{aligned} \quad (2)$$

where D_{KL} indicates the KL divergence.

Target Layout Diffusion In this stage, we aim to estimate the target scene layout B_{tg} using a diffusion model ε_b, conditioning on target scene graph G_{tg}, source scene graph G_s, source layout B_s, and language command l. The target scene layout $B_{tg} \in \mathbb{R}^{M\times 8}$ consists of position $T_{tg} \in \mathbb{R}^{M\times 3}$, size $S_{tg} \in \mathbb{R}^{M\times 3}$, and rotation $R \in \mathbb{R}^{M\times 2}$. During the training process, gaussian noises ϵ will be added to the target layout, and the layouts are encoded into the node features by MLP layers. Similar to the *Target Graph Diffusion*, we concatenate the source scene graph and source layout to the target scene graph and corrupted target layout as context. The language features are incorporated through cross-attention layers. The objective target is to estimate the added noises at each time step. The loss function can be written as:

$$L_b := \mathbb{E}_{B_{tg}^0, t, \epsilon}[||\epsilon - \varepsilon_b(B_{tg}^t, t, G_{tg}, G_s, B_s, l)] \quad (3)$$

Inference Process During the inference phase, the first step consists of transforming the source scene into a scene graph G_s and a corresponding layout B_s. Subsequently, the *Target Graph Generation* model predicts the target scene graph G_{tg}, conditioned on the source scene graph G_s and the language command l. This is followed by the *Target Layout Generation* model, which computes the target layout B_{tg}, leveraging all available variables as inputs. The final step in constructing the target scene, denoted as $T := (G_{tg}, B_{tg})$, involves retrieving the object meshes based on the estimated object features and arranging them according to the generated layout. This systematic approach enables the dynamic generation of scenes that are aligned with verbal instructions, ensuring that the resulting scenes accurately represent the specified conditions.

4 The EditRoom-DB Dataset

To support various basic editing operations, we introduce an automated data augmentation pipeline that generates editing pairs, subsequently forming the EditRoom-DB dataset. We utilize the bedroom, dining room, and living room scenes from the 3D-FRONT dataset (Fu et al., 2021a) as our initial scene sets, and the 3D-FUTURE dataset (Fu et al., 2021b) as the source for high-quality objects. The generation process accepts these 3D scenes and applies object-level modifications to simulate the 3D scene editing workflow. These modifications include `Add and Remove Objects`, `Pose and Size Changes`, and `Object Replacement`. The modified scenes are returned with a detailed template text describing the changes made.

Types	Train			Test		
	Bedroom	Diningroom	Livingroom	Bedroom	Diningroom	Livingroom
Translate	8.6k	3.2k	2.7k	61	58	74
Rotate	4.0k	1.3k	1.3k	38	35	27
Scale	12.7k	4.5k	3.9k	146	144	162
Add	8.9k	3.4k	2.8k	75	79	57
Remove	8.8k	3.3k	2.8k	129	142	127
Replace	6.8k	2.2k	2.1k	51	42	53
Total	49.8k	17.9k	15.6k	500	500	500

Table 1: **EditRoom-DB dataset statistics.** We collect around 83k training data across all room types and 500 test data for each room type.

Template commands, constructed with editing and target object descriptions, are captioned by the pretrained multimodal understanding model, LLAVA-1.6 (Liu et al., 2024b,a, 2023), using front view images of the objects. These template commands are then translated into natural language commands using GPT-4o for testing single operations and training baseline models. Additional prompts and examples are detailed in Figure 6 of the appendix.

For each scene in our initial sets, objects are randomly selected for iterative modification using basic editing operations. For the `Add and Remove Objects` pairs, the scene lacking the selected object serves as the target for removal, and the original scene serves as the source for addition. In `Pose and Size Changes`, random values are applied to the attributes of the selected objects, with collision checking ensuring the creation of collision-free editing pairs. During `Object Replacement`, objects within the same category are randomly chosen, with collision checking helping to avoid low-quality data samples. The dataset comprises 83k training samples across all room types and 500 test samples for each type. Detailed statistics are available in Table 1 and further details in Appendix C.

5 Experiments

5.1 Baselines and Evaluation Metrics

Baselines Since there is no previous work that accepts natural language commands for various editing types, we construct two baseline for comparisons: DiffuScene-E and SceneEditor-N:

- DiffuScen-N: DiffuScene-N is modified from the language-guided 3D scene synthesis work, DiffuScene (Tang et al., 2023), which includes a UNet-based diffusion model to generate scene layout. To enable it with language-

	Bedroom				Diningroom				Livingroom			
Model	IOU (↑)	S-IOU (↑)	LPIPS (↓)	CLIP (↑)	IOU (↑)	S-IOU (↑)	LPIPS (↓)	CLIP (↑)	IOU (↑)	S-IOU (↑)	LPIPS (↓)	CLIP (↑)
DiffuScene-N	0.6213	0.6122	0.1374	0.9550	0.4484	0.4338	0.1984	0.9247	0.4693	0.4507	0.1748	0.9328
SceneEditor-N	0.7254	0.7150	0.1071	0.9601	0.5189	0.5033	0.1572	0.9356	0.4797	0.4667	0.1638	0.9385
EditRoom	**0.7435**	**0.7344**	**0.0967**	**0.9644**	**0.5246**	**0.5095**	**0.1489**	**0.9450**	**0.4801**	**0.4724**	**0.1564**	**0.9463**

Table 2: **Performance on single operation with different room types.** From the table, we can find EditRoom outperforms baselines among all room types, which indicates that our methods can provide more accurate and coherent editing across room types.

	Translate				Rotate				Scale			
Model	IOU (↑)	S-IOU (↑)	LPIPS (↓)	CLIP (↑)	IOU (↑)	S-IOU (↑)	LPIPS (↓)	CLIP (↑)	IOU (↑)	S-IOU (↑)	LPIPS (↓)	CLIP (↑)
DiffuScene-N	0.5237	0.5115	0.1691	0.9488	0.5902	0.5770	0.1372	0.9510	0.5816	0.5691	0.1248	0.9510
SceneEditor-N	0.5611	0.5491	0.1488	0.9511	0.6269	0.6146	0.1313	0.9526	0.6191	0.6083	0.1150	0.9573
EditRoom	**0.5782**	**0.5673**	**0.1432**	**0.9553**	**0.6277**	**0.6158**	**0.1290**	**0.9538**	**0.6309**	**0.6216**	**0.1083**	**0.9610**
	Replace				Add				Remove			
	IOU (↑)	S-IOU (↑)	LPIPS (↓)	CLIP (↑)	IOU (↑)	S-IOU (↑)	LPIPS (↓)	CLIP (↑)	IOU (↑)	S-IOU (↑)	LPIPS (↓)	CLIP (↑)
DiffuScene-N	0.5662	0.5435	0.1439	0.9336	0.5075	0.4946	0.1826	0.9381	0.4291	0.4144	0.2060	0.9325
SceneEditor-N	0.6002	0.5746	0.1398	0.9390	0.5595	0.5472	0.1619	0.9341	0.5545	0.5412	0.1442	0.9496
EditRoom	**0.6114**	**0.5837**	**0.1369**	**0.9427**	**0.5657**	**0.5542**	**0.1571**	**0.9430**	**0.5556**	**0.5453**	**0.1363**	**0.9517**

Table 3: **Performance on single operation with different editing types.** From the table, we can notice EditRoom can provide better editing results across all basic editing types.

guided scene editing ability, we leverage their scene completion pipeline by incorporating the source scene as context for the diffusion process. During the training and testing, the model directly conditions natural commands for target scene layout generation.

- SceneEditor-N: To test our generalization ability, we experiment with another setting, where the *scene editor* directly trains on the natural commands got from the GPT-4o. During the inference time, the model conditions the natural commands and generates the final scenes.

Metrics To evaluate the models' performance, we utilize four metrics: IOU, S-IOU, LPIPS (Zhang et al., 2018), and CLIP (Radford et al., 2021) scores. The IOU scores are calculated by determining the 3D Intersection Over Union (IOU) between each object in the generated and target scenes, selecting pairs with the highest 3D IOU values. The S-IOU represents the semantic-weighted 3D IOU, where semantic similarities between matching objects are calculated using Sentence BERT (S-BERT) (Reimers and Gurevych, 2019) based on their captions. For visual evaluation, we render both the generated and target scenes from 24 fixed camera views. Visual similarity is assessed using the LPIPS metric for pixel similarity, and semantic similarity is evaluated using the CLIP image encoder (CLIP-ViT-B32).

5.2 Results

Single Operation To assess model performance on single operations, we test our model and baselines using the EditRoom-DB test set, which contains 500 samples per room type, with language commands generated by GPT-4o. Quantitative results are depicted in Tables 2 and 3, and qualitative outcomes are illustrated in Figure 3. Table 2 indicates that EditRoom consistently outperforms other baselines across all room types, with notably superior performance in bedrooms. According to Table 3, EditRoom also excels across all editing types. Comparisons between EditRoom and SceneEditor-N reveal that template-based instructions can simplify the learning process by more effectively aligning language commands with scene changes. Moreover, the LLM (GPT-4o) demonstrates a successful bridge between natural language and template commands. SceneEditor-N outperforms DiffScene-E across all metrics and editing types, suggesting that our graph-based diffusion method yields more coherent and accurate editing results compared to the UNet-based approach. Thus, EditRoom provides more precise and coherent atomic editing operations from natural language commands than its counterparts.

Analysis across different room types shows that all models perform better as the average number of objects in rooms decreases, highlighting potential for improvements in larger, more complex scenes. Evaluating different editing operations reveals that translating, adding, and removing operations score lower on IOU, demanding stronger spatial reasoning. Meanwhile, replacing and adding operations yield lower CLIP scores, indicating a need for better alignment between object descriptions and their

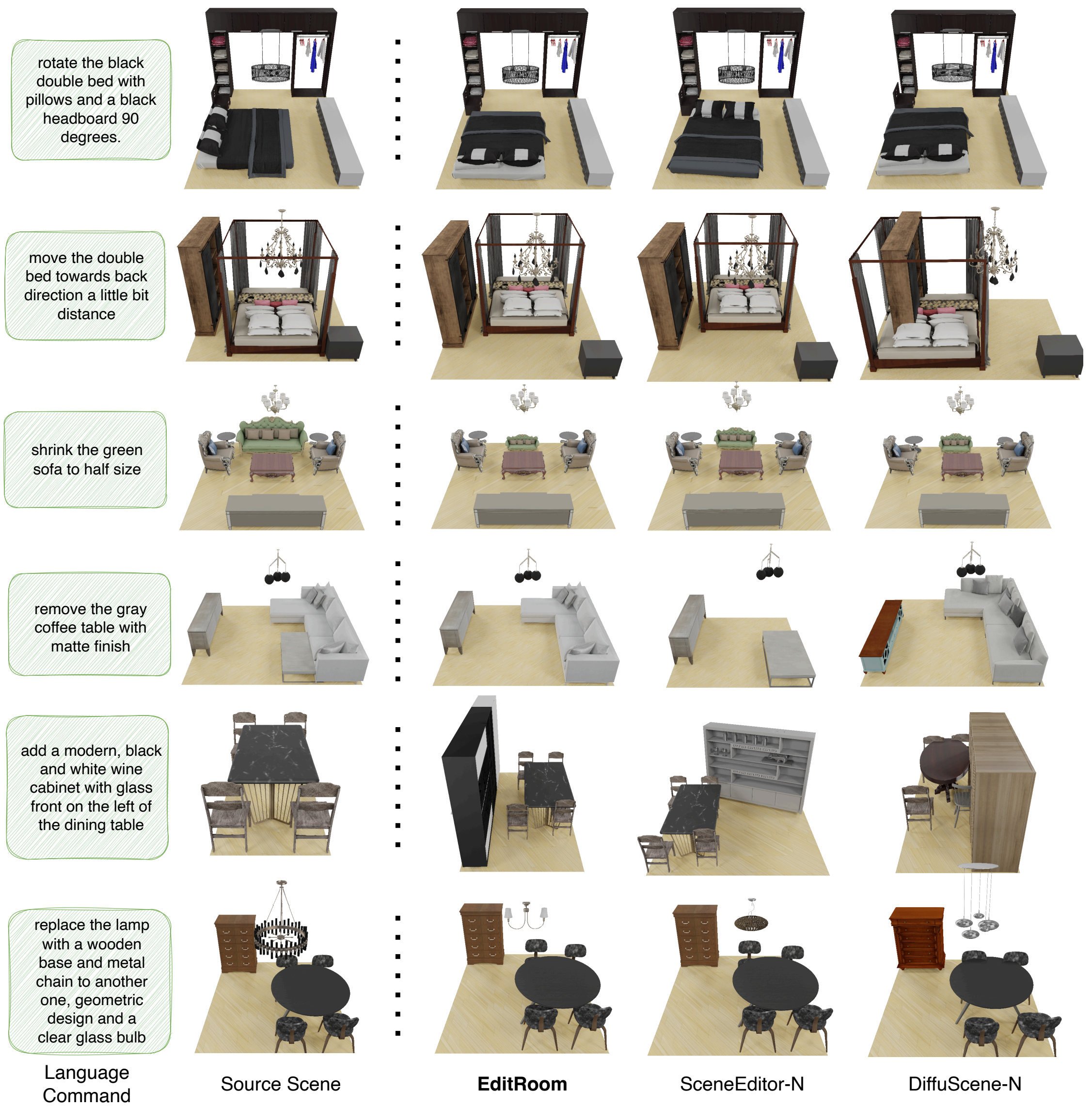

Figure 3: **Qualitative results on single operation commands.** The left column is the source scene with single operation commands for each basic editing type. From the examples, we can find that EditRoom can provide more coherent and appropriate editing operations across all editing types.

semantic features. This underscores the potential for further enhancement of models' spatial reasoning and object alignment capabilities.

Model	IOU (↑)	S-IOU (↑)	LPIPS (↓)	CLIP (↑)
EditRoom (Concat-Text)	0.5992	0.5835	0.1325	0.9547
EditRoom (Original)	**0.7435**	**0.7344**	**0.0967**	**0.9644**

Table 4: **Ablation on different condition types on the bedroom.** From the table, we can show that incorporating source information as context with the self-attention (our design) instead of the cross-attention mechanism can significantly improve model performance.

Complex Operations To demonstrate the generalization capabilities of EditRoom, we manually designed several test prompts that combine multiple atomic operations, and we assessed each model's performance qualitatively. Figure 4, shows that EditRoom provides more coherent and appropriate responses than the baseline models. For instance, the command in the first row requests a bed replacement and the addition of a wardrobe. EditRoom successfully interprets the natural language command and translates it into the corresponding atomic operations using an LLM, whereas other baseline models misinterpret the command and perform incorrect operations such as translation. These outcomes highlight the challenges of directly training models on natural language commands for compositional editing tasks. EditRoom, by contrast, effectively executes complex editing operations through strategic LLM planning.

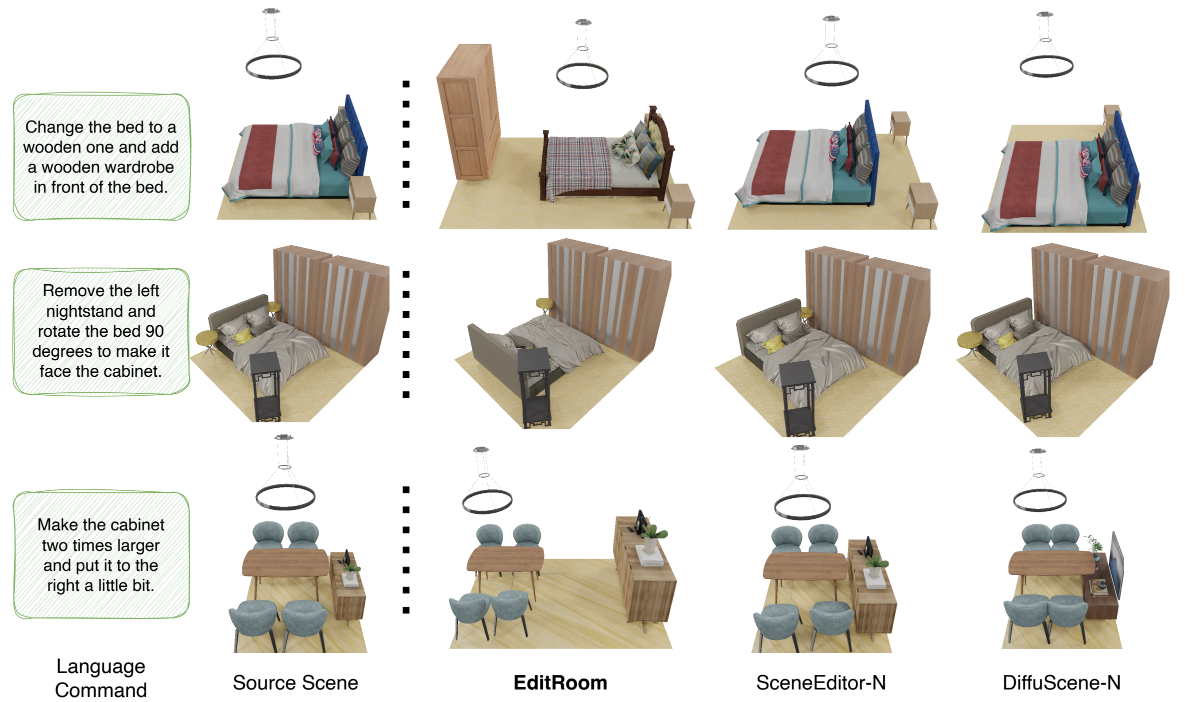

Figure 4: **Qualitative results on complex operation commands.** The left column is the source scene with complex operation commands. From the figure, we can find the EditRoom can successfully generalize to complex natural language commands with multiple operations without further training on the complex operation data, while baselines fail to execute coherent editing.

Model	IOU (↑)	S-IOU (↑)	LPIPS (↓)	CLIP (↑)
EditRoom (OpenCLIP-ViT-bigG-14)	0.6970	0.6788	0.1271	0.9490
EditRoom (Original)	**0.7435**	**0.7344**	**0.0967**	**0.9644**

Table 5: **Ablation on different text encoders on the bedroom.** Due to the limited size of training data, we find using the larger text encoder with high-dimensional features induces decreasing performance on editing, which indicates further exploration with 3D editing data generation.

Ablation on Condition Types To validate our model design, we experimented with an alternative conditioning approach, where a graph transformer encodes the source scene into a sequence of vectors that are then concatenated with text features. These combined features are incorporated into the cross-attention layers of our graph diffusion process. We specifically tested this method on the bedroom scene type, with results shown in Table 4. The table indicates a significant decrease in model performance, both in terms of layout accuracy and visual coherence. This outcome suggests that utilizing source scene information as the context for self-attention layers, rather than as conditions for cross-attention, yields better results.

Ablation on Text Encoders In an exploration of text encoder options, we replaced the CLIP-ViT-B32 text encoder (512 feature dimensions) with a larger pretrained text encoder, OpenCLIP-ViT-bigG-14 (1280 feature dimensions), used by OpenShape to align with object semantic features—consistent with the object features in our models. We conducted tests on the bedroom test set, with outcomes detailed in Table 5. The results indicate that the model equipped with the larger text encoder underperforms compared to the one using the original encoder. We attribute this decrease in performance to the limited size of our training dataset. Given that our diffusion models are trained from scratch, they require more data to effectively align with higher-dimensional features ($d = 512$ vs $d = 1280$). This finding underscores the need for further exploration into constructing larger scene editing datasets.

6 Conclusion

In this work, we introduce EditRoom, a language-guided 3D room layout editing method. EditRoom incorporates a graph diffusion-based scene editor that facilitates unified basic editing operations, and it utilizes an LLM for natural language planning. Our experiments demonstrate that EditRoom can effectively execute appropriate edits for both single and complex operations. We believe this work will inspire further research into language-guided 3D scene layout editing.

Limitation Since EditRoom leverages the LLM for the command planner, its performance is contingent upon the LLM's capability in 3D scene understanding and natural command comprehension. This dependency may lead to the generation of erroneous commands that prompt the scene editor to execute potentially problematic operations, such as collisions. However, because the training data predominantly consist of collision-free samples, there is an inherent trade-off between adhering strictly to the commands and avoiding collisions. If the commands deviate significantly from typical scenarios—such as moving an object 100 meters away—the model might instead perform a similar action that falls within the observed training distributions.

References

Edward Bartrum, Thu Nguyen-Phuoc, Chris Xie, Zhengqin Li, Numair Khan, Armen Avetisyan, Douglas Lanman, and Lei Xiao. 2024. Replaceanything3d: Text-guided 3d scene editing with compositional neural radiance fields. *arXiv preprint arXiv:2401.17895*.

Yiwen Chen, Zilong Chen, Chi Zhang, Feng Wang, Xiaofeng Yang, Yikai Wang, Zhongang Cai, Lei Yang, Huaping Liu, and Guosheng Lin. 2023. Gaussianeditor: Swift and controllable 3d editing with gaussian splatting. *arXiv preprint arXiv:2311.14521*.

Weixi Feng, Wanrong Zhu, Tsu-jui Fu, Varun Jampani, Arjun Akula, Xuehai He, Sugato Basu, Xin Eric Wang, and William Yang Wang. 2024. Layoutgpt: Compositional visual planning and generation with large language models. *Advances in Neural Information Processing Systems*, 36.

Huan Fu, Bowen Cai, Lin Gao, Ling-Xiao Zhang, Jiaming Wang, Cao Li, Qixun Zeng, Chengyue Sun, Rongfei Jia, Binqiang Zhao, et al. 2021a. 3d-front: 3d furnished rooms with layouts and semantics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10933–10942.

Huan Fu, Rongfei Jia, Lin Gao, Mingming Gong, Binqiang Zhao, Steve Maybank, and Dacheng Tao. 2021b. 3d-future: 3d furniture shape with texture. *International Journal of Computer Vision*, pages 1–25.

Ayaan Haque, Matthew Tancik, Alexei A Efros, Aleksander Holynski, and Angjoo Kanazawa. 2023. Instruct-nerf2nerf: Editing 3d scenes with instructions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 19740–19750.

Yining Hong, Haoyu Zhen, Peihao Chen, Shuhong Zheng, Yilun Du, Zhenfang Chen, and Chuang Gan. 2023. 3d-llm: Injecting the 3d world into large language models. *Advances in Neural Information Processing Systems*, 36:20482–20494.

Jiangyong Huang, Silong Yong, Xiaojian Ma, Xiongkun Linghu, Puhao Li, Yan Wang, Qing Li, Song-Chun Zhu, Baoxiong Jia, and Siyuan Huang. 2023. An embodied generalist agent in 3d world. *arXiv preprint arXiv:2311.12871*.

Johnny Huynh. 2009. Separating axis theorem for oriented bounding boxes. *URL: https://jkh.me/files/tutorials/Separating Axis Theorem for Oriented Bounding Boxes.pdf*.

Nazmul Karim, Umar Khalid, Hasan Iqbal, Jing Hua, and Chen Chen. 2023. Free-editor: Zero-shot text-driven 3d scene editing. *arXiv preprint arXiv:2312.13663*.

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

Chenguo Lin and MU Yadong. 2023. Instructscene: Instruction-driven 3d indoor scene synthesis with semantic graph prior. In *The Twelfth International Conference on Learning Representations*.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2024a. Improved baselines with visual instruction tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 26296–26306.

Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee. 2024b. Llava-next: Improved reasoning, ocr, and world knowledge.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual instruction tuning. *arXiv preprint arXiv:2304.08485*.

Minghua Liu, Ruoxi Shi, Kaiming Kuang, Yinhao Zhu, Xuanlin Li, Shizhong Han, Hong Cai, Fatih Porikli, and Hao Su. 2024c. Openshape: Scaling up 3d shape representation towards open-world understanding. *Advances in Neural Information Processing Systems*, 36.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Jiapeng Tang, Yinyu Nie, Lev Markhasin, Angela Dai, Justus Thies, and Matthias Nießner. 2023. Diffuscene: Scene graph denoising diffusion probabilistic model for generative indoor scene synthesis. *arXiv preprint arXiv:2303.14207*.

Alexander Vilesov, Pradyumna Chari, and Achuta Kadambi. 2023. Cg3d: Compositional generation for text-to-3d via gaussian splatting. *arXiv preprint arXiv:2311.17907*.

Xin Wang, Shinji Takaki, Junichi Yamagishi, Simon King, and Keiichi Tokuda. 2019. A vector quantized variational autoencoder (vq-vae) autoregressive neural f_0 model for statistical parametric speech synthesis. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:157–170.

Jianing Yang, Xuweiyi Chen, Shengyi Qian, Nikhil Madaan, Madhavan Iyengar, David F Fouhey, and Joyce Chai. 2023. Llm-grounder: Open-vocabulary 3d visual grounding with large language model as an agent. *arXiv preprint arXiv:2309.12311*.

Mingqiao Ye, Martin Danelljan, Fisher Yu, and Lei Ke. 2023. Gaussian grouping: Segment and edit anything in 3d scenes. *arXiv preprint arXiv:2312.00732*.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*.

Gengze Zhou, Yicong Hong, and Qi Wu. 2024a. Navgpt: Explicit reasoning in vision-and-language navigation with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 7641–7649.

Kaiwen Zhou, Kaizhi Zheng, Connor Pryor, Yilin Shen, Hongxia Jin, Lise Getoor, and Xin Eric Wang. 2023. Esc: Exploration with soft commonsense constraints for zero-shot object navigation. In *International Conference on Machine Learning*, pages 42829–42842. PMLR.

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

Jingyu Zhuang, Chen Wang, Liang Lin, Lingjie Liu, and Guanbin Li. 2023. Dreameditor: Text-driven 3d scene editing with neural fields. In *SIGGRAPH Asia 2023 Conference Papers*, pages 1–10.

A LLM as Command Planner

(Referred by Section 3.1) A detailed dialog between user and LLM (GPT-4o) is shown in Figure 5.

B Scene Editor Implementation Details

All language commands are encoded through the pretrained CLIP-ViT-B32 text encoder. Each graph diffusion model includes a five-layer graph transformer model with 512 hidden dimensions and 8 attention heads. Training is conducted using the AdamW optimizer over 300 epochs, with a batch size of 512 and a learning rate of 2×10^{-4}. All models are individually trained and tested on each room type. For EditRoom, template commands are employed for the scene editor during training, whereas other baseline models utilize natural language commands generated by GPT-4o. During testing, all models receive natural language commands as input.

C EditRoom-DB pipeline details

(Referred by Section 4) A detailed example of using LLM to generate natural language description from template command is shown in Figure 6.

Add and Remove Objects Removing each object in the scene separately could generate the modified scenes as the result after removal compared to the original scene. Conversely, the original scene could be treated as the result after object addition. The formatted editing description will be 'Add/Remove [object description]'. In order to consider the location of the addition and potential multiple objects in the scene, we will add the relative location description with the closest unique object in the scene, like 'location: [relative description] [reference object description]'.

Pose and Size Changes We can similarly repeat the pose change operation for every object in the scene as add/remove. Specifically, we design three operations: translation, rotation, and scaling. For translation, we create random translations as the mix of distances, sampled from 0.1 meters to 1.5 meters with step 0.1, and directions, sampled along the two axes directions (front/back and left/right). Then, collision checking is done for every translated object until we find a collision-free sample. The translation will be skipped if all the samples fail in collision checking. The formatted editing description will be 'Move object towards the front/back/left/right direction for [distance] : [object description]'

Similarly, we create random rotation angles as the mix of uniform direction samples, clockwise or counterclockwise, and random values between $15 - 180$ degrees with the step of 15 degrees, and check collision for each sample. The check stops when we find a collision-free sample or all samples fail the checking. The formatted editing description will be 'Rotate object [angle] degrees : [object description]'

For scaling, we separate it as shrinking and enlarging. The scaling factor is randomly generated between 0.5-0.8 or 1.2-1.5. The scaling factor uniformly applies to three dimensions. Since shrinking won't cause a collision with other objects, it can always result in a successfully modified scene. For enlarging, if collision checking fails on all trials, the enlarging is skipped. Otherwise, we save the largest collision-free scaling factor. The formatted editing description will be 'Shrink/Enlarge object by [scale_factor] times : [object description]'

Object Replacement For the replace operation, we access an object dataset with semantic class labels and 3D meshes. The system will retrieve several objects from the dataset with the same class label as the replaced object, and check their collision with other existing objects in the scene. If none of the objects could be placed without collision, we randomly select one object and shrink its bounding box to be equal or smaller than the replaced object to avoid collision. The formatted description is 'Replace source with target : [source object description] to [target object description]'.

Collision Detection Module The objects are abstracted as 3D bounding boxes and further decomposed into 2D bounding boxes on a horizontal plane and vertical range, as the objects can only rotate about the vertical axis. Then, the two objects are only in collision if their 2D bounding boxes overlap and their vertical ranges overlap. For 2D bounding box collision detection, we apply the Separating Axis Theorem (Huynh, 2009) to determine if the boxes intersect.

System Prompt

Imagine you are a indoor room designer and you are using provided API to control the 3D models in the scene.
Given one scene configuration and a command to edit the scene, you should use the provided APIs to do planning and achieve the target.
All sizes and centroids in scene configurations are in meters. The angles are defined in degrees. The dimension sequence is [x,y,z]. Vertical angles are the angles along the y-axis.
Sizes are the half lengths of the bounding box along the x, y, and z axes when the vertical angle is zero.
We define +x/-x as the right/left direction, +y/-y as the up/down direction, and +z/-z as the front/back direction.
Positive angles are counterclockwise, and negative angles are clockwise.

APIs:
1. Rotate an object: [Rotate, Target Object Description, Angle :(degrees)]
2. Translate an object: [Translate, Target Object Description, Direction :(x/y/z), Distance :(meters)]
3. Scale an object: [Scale, Target Object Description, Scale Factor]
4. Replace an object: [Replace, Source Object Description, Target Object Description]
5. Add an object: [Add, Target Object Description]
6. Remove an object: [Remove, Target Object Description]

Matters needing attention:
1. If there are multiple same objects in the scene and the command is related to the object, you should refer to the object locations.
When you refer to the object locations, you should this format: (Relative Description, Relative Object Description).
When you use add or remove command, you should refer to the object locations.
Relative Description: [left, right, in front of, behind, above, below, closely left, closely right, closely in front of, closely behind]. 'closely' means the distance between two object centroids are less than 1 meters in x-z plane.
For example, if you want to add a chair in front of the table, you should use the format: [Add, Chair, (in front of, Table)].
2. Translate, rotate, and scale commands should be executed in the order of scale, rotate, and translate.
Consider the protential collision between objects when you execute the commands.
3. When you scale an object, the object should be scaled uniformly. Scale factor should one float number.
4. Replace object will only replace the object with the same class. Replace command will only change the object appearance, not the object poses and sizes.
5. If Translate/Rotate/Scale commands can achieve the target, you should not use Replace/Add/Remove commands.
6. If image is provided, you should use the image to help you understand the scene.
7. Attempt to use the minimum number of commands to achieve the target.
8. If you want to remove and add the object within the same class, you should use the replace command.
9. Object descriptions should be detailed descriptions instead of class names. You can imagine the object descriptions if the object is not in the scene.
10. All apis should be able to converted to a list of strings and numbers, which can be directly processed by json.loads()

For example:
1. If you want to rotate a chair 90 degrees and there is only one chair in the scene, you should use the format: ['Rotate', 'chair', 90].
2. If you want to add a chair in front of the table, you should use the format: ['Add', 'chair', ('in front of', 'table')].
3. If you want to remove a chair, you should use the format: ['Remove', 'chair'].
4. If you want to replace a metal chair with a wooden one and this chair on the left of the bed, you should use the format: ['Replace', 'mental chair', 'wooden chair', ('left', 'bed')].

Think about it step by step. Summarize the used apis at the end by lines. The final output format should be ***api1;api2;...***.

User Input

[Scene configurations]:
Object 0: {"class": "double bed", "size": [1.01, 0.39, 1.08], "vertical angle": -90, "centroid": [3.22, 0.0, -2.48],
"description": "the double bed is a modern, minimalist design with a white color scheme and a simple, clean appearance."}
Object 1: {"class": "nightstand", "size": [0.28, 0.24, 0.22], "vertical angle": 0, "centroid": [2.05, 0.0, -4.52],
"description": "the nightstand is a modern, dark wood piece with a sleek, minimalist design."}
Object 2: {"class": "nightstand", "size": [0.33, 0.33, 0.23], "vertical angle": 0, "centroid": [4.54, 0.0, -3.35],
"description": "the nightstand is a simple, white, two-drawer piece with a smooth finish and a small, round knob on each drawer."}
Object 3: {"class": "wardrobe", "size": [1.04, 1.02, 0.32], "vertical angle": 90, "centroid": [1.51, 0.0, -2.48],
"description": "the wardrobe is a large, white, traditional style piece of furniture with a decorative top and multiple doors."}
Object 4: {"class": "desk", "size": [1.0, 0.38, 0.3], "vertical angle": -90, "centroid": [5.6, 0.0, -2.47],
"description": "the desk is a modern, minimalist design with a sleek, rectangular shape and a matte gray finish."}
[Command]:"replace the nightstand near the white cabinet with the same model of another nightstand near the black table,
then move it closer to the bed corner to be symmetric with another one."

replace the nightstand near the white cabinet with the same model of another nightstand near the black table, then move it closer to the bed corner to be symmetric with another one.

LLM Response

To achieve the command of replacing the nightstand near the white cabinet with the other nightstand near the black table and moving it to be symmetric with the other nightstand closer to the bed corner, we can follow these steps:
1. Identify the nightstands:
   a. Nightstand near the white cabinet (wardrobe): This is the darker wood nightstand at centroid [2.05, 0.0, -4.52].
   b. Nightstand near the black table: This is the white nightstand at centroid [4.54, 0.0, -3.35].
2. Replace the darker wood nightstand with the white nightstand.
3. Move this replaced nightstand closer to the bed so it is symmetric with the other nightstand:
   a. To maintain symmetry, the nightstand should be as positioned close to the bed corner, in the right direction.
Let's execute the steps to achieve the target:
1. **Replace the dark wood nightstand with the white nightstand:**
   ```json
   ['Replace', 'modern, dark wood nightstand', 'simple, white two-drawer nightstand', ('near', 'white cabinet')]
   ```
2. **Translate the replaced nightstand to the right direction to make it symmetric:**
   Considering there is another nightstand near the bed (on the right side), the new one should move closely to the same distance as the first one but in the symmetric position (left of the bed):
   ```json
   ['Translate', 'simple, white two-drawer nightstand', 'x', 1.22]
   ```
Final API commands used:
- **Replace the nightstand:**
  ```json
  ["Replace", "modern, dark wood nightstand", "simple, white two-drawer nightstand", ("near", "white cabinet")]
  ```
- **Translate the nightstand to ensure symmetry with another nightstand near the bed:**
  ```json
  ["Translate", "simple, white two-drawer nightstand", "x", 1.22]
  ```
Combining these together, the final output is:
Output:
*****["Replace", "modern, dark wood nightstand", "simple, white two-drawer nightstand", ("near", "white cabinet")];
["Translate", "simple, white two-drawer nightstand", "x", 1.22]*****

Figure 5: Dialog between user and LLM to generate an edit plan to rearrange the bedroom. The room image in 'User Input' is not given to LLM, just for illustration purpose.

System Prompt

Given scene configurations and templated commands, you should write new commands using natural language and spatial reference.
Templated commands will be in the 'action: target_object' format. If the location is provided in the templated commands, it can be considered as a hint for the target object's location compared to the existing object in the scene.
All sizes and centroids in scene configurations are in meters. The angles are defined in degrees. The dimension sequence is [x,y,z]. Vertical angles are the angles along the y-axis.
Sizes are the half lengths of the bounding box along the x, y, and z axes when the vertical angle is zero.
We define +x/-x as the right/left direction, +y/-y as the up/down direction, and +z/-z as the front/back direction.
When you design new commands, please refer to the spatial relations between objects in the scene.
When you design new commands, please consider correctness, conciseness, and naturalness.
You should attempt to make your command need reasoning.
If there are duplicate target objects in the scene, you should refer to object locations by relative spatial relations with one unique object in the scene.
If there are multiple templated commands, you should consider them as the same command with different representations.
If templated commands indicate to add an object where there is already a similar object, you should indicate this is about adding a new object in your command.
Enlarge and shrink in the command should be uniform.
You can add object descriptions according to the scene configurations, commands, and the image (if provided).
For example:
Example1:
[Templated commands]:['move object towards the ***left*** direction for 1 meters: a white bed with a red and white plaid comforter and a red and white plaid pillow.']
If there is a table (only one table inside the scene) on the left side of the bed and length of bed is 2 meters, you can write: 'move the white bed with red and white plaid towards the table around 1 meters.' or 'move the bed towards the left direction by half of bed length.'
Example2:
[Templated commands]:['move object towards the ***left*** direction for 0.5 meters: a wooden nightstand.]'
If there is a bed parallel to the nightstand and moving to the left will make the nightstand closer to the bed headboard, you can write: 'move the nightstand closer to the bed headboard by 0.5 meters'.
Example3:
[Templated commands]:['replace source with target : [Source] a white bed; [Target] a brown bed.']
You can write: 'replace the white bed with a brown bed.'
Example4:
[Templated commands]:['add object: a white bed; location: ***right*** a wardrobe.']
If there is a wardrobe in the scene, you can write: 'add a white bed on the right side of the wardrobe.'
Now you can start to design new commands based on the scene configurations and templated commands. You can supplement object descriptions on the command.
Think about it step by step and summarize your commands in the end. The final output format should be '###[natural command 1, natural command 2, ...]###', which is a list of strings and can be processed by ast.literal_eval() or json.loads().

User Input

[Scene configurations]:
Object 0: {"class": "dining table", "size": [0.55, 0.38, 0.23], "vertical angle": 0, "centroid": [-0.8, 0.0, -3.46],
"description": "the dining table is a modern, minimalist design with a black marble top and a silver metal frame."}
Object 1: {"class": "loveseat sofa", "size": [1.24, 0.43, 0.47], "vertical angle": 90, "centroid": [-3.78, 0.0, 1.38],
"description": "the loveseat sofa is brown with a modern design and has a variety of patterned throw pillows."}
Object 2: {"class": "coffee table", "size": [0.69, 0.23, 0.47], "vertical angle": 90, "centroid": [-2.48, 0.0, 1.4],
"description": "the coffee table is a modern, minimalist design with a geometric shape, featuring a combination of dark wood and lighter wood panels."}
Object 3: {"class": "lounge chair", "size": [0.37, 0.45, 0.37], "vertical angle": 153, "centroid": [-2.94, 0.0, 3.11],
"description": "the chair is a modern, minimalist design with a dark wood frame and a cushion featuring a geometric pattern."}
Object 4: {"class": "corner side table", "size": [0.22, 0.23, 0.22], "vertical angle": 90, "centroid": [-3.99, 0.0, -0.25],
"description": "a round, black marble table with a white base."}
Object 5: {"class": "dining chair", "size": [0.31, 0.45, 0.3], "vertical angle": -180, "centroid": [-0.47, 0.0, -2.79],
"description": "the chair is black with a modern design, featuring a high back and armrests."}
Object 6: {"class": "dining chair", "size": [0.31, 0.45, 0.3], "vertical angle": -180, "centroid": [-1.08, 0.0, -2.79],
"description": "the chair is black with a modern design, featuring a high back and armrests."}
Object 7: {"class": "corner side table", "size": [0.22, 0.23, 0.22], "vertical angle": 90, "centroid": [-3.99, 0.0, 3.13],
"description": "a round, black marble table with a white base."}
Object 8: {"class": "dining chair", "size": [0.31, 0.45, 0.3], "vertical angle": 0, "centroid": [-1.08, 0.0, -4.17],
"description": "the chair is black with a modern design, featuring a high back and armrests."}
Object 9: {"class": "dining chair", "size": [0.31, 0.45, 0.3], "vertical angle": 0, "centroid": [-0.48, 0.0, -4.17],
"description": "the chair is black with a modern design, featuring a high back and armrests."}
Object 10: {"class": "console table", "size": [0.7, 0.42, 0.15], "vertical angle": 0, "centroid": [-3.35, 0.0, -4.56],
"description": "the console table is a modern, black, three-tiered design with a flat top and a rectangular base."}
Object 11: {"class": "cabinet", "size": [0.62, 1.08, 0.29], "vertical angle": -90, "centroid": [-3.94, 0.0, -2.04],
"description": "the children's cabinet is a modern, minimalist design with a light wood frame and a blue and white color scheme, featuring a playful bunny motif on the doors."}
Object 12: {"class": "pendant lamp", "size": [0.18, 0.54, 0.18], "vertical angle": 90, "centroid": [-2.71, 1.53, 4.35],
"description": "the pendant lamp is a modern, metallic chandelier with a white finish, featuring a series of vertical, clear glass tubes that create a geometric pattern."}
Object 13: {"class": "pendant lamp", "size": [0.19, 0.52, 0.19], "vertical angle": 90, "centroid": [-2.26, 1.55, -0.3],
"description": "the pendant lamp is black with a woven design and a white interior."}

**[Templated commands]:["rotate object 135 degrees : the pendant lamp is black with a woven design and a white interior",
"obviously rotate object 135 degrees :the pendant lamp is black with a woven design and a white interior."]
Hint: The target object is the Object_13.**

LLM Response

*****###['Rotate the black pendant lamp with a woven design and a white interior by 135 degrees.']###*****

Figure 6: Dialog between user and LLM to generate natural language description from the template command. The room image in 'User Input' is not given to LLM, just for illustration purposes.