NS3D: Neuro-Symbolic Grounding of 3D Objects and Relations

Joy Hsu  
Stanford University  
joycj@stanford.edu

Jiayuan Mao  
Massachusetts Institute of Technology  
jiayuanm@mit.edu

Jiajun Wu  
Stanford University  
jiajunwu@cs.stanford.edu

Abstract

Grounding object properties and relations in 3D scenes is a prerequisite for a wide range of artificial intelligence tasks, such as visually grounded dialogues and embodied manipulation. However, the variability of the 3D domain induces two fundamental challenges: 1) the expense of labeling and 2) the complexity of 3D grounded language. Hence, essential desiderata for models are to be data-efficient, generalize to different data distributions and tasks with unseen semantic forms, as well as ground complex language semantics (e.g., view-point anchoring and multi-object reference). To address these challenges, we propose NS3D, a neuro-symbolic framework for 3D grounding. NS3D translates language into programs with hierarchical structures by leveraging large language-to-code models. Different functional modules in the programs are implemented as neural networks. Notably, NS3D extends prior neuro-symbolic visual reasoning methods by introducing functional modules that effectively reason about high-arity relations (i.e., relations among more than two objects), key in disambiguating objects in complex 3D scenes. Modular and compositional architecture enables NS3D to achieve state-of-the-art results on the ReferIt3D view-dependence task, a 3D referring expression comprehension benchmark. Importantly, NS3D shows significantly improved performance on settings of data-efficiency and generalization, and demonstrate zero-shot transfer to an unseen 3D question-answering task.

1. Introduction

Interacting with the physical world requires 3D visual understanding; it entails the ability to interpret 3D objects and relations among multiple entities, as well as reason about 3D instances in a scene from language expressions. However, due to the variability of the 3D domain, there are two prevalent challenges: the expense of annotating 3D labels and the complexity of 3D grounded language. In this paper, we tackle these two challenges on a specific task of 3D scene understanding, the referring expression comprehension (3D-REC) task. As shown in Figure 1, in a 3D-REC task, the input contains a sentence and a 3D scene, usually given as a collection of object point clouds; the goal is to identify the correct referred object in the scene. The task is challenging: obtaining high-quality annotations for such tasks is expensive; the referring expressions often require reasoning about multiple objects, such as anchoring speaker viewpoints (i.e., facing X, select the object Y behind Z) and utilizing multiple objects in the scene as reference points.

Many prior works have studied end-to-end methods to tackle this problem [1, 2, 16–18, 20, 30, 37, 39, 40], jointly attending over features from language and point clouds. These methods report strong performance, but generally require large amounts of data to train and are prone to dataset biases, such as object co-occurrences. Meanwhile, the learned 3D representations cannot be directly transferred to related downstream tasks, such as 3D question answering. In addition, most prior works in 3D grounding are based on Transformers [33], which reduce the set of realizable functions to a subset of reasoning tasks with binary relations [26, 33]. This has limited their ability to resolve complex 3D grounded languages, empirically leading to a noticeable performance

Figure 1. NS3D achieves grounding of 3D objects and relations in complex scenes, while showing state-of-the-art results in data efficiency, generalization, and zero-shot transfer.
To this end, we propose NS3D as a powerful neuro-symbolic approach to solve 3D visual reasoning tasks, with more faithful grounding of 3D objects and relations. NS3D first parses the referring expression from the free language form to a neuro-symbolic program form. We introduce the use of Codex [11], a large language-to-code model, for semantic parsing with a small number of prompting examples, leading to perfect identification of entities and program structures. Such program structures decompose each referring expression into a set of functional modules that are hierarchically chained together. Functional modules can perform an object-level grounding step, such as selecting the *bathroom vanity* from the input point clouds, and a relational grounding step, such as finding objects that are *behind* another reference object. This functional composition strategy can be easily extended to more complex functions that require multiple objects, such as view-dependent relation grounding. In NS3D, functional modules are implemented as different neural networks that take object features of the corresponding arity: e.g., object-level grounding modules take per-object features, while relation grounding modules take a set of vector encodings for each pair of objects. Importantly, NS3D extends prior neuro-symbolic approaches for visual reasoning [24] by introducing modules that execute high-arity programs, such as those for relation grounding over multiple objects, especially ubiquitous in the 3D domain.

The combination of compositional structures and modular neural networks fulfills many desiderata for 3D visual reasoning (see Figure 1). First, specializing neural modules for relations that involve multiple objects improves performance, particularly in resolving complex view-dependent referring expressions. Our approach is noticeably simpler and more effective than existing models that solve this task by fusing multiple view representations [18]. Second, the disentangled grounding of objects and relations brings significantly improved data efficiency. Third, by following symbolic structures to compose functional modules, NS3D generalizes better to scenarios with unseen object co-occurrences and scene types. Fourth, the compositional nature of the functional structures and the flexibility of our Codex-based parser enables NS3D to zero-shot generalize to novel reasoning tasks, such as 3D visual question answering (3D-QA). Furthermore, as a byproduct of our modular approach, NS3D enables better interpretability, allowing attribution to where visual grounding fails and succeeds; we show in ablations that NS3D learns almost perfect relation grounding.

We validate NS3D on the ReferIt3D benchmark, which evaluates referring expression comprehension in 3D scenes, and requires fine-grained object-centric and multi-object relation grounding [2]. We report state-of-the-art view-dependent accuracy and comparable overall accuracy to top-performing methods. We also present results on data efficiency and generalization to unseen object co-occurrences and new scenes, with our neuro-symbolic method outperforming all prior work by a large margin. Finally, we show NS3D’s ability to zero-shot transfer from the 3D reference task to a new 3D visual question answering task, achieving strong performance without any data in this novel setup.

To summarize, the contribution of this paper is three-fold: 1) We propose a neuro-symbolic method to ground 3D objects and relations that integrates the power of large language-to-code models and modular neural networks. 2) We introduce a neural program executor that reasons about high-arity relations as a principled solution to view-point anchoring and multi-object reference. 3) We show state-of-the-art view-dependent grounding results in 3D-REC tasks, high accuracy in data-efficient settings (a 24.5 percent point gain from prior work with 1.5% of data), significant improvements in generalization to different data distributions, and ability to zero-shot transfer to an unseen 3D-QA task.

## 2. Related Work

### 3D grounding

Many prior works that tackle the 3D-REC task employ end-to-end approaches that jointly attend over language and point clouds [7–10, 23], commonly leveraging a Transformer architecture [33]. These methods can be broadly categorized into two types: object-centric ones, and ones that model the full 3D scene. Most works based on full 3D scene modeling use a detection module to create object proposals. For example, Text-guided Graph Neural Network [17] conducts instance segmentation on the full scene to create candidate objects as input to a graph neural network [32]; InstanceRefer [39] selects instance candidates from the panoptic segmentation of point clouds; 3DVG-Transformer [40] uses outputs from an object proposal generation module to fully leverage contextual clues for cross-modal proposal disambiguation. The best performing work in this category, BUTD-DETR [20], uses box proposals from a pretrained detector and scene features from the full 3D scene to decode objects with a detection head. The Multi-View Transformer [18] separately models the scene by projecting the 3D scene to a multi-view space, to eliminate dependence on specific views and learn robust representations.

By contrast, object-centric models perform reasoning over an input set of object point clouds. ReferIt3DNet [2] utilizes a graph convolutional network with input objects as nodes of the graph. 3DRefTransformer [1], LanguageRefer [30] TransRefer [16], and SAT [37] are Transformer-based methods that operate on language and 3D object point clouds. 3DRefTransformer [1] is an end-to-end Transformer model that incorporates an object pairwise spatial relation loss. LanguageRefer [30] uses a Transformer architecture over bounding box embeddings and language embedding from DistillBert [31]. TransRefer [16] utilizes a Transformer-based network to extract entity-and-relation-aware represen-
NS3D is composed of three main components. a) A semantic parser parses the input language into a symbolic program. b) A 3D object-centric encoder takes input objects and learns object, relation, and ternary relation features. c) A neural program executor executes the symbolic program with the learned features to retrieve the target referred object.

The goal of the NS3D parser is to parse utterances $U$ into a symbolic program $P$ that resembles the underlying reasoning process of $U$. The program has a hierarchy of primitive operations defined in a minimal but powerful domain-specific language (DSL) for 3D visual reasoning tasks. Each operation is composed of a function name (e.g., anchor or filter), and arguments (e.g., shelf, door, right). A key feature of such programs is that the output of one operation can be the input (argument) to another operation, as shown in Figure 3. Informally, the anchor program grounds the viewpoint, the filter program takes all the input objects in the full scene and outputs those that are of the specified category, and the relate program returns objects that satisfy the given relationship constraint. We include a more formal definition of these operations and the DSL in the supplementary material.

Parsing the input language into this hierarchical program allows NS3D to disentangle the learning of different functional modules that perform object-level or relational grounding and reasoning. These neural programs can be trained and combined in different ways.

**Semantic parsing with Codex.** In contrast to most existing neuro-symbolic reasoning frameworks, e.g., [24], instead of using a pretrained or jointly-trained semantic parser, we introduce the use of large language-to-code models for parsing. Specifically, we use Codex [6,11] with the Synchromesh framework [27]. By specifying only a small number of examples of language input and expected programs, we gain perfect parsing capabilities across unseen categories and re-

3. NS3D

In this section, we describe NS3D applied to the task of referring expression comprehension (3D-REC). We set up the task following ReferIt3D [2]: given a set of $M$ objects in the scene $O = \{O_1, ..., O_M\}$, where each object is represented as an RGB-colored point cloud of $N$ points $O_i \in \mathbb{R}^{N \times 6}$, and given an utterance $U$, the goal is to predict the target referred object $T \in O$. Due to the existence of many distractor objects in $O$, it is crucial to parse the full referring expression to select the correct target.

NS3D is a neuro-symbolic approach that combines programmatic functional structures and modular neural networks. It consists of three main components (see Figure 2). The first is a semantic parser that parses the input language $U$ into a symbolic program $P$ that resembles a hierarchical reasoning process underlying $U$ (Section 3.1). The second is a 3D feature encoder that extracts an object-centric representa-
relations in the ReferIt3D task. Synchromesh constrains the output of Codex to be a valid, executable program, adhering to the syntactic rules of the DSL.

Using Codex as our semantic parser has two major advantages. First, it only requires a small number of example programs to achieve strong parsing accuracy, compared to defining rules for semantic parsers or training from scratch. This enables us to easily generalize to new DSLs and tasks, such as recombining the learned functional modules in a completely new way to answer visually grounded questions. Second, compared to existing works that assume a given set of prompting examples, Codex can automatically identify unseen concepts from language through its built-in knowledge, even if they never appear in the prompting examples. In our experiments, we show that Codex outperforms a T5-based parser [29] finetuned on the same set of prompting examples.

### 3.2. 3D object-centric encoder

NS3D’s 3D encoder generates object-centric and relational features for each scene in a latent space for 3D grounding (see Figure 4). Recall that the input to the encoder is a scene as input, and executes the program based on the 3D features, and returns the target referred object \( T \).

![Figure 4. The NS3D object-centric encoder](image)

![Figure 5. The NS3D neural program executor executes the symbolic program recursively with the learned 3D features, and returns the target referred object \( T \).](image)

clouds \( O^\text{pos}_i \in \mathbb{R}^{1024 \times 3} \), as we are interested in modeling the spatial relations between objects rather than object types. Next, NS3D concatenates the per-object encoding for \( O_1 \) and \( O_j \), and applies a small 2-layer multi-layer perceptron MLP\text{binary} to extract relational features for the object pair:

\[
\begin{align*}
    f^\text{rel}_{i,j} &= \text{MLP}^\text{binary} \left( \text{concat} \left( E^\text{rel}(O^\text{pos}_i), E^\text{rel}(O^\text{pos}_j) \right) \right),
\end{align*}
\]

where \( \text{concat} \) denotes the concatenation operation for two vectors. In our design, \( E^\text{rel} \) is a shallower network that uses a sparser amount of samples than the object feature encoder \( E^\text{obj} \), which requires more fine-grained encoding of point clouds to classify categories.

We also model ternary relations as seen in ReferIt3D (e.g., a vector embedding for each triple of objects \( (O_i, O_j, O_k) \)). Specifically, we propose using the encoder to also extract a ternary feature \( f^\text{ternary}_{i,j,k} \). As both the binary-relation features and ternary-relation features focus on encoding spatial relationships among objects, NS3D shares the underlying PointNet encoder for them. This reduces the time and memory cost for high-arity inference. Mathematically,

\[
\begin{align*}
    g^\text{ternary}_{i,j} &= \text{MLP}^\text{ternary} \left( f^\text{rel}_{i,j} \right),
    g^\text{ternary}_{i,j,k} &= \text{concat} \left( g^\text{ternary}_{i,j}, g^\text{ternary}_{j,k}, g^\text{ternary}_{k,i} \right),
\end{align*}
\]

where MLP\text{ternary} is another 2-layer multi-layer perceptron that shares the same architecture as MLP\text{binary}.

### 3.3. Neural program executor

The neural program executor takes the parsed program \( P \) and the learned representation \( (f^\text{obj}, f^\text{rel}, f^\text{ternary}) \) for each scene as input, and executes the program based on the 3D representation, returning the target object being referred to. At a high-level, NS3D follows the hierarchical structure in \( P \) and executes the program recursively (see Figure 5). Here, we describe the neural implementation for each operation.

The key representation we will be using during program execution is the object score vector, which is a vector of length \( M \) (the number of objects in the scene), indicating
whether an object has been selected or not. For example, semantically, the operation `filter` takes an input set of objects being selected and an object category, and outputs a subset of input objects belonging to the specified category. Both the input and output of the `filter` operation will be represented as such object score vectors. For numerical stability, such scores are represented in the log space. One interpretation is that each entry $v_i$ in a score vector represents the log probability of object $O_i$ being selected.

**scene**$(x) \rightarrow y$: the scene operation returns a object score vector representing “all objects in the scene.” Recall that the values are in the log space; therefore, $y_i = 0$, for all $i \in \{1, 2, \ldots, M\}$.

**filter**$(x, c) \rightarrow y$: the filter operation takes an input object score vector $x$, an object category $c$, and returns a new object score vector, selecting objects that are in $x$ and belongs to category $c$. Therefore, we first compute $\text{prob}_i^c = \text{MLP}^c(f_i^{ob})$, which is a score for object $i$ belonging to category $c$, where $\text{MLP}^c$ is a mapping (specialized for $c$) from the dimension of $f_i^{ob}$ to dimension 1. Next, we merge it with the input object score vector $x$. Overall,

$$y_i = \min(x_i, \text{prob}_i^c) = \min\left(x_i, \text{MLP}^c(f_i^{ob})\right).$$

**relate**$(x^f, x^r, \text{rel}) \rightarrow y$: the relate program takes as input two sets of objects, target objects $x^f$ and reference objects $x^r$, as well as a relational concept `rel`, and outputs target objects that satisfy the specified relation. As an example, the expression “the chair beside the shelf” will be parsed into the program `relate(filter(chair), filter(shelf), beside)`. In this case, $x^f$ will be the filter result for `chair`, while $x^r$ will be the filter result for `shelf`. The relate operation classifies whether each pair of objects satisfy the relation `rel`, and selects the objects in $x^f$ that have relation `rel` with objects in $x^r$:

$$\text{prob}_{i,j}^\text{rel} = \text{MLP}^\text{rel}(f_{i,j}^{ob})$$

$$y_i = \min(x_i, \sum_j \text{sx}(x^r)_j \cdot \text{prob}_{i,j}^\text{rel}),$$

where $\text{MLP}^\text{rel}$ is a linear layer with scalar output and specialized for concept `rel`, and $\text{sx}$ is the softmax function applied to $x_r$. The $\sum_j$ operator can be interpreted as a “soft” selection of the $j^\text{th}$ row in the relation matrix $\text{prob}^\text{rel}$, where $j^* = \arg \max x^r$, the index of the referred object.

**ternary**$_\text{relate}(x^f, x^{r1}, x^{r2}, \text{trel}) \rightarrow y$: we propose to extend the formulations above to handle object relationships that involve more than two objects. In this case, $x^{r1}$ and $x^{r2}$ are two reference objects. In ReferIt3D [2], there are two types of ternary relationships: spatial ternary relations (e.g., between), and view-dependent relations. Both are resolved with this operation. As an example, the sentence “Facing the front of the shelf, select the door that is on the right side of it.” yields the program `anchor(filter(shelf), ...)`. Internally, such `anchor` operation will be handled as a ternary relation function: `ternary`$_\text{relate}$(`filter(door), filter(shelf), filter(shelf), anchor-right`). The two reference objects (the reference for the relation “right” and the anchor for “facing”) are the same. Notably, NS3D’s ternary operation can be generalized as a principled solution to any high-arity relations that can be executed based on learned features of the corresponding arity. In our ternary case, it is executed as the following:

$$\text{prob}_{i,j,k}^\text{ternary} = \text{MLP}^\text{rel}(f_{i,j,k}^{\text{ternary}})$$

$$y_i = \min\left(x_i, \sum_j \sum_k \text{sx}(x^{r1})_j \cdot \text{sx}(x^{r2})_k \cdot \text{prob}_{i,j,k}^\text{rel}\right).$$

The NS3D neural program executor composes the above operations and outputs the final object score vector, whose maximum-valued index represents the referred object $T$.

### 3.4. Training

Modules in NS3D can be trained end-to-end with only the groundtruth referred objects as supervision; each can also be trained individually whenever additional labels are available. In this paper, we use a hybrid training objective similar to prior works [2, 20]. Specifically, we use the groundtruth object category to compute a per-object classification loss $\mathcal{L}_{\text{oce}}$ (applied to all $\text{prob}_c$, where $c$ is the category) and the groundtruth final target object to compute a per-expression loss $\mathcal{L}_{\text{oc}e}$. Both loss functions are standard cross-entropy losses. The final total loss, with $\alpha = 1$ in our experiments, is: $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{oce}} + \alpha(\mathcal{L}_{\text{oc}e})$. Practically, we perform a two-stage training: we first pretrain the model with $\mathcal{L}_{\text{oce}}$ until convergence, and then train with the full loss.

As a byproduct of NS3D’s modular structure, we gain improved model interpretability. We can validate whether specific programs yield correct outputs at each stage. This can help determine what types of additional data will be valuable. In our experiments, we find that object categorization is the most challenging part of the 3D-REC task.

NS3D does not need the full scene point cloud as its input, and instead only explicitly models a given object set $O$. Therefore, we can train on scenes with a small number of objects (10 objects in our experiments) and directly test on scenes with much more objects (88 objects maximum in the test set). This improves training efficiency and reduces the need for annotated 3D objects, which are expensive to acquire in 3D domains. It also enables generalization to more cluttered and complex scenes. In all of our experiments, we train NS3D on scenes with a sparse amount of objects and see no performance drop compared to training with a dense train set. By contrast, baseline methods yield significantly decreased performance under this setting; we present these results in the supplementary material.

---

\*For conciseness, we have used `filter(shelf)` as a short-hand notation for `filter(scene(), shelf)`.\*
4. Experiments

We evaluate NS3D and compare it to prior work on the ReferIt3D benchmark [2], a 3D referring expression comprehension (3D-REC) task. We specifically focus on the SR3D setting, which tackles spatially-oriented object referential language in 3D scenes. In Section 4.1, we compare performance against baselines and report ablations. In Section 4.2 and Section 4.3, we show experiments on data efficiency and generalization. Finally, in Section 4.4, we present NS3D’s ability to zero-shot transfer to a 3D-QA task.

4.1. 3D referring expression comprehension

In the ReferIt3D task, the input is a set of point clouds, one for each object in the scene, as well as the utterance, and the target object is assumed to be unique. Therefore, models can be evaluated by the accuracy of selecting the correct object. Figure 6 shows an example task instance in ReferIt3D and its NS3D execution trace.

Results. In Table 1, we compare NS3D to baselines. We group methods into two categories: methods that only use the per-object point clouds and methods that model the full scene. The results show that we outperform other object-centric methods and achieve comparable performance with top-performing methods. In addition, we demonstrate state-of-the-art view-dependent accuracy compared to all prior work, with an improvement of 3.6% against the top-performing baseline. NS3D shows close performance between overall and view-dependent accuracy, while all other methods yield a large gap. Note that unlike models such as MVT [18] that explicitly transform and encode point clouds from multiple views to improve multi-view performances, our model simply uses a general high-arity neural network.

Ablations. In Table 3, we first discuss the performance of relational grounding modules. Specifically, since the ReferIt3D dataset does not contain groundtruth labels for object relations, we study the performance of relation grounding modules by evaluating NS3D performance using the groundtruth object classification for all filter operations. We see that NS3D achieves almost perfect performance on the task, indicating that it learns relations and high-arity relations well. Our neuro-symbolic approach allows for such diagnostics of model performance: the primary challenge to NS3D is object classification, which can potentially be improved with more object labels.

We next explore the importance of separating object encoders \( E^{obj} \) and \( E^{rel} \). Having separate object and relation features allows each to specialize in different goals: object classification and relational reasoning. We see that overall performance decreases by 4.6% if they share the same feature encoder. We additionally present results on using an incorrect number of arguments for executing high-arity queries (i.e., spatial ternary relations and view-dependent relations) by removing the second reference object. It leads to a 5.8% drop in view-dependent accuracy, which supports the importance of high-arity modules.

4.2. Data efficiency

We report experiments on data efficiency compared to four top-performing prior work on ReferIt3D, two object-centric methods (SAT [37] and TransRefer [16]) and two

<table>
<thead>
<tr>
<th>Method</th>
<th>Overall</th>
<th>View-dep.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NS3D (ours)</td>
<td>0.627</td>
<td>0.620</td>
</tr>
<tr>
<td>SAT [37]</td>
<td>0.579</td>
<td>0.492</td>
</tr>
<tr>
<td>TransRefer [16]</td>
<td>0.574</td>
<td>0.499</td>
</tr>
<tr>
<td>LANGUAGERefer [30]</td>
<td>0.560</td>
<td>0.492</td>
</tr>
<tr>
<td>3DRefTransformer [1]</td>
<td>0.470</td>
<td>0.443</td>
</tr>
<tr>
<td>ReferIt3D [2]</td>
<td>0.408</td>
<td>0.392</td>
</tr>
</tbody>
</table>

Table 1. NS3D yields the highest overall accuracy on the SR3D task among object-centric methods, and state-of-the-art view-dependent accuracy across all methods.

\( U = \text{“ Facing the couch from the side you sit on it, choose the cabinet that is to the right of this couch.”} \)

Figure 6. Example of NS3D’s execution trace on a view-dependent 3D-REC task. NS3D returns the correct target cabinet object.

\( \text{We note that BUTD-DETR operates on a more constrained setting, assuming 485 classes instead of the full 607 classes in ReferIt3D.} \)
methods that model the full 3D scene (BUTD-DETR [20] and MVT [18]). We test on 0.5% (329 examples), 1.5% (987 examples), 2.5% (1,646 examples), 5% (3,292 examples), and 10% of data (6,584 examples) in the train set, with the same full test set. We note that BUTD-DETR [20] uses pretrained object classification results on the full ScanNet dataset [13], while others do not. Hence in Table 2, we report NS3D’s performance on both settings: pretrained object classification on the full ReferIt3D train set (NS3D + Full), and on the smaller train set only (NS3D).

Shown in Figure 7, we see that in both settings, our neuro-symbolic approach significantly outperforms prior work across all data-efficient settings, achieving only small drops in accuracy compared to training on the full train set. NS3D yields 52.0% accuracy with just 1.5% of the train data, with 987 examples only, while all other baselines report accuracy lower than 27.5%. We see this trend persist across data-efficient settings. NS3D sees only a 3.6% gap when using 5% vs 100% of data, while all other methods decrease in performance significantly. NS3D’s accuracy of 59.1% at 5% train data is higher than that of baselines at 100% train data, aside from BUTD-DETR [20] and MVT [18]. This is a significant improvement in the 3D domain, where data annotation is especially labor intensive and expensive.

4.3. Generalization

We present two additional generalization settings and compare our model against prior work in Table 4. The first setting (PAIRS) evaluates performance on unseen object-relation-object pairs. The referring expressions in the train set include the top 5 percent of object-relation-object pairs: i.e., the referred object category, relation type, and the refer-
The dataset consists of four main types of questions; see Figure 8 for examples of each type. The first are exist-typed questions, which ask whether an object of the specified class and relation exists. The second are count-typed questions, which ask for the number of objects that satisfies the specification. The third are object-typed questions, which ask for object categories, and the last are relation-typed questions, which ask for the relationship between the specified objects. Each type of question has view-dependent and ternary relation variants.

In the semantic parsing stage, NS3D parses the new input questions into symbolic programs by specifying only a handful of prompts for our Codex-based parser (10 sentence-program pairs). By simply specifying one prompt for each type of expected program structure, we gain perfect parsing capabilities of this new program structure. In the 3D feature encoding stage, NS3D can directly re-use learned object and relation grounding modules from 3D-REC for 3D-QA. The 3D object-centric features are the same across both tasks.

The new functional modules introduced in NS3D for 3D-QA task output text answers. The query_exist operation is implemented as max over a threshold, and the query_count operation as the sum over a threshold, both based on the object score vector. The query_object and query_relation operations return the category or relation label with the highest prediction probability across labels. Formal definitions for each operation are described in the supplementary material. Note that all modules require no additional training, and are built through composing learned models from 3D-REC.

We show that NS3D can zero-shot transfer across tasks in 3D, with no additional finetuning or training of neural networks required. In Table 5, we report accuracy, calculated as the exact match of the text output, for overall performance and for each question type. We compare against NS3D with a finetuned T5 conditional generation model [29] as its semantic parser (NS3D + T5), instead of with Codex (NS3D + Codex). The T5 model is pretrained then finetuned on the same set of examples that Codex received. NS3D with Codex outperforms NS3D with the finetuned T5 model by a large margin, due to T5’s inability to generalize to new words outside of its small train set as well as errors in parsing text into

<table>
<thead>
<tr>
<th>Pairs</th>
<th>Scene</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ALL</td>
</tr>
<tr>
<td>NS3D + FULL (OURS)</td>
<td>0.612</td>
</tr>
<tr>
<td>NS3D (OURS)</td>
<td>0.599</td>
</tr>
<tr>
<td>BUTD-DETR [20]</td>
<td>0.440</td>
</tr>
<tr>
<td>MVT [18]</td>
<td>0.420</td>
</tr>
<tr>
<td>SAT [37]</td>
<td>0.359</td>
</tr>
<tr>
<td>TRANSREFER [16]</td>
<td>0.322</td>
</tr>
</tbody>
</table>

Table 4. Generalization performance to unseen object co-occurrence pairs and scene. NS3D outperforms all baselines.

<table>
<thead>
<tr>
<th></th>
<th>ALL</th>
<th>EXIST</th>
<th>COUNT</th>
<th>OBJ</th>
<th>REL</th>
</tr>
</thead>
<tbody>
<tr>
<td>NS3D + Codex</td>
<td>0.68</td>
<td>0.80</td>
<td>0.67</td>
<td>0.60</td>
<td>0.60</td>
</tr>
<tr>
<td>NS3D + T5</td>
<td>0.30</td>
<td>0.40</td>
<td>0.13</td>
<td>0.00</td>
<td>0.30</td>
</tr>
<tr>
<td>RANDOM</td>
<td>0.16</td>
<td>0.40</td>
<td>0.07</td>
<td>0.00</td>
<td>0.10</td>
</tr>
</tbody>
</table>

Table 5. NS3D’s zero-shot transfer performance on the 3D-QA task, with comparison of semantic parsers (Codex vs T5), and with a randomly initialized model as baseline.

5. Conclusion

We have presented NS3D, a neuro-symbolic model for 3D grounding that leverages compositional programs and modular neural networks to solve complex 3D-REC tasks. It enables strong data-efficiency, generalization to novel data distributions, and zero-shot transfer of 3D knowledge to a new 3D-QA task. We show that we can integrate large language-to-code models with modular neural networks, and accurately parse language into symbolic programs for visual reasoning. We also present a neural program executor that implements high-arity operations effectively to ground complex semantic forms, such as view-dependent anchoring. Together, these components of NS3D form a powerful model for 3D visual understanding. As a future direction, combining NS3D with strong object localization models can potentially enable learning directly from 3D scenes.

Acknowledgments. This work is in part supported by Stanford HAI, NSF RI #2211258, ONR MURI N00014-22-1-2740, AFOSR YIP FA9550-23-1-0127, Amazon, Analog Devices, Bosch, JPMorgan Chase, Meta, and Salesforce.
References


