SceneScript: Supplementary Material

A *Aria Synthetic Environments*

Fig. 1: 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 1 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 2.

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

```
- 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. 2: 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.

Table 1: 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 1. 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 [19]. 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 [5] 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 [24] 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 [7] encoder that employs sparse 3D convolutions [22, 23] 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 [26]. 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 [8] 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 [25], 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 [15, 21].

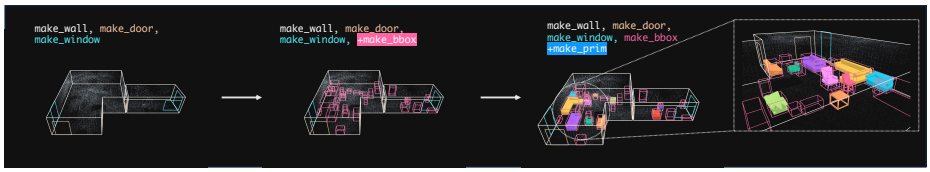

Fig. 3: An illustrative example of how the expressiveness of SceneScript's reconstruction increases through the addition of new commands. (left) Layout commands only: walls, doors and windows. (middle, left) Addition of `make_bbox` enriches the scene reconstruction with bounding boxes for detected objects. (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.

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 `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*) [15].

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 3 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

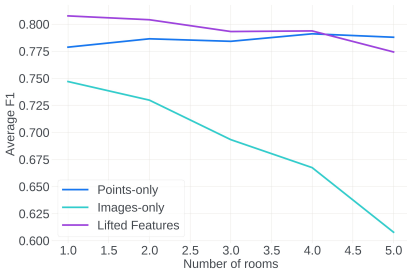

Fig. 4: F1-Score model performance graphs for our various encoder variants as functions of the number of rooms in a scene.

labels are modeled by cuboid and cylinder primitives. Detected bounding boxes of object instances with unsupported classes remain empty.

F.2 Model Performance with respect to Scene Complexity

The Average F1-score graphs of Figure 4 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.

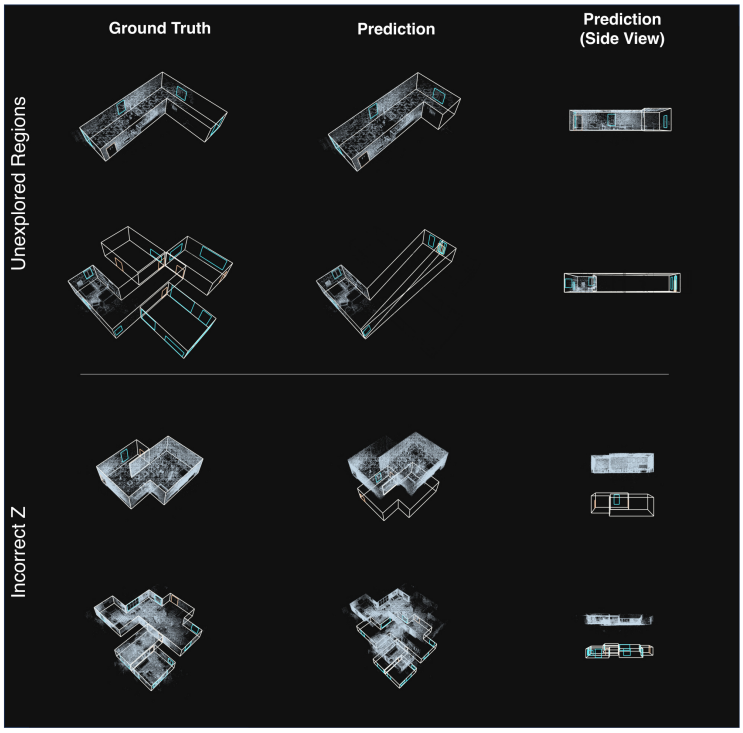

Fig. 5: Examples from two notable failure types observed in `SceneScript`predictions. (top) Limited exploration of the scene makes the ground truth difficult, or in some cases potentially impossible to predict. (bottom) Accurate overall room structure is predicted, but at an incorrect Z value.

We visualize a couple of examples for each of these failure types in Figure 5. 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. 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 2. 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 2: 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 [14]: 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 [3]: 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 [18]: 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 [27]: 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 [4], we use the axis-aligned bounding boxes for ground truth as extracted in [14, 17].

G.2 Sparse Encoder with 3DETR Head

Table 3: 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 [14]	Points	0.201	0.078
SparseCNN [22] + 3DETR [14]	Points	0.381	0.191

We run an experiment that confirms that 3DETR's standard settings are well-suited to ScanNet [4] and SUN RGB-D [20], 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.

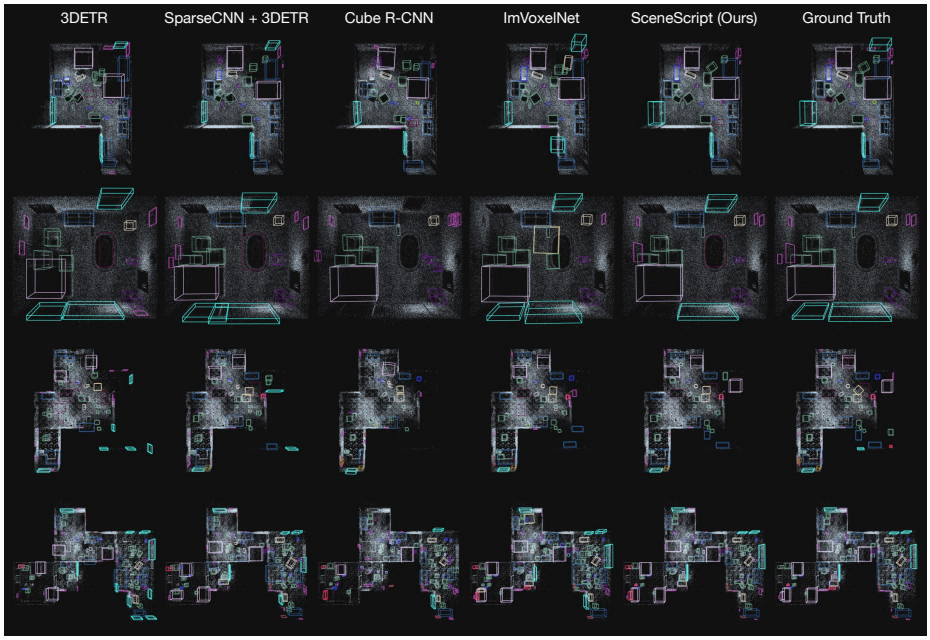

Fig. 6: Qualitative results of predicted bounding boxes on *Aria Synthetic Environments*. Each bounding box is colored by its class. The colors are: table, sofa, shelf, chair, bed, floor_mat, exercise_weight, cutlery, container, clock, cart, vase, tent, flower_pot, pillow, mount, lamp, ladder, fan, cabinet, jar, picture_frame, mirror, electronic_device, dresser, clothes_rack, battery_charger, air_conditioner, window.

In Table 3, we show that replacing 3DETR's Transformer encoder with a sparse CNN encoder [22,23] 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 6 and 7 qualitatively demonstrates more accurate predictions with this encoder replacement.

G.3 Qualitative Results on *Aria Synthetic Environments*

In Figure 6, 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

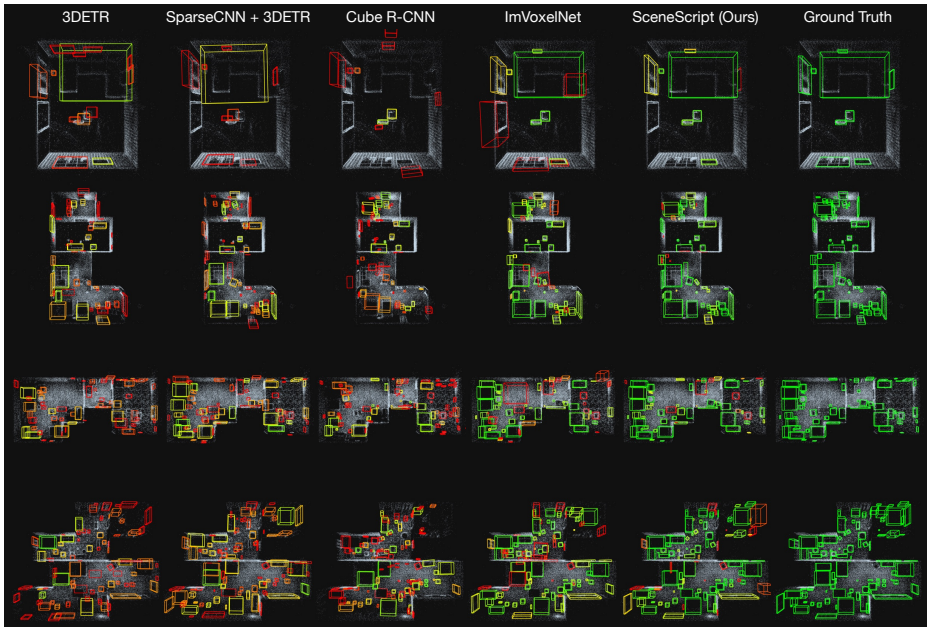

Fig. 7: Qualitative results of predicted bounding boxes on *Aria Synthetic Environments*. Each bounding box is colored by its IoU with its matched ground truth bounding box. The color is interpolated from green (IoU = 1.0) to yellow (IoU = 0.5) to red (IoU = 0).

IoU with predictions (see Figure 7 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 6 and 7 that it qualitatively performs better on *Aria Synthetic Environments* than a pure 3DETR model.

Table 4: mAP for baselines trained on *Aria Synthetic Environments*.

Method	Input	mAP @.25 IoU	mAP @.50 IoU
3DETR '21 [14]	Points	0.148	0.040
SparseCNN [22] + 3DETR [14]	Points	0.308	0.115
Cube R-CNN '23 [3]	RGB	0.383	0.181
ImVoxelNet '22 [18]	RGB	0.648	0.572

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

In Table 4, 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.

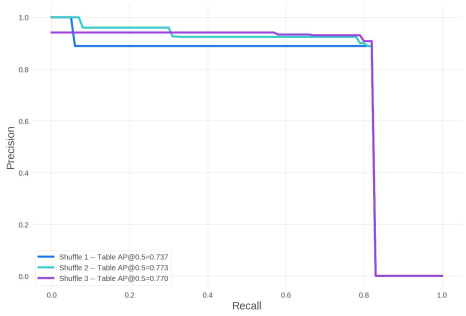

Fig. 8: Precision-recall variance with scene order. The precision-recall curves are plotted for `SceneScript`'s predictions of the *table* class on the same 10 scenes, however the order of those scenes is shuffled for each evaluation. The inability sort predictions across scenes leaves the AP@0.5IoU metric sensitive to the order that scenes are evaluated.

Table 5: 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 8 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 5.

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 5.

H Further Extensions of SceneScript

H.1 Extending `SceneScript` with Curved Entities

In previous layout estimation work, such as [6, 12], 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 [13], 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:

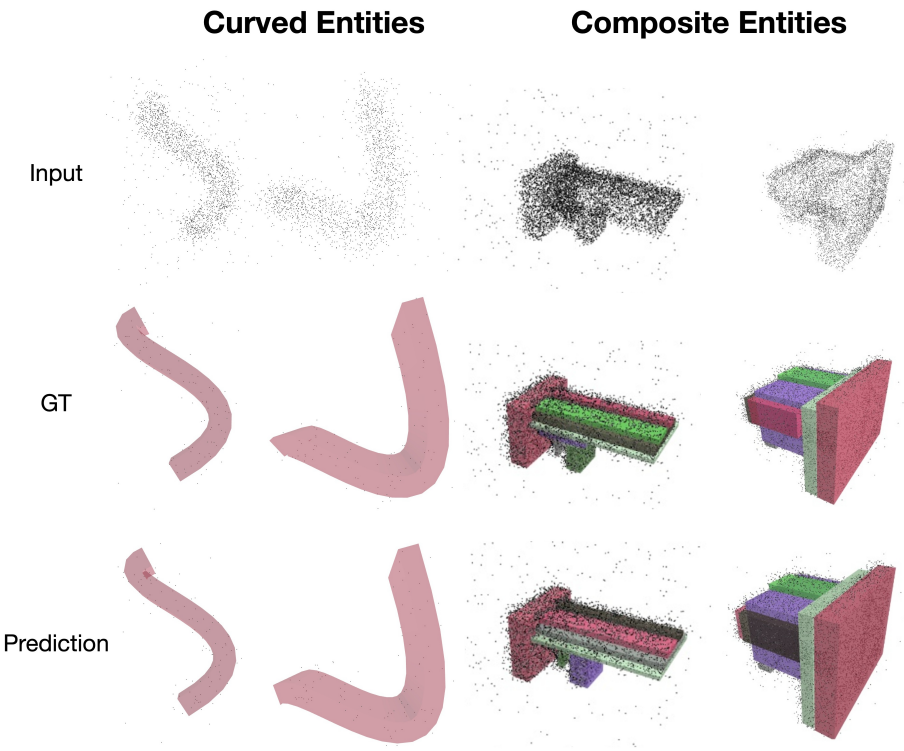

Fig. 9: Non-planar wall geometry extensions to `SceneScript`. Examples of input point clouds (top row), the prediction 3D shape (middle row), and ground truth wall shape (bottom row). (left) Examples of Bezier parameterisation for curved walls. (right) Results for wall primitive compositions. We observe 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 9 (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 6.

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,

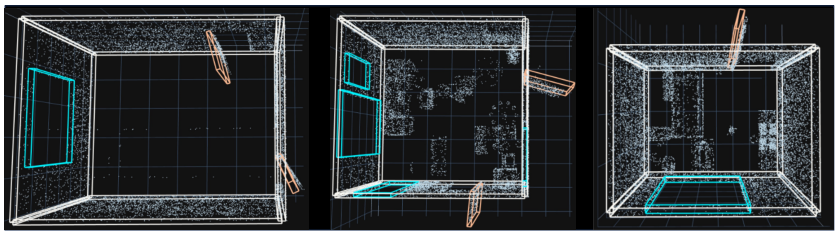

Fig. 10: Results for detecting door state estimation. We visualize the predicted layout on top of the input point cloud.

Table 6: 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 9 (right). In this proof-of-concept, the results of Table 7 clearly demonstrate the ability of the network to infer compositions of cuboids on base walls only from a noisy surface point cloud.

Table 7: 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

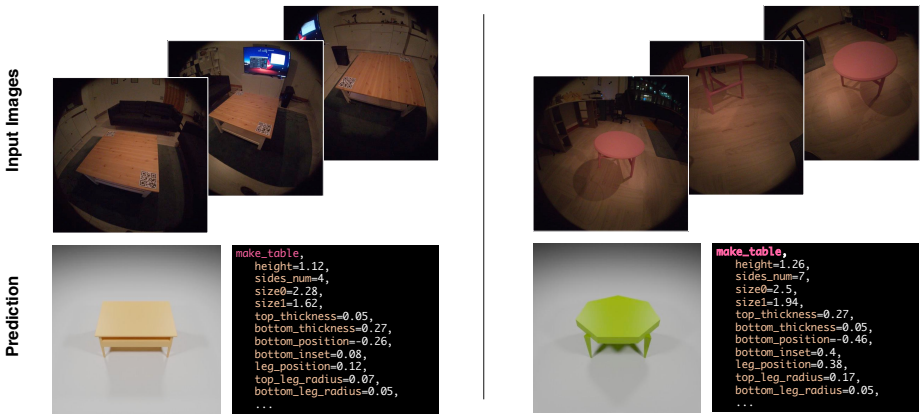

Fig. 11: Two real-world inferences based on a Blender Geometry Node [3] obtained online. Input RGB images are recorded on an Aria device. We visualize a subset of the predicted language as well as the geometry obtained by inputting that prediction into the Geometry Node.

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 10 (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 [9–11, 16]. 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 [2]. 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 [16].

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 11, 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.

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
2. Bash, B.: Procedural table with geometry nodes in blender (2023), `https://blenderbash.gumroad.com/l/daghsw`, accessed: 2023-05-23
3. 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)
4. 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)
5. Engel, J., Schöps, T., Cremers, D.: LSD-SLAM: Large-scale direct monocular SLAM. In: European Conference on Computer Vision (ECCV) (September 2014)
6. Furukawa, Y., Curless, B., Seitz, S.M., Szeliski, R.: Reconstructing building interiors from images. In: 2009 IEEE 12th international conference on computer vision (2009)
7. 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)
8. Holtzman, A., Buys, J., Du, L., Forbes, M., Choi, Y.: The curious case of neural text degeneration. arXiv preprint arXiv:1904.09751 (2019)
9. 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)
10. 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)
11. 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)
12. Liu, C., Kim, K., Gu, J., Furukawa, Y., Kautz, J.: Planercnn: 3d plane detection and reconstruction from a single image (2019)
13. Loop, C., Blinn, J.: Resolution independent curve rendering using programmable graphics hardware. In: ACM SIGGRAPH 2005 Papers, pp. 1000–1009 (2005)
14. 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)
15. OpenAI: Gpt-4 technical report (2023)

16. Pearl, O., Lang, I., Hu, Y., Yeh, R.A., Hanocka, R.: Geocode: Interpretable shape programs (2022)
17. 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)
18. 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)
19. 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)
20. 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)
21. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks (2014)
22. 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)
23. 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)
24. Tomasi, C., Kanade, T.: Detection and tracking of point. Int J Comput Vis **9**(137-154), 3 (1991)
25. 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)
26. 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)
27. 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)