SceneScript: Reconstructing Scenes With An Autoregressive Structured Language Model

Armen Avetisyan[1], Christopher Xie[1], Henry Howard-Jenkins[1], Tsun-Yi Yang[1], Samir Aroudj[1], Suvam Patra[1], Fuyang Zhang[2†], Duncan Frost[1], Luke Holland[1], Campbell Orme[1], Jakob Engel[1], Edward Miller[1], Richard Newcombe[1], and Vasileios Balntas[1]

[1] Meta Reality Labs [2] Simon Fraser University
https://projectaria.com/scenescript

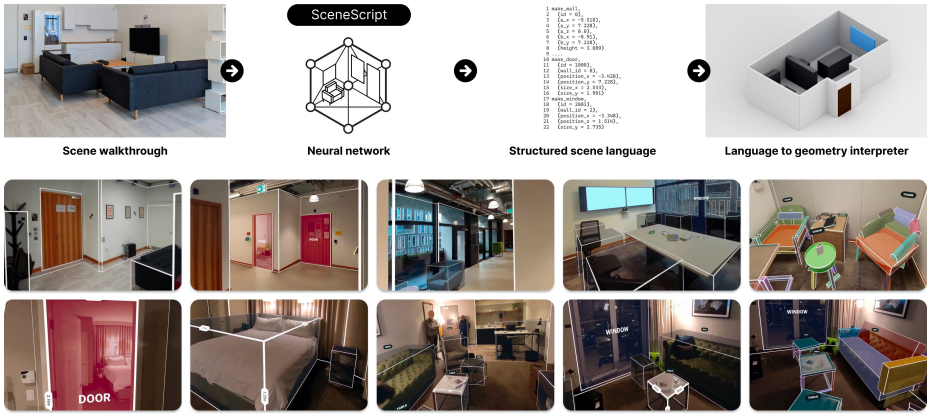

Fig. 1: footnotesize(*top*) Given an egocentric video of an environment, SceneScript directly predicts a 3D scene representation consisting of structured scene language commands. (*bottom*) Our method generalizes on diverse real scenes while being solely trained on synthetic indoor environments. (*last column, bottom*) A notable advantage of our method is its capacity to easily adapt the structured language to represent novel scene entities. For example, by introducing a single new command, SceneScript can directly predict object parts jointly with the layout and bounding boxes.

Abstract. We introduce SceneScript, a method that directly produces full scene models as a sequence of structured language commands using an autoregressive, token-based approach. Our proposed scene representation is inspired by recent successes in transformers & LLMs, and departs from more traditional methods which commonly describe scenes as meshes, voxel grids, point clouds or radiance fields. Our method infers the set of structured language commands directly from encoded visual data using a scene language encoder-decoder architecture. To train SceneScript, we generate and release a large-scale synthetic dataset called *Aria Synthetic Environments* consisting of 100k high-quality indoor scenes, with photorealistic and ground-truth annotated renders of egocentric scene walkthroughs. Our method gives state-of-the art results in architectural layout estimation, and competitive results in 3D object detection. Lastly, we explore an advantage for SceneScript, which is the ability to readily adapt to new commands via simple additions to the structured language, which we illustrate for tasks such as coarse 3D object part reconstruction.

[†]Work done while the author was an intern at Meta.

1 Introduction

Scene representations play a crucial role in machine learning and computer vision applications, enabling accurate understanding of the environment. Over the years, researchers have explored various options such as meshes, voxel grids, point clouds, and implicit representations, aiming to represent complex real-world scenes with high-fidelity. Each of these exhibits distinct advantages and limitations that impact their suitability for different tasks. Meshes offer detailed geometric information but can be expensive in both computation and memory. Voxel grids provide a volumetric representation but suffer from a trade-off between resolution and memory requirements. Point clouds are efficient in representing sparse scenes but lack semantics and explicit connectivity information. Implicit representations, such as DeepSDF [32] and NeRF [25], can be infinitely precise but lack interpretability and editability. The selection of an appropriate representation directly impacts the performance and efficacy of various tasks including object recognition, scene understanding, and 3D reconstruction. In this paper, we propose a novel scene representation based on structured language commands as a more efficient and versatile solution.

Our motivation stems from the recent advancements in the field of Large Language Models (LLMs) and "next token prediction" autoregressive methods [30], coupled with recent works on exploring generation of sequences to represent geometric structures. For example, PolyGen [28] demonstrated the ability to describe 3D meshes as a sequence of vertices and faces generated using transformers [45]. Similarly, CAD-as-Language [14] showcased the effectiveness of generating Computer-Aided Design (CAD) primitives to represent 2D CAD sketches. Our main goal is to **directly infer a metrically accurate representation of a full scene** as a text-based sequence of specialized structured language commands.

Our method, denoted `SceneScript`, autoregressively predicts a language of hand-designed structured language commands in pure text form. This language offers several distinct advantages: 1) As pure text, it is compact and reduces memory requirements of a large scene to only a few **bytes**. 2) It is crisp and complete since the commands are designed to result in sharp and well-defined geometry (similar to scalable vector graphics). 3) It is interpretable, editable and semantically rich by design via the use of *high-level parametric* commands such as `make_door(*door_parameters)`. 4) It can seamlessly integrate novel geometric entities by simply adding new structured commands to the language. 5) The fact that the scene representation is a series of language tokens similar to [30] opens up a plethora of potential new applications in the future such as ways to edit the scene, query it, or spin up chat interactions.

We mainly focus on the problems of architectural layout estimation and object detection as proxy tasks for the efficacy of our `SceneScript` language as a scene representation. Architectural entities such as walls, doors, and windows are highly structured entities, making them an ideal test-bed. However, one notable drawback of language models is that they require vast amounts of data for training. Since there is no existing dataset of scene walkthroughs and their cor-

responding structured language commands, we publicly released *Aria Synthetic Environments* (*ASE*), a synthetically generated dataset of 100k unique interior scenes. For each scene, we simulate egocentric trajectories with an entire suite of sensor data from Project Aria [24]. We also release multiple sources of ground truth including depth and instance segmentations. Importantly, for each rendered egocentric sequence, the architectural layout ground truth is given in our proposed `SceneScript` language.

While architectural layout serves as a test-bed, we demonstrate that our method `SceneScript` can easily be extended to new tasks via simple extensions to `SceneScript` language while keeping both the visual input and network architecture fixed. We illustrate this on the problem of 3D object detection, which results in a method that jointly infers architectural layout and 3D oriented bounding boxes. Additionally, we demonstrate more proof-of-concept experiments that show that our method results in a significantly lower barrier to entry for new tasks including representing coarse 3D object reconstruction, curved entities, composition of entities, and entity states.

Our core contributions are:

- We propose `SceneScript`, a method that jointly predicts architectural layout and object bounding boxes of a scene in the form of structured language commands given a video stream.
- We demonstrate that `SceneScript` can easily be extended to completely new tasks with simple additions of commands to our structured language, significantly lowering the barrier to entry for new tasks.
- We release a large-scale synthetic dataset, named *Aria Synthetic Environments*, comprized of **100k** unique high-quality 3D indoor scenes with GT, which will enable large scale ML training of scene understanding methods.
- We show that training `SceneScript` on *Aria Synthetic Environments* leads to generalization on real scenes (see videos/demos on the project page).

2 Related Works

2.1 Layout Estimation

Layout estimation is an active research area, aiming to infer architectural elements. Scan2BIM [27] proposes heuristics for wall detection to produce 2D floorplans. Ochmann et al. [29] formulate layout inference as an integer linear program using constraints on detected walls. Shortest path algorithms around birds-eye view (BEV) free space [5] and wall plus room instance segmentation [7] have also been explored.

Furukawa et al. [13] utilize a *Manhattan world*-based multi-view stereo algorithm [12] to merge axis-aligned depth maps into a full 3D mesh of building interiors. RoomNet [21] predicts layout keypoints while assuming that a fixed set of Manhattan layouts can occur in a single image of a room. LayoutNet [53] improves on this by predicting keypoints and optimising the Manhattan room layout inferred from them. Similarly, AtlantaNet [34] predicts a BEV floor or

Table 1: Complete set of structured language commands designed for detailing architectural layouts and object bounding boxes. Supported data types can include `int, float, bool`. It is important to note that the language's extensibility allows for easy augmentation by introducing new commands like `make_prim, make_pillar`, or enhancing existing commands, such as incorporating `is_double_door (bool)`.

make_wall (int)		make_door (int)		make_window (int)		make_bbox (int)	
id	int	id	int	id	int	id	int
a_x	float	wall0_id	int	wall0_id	int	class	int
a_y	float	wall1_id	int	wall1_id	int	position_x	float
a_z	float	position_x	float	position_x	float	position_y	float
b_x	float	position_y	float	position_y	float	position_z	float
b_y	float	position_z	float	position_z	float	angle_z	float
b_z	float	width	float	width	float	scale_x	float
height	float	height	float	height	float	scale_y	float
						scale_z	float

ceiling shape and approximates the shape contour with a polygon resulting in an *Atlanta world* prior. SceneCAD [2] uses a graph neural network to predict object-object and object-layout relationships to refine its layout prediction.

Our approach stands out by requiring neither heuristics nor explicitly defined prior knowledge about architectural scene structure. In fact, our method demonstrates geometric understanding of the scene which emerges despite learning to predict the GT scene language as a sequence of text tokens.

2.2 Geometric Sequence Modelling

Recent works have explored transformers for generating objects as text-based sequences. PolyGen [28] models 3D meshes as a sequence of vertices and faces. CAD-as-Language [14] represents 2D CAD sketches as a sequence of triplets in protobuf representation, followed by a sequence of constraints. Both SketchGen [31] and SkexGen [48] use transformers to generate sketches. DeepSVG [6] learns a transformer-based variational autoencoder (VAE) that is capable of generating and interpolating through 2D vector graphics images. DeepCAD [47] proposes a low-level language and architecture similarly to DeepSVG, but applies it to 3D CAD models instead of 2D vector graphics. Our approach stands out by utilising *high-level* commands, offering interpretability and semantic richness. Additionally, while low-level commands can represent arbitrarily complex geometries, they lead to prohibitively longer sequences when representing a full scene.

The closest work to ours is Pix2Seq [8]. Pix2Seq proposes a similar architecture to ours but experiments only with 2D object detection, thus requiring domain-specific augmentation strategies. Another closely related work is Point2Seq [49] that trains a recurrent network for autoregressively regressing continuous 3D bounding box parameters. Interestingly, they find the autoregressive ordering of parameters outperforms current standards for object detection architectures, including anchors [15] and centers [51].

Fig. 2: *Aria Synthetic Environments*: (top) Random samples of generated scenes showing diversity of layouts, lights and object placements. (bottom - left to right) A top down view of a scene filled with objects, a simulated trajectory (blue path), renderings of depth, RGB, and object instances, and lastly a scene pointcloud.

3 `SceneScript` Structured Language Commands

We first describe our structured language commands that define a full scene representation including both layout and objects. After this, we introduce our corresponding large scale training dataset: *Aria Synthetic Environments*.

3.1 Commands and Parameters

We begin with a parameterization that captures the most common layout elements. For this purpose we use three commands: `make_wall`, `make_door`, and `make_window`. Each command comes with a set of parameters that results in well-defined geometry. For example, the full set of parameters for `make_wall` specifies a gravity-aligned 2D plane, while `make_door` and `make_window` specify box-based cutouts from walls. It is worth noting that this parametrization is arbitrary, and is only made in the context of presenting a proof-of-concept `SceneScript` system. There are infinitely many parametrization schemes, in this work we opt for one that prioritizes ease of use and research iteration speed.

In addition to representing these three major layout entities, we aim to jointly infer objects as oriented bounding boxes. Thus, we introduce a fourth command:

```
make_bbox: id, class, position_x, position_y,
    position_z, angle_z, scale_x, scale_y, scale_z
```

This simple parametrization represents an oriented 3D bounding box that is assumed to be aligned with gravity (assuming it points in the $-z$ direction). A summary of these commands and their respective parameters are shown in Table 1.

While we have described just four commands to capture structure and objects in an indoor environment, importantly, this text-based parametrization can readily be extended geometrically and/or even semantically to include states or other functional aspects. For example, changing the mentioned `make_door` command to include parameters such as `open_degree` allows the language to represent door states. In Section 6, we demonstrate how such extensions to the language allows for `SceneScript` to readily adapt to new tasks including coarse 3D object reconstruction.

3.2 Scene Definition

A single scene can be described as a sequence of our proposed structured language commands. The sequence requires no specific ordering, and can be of arbitrary length. From this definition, the 3D scene can easily be obtained by parsing the commands through a simple custom interpreter.

3.3 Training Dataset

To enable practical indoor scene reconstruction based on our structured language commands (Section 3.1), we publicly released a new dataset called *Aria Synthetic Environments*. It consists of a large number of training pairs of egocentric scene walkthroughs linked with corresponding ground truth command sequences.

Since transformers require vast amounts of data, we generated $100k$ synthetic scenes, which is in comparison infeasible for real-world data. Each synthetic scene comes with a floor plan model, a corresponding complete 3D scene as well as a simulated agent trajectory and a photo-realistic rendering of this trajectory. Fig. 2 illustrates the basics of *Aria Synthetic Environments*. For brevity, we refer the reader to Appendix A for further details.

4 `SceneScript` Network Architecture

Our pipeline is a simple encoder-decoder architecture that consumes a video sequence and returns `SceneScript` language in a tokenized format. Figure 3 illustrates a high-level overview of our method.

We examine three encoder variants: a pointcloud encoder, a posed image set encoder, and a combined encoder. The decoder remains the same in all cases.

4.1 Input Modalities and Encoders

The encoder computes a latent scene code in the form of a 1D sequence from video walkthrough of the scene. The decoder is designed to consume these 1D sequences as input. This enables the integration of various input modalities within a unified framework. As a preliminary, for each scene we assume access to a set of M posed camera images $\{\mathbf{I_1}, ..., \mathbf{I_M}\}$, e.g., *SLAM* output.

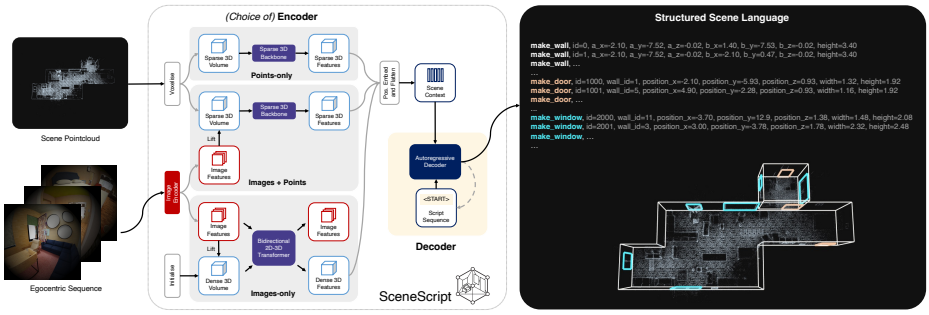

Fig. 3: `SceneScript` core pipeline overview. Raw images & pointcloud data are encoded into a latent code, which is then autoregressively decoded into a sequence of commands that describe the scene. Visualizations are shown using a customly built interpreter. Note that for the results in this paper, the the point clouds are computed from the images using Aria MPS [37] – i.e. are not using a dedicated RGB-D / Lidar sensor.

Point Clouds. A point cloud $\mathbf{P} = \{\mathbf{p_1}, ..., \mathbf{p_N}\}$ consists of N points, where $\mathbf{p_i}$ is a 3D point coordinate. It can come from passive images using *SLAM* or *SfM*, or RGB-D / Lidar sensors.

Specifically, we use the Semi-dense Pointclouds from Project Aria's Machine Perception Services [37], that are obtained from a visual-inertial SLAM system using Aria's monochrome cameras and IMUs. We discretize the point cloud to 5cm resolution, then employ a sparse 3D convolution library [40,41] to generate pooled features. The encoder $\mathcal{E}_{geo}$ applies a series of down convolutions, resulting in a reduction of the number of points in the lowest level.

$$\mathbf{F}_{geo} = \mathcal{E}_{geo}(\mathbf{P}), \quad \mathbf{P} \in \mathbb{R}^{N\times 3}, \mathbf{F}_{geo} \in \mathbb{R}^{K\times 512} \tag{1}$$

where $K \ll N$. $\mathbf{F}_{geo}$ is a condensed latent representation of the point cloud that contains the necessary scene context. For later use in the transformer decoder, we treat $\mathbf{F}_{geo}$ as a sequence of feature vectors where the entries $\mathbf{f_i}$, $i \in 1...K$ are sorted lexicographically according to the coordinate of the active site $\mathbf{c_i}$, $i \in 1...K$. To incorporate positional encoding, we append the coordinates of the active sites to their respective feature vectors $\mathbf{f}_i \leftarrow \texttt{cat}(\mathbf{f_i}, \mathbf{c_i})$.

Point Clouds with Lifted Features. We additionally explore augmenting the point cloud with image features. From the original egocentric sequence and associated trajectory, we sample a set of M keyframes, $\mathbf{I}_i$ where $i \in 1...M$, and compute a set of image features $\mathbf{F}_i$ for each. We then project each point into the set of keyframe cameras and retrieve the feature vector (output by a CNN) at the pixel location:

$$\mathbf{f}_{ip} = F_i(\pi(\mathbf{p})) \qquad \mathbf{p} \in \mathbf{P}, i \in 1...M, \tag{2}$$

where $\pi(\cdot)$ represents the projection function of a 3D point into the camera. If $\pi(\mathbf{p})$ falls outside the image bounds, no feature is retrieved. We combine the set

of lifted features for a point through an average, resulting in a single feature vector for each point: $\mathbf{f}_p = 1/M \sum_{i=1}^{M} \mathbf{f}_{ip}$.

We form our lifted-feature point cloud, $\mathbf{P'} = \{\mathbf{p'_1}, ..., \mathbf{p'_M}\}$, by concatenating each point's lifted feature with the original XYZ location: $\mathbf{p'} = \texttt{cat}(\mathbf{f}_p, \mathbf{p})$. $\mathbf{P'}$ is then encoded into a context sequence using sparse 3D convolutions, with only the input feature channels adjusted to match the new point feature dimension.

End-to-end Encoding of Posed Views. In order to encode the egocentric sequence more directly without a pre-computed point cloud, we adopt a 2D $\leftrightarrow$ 3D bidirectional transformer encoder following the form defined in RayTran [44].

In this formulation, we initialize a volume representation of the scene as a dense voxel grid of features, $\mathbf{V}$, that coincides with the scene geometry. In turn, we sample a subset of M keyframes, $\mathbf{I}_i$ where $i \in 1...M$, from the full stream of posed images. And for each of these keyframes we compute image features from a CNN, $\mathbf{F}_i$. Repeated layers of bidirectional attention enable the image and voxel grid features to be refined iteratively in successive transformer blocks through the aggregation of view-point and global-scene information. As in RayTran [44], the interaction between the two representations is guided by the image-formation process by leveraging known camera parameters and poses. Attention in these transformer blocks is restricted by patch-voxel ray intersections, where each image patch attends to the voxels it observes and each voxel location attends to all the patches that observe it. The resulting voxel grid of features is flattened, concatenated with an encoded represention of its XYZ location, and passed to the decoder.

4.2 Language Decoder

We utilize a transformer decoder [45] to decode the scene latent code into a sequence of structured language commands. The sequence of tokens passes through an embedding layer, followed by a positional encoding layer. Together with the encoded scene code (Section 4.1), the embedded tokens are passed into the several transformer decoder layers where a causal attention mask is used to ensure autoregressive generation. More implementation details can be found in Appendix C.2.

4.3 Language Tokenization

We refer to the serialization the structured language into a sequence of tokens as **tokenization**. The goal is to construct a bijective mapping between a sequence of structured language commands (Section 3) and a sequence of integer tokens that can be predicted by the transformer decoder architecture. We utilize the following schema:

$$[\texttt{START}, \texttt{PART}, \texttt{CMD}, \texttt{PARAM\_1}, \texttt{PARAM\_2}, \ldots, \texttt{PARAM\_N}, \texttt{PART}, \ldots, \texttt{STOP}]$$

For example, a sample sequence for a `make_door` command may look like:

$$[\texttt{START}, \texttt{PART}, \texttt{MAKE\_DOOR}, \texttt{POSITION\_X}, \texttt{POSITION\_Y}, \texttt{POSITION\_Z},$$
$$\texttt{WALL0\_IDX}, \texttt{WALL1\_IDX}, \texttt{WIDTH}, \texttt{HEIGHT}, \texttt{PART}, \ldots, \texttt{STOP}]$$

This schema enables 1D packing of tokens without requiring fixed-size slots or padding like other sequence modelling methods such as [47]. Additionally, it does not impose any limitations on the number or the hierarchy of sub-sequences, as they are flexibly separated by a `PART` token. This allows for arbitrarily complex scene representations.

The tokenized sequence is discretized into integers at a 5cm resolution, then translated into a sequence of embeddings via learnable lookup table. Note that by designing the `SceneScript` language, we also design the tokenization. This tokenization scheme is notably different from standard NLP tokenization, which involves Byte-Pair Encodings (BPE) [30].

5 Results

In this section, we introduce the metrics we define to measure performance, and we discuss some qualitative and quantitative results that give insights to our proposed `SceneScript` method.

5.1 Metrics

To evaluate accuracy of the layout estimation, we make use of geometric metrics applied between the ground-truth room layout and our predicted `SceneScript` language. To do so, we define an *entity distance*, d_E, between a pair of entities of the same class. Each entity, E, is represented as a 3D plane segment comprising of 4 corners $\{c_1, c_2, c_3, c_4\}$. The distance between two entities, E and E' is computed as the maximum Euclidean distance between each corner and its counterpart assigned via Hungarian mathcing, i.e.: $d_E(E, E') = \max\{||c_i - c'_{\pi(i)}|| : i = 1, ..., 4\}$, where $\pi(i)$ is the permutation also found via Hungarian matching.

We threshold d_E to define the success criteria for the predicted entities. We compute the following metrics:

- F1 Score @ *threshold* – the F1 score of the set of predictions is computed at a single d_E threshold.
- Average F1 Score – the F1 score is computed across a range of entity distance thresholds and averaged.

The scores are computed for each class independently and averaged to overall score. In addition, scores are computed for each scene and averaged across the dataset. We use the following range of thresholds (cm) for the average F1 scores: $T = \{1, 2, ..., 9, 10, 15, 25, 30, 50, 75, 100\}$.

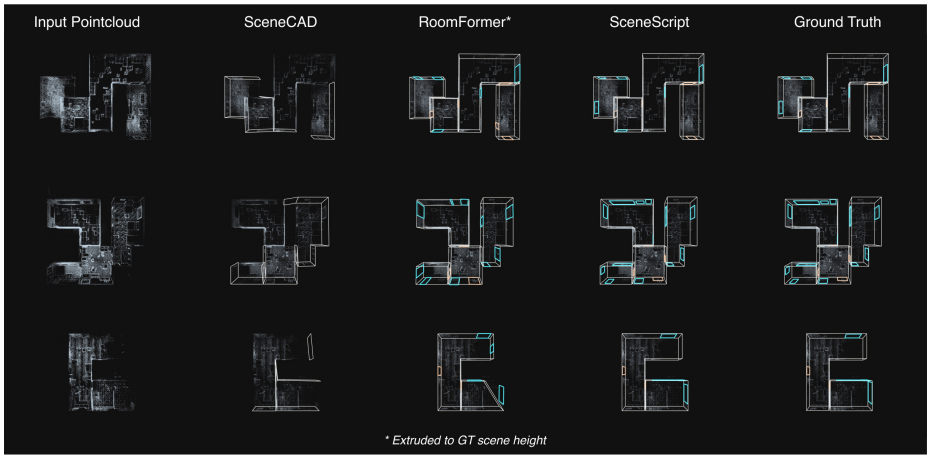

Fig. 4: Qualitative samples between our model and SOTA methods on *Aria Synthetic Environments*'s test set. Hierarchical methods like SceneCAD suffer from error cascading which leads to missing elements in the edge prediction module. RoomFormer (a 2D method extruded to 3D) primarily suffers from lightly captured scene regions which leave a unnoticeable signal in the density map.

Table 2: Layout Estimation on *Aria Synthetic Environments* Quantitative comparison between our method and related recent work.

	F1 @5cm				Avg F1			
Method	mean	wall	door	window	mean	wall	door	window
SceneCAD '20 [2]	-	0.048	-	-	-	0.275	-	-
RoomFormer '23 [52]	0.139	0.159	0.148	0.110	0.464	0.505	0.481	0.407
Ours (Point cloud)	0.848	0.930	0.922	0.692	0.784	0.816	0.811	0.724
Ours (Lifted features)	**0.903**	**0.943**	**0.959**	**0.806**	**0.801**	**0.818**	**0.822**	**0.764**
Ours (Image-only)	0.661	0.687	0.798	0.497	0.719	0.727	0.772	0.658

5.2 Layout Estimation

We perform scene layout estimation with the the three encoder variants of `SceneScript`: a baseline model with sparse3d convolution pointcloud encoder, an RGB RayTran-based feature volume encoder [44], and our proposed lifted feature point encoder. The same transformer decoder is being used in all three scenarios.

To provide comparison to existing works, we include results from two baseline methods, namely SceneCAD [2] and the recent RoomFormer [52]. For these experiments SceneCAD and RoomFormer were both trained on *Aria Synthetic Environments*. Note that SceneCAD only predicts walls.

Table 2 shows the main results for our F1-based metrics on *Aria Synthetic Environments*. `SceneScript` exhibits a substantial performance advantage over SOTA layout estimation baselines across multiple metrics. Both baseline meth-

Table 3: 3D Object Detection Performance comparison against state-of-the-art methods on an 3D object detection task trained and evaluated by F1-score at 0.25 and 0.5 IoU thresholds. By simply adding a `make_bbox` command `SceneScript` can achieve competitive object detection results.

(a) *Aria Synthetic Environments*

Method	Input	F1 @.25 IoU	F1 @.50 IoU
3DETR '21 [26]	Points	0.201	0.078
Cube R-CNN '23 [4]	RGB	0.394	0.228
ImVoxelNet '22 [36]	RGB	0.584	0.516
Ours	Points	**0.620**	**0.577**

(b) ScanNet [10]

Method	Input	F1 @.25 IoU	F1 @.50 IoU
3DETR '21 [26]	Points	0.480	0.349
3DETR-m '21 [26]	Points	0.536	0.407
SoftGroup '22 [46]	RGB Points	**0.622**	**0.573**
Ours	RGB Points	0.506	0.406

ods encounter a significant decline in accuracy when dealing with finer details. See Figure 4 for qualitative comparisons between our method and baseline methods.

Encoder Ablation. The results demonstrate that `SceneScript` is robust to the encoding strategy chosen to encode the egocentric capture. It is able to infer highly accurate scene layouts in all configurations tested, and in each case `SceneScript` outperforms the included baselines by a significant margin.

Relative comparison of the encoder strategies reveals that leveraging the pointclouds from a highly specialized and well-tuned system is still able to offer advantages of an, almost, entirely learned approach such as RayTran [44]. A light extension in the form of lifted image features can widen this gap even further. In particular, we observe that the discrepancy between the encoders becomes particularly apparent as the complexity of the scene increases in the form of increased room count – more details are included in Appendix F.2. In Appendix F.2, we also show a quantitative evaluation of per-entity error distances, which aids in further attribution of relative performance gains between the encoding methods.

5.3 Object Detection

In this section, we perform evaluation of `SceneScript` for object detection on both *Aria Synthetic Environments* and ScanNet [10]. For comparison, we include recent and state-of-the-art baseline methods, namely Cube-RCNN [4], ImVoxelNet [36], 3DETR [26], and SoftGroup [46].

Worth noting is that `SceneScript` does not predict a confidence score for each `make_bbox` command. This means that fair computation of the conventional mAP metric for this task is not possible, as detections cannot be ranked across scenes. Among other issues, this results in a metric that varies with the order in which scenes are evaluated. We therefore report F1-score-based metrics, which do not exhibit this order variance. Further discussion of this, and mAP numbers for the baselines for reference, can be found in Appendix G.4.

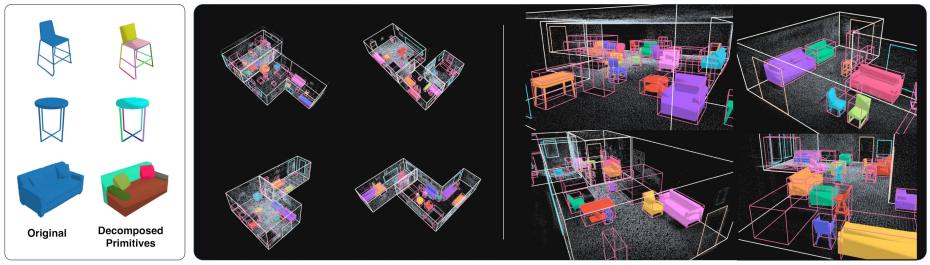

Fig. 5: Example scene reconstructions on scenes from *Aria Synthetic Environments*. (left) Visualisation of the decomposed meshes used to create `make_prim` training pairs. (right) Views of full scene predictions, as well as close ups highlighting the fidelity of object reconstruction through the prediction volumetric primitives enabled by `make_prim`.

In Table 3a, all methods were trained on *Aria Synthetic Environments*. 3DETR performs poorly due to its encoder being designed for relatively clean datasets such as ScanNet [10], while semi-dense point clouds from *Aria Synthetic Environments* exhibit more noise and less uniformity (see Figure 3 for an example). Cube R-CNN and ImVoxelNet both operate on RGB images, and detections are tracked for the entire sequence via a tracker [4] to provide competitive performance. In Table 3b, our method provides similar performance to both 3DETR and SoftGroup.

Through the addition of the `make_bbox` command, `SceneScript` demonstrates object detection performance on par with SOTA baselines on multiple object detection datasets. This result illustrates the extensibility of a token-based approach, and that our proposed `SceneScript` language representation does not suffer compared to specialised object detection network architectures.

6 Extending the `SceneScript` Structured Language

A key advantage offered by `SceneScript`'s structured language prediction paradigm is that the expressiveness of its reconstruction can be tailored without requiring a change to the method. Up to now, we have focussed on showcasing the efficacy of `SceneScript` for representing simple layout elements and objects as bounding boxes. In this section, we showcase this characteristic by increasing the fidelity of our scene representation by introducing coarse 3D object reconstruction.

6.1 Objects as Volumetric Primitives

We turn to a language based on volumetric primitives, motivated by works such as [43, 50]. Using simple primitives such as cuboids and extruded cylinders, enables us to coarsely represent arbitrary object categories while maintaining object semantics (e.g. tabletops can be represented by a single cuboid). Thus, this language can describe many object categories simultaneously.

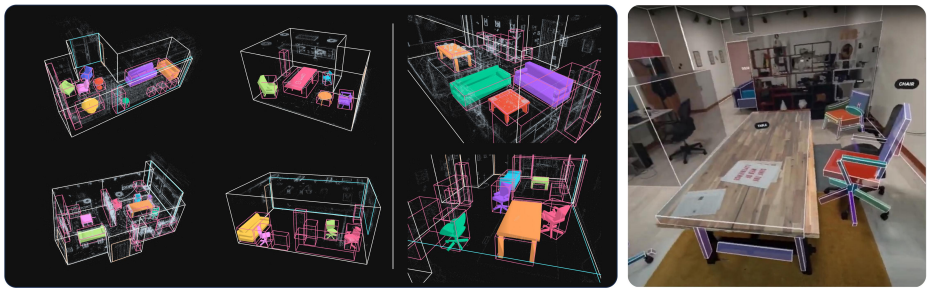

Fig. 6: Example scene reconstructions on **real scenes** with the addition of the `make_prim` command. Note that `SceneScript` was trained only on synthetic data.

This representation requires only one additional command over the layout and box commands already discussed previously, namely:

```
make_prim: bbox_id, prim_num, class, center_x, center_y
    , center_z, angle_x, angle_y, angle_z, scale_x,
    scale_y, scale_z
```

This command and its parameters describe a volumetric primitive (cuboid or extruded cylinder) via its 3D center, 3D rotation, and 3D scale. The `prim_num` parameter can be associated with semantics, e.g. tabletops of different tables typically have the same `prim_num`.

Dataset. To obtain ground truth `make_prim` commands that align with the objects in *Aria Synthetic Environments*, we first run an extension of Yang *et al*. [50] to obtain cuboid and extruded cylinder primitives of a database of 3D CAD models (ABO [9], which was used to populate *Aria Synthetic Environments*). See Figure 5 (left) for example decompositions. For this proof-of-concept experiment, we use three categories: *table*, *chair*, and *sofa*. We then convert these decomposed primitives into `make_prim` commands that are aligned with the objects in the dataset, which results in training pairs.

Results. We show qualitative results on *Aria Synthetic Environments* in Figure 5. In Figure 6, we show inferences in a few real-world environments despite only having trained on our simulated dataset. These results demonstrate that `SceneScript`'s general purpose architecture is capable of coarsely reconstructing objects of multiple categories through addition of a new command.

6.2 Further Extensions

In this section, we have explored just one extension of `SceneScript`'s structured scene language in order to demonstrate its flexibility. The result was greatly increased expressiveness of the scene reconstruction. In Appendix H, we

include additional explorations that can further increase the fidelity and accuracy of SceneScript's reconstructions. These explorations include reconstructing curved walls, inferring object states (such as door opening angles), as well as direct prediction of parametric models using object models deployed commonly by tech artists (e.g., blender geometry nodes) for object reconstruction.

7 Interactive Scene Reconstruction

Scene reconstruction often occurs offline, on pre-recorded walkthroughs of an environment, or derivatives of them, such as a point cloud. However, we take advantage of SceneScript's inference occurring at an interactive rate, which takes between 2-5s depending on the size of the environment, by implementing streaming of SceneScript's live reconstructions into a modern VR headset. This enables an interactive experience where the user can see the reconstruction overlaid onto the environment they are exploring in real-time. See the video recording on the website for examples.

With this the user can actively refine the results, for example through more thorough exploration of areas that may have been missed. Visualisations of this interface are included in the bottom half for Figure 1.

8 Limitations and Future Work

SceneScript exhibits certain limitations that should be acknowledged. First, the structured language commands are manually defined, which requires human intervention at this stage. Secondly, due to the higher-level nature of our commands, it can be challenging to capture fine-grained geometric details with extremely high precision (i.e. mm). As a consequence, the resulting reconstructions based on this representation tend to lead to simpler and coarser geometries, potentially missing intricate nuances at the very high detail level. These limitations potentially highlight areas for future research and optimization, aiming to automate the command definition process and explore techniques to enhance the representation's ability to capture intricate geometric details accurately. However, we believe that the ability to built scene representations that are based on structured language commands will be a key part in complex and efficient methods of scene reconstruction in the future, enabling them to be used in conjunction with general purpose LLMs.

9 Conclusion

We introduced SceneScript, a novel reconstruction method that is based on a tokenized scene representation. SceneScript autoregressively predicts a sequence of structured scene language commands given a video stream of an indoor space. This tokenized scene representation is compact, editable and intepretable. In addition, we showed that a strength of SceneScript is the ability to extend to

arbitrary elements (e.g. Bezier curves, object part decomposition) with minimal changes. This research opens up new directions in representing 3D scenes as language, bringing the 3D reconstruction community closer to the recent advances in large language models such as GPT-class models [30].

References

1. Aria, P.: Project aria machine perception services (2023), https://facebookresearch.github.io/projectaria_tools/docs/data_utilities/core_code_snippets/mps, accessed: 2023-11-27 21, 28
2. Avetisyan, A., Khanova, T., Choy, C., Dash, D., Dai, A., Nießner, M.: Scenecad: Predicting object alignments and layouts in rgb-d scans. In: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXII 16. pp. 596–612. Springer (2020) 4, 10
3. Bash, B.: Procedural table with geometry nodes in blender (2023), https://blenderbash.gumroad.com/l/daghsw, accessed: 2023-05-23 34
4. Brazil, G., Kumar, A., Straub, J., Ravi, N., Johnson, J., Gkioxari, G.: Omni3D: A large benchmark and model for 3D object detection in the wild. In: CVPR. IEEE, Vancouver, Canada (June 2023) 11, 12, 26, 29
5. Cabral, R., Furukawa, Y.: Piecewise planar and compact floorplan reconstruction from images. In: 2014 IEEE Conference on Computer Vision and Pattern Recognition (2014) 3
6. Carlier, A., Danelljan, M., Alahi, A., Timofte, R.: Deepsvg: A hierarchical generative network for vector graphics animation. In: Advances in Neural Information Processing Systems (2020) 4
7. Chen, J., Liu, C., Wu, J., Furukawa, Y.: Floor-sp: Inverse cad for floorplans by sequential room-wise shortest path. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (2019) 3
8. Chen, T., Saxena, S., Li, L., Fleet, D.J., Hinton, G.: Pix2seq: A language modeling framework for object detection. In: International Conference on Learning Representations (ICLR) (2022) 4
9. Collins, J., Goel, S., Deng, K., Luthra, A., Xu, L., Gundogdu, E., Zhang, X., Yago Vicente, T.F., Dideriksen, T., Arora, H., Guillaumin, M., Malik, J.: Abo: Dataset and benchmarks for real-world 3d object understanding. CVPR (2022) 13
10. Dai, A., Chang, A.X., Savva, M., Halber, M., Funkhouser, T., Nießner, M.: Scannet: Richly-annotated 3d reconstructions of indoor scenes. In: Proc. Computer Vision and Pattern Recognition (CVPR), IEEE (2017) 11, 12, 27
11. Engel, J., Schöps, T., Cremers, D.: LSD-SLAM: Large-scale direct monocular SLAM. In: European Conference on Computer Vision (ECCV) (September 2014) 21
12. Furukawa, Y., Curless, B., Seitz, S.M., Szeliski, R.: Manhattan-world stereo. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition (2009) 3
13. Furukawa, Y., Curless, B., Seitz, S.M., Szeliski, R.: Reconstructing building interiors from images. In: 2009 IEEE 12th international conference on computer vision (2009) 3, 31
14. Ganin, Y., Bartunov, S., Li, Y., Keller, E., Saliceti, S.: Computer-aided design as language. Advances in Neural Information Processing Systems **34**, 5885–5897 (2021) 2, 4

15. Girshick, R.: Fast r-cnn. In: Proceedings of the IEEE international conference on computer vision. pp. 1440–1448 (2015) 4
16. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016) 22
17. Holtzman, A., Buys, J., Du, L., Forbes, M., Choi, Y.: The curious case of neural text degeneration. arXiv preprint arXiv:1904.09751 (2019) 22
18. Jones, R.K., Barton, T., Xu, X., Wang, K., Jiang, E., Guerrero, P., Mitra, N.J., Ritchie, D.: Shapeassembly: Learning to generate programs for 3d shape structure synthesis. ACM Transactions on Graphics (TOG) **39**(6), 1–20 (2020) 34
19. Jones, R.K., Charatan, D., Guerrero, P., Mitra, N.J., Ritchie, D.: Shapemod: macro operation discovery for 3d shape programs. ACM Transactions on Graphics (TOG) **40**(4), 1–16 (2021) 34
20. Jones, R.K., Walke, H., Ritchie, D.: Plad: Learning to infer shape programs with pseudo-labels and approximate distributions. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (2022) 34
21. Lee, C.Y., Badrinarayanan, V., Malisiewicz, T., Rabinovich, A.: Roomnet: End-to-end room layout estimation. In: Proceedings of the IEEE international conference on computer vision (2017) 3
22. Liu, C., Kim, K., Gu, J., Furukawa, Y., Kautz, J.: Planercnn: 3d plane detection and reconstruction from a single image (2019) 31
23. Loop, C., Blinn, J.: Resolution independent curve rendering using programmable graphics hardware. In: ACM SIGGRAPH 2005 Papers, pp. 1000–1009 (2005) 31
24. Meta: Project aria. https://projectaria.com/ (2022), accessed: 2023-08-30 3
25. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: Nerf: Representing scenes as neural radiance fields for view synthesis. Communications of the ACM **65**(1), 99–106 (2021) 2
26. Misra, I., Girdhar, R., Joulin, A.: An end-to-end transformer model for 3d object detection. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 2906–2917 (2021) 11, 26, 27, 29
27. Murali, S., Speciale, P., Oswald, M.R., Pollefeys, M.: Indoor scan2bim: Building information models of house interiors. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2017) 3
28. Nash, C., Ganin, Y., Eslami, S.A., Battaglia, P.: Polygen: An autoregressive generative model of 3d meshes. In: International conference on machine learning. pp. 7220–7229. PMLR (2020) 2, 4
29. Ochmann, S., Vock, R., Klein, R.: Automatic reconstruction of fully volumetric 3d building models from oriented point clouds. ISPRS journal of photogrammetry and remote sensing **151**, 251–262 (2019) 3
30. OpenAI: Gpt-4 technical report (2023) 2, 9, 15, 22, 23
31. Para, W., Bhat, S., Guerrero, P., Kelly, T., Mitra, N., Guibas, L.J., Wonka, P.: Sketchgen: Generating constrained cad sketches. In: Advances in Neural Information Processing Systems (2021) 4
32. Park, J.J., Florence, P., Straub, J., Newcombe, R., Lovegrove, S.: Deepsdf: Learning continuous signed distance functions for shape representation. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (2019) 2
33. Pearl, O., Lang, I., Hu, Y., Yeh, R.A., Hanocka, R.: Geocode: Interpretable shape programs (2022) 34
34. Pintore, G., Agus, M., Gobbetti, E.: Atlantanet: inferring the 3d indoor layout from a single 360 image beyond the manhattan world assumption. In: Proceedings of the European conference on computer vision (ECCV) (2020) 3

35. Qi, C.R., Litany, O., He, K., Guibas, L.J.: Deep hough voting for 3d object detection in point clouds. In: proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 9277–9286 (2019) 27
36. Rukhovich, D., Vorontsova, A., Konushin, A.: Imvoxelnet: Image to voxels projection for monocular and multi-view general-purpose 3d object detection. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. pp. 2397–2406 (2022) 11, 27, 29
37. Somasundaram, K., Dong, J., Tang, H., Straub, J., Yan, M., Goesele, M., Engel, J.J., De Nardi, R., Newcombe, R.: Project aria: A new tool for egocentric multimodal ai research. arXiv preprint arXiv:2308.13561 (2023) 7, 20
38. Song, S., Lichtenberg, S.P., Xiao, J.: Sun rgb-d: A rgb-d scene understanding benchmark suite. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 567–576 (2015) 27
39. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks (2014) 22
40. Tang, H., Liu, Z., Li, X., Lin, Y., Han, S.: TorchSparse: Efficient Point Cloud Inference Engine. In: Conference on Machine Learning and Systems (MLSys) (2022) 7, 22, 27, 28, 29
41. Tang, H., Liu, Z., Zhao, S., Lin, Y., Lin, J., Wang, H., Han, S.: Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution. In: European Conference on Computer Vision (ECCV) (2020) 7, 22, 28
42. Tomasi, C., Kanade, T.: Detection and tracking of point. Int J Comput Vis **9**(137-154), 3 (1991) 21
43. Tulsiani, S., Su, H., Guibas, L.J., Efros, A.A., Malik, J.: Learning shape abstractions by assembling volumetric primitives. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017) 12
44. Tyszkiewicz, M.J., Maninis, K.K., Popov, S., Ferrari, V.: Raytran: 3d pose estimation and shape reconstruction of multiple objects from videos with ray-traced transformers. In: Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part X. pp. 211–228. Springer (2022) 8, 10, 11, 22
45. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. NeurIPS **30** (2017) 2, 8, 22
46. Vu, T., Kim, K., Luu, T.M., Nguyen, X.T., Yoo, C.D.: Softgroup for 3d instance segmentation on 3d point clouds. In: CVPR (2022) 11, 27
47. Wu, R., Xiao, C., Zheng, C.: Deepcad: A deep generative network for computer-aided design models. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (2021) 4, 9
48. Xu, X., Willis, K.D., Lambourne, J.G., Cheng, C.Y., Jayaraman, P.K., Furukawa, Y.: Skexgen: Autoregressive generation of cad construction sequences with disentangled codebooks. In: International Conference on Machine Learning (ICML) (2022) 4
49. Xue, Y., Mao, J., Niu, M., Xu, H., Mi, M.B., Zhang, W., Wang, X., Wang, X.: Point2seq: Detecting 3d objects as sequences. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8521–8530 (2022) 4
50. Yang, K., Chen, X.: Unsupervised learning for cuboid shape abstraction via joint segmentation from point clouds. ACM Transactions on Graphics (TOG) **40**(4), 1–11 (2021) 12, 13
51. Yin, T., Zhou, X., Krahenbuhl, P.: Center-based 3d object detection and tracking. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (2021) 4

52. Yue, Y., Kontogianni, T., Schindler, K., Engelmann, F.: Connecting the dots: Floorplan reconstruction using two-level queries. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (2023) 10
53. Zou, C., Colburn, A., Shan, Q., Hoiem, D.: Layoutnet: Reconstructing the 3d room layout from a single rgb image. In: Proceedings of the IEEE conference on computer vision and pattern recognition (2018) 3

A *Aria Synthetic Environments*

Fig. 7: Randomly selected scenes from *Aria Synthetic Environments*. (top) Birds eye view renderings demonstrating room layouts and furniture arrangements. (bottom) Ego-centric close-up renderings showing scene details.

A.1 Large Scale Training Dataset

Aria Synthetic Environments consists of 100k training pairs with photo-realistically rendered indoor scenes coupled with structured language commands. In addition to these training sequences, *Aria Synthetic Environments* also provides an additional 1k scenes for testing. Figure 7 presents example scenes from the dataset. To the best of our knowledge, this is the largest synthetically generated and annotated dataset to date.

Specifically, a training pair for `SceneScript` consists of a 3D scene model represented through a rendered video sequence (input) and associated with a sequence of commands (ground truth). An example training pair for our method is shown in Figure 8.

Generation. Each of our synthetic indoor scenes are generated in the following way:

1. Start with:
   - A floor plan defining the layout of all rooms
   - A corresponding 3D model (room & object geometry, materials and illumination)
2. A trajectory is estimated going through the 3D scene model, simulating an agent walking around wearing an Aria sensor
3. A photo-realistic RGB rendering of the trajectory is generated with additional annotation data (depth and instance segmentation maps)

```
- make_wall, id=0, a_x=2.1, a_y=3.9, a_z=0.0,
  b_x=7.8, b_y=3.9, b_z=0.0, height=2.7
- make_wall, id=1, a_x=7.8, a_y=3.9, a_z=0.0,
  b_x=7.8, b_y=0.3, b_z=0.0, height=2.7
- make_wall, id=2, a_x=7.8, a_y=0.3, a_z=0.0,
  b_x=2.1, b_y=0.3, b_z=0.0, height=2.7
- make_wall, id=3, a_x=2.1, a_y=0.3, a_z=0.0,
  b_x=2.1, b_y=3.9, b_z=0.0, height=2.7
- make_door, id=4, wall0_id=1, wall1_id=-1,
  position_x=7.8, position_y=1.5, position_z=1.0, width=1.0, height=1.9
- make_window, id=5, wall0_id=2, wall1_id=-1,
  position_x=5.3, position_y=0.3, position_z=1.4, width=2.3, height=2.5
- make_window, id=6, wall0_id=3, wall1_id=-1,
  position_x=2.1, position_y=2.1, position_z=1.4, width=2.2, height=2.1
```

Fig. 8: Example of training data pair. A scene with objects is shown on the left, while the respective GT `SceneScript` language description is shown on the right.

4. A *SLAM*-based point cloud inferred from the synthetic video sequence and aligned with the initial 3D scene model/floor plan

Table 4: Comparison between existing indoor datasets. P-R: Photo-realistic; Ego: Egocentric; Camera: Either Fisheye(F) or Pinhole(P), or panorama photo(360); Seg: object and layout segmentations rendered to images; L-GT: Layout entity ground-truth, such as individual wall/window/door parameters.

Type	Name	Scenes	Trajectory	P-R	Ego	Camera	Depth	Seg	L-GT	License
Syn	ASE (**Ours**)	100k	✓	✓	✓	F	✓	✓	✓	Agreement needed
	ProcTHOR	10k	×	×	✓	P	✓	✓	✓	Apache2.0
	HyperSim	461	✓	✓	✓	P	✓	✓	×	Special license
	Structured3D	3,500	×	✓	✓	P	✓	✓	✓	MIT
	SceneNet RGB-D	57	✓	✓	✓	P	✓	✓	×	Special license
	InteriorNet	10k/1.7M	✓	✓	✓	F, P	✓	✓	×	Agreement needed
Real	Zillow Indoor	2,564	×	✓	×	360	×	×	✓	Apache 2.0
	HM3D	1,000	×	✓	×	P	✓	×	×	MIT
	ScanNet	1,513	✓	✓	×	P	✓	✓	×	Special license (data)

Dataset Properties. An overview of properties of existing indoor datasets is given in Table 4. Note that we define ego-centric as wearing a real camera on the head or chest or having a synthetic camera with similar trajectory. In particular, we follow the Aria ego-centric specifications [37]. Also note that *InteriorNet* contains 15k sequences rendered from 10k randomly selected layouts and 5M images from 1.7M randomly selected layouts, with 3 images per layout. *Aria Synthetic Environments* is especially useful for machine learning tasks not just due to its

sheer scale (allowing algorithms to generalize well), but also the wide variety of available ground truth annotations. For example, the layout ground truth (L-GT) is critical for training `SceneScript`, but not included in the majority of other datasets.

Aria Synthetic Environments is made publicly available to the research community. Users must agree to a standard licence to prevent data misuse and it is to be used for non-commercial purposes only.

A.2 Semi-dense Point Clouds from Video Sequences

We utilize the open-source Machine Perception Services (MPS) from Project Aria [1] to estimate a SLAM trajectory and generate a point cloud map of the scene. Similarly to LSD-SLAM [11] they maximize the extracted geometric information from a video sequence by estimating points for all image regions with non-negligible gradient. Each point is parameterized by an inverse distance and its associated Gaussian uncertainty in the frame in which it is first observed. KLT-based [42] epipolar line-searches in subsequent frames provide sub-pixel accurate short and large-baseline measurements that are absorbed using a Kalman filter update. While points are associated with a final estimated uncertainty, they consider utilizing this information in a probabilistically-sound way as beyond the scope of their work, and instead choose to sub-select points whose uncertainty is below a predefined threshold.

B Structured Language Commands

Command parameters can have data types such as `float` or `int`. The full list of parameters for each command can be found in Table 1 of the main paper. Below, we provide detailed descriptions of each parameter:

- Wall parameters
  - `id`: The ID of the wall.
  - `(a_x,a_y,a_z)/ (b_x,b_y,b_z)`: The x, y, z coordinates of the first / second corner of the wall.
  - `height`: The height of the wall. Note that we assume the walls are straight and gravity-aligned.
- Door/Window parameters
  - `id`: The ID of the door/window.
  - `wall0_id,wall1_id`: The IDs of the (potentially two) walls that a door/window is attached to.
  - `position_x,position_y,position_z`: The x, y, z coordinates of the centre of the door/window.
  - `width, height`: The width and height of the door/window.

C Network Architectures

C.1 Point Cloud Encoder

The point cloud encoder is essentially a ResNet-style [16] encoder that employs sparse 3D convolutions [40, 41] in place of standard 3D convolutions. It uses a total of 5 down convolutional layers with a kernel size of 3 and a stride of 2. This architecture effectively reduces the number of points (i.e. active sites) by ≈ 1000. As a result, the feature sizes are manageable in terms of size and can be used effectively in the subsequent Transformer decoder [45]. The point cloud encoder consists of $\approx 20M$ optimizable parameters, which contribute to its capacity and ability to capture intricate geometric information.

C.2 Transformer Decoder

Our implementation includes a transformer decoder consisting of 8 layers, each with 8 heads for multi-head attention, and a feature dimension of 512. This configuration results in a relatively modest set of $\approx 35M$ parameters. Our vocabulary size is 2048, which we also use as the maximum sequence length of tokens. While we could theoretically increase the vocabulary size to accommodate a larger number of tokens, in practice the majority of the released rendered scenes can be accurately represented using significantly fewer tokens than 2048.

In some scenarios, we employ nucleus sampling [17] with a top-p probability mass during autoregressive inference. By using nucleus sampling, we limit the selection of next tokens to a subset with a cumulative probability threshold, allowing for greater exploration at inference time. Quantitative results are decoded greedily.

D Training Methodology

We use the AdamW optimizer with a default initial learning rate of 10^{-4} and weight decay as well as dropout enabled. For the image-only Raytran-based encoder model [44], we found that an initial learning rate of 10^{-3} provided better convergence. We train all our methods with an effective batch size of 64, which may be distributed across multiple nodes. During training, the only augmentations we perform are: 1) up to 360 degrees of rotation around the z-axis (since the scenes are assumed to be gravity-aligned), and 2) random subsampling of point clouds up to a specificed maximum number of points (500k). Training times lasted approximately between 3 and 4 days.

Our training loss is the standard cross-entropy loss on next token prediction, similar to how most LLMs are trained [30, 39].

E Tokenization Details

The conversion of a parameter `x` into an integer token `t` is determined by two factors: its data type and its magnitude. In general, parameters with `int` and

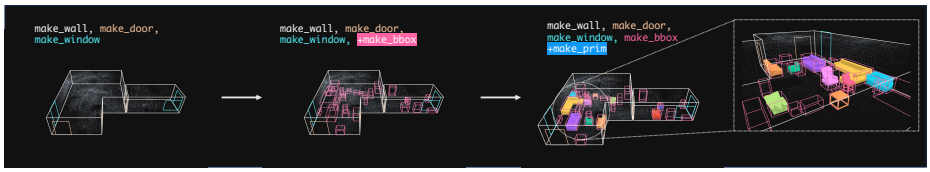

Fig. 9: An illustrative example of how the expressiveness of SceneScript's reconstruction increases through the addition of new commands. (left) Layout commands only: walls, doors and windows. (middle, left) Addition of `make_bbox` enriches the scene reconstruction with bounding boxes for detected objects. (middle, right) Addition of `make_prim` adds volumetric primitives for detected chairs, sofas and tables. (right) Close-up illustrating the fidelity possible with just these five commands.

`bool` data types are converted using the `t = int(x)` operation. For `float` parameters, the conversion involves `t = round(x/res)`, where `res` represents the resolution. Note that by designing the `SceneScript` language, we also design the tokenization. This is notably different from standard NLP tokenization, which involves *B*yte-*P*air *E*ncodings (*BPE*) [30].

F Additional Results: Layout Prediction

In this section, we present additional qualitative results, comparisons with related works and more evaluations with respect to extending `SceneScript`.

F.1 Visualization of High-Level Commands

Figure 9 presents a visual example of how the fidelity of `SceneScript`'s reconstruction can be increased through the addition of new commands. Initially, structural room layouts are represented by three commands: `make_wall`; `make_door`; and `make_window`. Just through the addition of `make_bbox`, scene content is now present in the reconstruction in the form of object detections. Finally for the commands discussed in the main paper, `make_prim` for the three selected target classes enables not just the capture of the scene's overall structure and content, but also much finer reconstruction of scene objects themselves.

Importantly, each of these levels of detail is enabled without change to `SceneScript`'s network architecture, and instead just by increasing the expressiveness of the structured language it infers.

Note that the volumetric primitive commands for detected objects are a proof of concept. We trained our models for the object primitive commands only on a subset of the available object types from the *Aria Synthetic Environments*. Supported object class labels are "*chair*", "*sofa*" and "*table*". Objects with these labels are modeled by cuboid and cylinder primitives. Detected bounding boxes of object instances with unsupported classes remain empty.

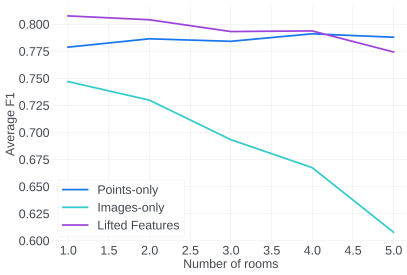

Fig. 10: F1-Score model performance graphs for our various encoder variants as functions of the number of rooms in a scene.

F.2 Model Performance with respect to Scene Complexity

The Average F1-score graphs of Figure 10 demonstrate the performance of our `SceneScript` model with varying encoders as a function of the number of rooms in a scene. Our `SceneScript` model performs constantly well when inputting points only or lifted features. As opposed to this, performance drops drastically with increasing room number when encoding scenes only using images. We posit that the decrease in performance is due to the model's lack of occlusion reasoning. With increasing number of layout elements, the rays linked to image observations traverse more scene space by going through a higher number of rooms when ignoring wall intersections. This likely results in our model falsely attending to occluded image observations.

F.3 Failure Cases

In this section, we detail observed failure types for the task of layout estimation on *Aria Synthetic Environments*. Aside from expected errors such as slightly incorrect wall corner, window and door placement, or entirely missed, we observe two notable failure modes for `SceneScript`.

The more common of the two occurs due to non-complete exploration of the target scene. In this scenario there are significant portions of the scene structure that are poorly observed, potentially not at all, making the ground truth structure near unpredictable.

An especially interesting failure mode is the reconstruction of accurate room structure, but at an incorrect Z-value. For the point cloud-based encoder configurations, we suspect that this failure mode is caused by particular sensitivity to noise to point outliers in the Z-direction. This failure mode is also observed in the image-only encoder configuration, suggesting it also exhibits more sensitivity to in the Z direction than XY.

We visualize a couple of examples for each of these failure types in Figure 11. Worth noting is that this figure is comprized of scenes taken from the worst 10 predictions out of the 1000 scene test set, as defined by wall entity distance.

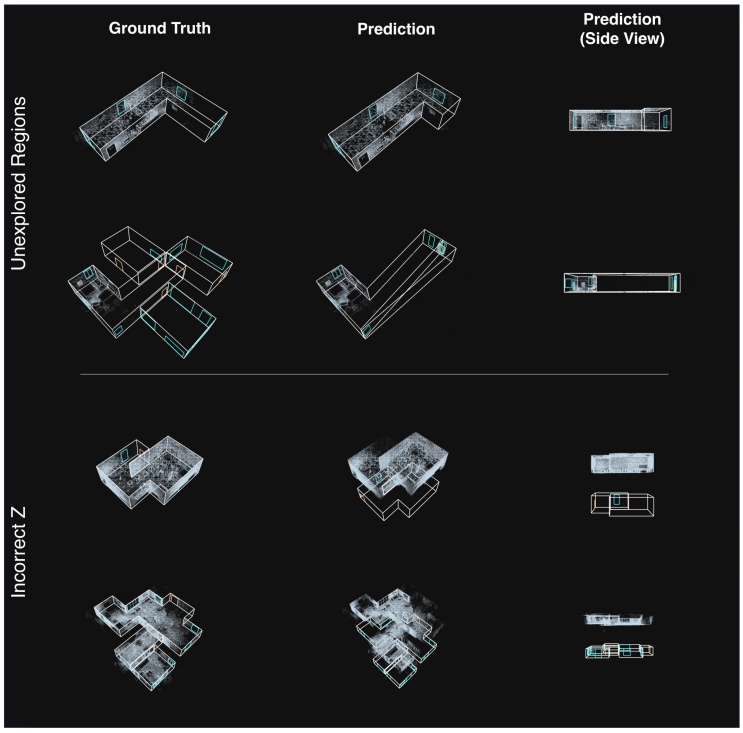

Fig. 11: Examples from two notable failure types observed in `SceneScript`predictions. (top) Limited exploration of the scene makes the ground truth difficult, or in some cases potentially impossible to predict. (bottom) Accurate overall room structure is predicted, but at an incorrect Z value.

Therefore, while clearly illustrating the failures described, they should not be taken as representative of general prediction quality.

F.4 Quantitative Evaluation of Layout Predictions

We include an additional breakdown of the entity distance accuracy metrics in Table 5. This breakdown of accuracy makes apparent that the improvement offered by lifting image features onto the semi-dense point cloud comes largely in the prediction of windows and doors. Following the same trend as the results included in the main paper, we observe that windows appear to be the most challenging class to predict accurately. However in spite of this challenge, the 90th percentile of window predictions falls within 0.5m of the ground truth for all encoder setups tested.

Table 5: Accuracy reported as raw entity distance for the encoder setups tested for `SceneScript`.

	Entity Distance (cm)					
	Wall		Door		Window	
Method	med.	p90	med.	p90	med.	p90
Point Cloud	4.7	7.2	5.0	6.7	6.9	37.6
Lifted Features	4.8	7.1	4.8	6.1	5.9	26.2
Image-only	6.7	17.3	5.8	8.9	9.0	45.7

G Additional Results: Object Detection

G.1 Implementation Details

Training Details of `SceneScript`. As outlined in the paper, a significant advantage of `SceneScript` lies in its seamless adaptability to other tasks through the addition of new language commands. Here, for instance, we integrate `make_bbox` to denote 3D oriented bounding boxes.

Notably, no architectural changes to `SceneScript` have been implemented to facilitate training for object detection. We utilize the point cloud encoder and language decoder detailed in Section C. The entire training objective is a single cross-entropy loss, which stands as the de facto-standard in training LLMs. The model is trained for $\approx 200k$ iterations using an effective batch size of 64. For this experiment, we only trained a point cloud version of SceneScript.

Baseline Implementation Details. We list implementation details for each method below:

3DETR [26]: We downloaded the weights for both 3DETR and 3DETR-m, trained for 1080 epochs on ScanNet, from the official Github repository. We evaluated both models on each ScanNet validation examples, subsampled to $40k$ points. Predictions were thresholded at probability 0.5. We attemped to run NMS in 3D, but achieved worse results, thus the numbers reported in the main paper do not include NMS.

We trained 3DETR (not 3DETR-m) on *Aria Synthetic Environments* using almost the same configuration as trained on ScanNet. The differences include: a batch size of 128, a PointNet set aggregation downsampling to 4096 (compared to 2048 for ScanNet), 512 furthest point samples as detection queries (compared to 256 for ScanNet), and 200k points per example.

Cube R-CNN [4]: This method predicts 3D bounding boxes from a single RGB image. To obtain 3D bounding box predictions for an entire scene, we accumulate per-frame predictions via a matching-based tracker. At a high-level, we match predictions of the same class between frames with Hungarian matching with a cost based on IoU and bounding box distances. Then the final bounding

box parameters are computed as a running average of the matched and tracked predictions. For evaluation, the accumulated predicted boxes were thresholded at probability 0.5.

ImVoxelNet [36]: This model predicts 3D bounding boxes from a set of RGB images. We trained this method using 10 consecutive frame snippets from *Aria Synthetic Environments*. During evaluation, we run the model on overlapping 10-frame snippets and apply the same bounding box tracker as described for Cube R-CNN. For evaluation, the accumulated predicted boxes were thresholded at probability 0.1.

SoftGroup [46]: Since this is primarily a 3D semantic instance segmentation method, we extract axis-aligned bounding boxes from the predictions by utilizing the predicted instance masks and computing the *minimum* and *maximum* extents of the point set belonging to each instance. The geometric mean of these extents serves as the box center, and the difference between the maximum and minimum extents provides the box scale. Since the bounding boxes are intended to be axis-aligned, the angle is kept at 0. By combining this information with the predicted semantic class, one can conduct evaluations over 3D bounding boxes. We used a publically available checkpoint provided by the authors to conduct inference and extract bounding boxes for evaluation following the aforementioned procedure.

Note that for ScanNet [10], we use the axis-aligned bounding boxes for ground truth as extracted in [26, 35].

G.2 Sparse Encoder with 3DETR Head

Table 6: Replacing the 3DETR encoder with a SparseCNN results in better performance on *Aria Synthetic Environments*.

Method	Input	F1 @.25 IoU	F1 @.50 IoU
3DETR '21 [26]	Points	0.201	0.078
SparseCNN [40] + 3DETR [26]	Points	0.381	0.191

We run an experiment that confirms that 3DETR's standard settings are well-suited to ScanNet [10] and SUN RGB-D [38], but perform poorly on *Aria Synthetic Environments*. For this experiment, we use the same sparse point cloud encoder that SceneScript uses (see Section C.1) while using the 3DETR decoder. Similar to the pure 3DETR model trained on *Aria Synthetic Environments*, we increased the number of detection queries to 512, and used 200k points per example for training. We denote this model as SparseCNN+3DETR. Due to lack of resources, this model was only partially trained.

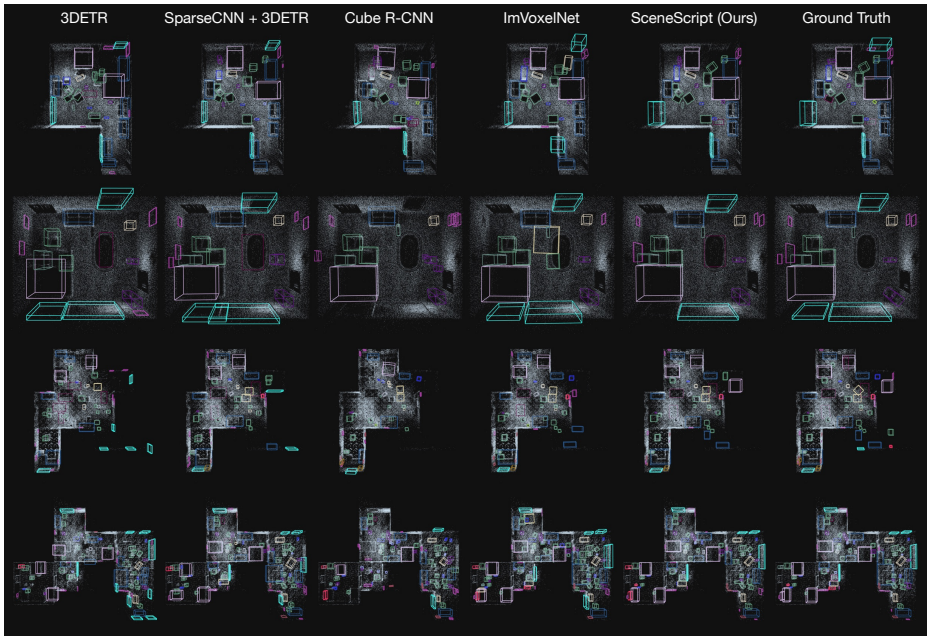

Fig. 12: Qualitative results of predicted bounding boxes on *Aria Synthetic Environments*. Each bounding box is colored by its class. The colors are: table, sofa, shelf, chair, bed, floor_mat, exercise_weight, cutlery, container, clock, cart, vase, tent, flower_pot, pillow, mount, lamp, ladder, fan, cabinet, jar, picture_frame, mirror, electronic_device, dresser, clothes_rack, battery_charger, air_conditioner, window.

In Table 6, we show that replacing 3DETR's Transformer encoder with a sparse CNN encoder [40, 41] results in stronger performance. We hypothesis that this is due to the non-uniformity of the point clouds arising from Project Aria's semi-dense point clouds from its Machine Perception Services [1]. The first two columns of Figures 12 and 13 qualitatively demonstrates more accurate predictions with this encoder replacement.

G.3 Qualitative Results on *Aria Synthetic Environments*

In Figure 12, we show qualitative results of all the methods trained on *Aria Synthetic Environments*. This figure demonstrates the difficult of predicting objects in *Aria Synthetic Environments* as it is very cluttered. Also, due to the generated trajectories not necessarily visiting every part of the scene, some ground truth bounding boxes have very little points associated with them (see the ground truth in row 3. The lower right corner has very few points yet there are bounding boxes present).

Most methods tend to correctly predict the larger categories (e.g. bed and sofa). However, the small object categories (e.g. jar and flower_pot) are much harder to detect, thus the ground truth for these categories typically have 0

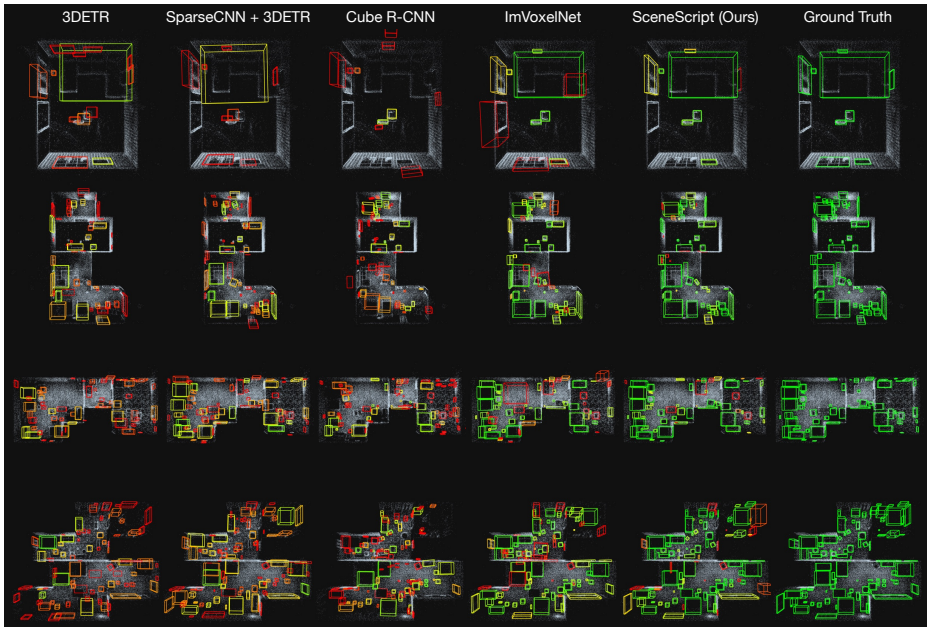

Fig. 13: Qualitative results of predicted bounding boxes on *Aria Synthetic Environments*. Each bounding box is colored by its IoU with its matched ground truth bounding box. The color is interpolated from green (IoU = 1.0) to yellow (IoU = 0.5) to red (IoU = 0).

IoU with predictions (see Figure 13 for qualitative predictions visualised with IoU scores). This leads to relatively low F1 scores for some of the baselines (e.g. 3DETR) due to averaging the F1 scores across classes, while visually the predictions look relatively reasonable. We also include results from SparseCNN+3DETR (details can be found in Section G.2). It can be seen from Figures 12 and 13 that it qualitatively performs better on *Aria Synthetic Environments* than a pure 3DETR model.

Table 7: mAP for baselines trained on *Aria Synthetic Environments*.

Method	Input	mAP @.25 IoU	mAP @.50 IoU
3DETR '21 [26]	Points	0.148	0.040
SparseCNN [40] + 3DETR [26]	Points	0.308	0.115
Cube R-CNN '23 [4]	RGB	0.383	0.181
ImVoxelNet '22 [36]	RGB	0.648	0.572

G.4 mAP Metrics for Baselines trained on *Aria Synthetic Environments*

In Table 7, we list the mAP values for methods trained on *Aria Synthetic Environments*.

G.5 Discussion of Average Precision Metric

Average precision (AP) has become a standard metric for measuring 3D object detection performance. A general outline of the procedure required to calulate this metric is to collect detections across a number of scenes, rank each by descending confidence. Average precision is then computed from this detection pool by framing it as an information retrieval task: order of retrieval determined by the confidence ranking; success of a retrieval determined by an IoU threshold (typically 0.25 or 0.5 for 3D object detection). This framing enables the generation of a precicision-recall curve for the detector, with the average precision given by an approximation of the area underneath this curve.

A drawback of this information retrieval framing is that it is order variant, and requires that the relative certainty of detections across scenes be determinable. While many prior detection methods regress a logit that can naturally represent this certainty, *e.g.* the classification logit is often used, `SceneScript`'s detections are more binary: either the object is present in the predicted sequence or not. Within a single scene, we may be able to leverage a heuristic such as sequence order to determine relative certainty, *i.e.* most certain detections appear sooner in the prediction order (although we have not investigated whether this actually occurs). However, to determine a similar heuristic between scenes would require too many assumptions to be considered a robust and fair evaluation configuration.

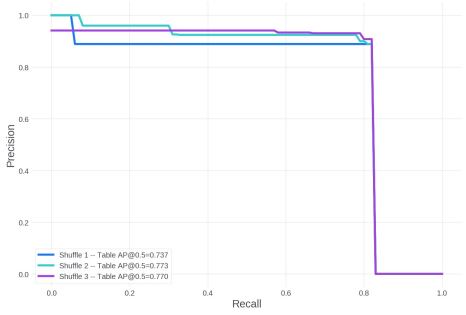

Fig. 14: Precision-recall variance with scene order. The precision-recall curves are plotted for `SceneScript`'s predictions of the *table* class on the same 10 scenes, however the order of those scenes is shuffled for each evaluation. The inability sort predictions across scenes leaves the AP@0.5IoU metric sensitive to the order that scenes are evaluated.

Table 8: Illustration of how average precision is negatively affected by the inability to sort across scenes. Two idential sets of detections are produced by detectors 1 and 2. Detector 1 outputs an absolute measure of confidence allowing for sorting across scenes. However, it is only possible to determine the relative confidence of predictions within a scene for detector 2. This results in a lower AP, as there is no opportunity to rank good predictions from scene B above bad predictions from scene A. We assume there are 3 GT entities in each scene for AP and F1 computation.

	Detector 1 w/ absolute conf.						Detector 2 relative conf. only					
Scene	A	B	B	A	B	A	A	A	A	B	B	B
Certainty	high	high	high	med.	low	low	-	-	-	-	-	-
Success	1	1	1	0	0	0	1	0	0	1	1	0
AP	0.5						0.34					
F1	0.5						0.5					

To further illustrate this point, we consider an evaluation setup where we use prediction order within scenes as a proxy for relative certainty, without sorting across scenes. In Figure 14 we show precision-recall curves computed over 10 scenes from *Aria Synthetic Environments* validation set using the assumption. Importantly, each curve on this graph are the *same detections on the same scenes*, but with the scenes simply evaluated in a new, random order each time. Not only is the resulting metric variant with the order of scenes, but low certainty predictions at the end of ascene's predictions may appear earlier in the ranked pool of detections than high certainty predictions from another scene. If these are incorrect, they will arteficially lower the precision achievable, and in turn lower the average precision for a method. A toy example of this is included in Table 8.

For these reasons, in the main paper we choose to use a F1-score-based metrics to evaluate detection performance. These are not sensitive to ordering as also illustrated in Table 8.

H Further Extensions of SceneScript

H.1 Extending `SceneScript` with Curved Entities

In previous layout estimation work, such as [13, 22], methods leverage a planar assumption and use robust estimation procedures custom tailored to planar primitive fitting. Extending such methods to more complex, non-planar entities is non-trivial and requires significant effort. In contrast, we show that our structured language-based approach makes it straightforward to extend to curved walls, as for example extruded Bezier curves [23], simply by defining a new `make_curved_wall` command.

The curved wall command is a simple Bezier parametrisation consisting of 4 additional parameters: the x, y values of the 2 control points that define the wall curvature. Explicitly, our planar wall command changes to:

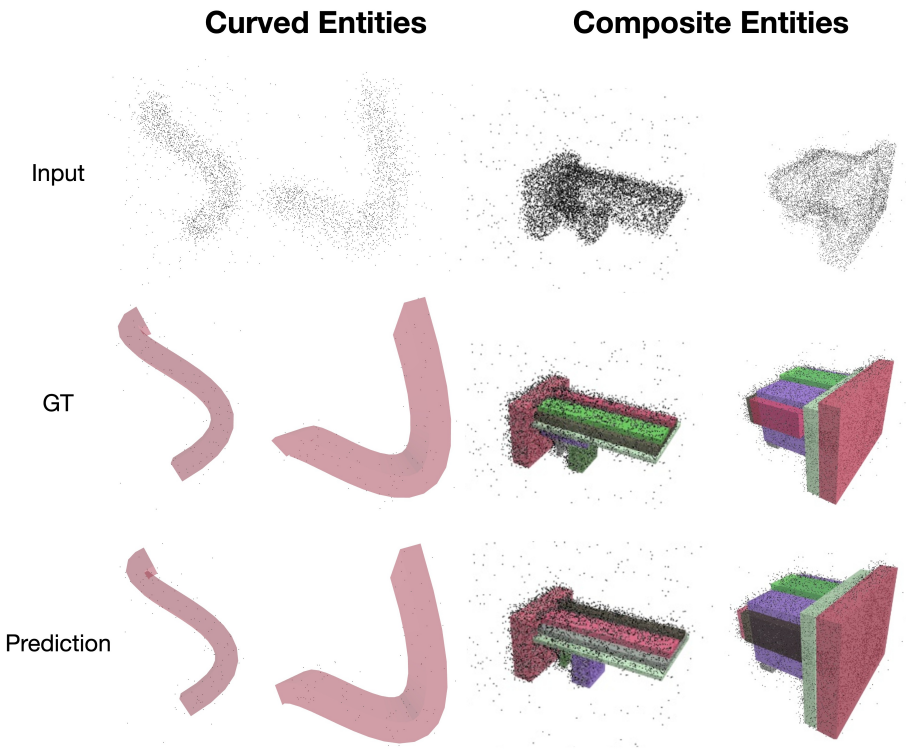

Fig. 15: Non-planar wall geometry extensions to `SceneScript`. Examples of input point clouds (top row), the prediction 3D shape (middle row), and ground truth wall shape (bottom row). (left) Examples of Bezier parameterisation for curved walls. (right) Results for wall primitive compositions. We observe that both simple extensions to the parameterisation of the walls can be accurately described and predicted by SceneScript.

```
make_curved_wall: a_x, a_y, a_z, b_x, b_y, b_z, c1_x,
    c2_y, c2_x, c2_y, height, thickness
```

where $c_{1_x}, c_{1_y}, c_{2_x}, c_{2_y}$ are the Bezier control points.

We generate a synthetic curved walls dataset to train `SceneScript`. Example Bezier walls with a qualitative evaluation are in Figure 15 (left). The predictions are nearly indistinguishable compared to ground truth, indicating that our method can learn to predict such complex primitives.

In a synthetic test bed, we evaluate the capability of our model to infer the control points of walls parameterized on extruded Bezier curves. Quantitative results are shown in Table 9.

H.2 Extending `SceneScript` to Compositions of Wall Primitives

To demonstrate the extensibility of `SceneScript`'s structured language, and similarly to the reconstruction of object primitives explored in the main paper,

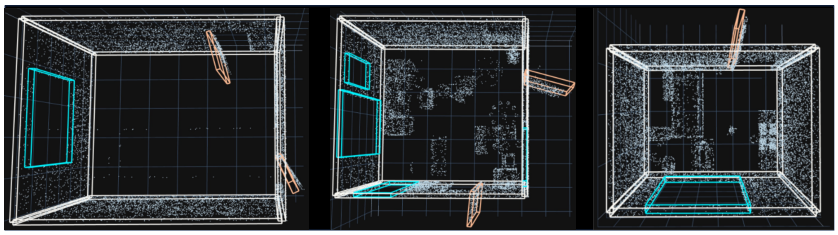

Fig. 16: Results for detecting door state estimation. We visualize the predicted layout on top of the input point cloud.

Table 9: Quantitative assessment of the reconstruction of curved walls using extruded Bezier curves as parameters. Token accuracies gauge performance based on a tokenized 1D sequence of the structured language, allowing for a specified slack of +/- N tokens. The IOU is calculated by comparing the interpreted geometry with the GT geometry. We achieve virtually error-free results indicating efficient interplay between parameterization and modelling capability of our method.

Token Acc. Slack 1	Token Acc. Slack 3	IOU
0.993	1.0	0.990

we demonstrate representing complexly shaped walls as compositions of cuboids. We define a new parametrization for this class of walls as follows:

```
make_wall: a_x, a_y, ...
make_wall_prim: pos_x, pos_y, pos_z, size_x, size_y,
    size_z
```

where the `make_wall_prim` command describes a cuboid to be composed with its parent wall entity. We added such cuboid compositions to a base wall in Figure 15 (right). In this proof-of-concept, the results of Table 10 clearly demonstrate the ability of the network to infer compositions of cuboids on base walls only from a noisy surface point cloud.

Table 10: Correctly predicted parameters of composite walls as a percentage . Slack n indicates estimation of composite wall parameters within bounds of $n * 5cm$.

		Occlusion Levels		
		No	Light	High
Slack	1	99.6	95.9	92.6
	3	99.9	96.4	93.3
	5	100	98.2	95.5

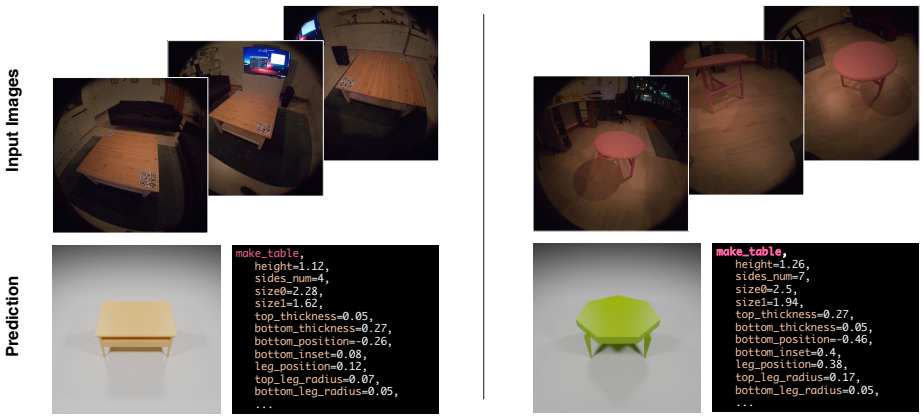

Fig. 17: Two real-world inferences based on a Blender Geometry Node [3] obtained online. Input RGB images are recorded on an Aria device. We visualize a subset of the predicted language as well as the geometry obtained by inputting that prediction into the Geometry Node.

H.3 Extending `SceneScript` to Object States

Yet another simple extension to our `SceneScript` language allows us to represent door states, w.r.t their opening configuration. For this, we simply change the original door representation to include a list of parameters that define door state as follows:

```
make_door: id, wall_id, pos_x, pos_y, pos_z, width,
    height, open_degree, hinge_side, open_direction
```

`hinge_side` represents which side of the door the hinge is on, `open_direction` determines whether the door opens into the room or outside of the room, and `open_degree` is the angle of the opening. In Figure 16 (second), we qualitatively demonstrate object state estimation. We annotated our doors with a new command parameterisation extended by door hinge position, wall opening side and opening angle. As with our other extensions, our model is able to handle this situation without issue. This small GT language extension demonstrates effective state estimation while the input and network architecture remain unchanged.

H.4 Extending `SceneScript` to Blender Parametric Object Models

Parametric modelling offers detailed high-quality geometry representations along with interpretability and editability by design [18–20, 33]. The Blender community offers readily accessible Geometry Nodes of diverse object categories as a procedural language. We investigate the use of a particular Geometry Node for tables [3]. Not only can we directly incorporate this parametric model into our `SceneScript` language, but we can also use it to generate data by randomly sampling its parameters similar to [33].

We design a simple proof-of-concept experiment where we render synthetic RGB images of random tables, composite them on a random image background, and learn to predict the ground truth Blender procedural language. In Figure 17, we demonstrate two real-world inferences of tables using this language, showing our method is capable of predicting reasonable parameters to reconstruct these tables. Interestingly, in the second example the model predicts a high `sides_num` to approximate the circular tabletop, which was not on the training set.