PlaceIt3D: Language-Guided Object Placement in Real 3D Scenes

Ahmed Abdelreheem[2,*] Filippo Aleotti[1] Jamie Watson[1] Zawar Qureshi[1] Abdelrahman Eldesokey[2]
Peter Wonka[2] Gabriel Brostow[1,3] Sara Vicente[1] Guillermo Garcia-Hernando[1]
[1]Niantic Spatial [2]KAUST [3]UCL

https://nianticlabs.github.io/placeit3d/

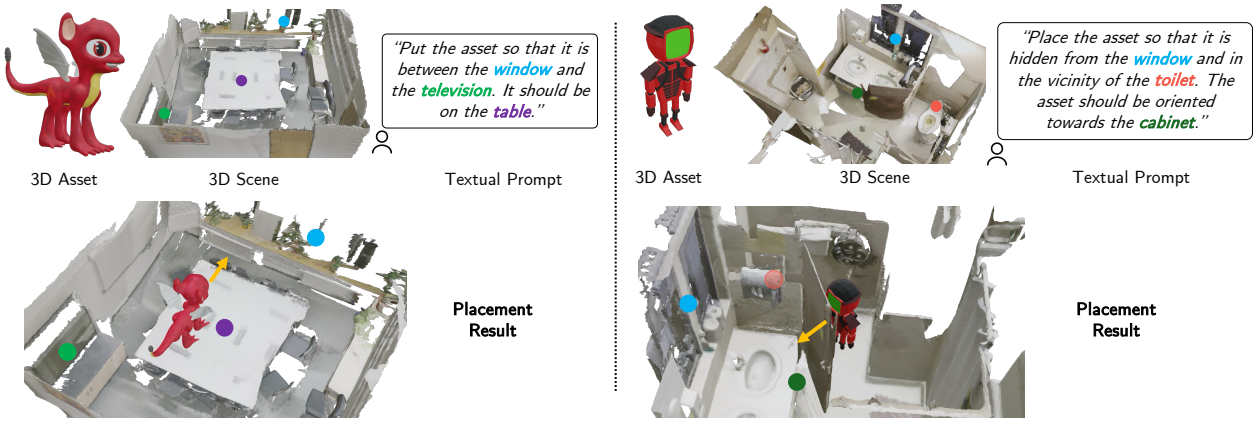

Figure 1. **Language-guided 3D Object Placement**: Our new task involves finding a valid placement for an asset according to a text prompt. This task requires semantic and geometric understanding of the scene, knowledge of the asset's geometry, and reasoning about object relationships and occlusions. The colored dots represent the positions of the objects mentioned in the prompt (provided only for visualization purposes and not given to the model), while the yellow arrow indicates the predicted frontal direction of the asset.

Abstract

We introduce the novel task of Language-Guided Object Placement in Real 3D Scenes. Our model is given a 3D scene's point cloud, a 3D asset, and a textual prompt broadly describing where the 3D asset should be placed. The task here is to find a valid placement for the 3D asset that respects the prompt. Compared with other language-guided localization tasks in 3D scenes such as grounding, this task has specific challenges: it is ambiguous because it has multiple valid solutions, and it requires reasoning about 3D geometric relationships and free space. We inaugurate this task by proposing a new benchmark and evaluation protocol. We also introduce a new dataset for training 3D LLMs on this task, as well as the first method to serve as a non-trivial baseline. We believe that this challenging task and our new benchmark could become part of the suite of benchmarks used to evaluate and compare generalist 3D LLM models.

1. Introduction

At two to three years old, neurotypical children learn to follow two-step instructions like *"Get your shoes and put them on the shelf"* [33]. These may seem like simple tasks, but children need time to understand basic vocabulary and to learn physical affordances of both 3D objects and scene layout. Perhaps AIs could obtain similar capabilities.

In this paper, we focus on the novel task of *language-guided 3D object placement* in a 3D scene. Like in the shoe example, the specific task here is to find a valid placement of the object among multiple configurations that satisfy the instructions. It must also respect the physical constraints of the space and the 3D asset (see Figure 1). Excelling at this task would unlock applications, such as instructing a robot using language to move a real object to a new location. It also has applications in augmented reality (AR) or virtual

*Work done during an internship at Niantic.

reality (VR). For example, a player of a virtual game can use language to give orders to virtual characters or move assets in a scene.

LLMs have recently been shown to be effective for tasks related to 3D scene understanding [13, 17, 46], particularly, visual question answering [14], grounding [18], and captioning [27]. Our task is most related to grounding; however, ours has a few properties that make it more challenging. First, grounding usually has a well-defined and unique solution, while our task is inherently ambiguous because multiple valid solutions exist. The ambiguity of our task makes defining a benchmark and constructing a training dataset non-trivial, since both have to account for the validity of multiple solutions. Second, the task cannot be easily solved by using 2D information alone, since many of the constraints are geometric and require 3D reasoning. For example, a language instruction like *"place the asset in between the chair and the table, facing the television"* requires reasoning about free space as well as geometric relationships, which are intrinsically 3D. Finally, the task requires us to analyze not only the scene but also the asset. This is because the size and shape of the asset constrain the set of valid placements. For example, given the same scene and language prompt, we expect a large object to have fewer valid possible placements than a smaller object.

The highlight of this paper is the introduction of a challenging novel task: language-guided 3D placement. To the best of our knowledge, there are no benchmarks or datasets that meet our needs, so we drive progress on this task with three main contributions, summarized here:

- We provide a benchmark for language-guided placement containing 3,300 evaluation examples, where each example is composed of a real scene from ScanNet [10], an asset from the PartObjaverse-Tiny dataset [40], and a guiding language prompt. The benchmark includes an evaluation protocol that takes into account the ambiguity of the placement task, which in general, has multiple valid solutions. This benchmark will allow future methods to evaluate progress on this challenging task.
- We propose PlaceIt3D, a large-scale dataset that can be used for training 3D LLM models for the task of language-guided placement. Similarly to the benchmark, it uses real scenes from ScanNet [10] and assets from PartObjaverse-Tiny [40]. Each example in the data set (made up of a scene, asset, and prompt triplet) includes all the valid placements. This training corpus contains a total of 97,020 training and 2,619 validation samples.
- We introduce a proto-method for this guided placement task that we call PlaceWizard. It builds on recent work in 3D LLMs [17] and outperforms baselines. It employs a modified form of spatial aggregation, an asset encoder, and rotation prediction.

2. Related Work

3D Scene understanding and language. Scene understanding is a long-standing challenge in computer vision, crucial for improving autonomous systems [28] and augmented reality (AR) [22]. A popular approach to tackle problems of scene understanding involves building a 3D representation of the scene and processing it using task-specific networks [9, 16, 29, 31]. Now, Large Language Models (LLMs) are rapidly expanding beyond textual inputs to encompass multi-modal data, paving the way toward more human-like interactions. LLaVa [25] was the first method to extend instruction tuning protocols to the vision domain. Subsequent works like [21, 23] unlocked multiple images as input, spatial reasoning capabilities [4, 5, 8], and segmentation [19]. Particularly relevant to our work are those LLM-based systems that process 3D input data. To tackle tasks such as grounding or question answering, 3D-LLM [13] takes its input from point clouds with features extracted from 2D images. Reason3D [17] employs a pre-trained point encoder to extract features from point clouds. These features are then projected into the embedding space of the LLM using Q-Former [24]. Subsequently, a decoder infers object masks based on the LLM's output. ScanReason [46] interleaves grounding and reasoning steps at inference time. Noticeably, none of these methods is designed for object placement, which requires reasoning not just about objects present in the scene, but also about empty space, relationships between objects, and the input asset.

Datasets for language-guided 3D tasks. Multiple datasets for language-guided 3D tasks [1–3, 6, 7, 43] have been built on top of ScanNet [10], taking advantage of the readily available annotations. These datasets provide examples and annotations for new tasks such as text-based object grounding [1, 2, 6, 18], visual question answering [3], captioning [7], and navigation [43]. More recent works have also proposed larger-scale [38] and synthetic datasets [39] for 3D language tasks. Although our data set and benchmark are also built on top of ScanNet, none of the existing 3D datasets focuses on the task of guided placement, with its challenges and range of acceptable placements, making our dataset complementary to the existing ones.

Object placement in 2D images and 3D scenes. Several existing methods [26, 30, 32, 44, 47] address the problem of *common-sense placement* in images, *i.e*. predicting 2D image regions where asset placement is semantically plausible, such as a book on a bookshelf. While this is also a placement task, we emphasize that our *language guided* placement task is intrinsically different, since the placement is directed by the language instructions not by common-sense principles. More similar to ours, Robopoint [42] addresses the task of language-guided placement in 2D images. Their placement predictions are lifted to 3D by using depth maps. In contrast, our method uses the full 3D scene

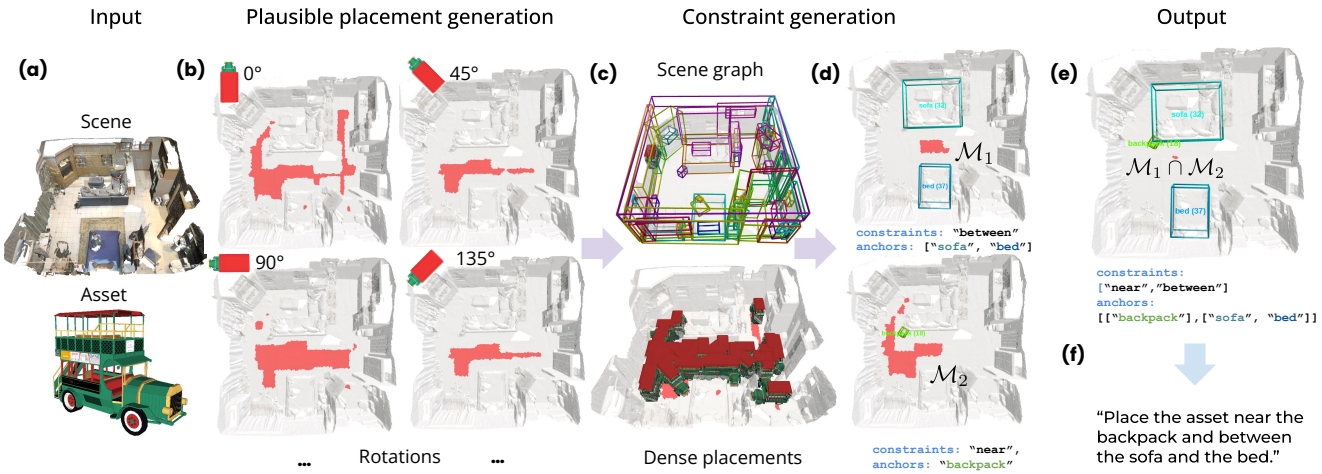

Figure 2. **PlaceIt3D dataset creation**. Given a scene and an asset as input **(a)** the goal is to create a prompt **(f)** and corresponding mask $\mathcal{M}$ of valid placements **(e)**. We start by finding the set of points which are physically plausible placements, shown in red in **(b)**. We consider eight equally spaced rotation angles, which condition the valid placements. For this example, angle 0° has more valid placements than 45°. To generate the language constraints, we use the ground truth scene graph **(c)**. Object anchors are selected from the scene graph and combined with relationship types to create a constraint and corresponding validity mask **(d)**. The different placement constraints are combined in the final output by intersecting the validity masks **(e)** given a mask of valid *dense* placements. Based on each selected set of anchors and constraint relationships, a natural language prompt is created using templates (please, see supplemental for more details).

as input, which provides more context and allows reasoning about anchor objects which might be out of view in a single image. The concurrent work, FirePlace [15], proposes a zero-shot pipeline to insert 3D objects into synthetic 3D scenes using common-sense reasoning. While we share the geometric reasoning aspect of the task, our approach goes beyond common-sense reasoning by explicitly incorporating purely geometric constraints, such as object rotation and visibility. Additionally, our method operates beyond synthetic scenarios, as it focuses specifically on reasoning within predicted 3D scenes and objects.

3. Language-Guided 3D Object Placement

We introduce the task of language-guided 3D object placement. Given the pointcloud of a 3D scene, a 3D asset, and text describing where the asset should be placed in the scene, the goal is to find a valid position and orientation for the asset that is physically plausible and adheres to the language prompt.

This task is inherently ambiguous because, in general, multiple valid placements exist. The multiple placements in Fig. 2 (c) demonstrate this ambiguity and illustrate the complexity of our task when compared with related tasks like object grounding, which typically has a single solution.

Simplifying assumptions. Given the ambiguity and complexity of our task, we make some simplifying assumptions to make the problem tractable. First, we assume the vertical orientation of the scene is fixed and given by the Z-axis. We also assume we know the vertical orientation of the asset as well as its frontal direction. The asset is always placed on a horizontal surface, and only the yaw angle is considered, *i.e*. rotation around the vertical axis.

3.1. Physical plausibility and language constraints

We stipulate that valid placement in this task should take into account *at least* a specific set of commonly-occurring constraints. The first constraint is **physical plausibility**, which enforces that all the placements are realistic and viable. A placement is viable if the object does not intersect with the scene mesh. This constraint is agnostic to the language instructions and must always be satisfied.

Beyond physical plausibility, the language prompt dictates how the object should be placed in the scene. Placement is often relative to an "anchor," which is a named object instance and is either available in the ground truth of a scene graph, or is inferred at test-time. In practice, "language constraints" capture both semantic and physical aspects of the placement. These language constraints are organized into three distinct groups:

Spatial constraints: These constraints specify the object's location relative to one or more scene anchors. This group includes: (i) *near* and *adjacent*: the object is positioned within a specified distance from an anchor. (ii) *on*: the object should directly rest on top of an anchor. (iii) *between*: the object must be placed between two anchors. (iv) *above* and *below*: the object is located above or below an anchor.

Rotational constraint: This constraint focuses on the orientation of the object relative to scene anchors. The object is positioned so that it faces toward the anchor.

Visibility constraints: The object is positioned so that it sits within the line-of-sight of an anchor, and can be *visible* or *not visible*.

All placements should be physically plausible. Beyond that, each example includes a combination of language constraints in the prompt. A candidate placement is considered a valid placement if and only if it simultaneously satisfies every constraint in that prompt.

Prompt creation: As a proxy for humans typing in desired constraints, we create language prompts using a template-based system, shown in the supplemental material. For each of the language constraints, we have a set of predefined template language sentences that describe it. We combine a random sample of the constraints to get the final prompt. The anchor objects are also randomly selected from the ScanNet annotations. We have a verification step where we discard prompts that cannot be satisfied, since there is no valid placement that follows it. This can happen if the selected random constraints are too restrictive and incompatible, for example: "*Place the asset on the table and below the desk.*"

3.2. Benchmark

Each benchmark example is composed of a 3D mesh of the scene, the 3D asset, and the language prompt, which is composed of one or more 3D language constraints. A method for placement would take this triplet as input and predict at least one placement, parameterized as a 3D translation vector $\mathbf{t}$ and a single rotation angle α that corresponds to the yaw angle. We use "posed object" to refer to the asset after applying the predicted transformation. Our evaluation protocol verifies whether the placement satisfies each 3D constraint individually and all of them collectively.

3.2.1. Checking validity of each 3D constraint

We use a rule-based system to check whether a predicted placement satisfies a 3D constraint. For the anchor objects, we use their ground truth oriented bounding boxes provided by ScanNet annotations. Since we consider a diversity of constraints with distinct properties, we describe how the validity of each of them is evaluated. Most constraints will depend on thresholds and parameters, to allow for small deviations. Please see supplemental material for more details.

Physical plausibility: We use Open3D [45] to check if the mesh of the posed object intersects with the scene mesh.

Spatial constraints: For the *near* and *adjacent* constraints, we compute the distance from the posed asset to the anchor object. We check the *on*, *above*, and *below* relationships by comparing the value of the z-coordinate of the placed object with the z-coordinate of the anchor object. For the *between* relationship, we check if the placed asset is close to a line connecting the centers of the two anchor objects.

Rotational constraint: We compute a cone around the frontal vector of the posed asset and check that the anchor object intersects with that cone.

Visibility constraint: To check if the placed object is visible from a specified anchor, we render it together with the scene from a camera centered at that anchor and facing the object. We then check if any of the pixels in the rendered image correspond to the asset. The asset is considered visible if any pixels correspond to it; otherwise, it is considered not visible

3.2.2. Benchmark metrics

To quantitatively assess the performance of a placement algorithm, we compute these metrics that capture the validity of the constraints both overall and for each subgroup:

Global Constraint Accuracy: The percentage of all constraints (across all groups) that are correctly satisfied over the entire dataset. It provides a holistic measure of the overall placement quality.

Complete Placement Constraint Success: The percentage of perfect placements, where every constraint is satisfied. It indicates the overall robustness of the placement method.

Subgroup Metrics: In addition to the overall metrics, we report accuracies across constraint groups.

3.2.3. Benchmark statistics

The benchmark contains 3,300 evaluation examples, combining a total of 142 different scenes from ScanNet [10] and 20 different assets from the PartObjaverse-Tiny dataset [40]. Statistics for the number of constraints and type of constraints are shown in Table 1.

Constraints per sample	#	Type	#
One	1,000	Spatial	3,679
Two	1,509	Rotational	991
Three or more	791	Visibility	970

Table 1. Benchmark statistics for the number of language constraints per benchmark sample and type of constraint. Note that the **physical plausibility** is enforced for all examples, so it is not counted as a constraint in this table.

3.3. Training dataset

Although our benchmark protocol allows offline method evaluation, we need a practical lower computational-cost approach to create a large-scale dataset for training, especially for obtaining the full set of valid placements.

Here we describe PlaceIt3D, our training dataset for the task of guided placement. The dataset consists of 97,020 training examples, sourced from 565 distinct ScanNet scenes and 20 unique assets. It includes a total of 68,561 spatial constraints, 53,009 rotational constraints, and 26,192 visibility constraints. Among these examples, 63,131 contain a single constraint, 23,420 have two constraints, and 10,469 include three or four constraints. Used throughout this paper, this training dataset is a subset, for

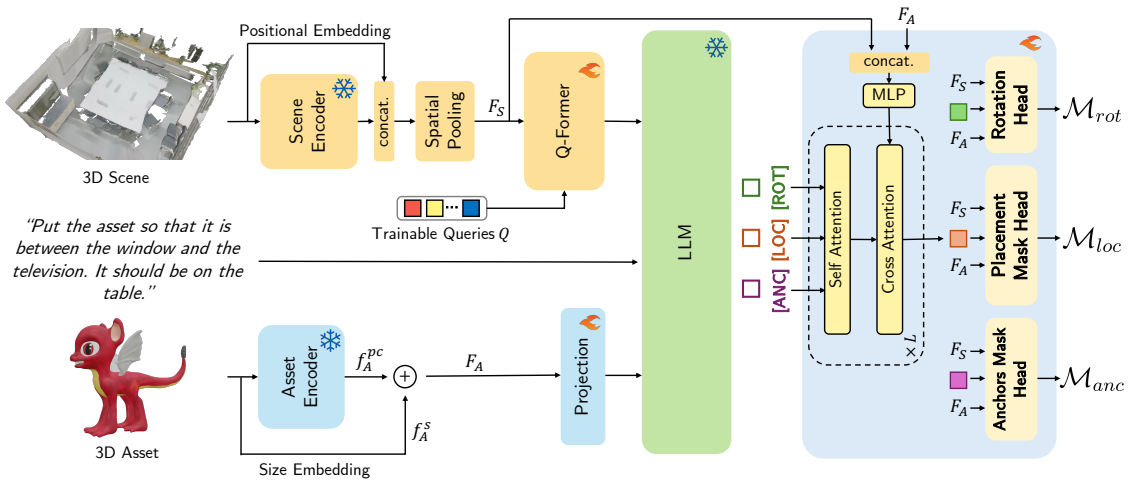

Figure 3. **Method overview.** A point encoder extracts features from the 3D scene, which are then complemented with positional embeddings. Spatial pooling reduces feature dimensions, and a Q-Former merges the pooled features with trainable queries Q (Section 4.1). The asset is encoded into a single vector by using a pretrained asset encoder followed by max-pooling (Section 4.2). This vector together with a size embedding is passed to a projection layer that aligns the features with the LLM space. The LLM take as input (i) the output of the Q-Former, (ii) the text prompt, and (iii) the projected asset features and predicts three special tokens [ANC], [LOC] and [ROT]. A transformer based decoder takes as input the features associated with the three special tokens and the pooled scene features and performs a few self and cross attention operations (Section 4.3). Three heads produce the final outputs: $\mathcal{M}_{loc}$ the valid placement mask; $\mathcal{M}_{anc}$ an auxiliary mask that localizes the object anchors; and $\mathcal{M}_{rot}$ a mask indicating which rotation angles are valid at each location.

practical purposes, of the PlaceIt3D-Full corpus we are also sharing. PlaceIt3D-Full has ∼4M examples: the 565 scenes x 140 objects x 50 prompts.

Dataset parametrization We denote the point cloud of the scene as $\mathbf{X} \in \mathbb{R}^{N\times 6}$, where each point $\mathbf{x}_i, i \in \{0, ..., N-1\}$ contains the 3D position for that point, as well as color information. Given a scene, an asset, and a prompt, we represent the set of valid ground truth placements for the asset as a binary mask $\mathcal{M}$ defined over the point cloud of the scene, associating a label $m_i \in \{0, 1\}$ to each 3D point $\mathbf{x}_i$. For each point i with label $m_i = 1$, *i.e.* a valid placement, we also define a binary mask over a discretized set of yaw angles indicating if the angle is valid for that specific location: $\alpha_i = \{\alpha_i^y \in \{0, 1\} | y = 0, ..., 7\}$, where each y corresponds to a $45°$ interval. Note that there is a fixed transform between the parametrizations used in the benchmark and the training dataset. While for the benchmark we parametrize the position of the center of the asset, for the training set, we consider contact points between the scene geometry and the asset's bottom surface.

Computing valid placement masks We create the valid placement masks $\mathcal{M}$ by using a combination of the rule-based system defined above and a few approximations to make it more efficient. More details on the approximations are available in the supplemental material. We treat each constraint independently, obtaining a valid mask per constraint $\mathcal{M}_c$ with $c \in \mathcal{C}$, where $\mathcal{C} = \{\text{physical}, \text{spatial}, \text{rotational}, \text{visibility}\}$. The final mask is given by the intersection of all the constraints that apply to that example, so

$$\mathcal{M} = \bigcap_{c\in\mathcal{C}} \mathcal{M}_c. \quad (1)$$

For the physical plausibility constraint we use a set of heightmaps to capture the different horizontal surfaces of the scene. We then compute the asset height and footprint and, for each point on a horizontal surface, check if the placement is valid. For the visibility constraint we use the same procedure as the benchmark, but use two approximations for efficiency: the asset is replaced by its bounding box, and a fixed rotation angle is used.

4. PlaceWizard: Method Description

Background. We briefly introduce Reason3D [17] as our method builds upon it. Given a textual prompt and a colored point cloud $\mathbf{X} \in \mathbb{R}^{N\times 6}$ as input, Reason3D performs dense 3D grounding, finding all the points in the point cloud that satisfy the prompt. The features $F_X \in \mathbb{R}^{N\times d}$, where d is the feature dimension, extracted by a point encoder [35] from the input point cloud are aggregated into superpoints [20] obtaining superpoint features $F_S \in \mathbb{R}^{M\times d}$, with $M \ll N$, reducing the overall complexity.

Next, the superpoint features F_s are projected into the embedding space of an LLM via a Q-Former block [24]. This model updates the learnable query vectors Q, resulting in Q'. From Q' and the input text the LLM generates a response containing two special tokens, namely [LOC] and

[SEG]. These tokens guide the model in two stages: coarse localization followed by precise mask prediction.

In practice, the Reason3D method uses a single token, [LOC], for datasets that contain small scenes, such as ScanNet, since hierarchical subdivision is not required. We will describe their method using this simplified version.

Finally, the last-layer embeddings associated to [LOC] are first projected via an MLP and then given as input to the Mask Decoder, which performs cross-attention [37] with F_s. The decoder produces an object-level binary segmentation mask over superpoints, which is upsampled into $\mathcal{M}_{loc} \in \{0,1\}^N$ to provide a segmentation mask on the full point cloud.

Figure 3 provides an overview of our method. In the following subsections, we detail our approach and emphasize the key modifications to the Reason3D architecture necessary for addressing *guided placement instead* of standard 3D visual grounding.

4.1. Scene encoding

Similarly to Reason3D, we use the point encoder from [35] to extract features $F_X \in \mathbb{R}^{N \times d}$ from the 3D scene. We use an additional positional embedding feature $F_X^{pos} \in \mathbb{R}^{N \times d^*}$, for points in the point cloud, encoding their location, which is concatenated with the previous features.

Spatial pooling. Reason3D uses superpoints [20] to reduce computational complexity and memory usage by pooling individual point features into a single feature per superpoint. While effective for their task, this coarse representation limits performance for our placement task.

For example, superpoints will generally cluster all points belonging to horizontal or vertical surfaces –such as floors, tabletops and walls– into single superpoints, which is clearly undesirable for accurate 3D placement of assets. To address this, we instead use uniform spatial pooling to aggregate features. Specifically, we use farthest point sampling [12] to select M centers and then assign points to their nearest center using Euclidean distance. By doing so, our method remains computationally tractable, while also being able to predict with sufficient granularity for accurate asset placement. This is shown in Table 2 in the comparison between row A and row B. See supplementary material for a visualization.

Our spatially aggregated features F_S are passed as input to the Q-Former block [24], which also takes as input a set of trainable queries and learns to project the features into the LLM embedding space.

4.2. Asset encoding

When compared with other tasks, our language-guided placement task has an additional input, the 3D asset point cloud. We encode the asset using a pre-trained Point-BERT encoder [41] trained on the Objaverse [11] dataset. This encoder predicts a sequence of feature vectors that are max-pooled to obtain a single feature embedding.

Encoding the scale of the input asset is essential to facilitate a valid placement. Since the asset encoder assumes a normalized point cloud in a unit sphere, we separately encode the size of the asset by taking the asset's dimensions in the x, y, and z axes. The F_A feature for the asset is a combination of the asset encoding and scale embeddings and is projected to the LLM space using an MLP.

4.3. Placement decoder

We instruct our LLM to output three special tokens, namely a [LOC] token, an [ANC] token, and a [ROT] token. The features associated with the three special tokens are passed as input to the decoder, where they undergo a few self-attention layers. These are followed by a few cross-attention layers between the updated token features and the asset features F_A and the pooled scene features F_S.

Each individual head takes the feature of the associated token after attention, the asset feature F_A, and the scene feature F_S and predicts the corresponding output. The **Placement Mask Head** takes the [LOC] token embedding and predicts $\mathcal{M}_{loc} \in [0,1]^N$, a mask over the scene point cloud encoding the regions where the input asset can be placed satisfying the input prompt. The **Rotation Head** takes the [ROT] token embedding and predicts $\mathcal{M}_{rot} \in [0,1]^{N \times 8}$ indicating for each point in the point cloud, the validity of a discretized set of rotation angles. Finally, the **Anchors Mask Head** takes the [ANC] token and predicts $\mathcal{M}_{anc} \in [0,1]^N$, a mask encompassing the masks of all the anchor objects. This is used only as an auxiliary task, to help the network identifying anchors in the prompt.

4.4. Losses

We use a combination of Binary Cross Entropy (BCE) and Dice [34] losses when comparing a ground truth mask $\bar{\mathcal{M}}$ with a predicted mask $\mathcal{M}$, so

$$\mathcal{L}_{seg}(\bar{\mathcal{M}}, \mathcal{M}) = \text{BCE}(\bar{\mathcal{M}}, \mathcal{M}) + \text{Dice}(\bar{\mathcal{M}}, \mathcal{M}). \quad (2)$$

The loss for the rotation prediction is given by

$$\mathcal{L}_{rot} = \text{BCE}(\bar{\mathcal{M}}_{rot}, \mathcal{M}_{rot}), \quad (3)$$

where $\bar{\mathcal{M}}_{rot} \in \mathcal{M}_{rot} \in \{0,1\}^{N \times 8}$ is the ground truth indicator mask for valid rotation angles, per point in the scene point cloud.

The loss for the LLM is a cross-entropy loss, comparing the ground truth text $\bar{Y}$ with the predicted text Y: $\mathcal{L}_L = \text{CE}(\bar{Y}, Y)$. Note that the ground truth text $\bar{Y}$ for our task, follows a simple format, *e.g.* "Sure, it is [LOC][ANC][ROT]", since the LLM is not required to predict articulated responses or explain placement decisions.

Instead, the information useful for placement should be encoded in the embeddings for the special tokens. Finally, our total loss is defined as

$$\mathcal{L} = \mathcal{L}_{seg}(\bar{\mathcal{M}}_{loc}, \mathcal{M}_{loc}) + \mathcal{L}_{rot} + \mathcal{L}_{seg}(\bar{\mathcal{M}}_{anc}, \mathcal{M}_{anc}) + \mathcal{L}_{L}. \quad (4)$$

4.5. Inference

At inference time, our method takes the network predictions for placement, $\mathcal{M}_{loc}$, and rotation, $\mathcal{M}_{rot}$, and extracts a single valid placement by finding the point in the point cloud with the maximum value in $\mathcal{M}_{loc}$: $\hat{\mathbf{x}} = \text{argmax}_{m \in \mathcal{M}_{loc}} m$. We apply a fixed offset to point $\hat{\mathbf{x}}$, half the asset height, to get the predicted 3D translation vector $\hat{t}$. This is due to the differences in parametrization between the training dataset and the benchmark. To predict the rotation angle, we use $\mathcal{M}_{rot}^{\hat{\mathbf{x}}} \in [0, 1]^8$, which encodes the validity of discretized rotations for $\hat{\mathbf{x}}$. The predicted angle $\hat{\alpha}$ is obtained by taking the argmax over this vector.

5. Experiments

We validate our method PlaceWizard for the task of language-guided object placement. Naturally, we use the benchmark described in Section 3.2.

Implementation details for our method are provided in the supplemental material. Our **metrics**, discussed in detail in Section 3.2.2, measure the validity of the predictions. All values represent percentages, and higher is better on all metrics.

5.1. Quantitative results

In the absence of a readily available language-guided placement method, we implemented a baseline system by combining an open-world grounding method, OpenMask3D [36], with our rule-based system for asset placement, and we compare it against our PlaceWizard on the benchmark. Since OpenMask3D requires object queries, we use ground truth anchor descriptions instead of the full placement instructions. Ground truth masks are used to locate the floor (as OpenMask3D rarely predicts floor masks), while other anchors are selected based on the highest similarity score. Finally, our rule-based system places assets using the detected anchor masks. Table 2 shows both comparisons to baselines and ablation results. PlaceWizard outperforms OpenMask3D+rules across all metrics. Additionally, PlaceWizard's end-to-end approach removes the need for a rule-based system, which can be computationally expensive for larger scenes. For instance, generating our training dataset required approximately 10,000 single-CPU hours.

5.1.1. Ablations

Table 2 also shows results for different ablations of our method. We start with an adaptation of the Reason3D [17] model to our task. One by one, we incrementally modify it using our novel components. Each row in the table introduces a single new modification, as compared to the previous row. We evaluate and report the model's performance until we reach PlaceWizard, our final method. All models are trained on our training dataset PlaceIt3D. For the methods that do not predict rotation (rows A, B, C, D, and E) we set the predicted rotation angle to 0. We describe the different variants below.

A. The asset dimensions are encoded in text and provided as part of the prompt: "*The asset dimensions are X Y Z cm*", where X, Y, and Z are integer values in cm.

B. This variant uses our proposed uniform spatial pooling approach instead of the original superpoints pooling.

C. Positional embedding features F_X^{pos} for points in the point cloud are added to the scene encoding.

D. We incorporate the asset encoder instead of only providing the asset dimensions in the text prompt to the LLM.

E. We introduce the anchor prediction auxiliary loss. In [1], predicting anchor objects leads to better 3D visual grounding. We find that this holds for our task as well.

F. The rotation prediction head is introduced, allowing the model to predict not only the placements mask $\mathcal{M}_{loc}$ but also $\mathcal{M}_{rot}$.

G (Ours). This variant constitutes our final PlaceWizard method. The asset feature F_A is added as an extra input to the placement decoder. We expect this integration to enable the placement decoder to perform more effective reasoning about the asset's geometry relative to the scene geometry.

The results in Table 2 validate our design choices. Using spatial aggregation instead of superpoints improves over all metrics (compare row B with row A). The inclusion of the anchor prediction head as an auxiliary sub-task also improves performance (row E vs row D). Finally, the use of our rotation head combined with passing the asset encoding as input to the decoder gives our final best-performing method (row G, which we use in the qualitative results).

5.2. Qualitative Results

In Figure 4, we show the results of our method PlaceWizard on benchmark examples, demonstrating its ability to follow language instructions and satisfy constraints. While most placements are accurate, some cases exhibit minor intersections with the scene mesh or constraint failures. We show additional results in the supplementary material.

6. Limitations and Future Work

The formulation of our novel task currently has some limitations. First, we only consider placement of objects on horizontal surfaces. Further generalization would allow to define arbitrary contact points, unlocking new scenarios, *e.g.* hanging a clock on a vertical wall. Second, our dataset and method do not address the issue of "incorrect guidance",

Method ablation							Subgroup metrics		Global metrics	
Name	Spatial aggregation	Pos. emb.	Asset encoder	Anchor pred.	Rot pred.	Decode asset	Spatial	Rotational	Global constraint accuracy	Complete placement success
Baseline — OpenMask3D [36] + rules							28.6	6.5	25.5	21.8
A [17]	Superpoints	–	text	–	–	–	37.5	6.6	40.6	18.1
B	uniform	–	text	–	–	–	45.6	7.0	46.8	22.5
C	uniform	✓	text	–	–	–	45.8	5.7	46.7	22.9
D	uniform	✓	PointBert	–	–	–	46.2	7.7	47.5	22.6
E	uniform	✓	PointBert	✓	–	–	53.6	7.3	52.0	27.8
F	uniform	✓	PointBert	✓	✓	–	50.8	9.5	50.3	26.7
G **(ours)**	uniform	✓	PointBert	✓	✓	✓	**54.1**	**12.1**	**52.6**	**29.4**

Table 2. **Quantitative results:** We compare our full method with variations where some components are removed. The results validate our design choices, and they show improvements over OpenMask3D [36] with rule-based asset placement and Reason3D [17].

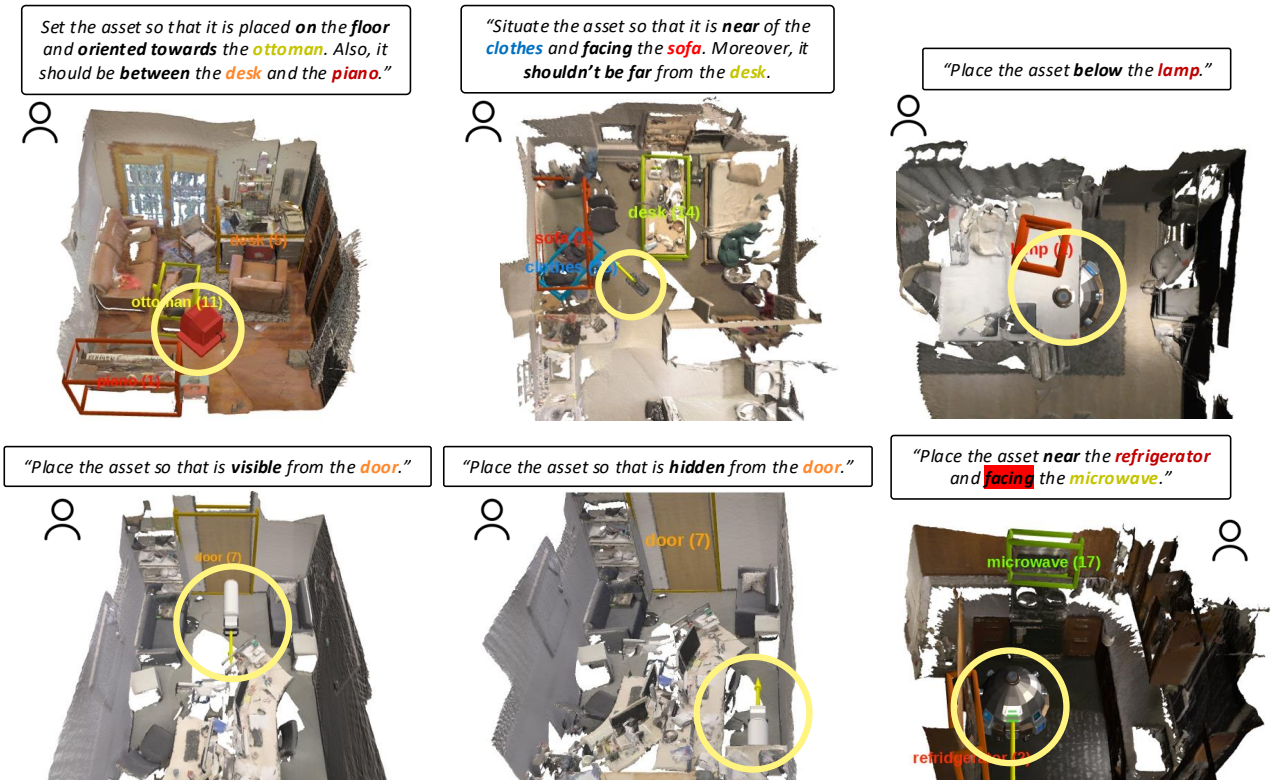

Figure 4. **Qualitative benchmark results**. Colored highlights indicate anchors referenced in the textual prompts (predictions are generated entirely from point clouds, with anchor information provided only as text). The asset position is marked with a yellow circle, and a yellow arrow denotes the frontal orientation. Our method successfully follows language instructions and meets the specified constraints. The top-right example illustrates a placement that satisfies constraints but slightly intersects with the scene mesh. The bottom-right example demonstrates a failure case where one constraint is not met (highlighted in red).

i.e. what to do when the language guidance is inconsistent with the scene. Despite these limitations, we believe that our proposed dataset, benchmark and method will enable further investigations in the area.

Our method can be considered a specialist model since we only train and evaluate it on our task of guided placement. We leave as future work the combination of our task with a generalist model, so that the same model can be used for other 3D understanding tasks, different from placement.

7. Conclusion

We introduced the novel task of language-guided object placement in real 3D scenes, bridging natural language understanding with 3D spatial reasoning about *both* scenes and assets. To facilitate research, we created both a benchmark and a large-scale dataset, explicitly designed to account for the inherent ambiguities of placement tasks, where multiple valid solutions may exist. Additionally, we proposed a proto-method for guided placement, building on advances in 3D large language models (LLMs). Ablations validated that key design choices have impact on the task. We hope PlaceIt3D will make it easier to develop AIs with human-like abilities to follow two-step instructions.

References

[1] Ahmed Abdelreheem, Kyle Olszewski, Hsin-Ying Lee, Peter Wonka, and Panos Achlioptas. Scanents3d: Exploiting phrase-to-3d-object correspondences for improved visio-linguistic models in 3d scenes. In *WACV*, 2024. 2, 7

[2] Panos Achlioptas, Ahmed Abdelreheem, Fei Xia, Mohamed Elhoseiny, and Leonidas Guibas. Referit3d: Neural listeners for fine-grained 3d object identification in real-world scenes. In *ECCV*, 2020. 2

[3] Daichi Azuma, Taiki Miyanishi, Shuhei Kurita, and Motoaki Kawanabe. Scanqa: 3d question answering for spatial scene understanding. In *CVPR*, 2022. 2

[4] Wenxiao Cai, Yaroslav Ponomarenko, Jianhao Yuan, Xiaoqi Li, Wankou Yang, Hao Dong, and Bo Zhao. Spatialbot: Precise spatial understanding with vision language models. *arXiv 2406.13642*, 2024. 2

[5] Boyuan Chen, Zhuo Xu, Sean Kirmani, Brain Ichter, Dorsa Sadigh, Leonidas Guibas, and Fei Xia. SpatialVLM: Endowing vision-language models with spatial reasoning capabilities. In *CVPR*, 2024. 2

[6] Dave Zhenyu Chen, Angel X Chang, and Matthias Nießner. Scanrefer: 3d object localization in rgb-d scans using natural language. 2020. 2

[7] Zhenyu Chen, Ali Gholami, Matthias Nießner, and Angel X Chang. Scan2cap: Context-aware dense captioning in rgb-d scans. In *CVPR*, 2021. 2

[8] An-Chieh Cheng, Hongxu Yin, Yang Fu, Qiushan Guo, Ruihan Yang, Jan Kautz, Xiaolong Wang, and Sifei Liu. SpatialRGPT: Grounded spatial reasoning in vision language model. In *NeurIPS*, 2024. 2

[9] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4D spatio-temporal convnets: Minkowski convolutional neural networks. In *CVPR*, 2019. 2

[10] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *CVPR*, 2017. 2, 4

[11] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *CVPR*, 2022. 6

[12] Meng Han, Liang Wang, Limin Xiao, Hao Zhang, Chenhao Zhang, Xiangrong Xu, and Jianfeng Zhu. Quickfps: Architecture and algorithm co-design for farthest point sampling in large-scale point clouds. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023. 6

[13] Yining Hong, Haoyu Zhen, Peihao Chen, Shuhong Zheng, Yilun Du, Zhenfang Chen, and Chuang Gan. 3D-LLM: Injecting the 3D world into large language models. In *NeurIPS*, 2023. 2

[14] Haifeng Huang, Zehan Wang, Rongjie Huang, Luping Liu, Xize Cheng, Yang Zhao, Tao Jin, and Zhou Zhao. Chat-scene: Bridging 3d scene and large language models with object identifiers. In *NeurIPS*, 2023. 2

[15] Ian Huang, Yanan Bao, Karen Truong, Howard Zhou, Cordelia Schmid, Leonidas Guibas, and Alireza Fathi. FirePlace: Geometric Refinements of LLM Common Sense Reasoning for 3D Object Placement. In *CVPR*, 2025. 3

[16] Jingwei Huang, Haotian Zhang, Li Yi, Thomas Funkhouser, Matthias Nießner, and Leonidas J Guibas. Texturenet: Consistent local parametrizations for learning from high-resolution signals on meshes. In *CVPR*, 2019. 2

[17] Kuan-Chih Huang, Xiangtai Li, Lu Qi, Shuicheng Yan, and Ming-Hsuan Yang. Reason3D: Searching and reasoning 3d segmentation via large language model. In *3DV*, 2025. 2, 5, 7, 8, 12, 13

[18] Baoxiong Jia, Yixin Chen, Huangyue Yu, Yan Wang, Xuesong Niu, Tengyu Liu, Qing Li, and Siyuan Huang. Sceneverse: Scaling 3d vision-language learning for grounded scene understanding. In *ECCV*, 2024. 2

[19] Xin Lai, Zhuotao Tian, Yukang Chen, Yanwei Li, Yuhui Yuan, Shu Liu, and Jiaya Jia. Lisa: Reasoning segmentation via large language model. In *CVPR*, 2024. 2

[20] Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *CVPR*, 2018. 5, 6, 13

[21] Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Yanwei Li, Ziwei Liu, and Chunyuan Li. Llava-onevision: Easy visual task transfer. *arXiv 2408.03326*, 2024. 2

[22] Changyang Li, Wanwan Li, Haikun Huang, and Lap-Fai Yu. Interactive augmented reality storytelling guided by scene semantics. In *SIGGRAPH*, 2022. 2

[23] Feng Li, Renrui Zhang, Hao Zhang, Yuanhan Zhang, Bo Li, Wei Li, Zejun Ma, and Chunyuan Li. Llava-next-interleave: Tackling multi-image, video, and 3d in large multimodal models. *arXiv 2407.07895*, 2024. 2

[24] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *ICML*, 2023. 2, 5, 6

[25] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *NeurIPS*, 2024. 2

[26] Liu Liu, Zhenchen Liu, Bo Zhang, Jiangtong Li, Li Niu, Qingyang Liu, and Liqing Zhang. Opa: object placement assessment dataset. *arXiv 2107.01889*, 2021. 2

[27] Tiange Luo, Chris Rockwell, Honglak Lee, and Justin Johnson. Scalable 3d captioning with pretrained models. In *NeurIPS*, 2023. 2

[28] Muzammal Naseer, Salman Khan, and Fatih Porikli. Indoor scene understanding in 2.5/3d for autonomous agents: A survey. *IEEE access*, 2018. 2

[29] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*, 2017. 2

[30] Ram Ramrakhya, Aniruddha Kembhavi, Dhruv Batra, Zsolt Kira, Kuo-Hao Zeng, and Luca Weihs. Seeing the unseen: Visual common sense for semantic placement. In *CVPR*, 2024. 2

[31] Jonas Schult, Francis Engelmann, Alexander Hermans, Or Litany, Siyu Tang, and Bastian Leibe. Mask3D: Mask transformer for 3d semantic instance segmentation. In *ICRA*, 2023. 2

[32] Aditya Sharma, Luke Yoffe, and Tobias Höllerer. Octo+: A suite for automatic open-vocabulary object placement in mixed reality. In *AIXVR*, 2024. 2

[33] R. Siegler, J. Saffran, E. Gershoff, N. Eisenberg, and J. DeLoache. *How Children Develop*. Macmillan Learning, 2020. 1

[34] Carole Helene Sudre, Wenqi Li, Tom Kamiel Magda Vercauteren, Sébastien Ourselin, and M. Jorge Cardoso. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. In *MICCAI workshop*, 2017. 6

[35] Jiahao Sun, Chunmei Qing, Junpeng Tan, and Xiangmin Xu. Superpoint transformer for 3d scene instance segmentation. In *AAAI*, 2023. 5, 6

[36] Ayça Takmaz, Elisabetta Fedele, Robert W. Sumner, Marc Pollefeys, Federico Tombari, and Francis Engelmann. OpenMask3D: Open-Vocabulary 3D Instance Segmentation. In *NeurIPS*, 2023. 7, 8

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 6

[38] Tai Wang, Xiaohan Mao, Chenming Zhu, Runsen Xu, Ruiyuan Lyu, Peisen Li, Xiao Chen, Wenwei Zhang, Kai Chen, Tianfan Xue, Xihui Liu, Cewu Lu, Dahua Lin, and Jiangmiao Pang. Embodiedscan: A holistic multi-modal 3d perception suite towards embodied ai. In *CVPR*, 2024. 2

[39] Jianing Yang, Xuweiyi Chen, Nikhil Madaan, Madhavan Iyengar, Shengyi Qian, David F. Fouhey, and Joyce Chai. 3d-grand: A million-scale dataset for 3d-llms with better grounding and less hallucination. *arXiv 2406.05132*, 2024. 2

[40] Yunhan Yang, Yukun Huang, Yuan-Chen Guo, Liangjun Lu, Xiaoyang Wu, Lam Edmund Y., Yan-Pei Cao, and Xihui Liu. Sampart3d: Segment any part in 3d objects. *arXiv 2411.07184*, 2024. 2, 4

[41] Xumin Yu, Lulu Tang, Yongming Rao, Tiejun Huang, Jie Zhou, and Jiwen Lu. Point-bert: Pre-training 3d point cloud transformers with masked point modeling. In *CVPR*, 2022. 6

[42] Wentao Yuan, Jiafei Duan, Valts Blukis, Wilbert Pumacay, Ranjay Krishna, Adithyavairavan Murali, Arsalan Mousavian, and Dieter Fox. Robopoint: A vision-language model for spatial affordance prediction for robotics. In *CoRL*, 2024. 2

[43] Haochen Zhang, Nader Zantout, Pujith Kachana, Zongyuan Wu, Ji Zhang, and Wenshan Wang. Vla-3d: A dataset for 3d semantic scene understanding and navigation. In *RSS Workshop*, 2024. 2

[44] Hengshuang Zhao, Xiaohui Shen, Zhe Lin, Kalyan Sunkavalli, Brian Price, and Jiaya Jia. Compositing-aware image search. In *ECCV*, 2018. 2

[45] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv 1801.09847*, 2018. 4

[46] Chenming Zhu, Tai Wang, Kai Chen, and Xihui Liu. Scanreason: Empowering 3d visual grounding with reasoning capabilities. In *ECCV*, 2024. 2

[47] Sijie Zhu, Zhe Lin, Scott Cohen, Jason Kuen, Zhifei Zhang, and Chen Chen. Topnet: Transformer-based object placement network for image compositing. In *CVPR*, 2023. 2

PlaceIt3D: Language-Guided Object Placement in Real 3D Scenes

Supplementary Material

8. Additional details on the training dataset

8.1. Training dataset creation

We give some details on the training set creation, particularly how the physically plausible constraint and visibility constraint are computed.

Spatial constraints Each constraint uses geometric criteria on 3D oriented bounding boxes and is governed by the following parameters. We use the same values both in the training dataset and the evaluation benchmark:

- **"near"**: asset in a proximity of the anchor object. The threshold distance is proportional to the size of the room (1%).
- **"adjacent"**: asset close to the anchor object. We set a tolerance distance of 3 cm.
- **"above" / "below"**: asset vertically aligned above / below the anchor object. Vertical Intersection over Minimum (IoM) $>=$ 0.5 and a minimum of 1 cm above/below the anchor.
- **"on"**: resting on top of the anchor object, considering vertical stacking and size constraints: vertical IoM $>=$ 0.5 and a tolerance for vertical distance of 1 cm.
- **"between"**: Determines if the asset object lies between two anchor objects in both xy and z planes. Parameters: between IoM (0.5) in projection planes (xy and z). Overlap threshold (0.3): maximum IoM that ensures the asset does not overlap excessively with either object. Distance threshold: filters anchors beyond 1.5 m

Rotational constraint. The "facing" constraint determines which objects an asset is oriented towards in a 3D scene by evaluating directional alignment, proximity, and spatial overlap. It uses the asset's front direction to identify candidate objects within its field of view. We use a maximum distance threshold of 2 meters, an angular threshold of 30 degrees and an IoM for lateral overlap of 0.5.

Physically plausible constraint The first constraint that we consider is whether an object can physically be placed at a particular location in a scene. To compute valid placements in a scene efficiently, we make use of a heightmap based representation, where we raycast the mesh from above using a grid of rays with a predefined resolution, and store all points of intersection. Next, we create a set of heightmaps, where each cell represents a different ray, each layer represents a different intersection per ray, and the value is the height of the intersection point. We construct the first heightmap using the intersection points with the minimum height per ray. Each subsequent heightmap will contain either the next intersection point for each ray, or if there are no remaining points for a cell, will contain the maximum intersection point. Additionally, we raycast each asset from above to obtain an asset heightmap and 2D bounding box per possible rotation. Given these heightmaps, we check for physical plausibility by:

- Extracting overlapping patches of the mesh heightmap, with patch size equal to the asset bounding box
- Extract minimum height and maximum height of the heightmap for each patch, using the asset heightmap to generate a mask. If this differs by more than 10cm, this point is not valid
- Check that the asset can also fit in the Z direction using the next heightmap - is the asset height for this cell less than the height of the next surface

If these 2 conditions are true, we deem a location to be physically plausible. Finally, we generate labels for the mesh vertices by assigning them the labels of their nearest location in the heightmap.

Visibility constraint Our visibility constraint determines whether an asset is visible from a specific location in the scene, which corresponds to one of the object anchors. To evaluate this, we use mesh rendering. To assess visibility efficiently, we first place the asset in a physically feasible position. Instead of rendering the full asset mesh, we approximate it using a simple cuboid with the same dimensions as the asset bounding box, reducing computational overhead, we also consider only 1 rotation for the asset. The virtual camera's position is then determined by computing the centroid of the vertices associated with the anchor. The camera center is set to the vertex within the anchor that is closest to this centroid, and the camera is oriented to face the asset.

We then render the scene and check whether any pixels from the asset's bounding cuboid appear in the rendered image. This process is repeated for all valid asset placements across all scenes. The virtual camera locations correspond to TVs, doors, and windows. When multiple instances of the same object class exist in a scene, we select the largest instance.

We use a virtual camera with a field of view (FOV) of 60° and we render images at a resolution of 64×64 pixels.

In the benchmark, we follow the same procedure as stated above for generating the training data with 2 differences: we render the original asset mesh instead of the

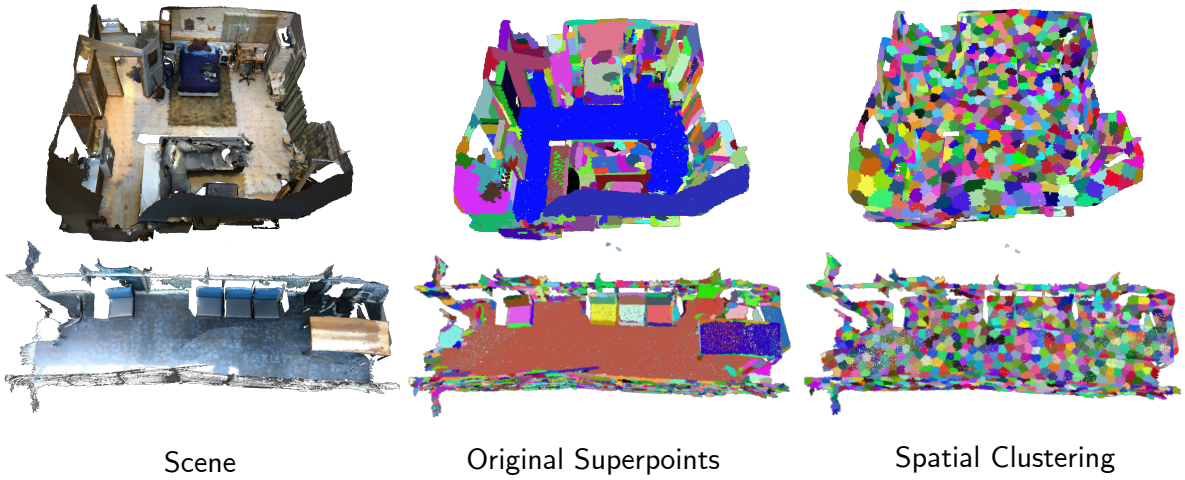

Figure 5. Comparison of our spatial pooling vs the superpoints used in [17]. Our regions are more local and more adequate to the task of object placement.

cuboid and render images at a resolution of 256×256 pixels.

8.2. Templates for prompts

We report the templates used to generate placement instructions.

```
relationships:
  - name: plausible
    templates:
    - in a plausible location
    - in a sensible location
    - in a reasonable spot
    - in a suitable position
    - in a feasible area
    - somewhere stable within the scene
    - at a steady spot in the scene
    - in a secure location within the scene
    - in a firm position in the scene
    - in an area that suits the scene's layout
  - name: adjacent
    templates:
      - adjacent to the {anchor_class}
      - next to the {anchor_class}
      - beside the {anchor_class}
      - right beside the {anchor_class}
      - alongside the {anchor_class}
      - abutting the {anchor_class}
  - name: between
    templates:
      - between the {anchor1_class} and the
          {anchor2_class}
      - in the space between the {anchor1_class}
          and the {anchor2_class}
      - positioned between the {anchor1_class} and
          the {anchor2_class}
      - in the middle of the {anchor1_class} and
          the {anchor2_class}
  - name: facing
    templates:
      - facing the {anchor_class}
      - directed at the {anchor_class}
      - pointing towards the {anchor_class}
      - oriented towards the {anchor_class}
      - looking at the {anchor_class}
      - angled toward the {anchor_class}
      - turned towards the {anchor_class}
  - name: near
    templates:
      - near the {anchor_class}
      - close to the {anchor_class}
      - in the vicinity of the {anchor_class}
      - not far from the {anchor_class}
      - within reach of the {anchor_class}
      - a short distance from the {anchor_class}
  - name: on
    templates:
      - on the {anchor_class}
      - resting on the {anchor_class}
      - placed on the {anchor_class}
      - sitting on the {anchor_class}
      - lying on the {anchor_class}
  - name: above
    templates:
      - above the {anchor_class}
      - over the {anchor_class}
      - higher than the {anchor_class}
      - up above the {anchor_class}
  - name: below
    templates:
      - below the {anchor_class}
      - under the {anchor_class}
      - beneath the {anchor_class}
      - underneath the {anchor_class}
      - lower than the {anchor_class}
      - situated under the {anchor_class}
      - right below the {anchor_class}
  - name: is_visible
    templates:
      - visible from the {anchor_class}
      - in view of the {anchor_class}
      - within sight of the {anchor_class}
      - seen from the {anchor_class}
      - not obstructing the view to the
          {anchor_class}
```

```
    - keeping the view to the {anchor_class}clear
    - positioned to avoid blocking the
        {anchor_class}
    - allowing an unobstructed view of the
        {anchor_class}
- name: not_visible
  templates:
    - not visible from the {anchor_class}
    - out of sight of the {anchor_class}
    - hidden from the {anchor_class}
    - obstructing the view to the {anchor_class}
    - blocking the view to the {anchor_class}
    - in the way of the {anchor_class}
    - preventing a clear view of the
        {anchor_class}
```

Dataset Examples In Figures 7 and 8, we provide examples from our proposed dataset.

9. PlaceWizard Implementation Details

We conduct all our experiments using eight NVIDIA Tesla A100 GPUs, with a training batch size of 28 per single gpu. Following Reason3D, we utilize the AdamW optimizer with parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$, a weight decay of 0.05, and a linear warm-up strategy for the learning rate during the initial 1000 steps gradually increasing it from 10^{-8} to 10^{-4} followed by a cosine decay schedule. We train for 50 epochs. We also use a pretrained FlanT5XL model, keeping most of its pre-trained weights frozen, except for adapting the weights of the newly added tokens, as similarly done in Reason3D. For spatial pooling, we employ 1024 groups for each ScanNet scan.

10. Visualization of superpoints

Figure 5 shows the difference between the superpoints [20] used in Reason3D [17] and our proposed spatial pooling. While [20] generates large clusters, such as for the floor, our method produces clusters at a finer granularity.

11. Further Qualitative Results

In Figure 6 we show the confidence scores predicted by our model for the spatial clusters in two example scenes from our dataset.

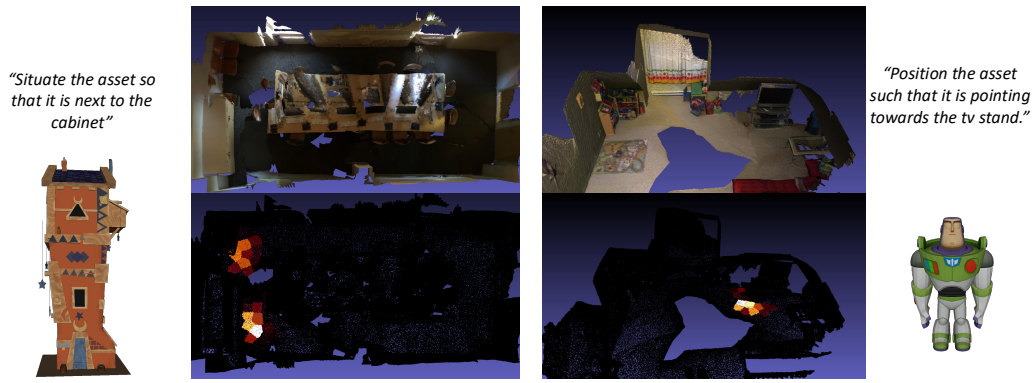

Figure 6. Heatmap visualization of the predicted confidence scores by our model for spatial clusters in two examples from our dataset, across two scenes, given different assets and textual prompts. Warmer colors indicate higher confidence regions for asset placement, with white representing the highest confidence.

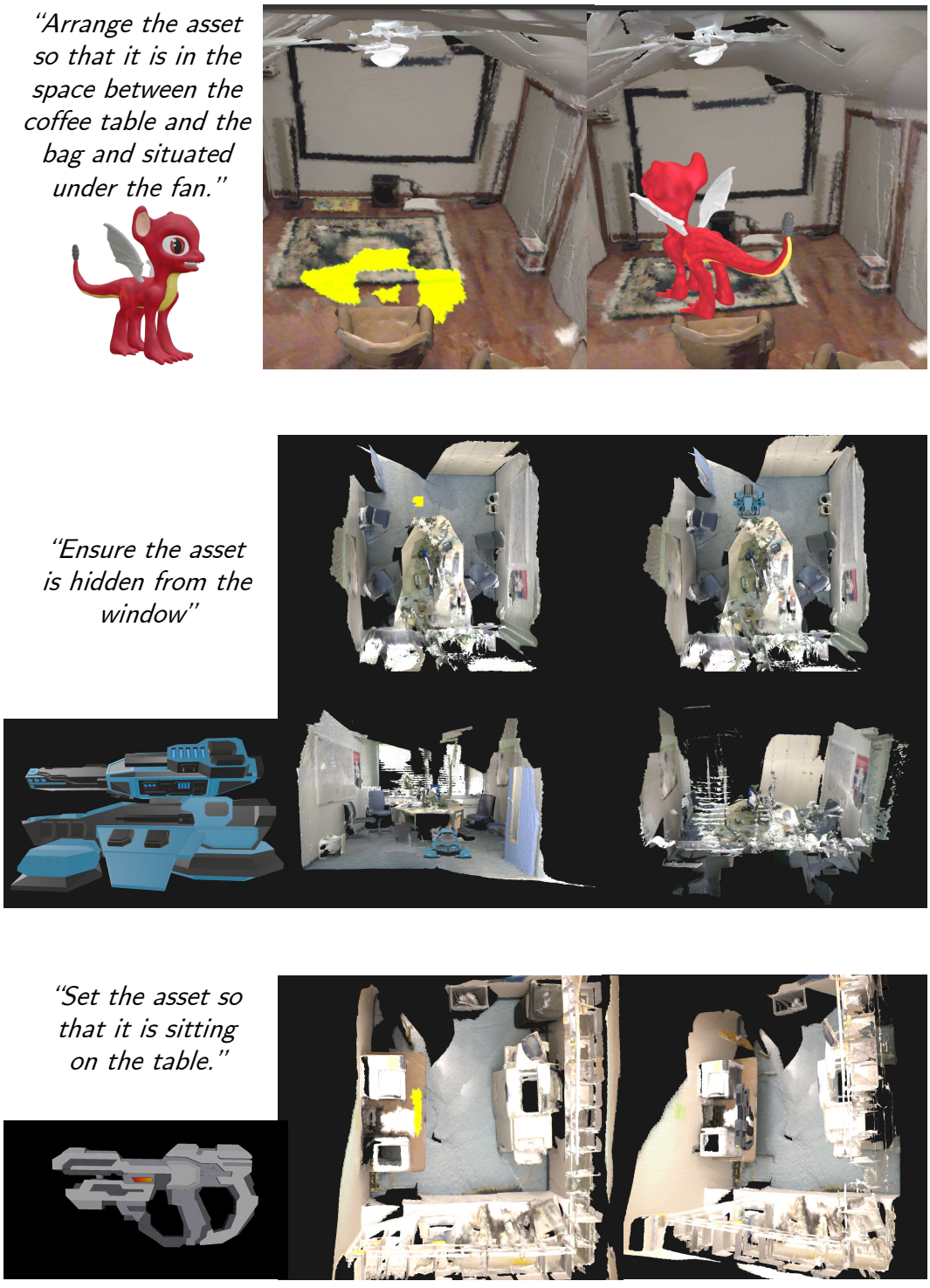

Figure 7. Examples from our proposed dataset illustrating prompts with different constraints, along with the corresponding placement mask and a sample placed asset.

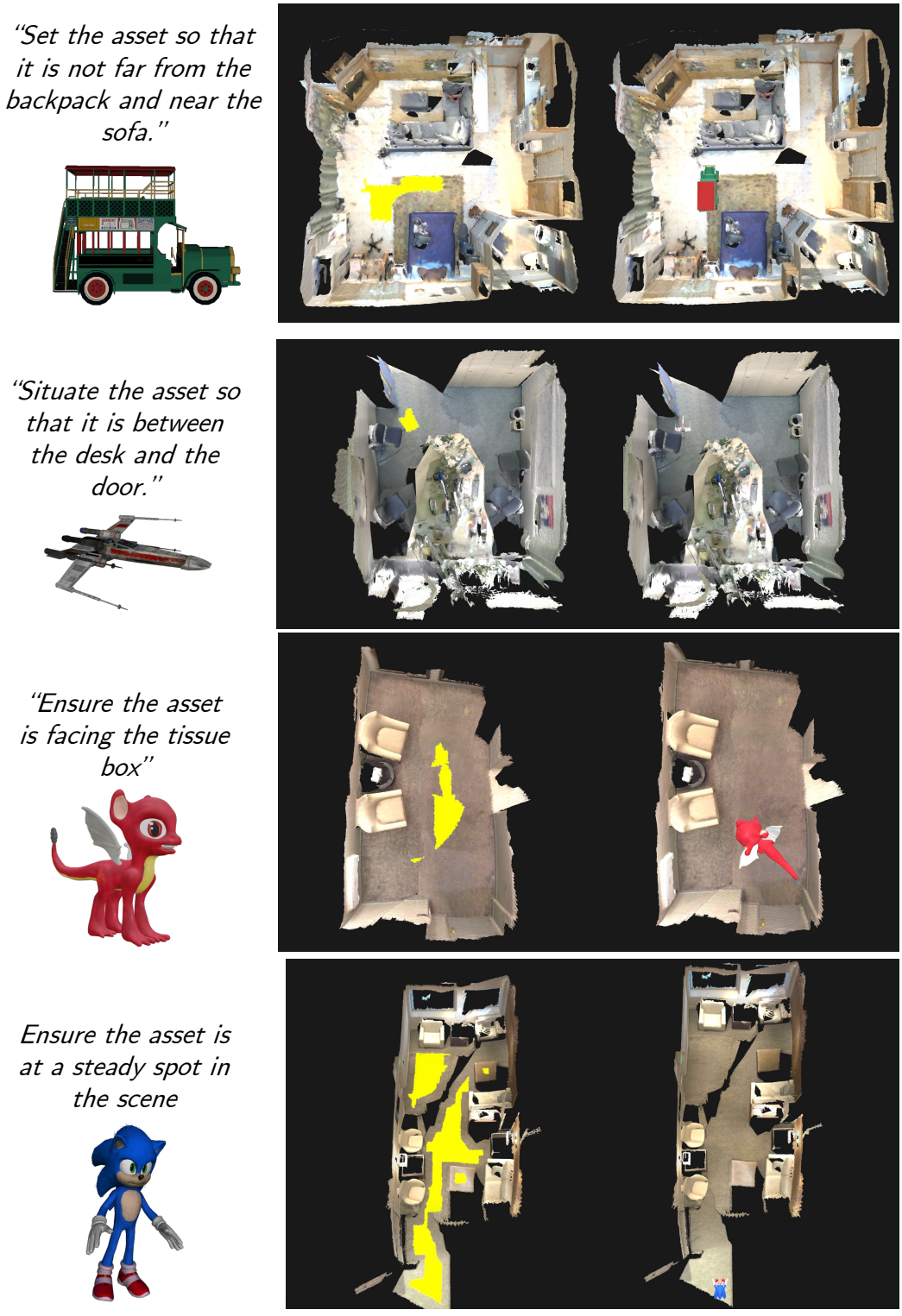

Figure 8. Examples from our proposed dataset illustrating prompts with different constraints, along with the corresponding placement mask and a sample placed asset.