Aligning Text, Images and 3D Structure Token-by-Token

Supplementary Material

Contents

A Dataset details . . . 1
A.1 Data generation . . . 1
A.2 3D scene serialization . . . 2
A.3 Tokenization . . . 3
A.4 Task sequence formation . . . 3
B Additional qualitative examples . . . 4
C Evaluation . . . 8
C.1 3D scenes . . . 8
C.2 Images . . . 9
C.3 3D assets . . . 12
C.4 Text . . . 12
D Additional implementation details . . . 12
D.1 Image VQGAN architecture . . . 12
D.2 3D VQ-VAE training . . . 13
D.3 Encoding of numbers . . . 14
D.4 Compute resources and time . . . 14
E Additional experiments and observations . . . 14
F Failure cases . . . 16

A. Dataset details

In this section, we present a detailed overview of the four datasets used in our experiments. We discuss dataset creation, statistics, serialization, tokenization, and task formulation.

A.1. Data generation

CLEVR. We generate CLEVR scenes using the dataset creation code from [5] and render the corresponding images with Blender. Each scene outputs a JSON file describing the scene along with its rendered image.

Rendering. We generate 120,000 unique CLEVR scenes as training data for rendering. The JSON files serve as input for the 3D scene representation, while the corresponding images are used to generate output sequences. Further details are provided in the following sections. For evaluation, we use a test set of 2,000 image-JSON pairs.

Recognition. We use the same training data as in rendering but with reversed input-output roles. Here, images serve as input sequences, while JSON files generate the 3D scene output. For evaluation, we use a test set of 2,000 JSON-image pairs.

Instruction-following. We consider four different types of instructions for 3D scene modification: (1) *modifying the appearance of objects*, (2) *adding new objects*, (3) *removing objects*, and (4) *moving an object to a desired location*. Using 16 to 28 text instruction templates, we generate 20,000 input-output pairs per instruction type that we build on top of the initial 20,000 CLEVR scenes. Specifically, we sample a CLEVR scene, apply an instruction template, and generate the corresponding modified scene. Additionally, for instructions involving object appearance modification, we generate an extra 20,000 pairs that do not reference other objects, ensuring modifications apply solely to uniquely identifiable objects within the scene. This results in a total of 100,000 input-output pairs forming the training set. For evaluation, we sample 500 input-output pairs per instruction type, creating a test set of 2,500 pairs. This approach enhances dataset diversity, improving the model's ability to generalize across

different instruction types. Example text instructions for each type are listed in Table 1.

Question-answering. For question-answer pair generation, we use the question generation engine by [5] that uses functional programs and generate 20,000 question-answer pairs for the training data. For evaluation, we use a test set of 2,000 question-answer pairs.

ObjaWorld. We extend the CLEVR framework to support 3D Blender assets from Objaverse [2], for inclusion of objects beyond basic geometric shapes such as cubes, cylinders, and spheres. Specifically, we adapt the CLEVR code to include objects like person, bird, bench, *etc.*.

First, for experiments showing generalization to complex scene layouts (Section 3.3), we consider two scene setups: *park* and *living room*. *Park* scenes are composed of the assets *person, bird, bench*, and *lamppost*, while *living room* scenes use *person, sofa*, and *table* to construct the scenes. For training, we generate 50,000 scenes for each setup, resulting in a total of 100,000 scenes. Our test set comprises 4,000 scenes, evenly split between 2,000 park and 2,000 living room scenes.

For experiments showing unified 3D shape and scene understanding (Section 3.4), we select 20 complex Objaverse objects like barrel, chicken, cheeseburger, *etc.*, and generate 100,000 training scenes containing 2-3 randomly sampled objects. The test set contains 1,000 unseen scenes.

Objectron. We adhere to the official dataset splits by [1] to construct our training and test sets. For each object in a scene, we extract the category name, 3D center camera coordinates, and object dimensions from the annotation files, generating image–3D scene pairs.

ARKitScenes. Similar to Objectron, we follow [1] and use the provided dataset splits and extract object category labels, 3D center coordinates, and dimensions from annotations to generate image-3D scene pairs.

Objaverse. For training the 3D VQ-VAE, we use $\sim$ 168k Objaverse assets from the Sketchfab subset. However, the computational cost of extracting slats from assets is high using the original TRELLIS pipeline, as it requires extracting 150 renders of the assets which are then passed through the DINOv2 encoder. Instead, we extract a single render of each asset only, and pass it as image-conditioning to the pre-trained TRELLIS slat generator. We use the resultant slats as the inputs to the 3D VQ-VAE encoder during training. We found that the slats generated via this synthetic pipeline are relatively faithful to the original asset. Further, the synthetic slats are effective substitutes in the training data as during evaluation, we use slats extracted from the original TRELLIS pipeline and find that performance transfers well, indicating the distributions of slats are similar with the two methods.

A.2. 3D scene serialization

As described above, each scene consists of an image paired with a structured 3D scene representation in JSON format. Before tokenization, we preprocess these JSON files by parsing and converting them into a single string representation. Specifically, we extract relevant attributes from the JSON and structure them using special markers like `[SHAPE]`, `[LOCATION]`, *etc.*, which vary depending on the dataset. These special markers are registered as special tokens in the tokenizer, which we discuss in the next section. For instance, in ObjaWorld, when we encode the explicit geometry, 512 tokens fetched from the 3D VQ-VAE codebook follow the `[SHAPE]` marker. We provide examples of serialized outputs for each dataset below.

CLEVR:

```
[SCENE-START][OBJECT-START][SIZE]large
[COLOR]cyan[MATERIAL]metal[SHAPE]cube
[LOCATION]-0.55 0.05 0.70
[OBJECT-END][OBJECT-START][SIZE]small
[COLOR]yellow[MATERIAL]metal[SHAPE]cylinder
[LOCATION]1.25 2.50 0.35
[OBJECT-END][SCENE-END]
```

ObjaWorld:

```
[SCENE-START][OBJECT-START][SHAPE]table
[LOCATION]-2.70 -2.20 0.20[POSE]0.00 0.00
-0.10
[OBJECT-END][OBJECT-START][SHAPE]person
[LOCATION]-0.20 -0.70 0.85[POSE]0.00 0.00
0.55[OBJECT-END][OBJECT-START][SHAPE]person
[LOCATION]-0.75 -2.80 0.85[POSE]0.00 0.00
-2.55[OBJECT-END][OBJECT-START][SHAPE]table
[LOCATION]2.75 1.90 0.20[POSE]0.00 0.00
1.95[OBJECT-END][OBJECT-START][SHAPE]sofa
[LOCATION]0.40 2.75 0.30[POSE]0.00 0.00
-0.95[OBJECT-END][SCENE-END]
```

ObjaWorld (with explicit shape representations):

```
[SCENE-START][OBJECT-START][SHAPE]<v_1^1,
v_2^1,...,v_512^1>[LOCATION]-2.45 0.60 1.20
[POSE]0.00 0.00 2.30
[OBJECT-END][OBJECT-START][SHAPE]<v_1^2,v_2^2,...,
v_512^2>[LOCATION]2.50 -1.35 0.00[POSE]0.00
0.00 1.55[OBJECT-END][OBJECT-START][SHAPE]<
v_1^3,v_2^3,...,v_512^3>[LOCATION]-1.80 -0.90 0.00
[POSE]0.00 0.00 1.45[OBJECT-END][SCENE-END]
```

Objectron:

```
[SCENE-START][OBJECT-START][CATEGORY]bicycle
[CENTER_CAM]0.00 -0.10 2.45[DIMENSIONS]0.60
 1.10 1.00[OBJECT-END][SCENE-END]
```

ARKitScenes:

```
[SCENE-START][OBJECT-START][CATEGORY]sofa
[CENTER_CAM]-0.14 0.04 1.50[DIMENSIONS]1.70
```

Instruction type	# pairs	Example text instruction
Modifying the appearance of objects (no reference to other objects)	20,000	*"Change the gray object to have purple color"* *"Transform the small yellow rubber sphere to have metal material"*
Modifying the appearance of objects	20,000	*"Change the size of the small purple metal cylinder to the behind of large green rubber sphere to large"* *"Set the material of the gray metal cube object to the left of small purple rubber cylinder to rubber"*
Adding new objects	20,000	*"Put a small gray rubber cylinder to the left of small yellow rubber sphere"* *"Insert a red rubber cylinder object to the front of large cyan rubber cube"*
Removing objects	20,000	*"Remove the small red rubber cylinder to the right of large yellow rubber cube"* *"Take out the small rubber sphere object to the right of small gray rubber cylinder"*
Moving an object to a desired location	20,000	*"Move the small cyan rubber cylinder object to left and behind"* *"Change the position of the large rubber sphere object towards right and behind"*

Table 1. **Instruction templates.** Each template type, its pair count, and two representative text instructions.

```
 0.80 0.90
[OBJECT-END][OBJECT-START][CATEGORY]table
[CENTER_CAM]0.02 0.08 1.60[DIMENSIONS]0.50
0.40 0.50
[OBJECT-END][OBJECT-START][CATEGORY]table
[CENTER_CAM]-0.30 0.22 0.00[DIMENSIONS]1.30
 0.80 0.40
[OBJECT-END][OBJECT-START][CATEGORY]cabinet
[CENTER_CAM]0.46 -0.02 0.00[DIMENSIONS]0.70
 0.80 0.30
[OBJECT-END][OBJECT-START][CATEGORY]cabinet
[CENTER_CAM]0.40 0.24 0.00[DIMENSIONS]1.90
0.90 0.70[OBJECT-END][SCENE-END]
```

A.3. Tokenization

Text. We employ an off-the-shelf text tokenizer from Llama-3.2 [4] with a vocabulary size of 128,000 for tokenization.

Images. For image tokenization, we train a domain-specific VQGAN on the training set of each dataset. This model encodes images into discrete representations by mapping them to codebook indices, which are then used for downstream processing.

3D Scenes. To enable tokenization of 3D scene representations, we augment the vocabulary of the Llama-3.2 tokenizer with two additional token types: (1) special tokens that serve as markers for scene attributes and (2) numerical tokens, ensuring that each location coordinate is encoded as a distinct token.

3D Shapes. For encoding the explicit shape geometries for the assets, we train a 3D VQ-VAE for tokenization as discussed in Section 3.2.1 of the main paper. We provide more details on training the 3D VQ-VAE in Section D.2 below.

A.4. Task sequence formation

After tokenizing all three modalities, we construct sequences tailored to the specific task, which are then used to train Kyvo. In this section, we present examples of complete sequences for each of the tasks across the four datasets.

CLEVR:

Rendering: 3D $\rightarrow$ Image

```
[BOS][SCENE-START][OBJECT-START][SIZE]small
[COLOR]green[MATERIAL]metal[SHAPE]sphere
[LOCATION]0.85 1.85 0.35
[OBJECT-END][OBJECT-START][SIZE]small
[COLOR]green[MATERIAL]metal[SHAPE]sphere
[LOCATION]0.80 -2.00 0.35
[OBJECT-END][OBJECT-START][SIZE]large
[COLOR]brown[MATERIAL]metal[SHAPE]cylinder
[LOCATION]-1.35 2.65 0.70
[OBJECT-END][OBJECT-START][SIZE]small
[COLOR]purple[MATERIAL]rubber[SHAPE]sphere
[LOCATION]-0.90 -2.20 0.35
[OBJECT-END][OBJECT-START][SIZE]large
[COLOR]red[MATERIAL]rubber[SHAPE]cylinder
[LOCATION]-2.70 -2.90 0.70
[OBJECT-END][OBJECT-START][SIZE]large
[COLOR]red[MATERIAL]metal[SHAPE]cylinder
[LOCATION]2.25 -1.15 0.70
[OBJECT-END][OBJECT-START][SIZE]small
[COLOR]red[MATERIAL]metal[SHAPE]cube
[LOCATION]2.15 -2.65 0.35
[OBJECT-END][SCENE-END][OUTPUT-SEP]
[IMAGE-START]<image-tokens>[IMAGE-END][EOS]
```

Recognition: Image $\rightarrow$ 3D

```
[BOS][IMAGE-START]<image-tokens>
[IMAGE-END][OUTPUT-SEP][SCENE-START]
[OBJECT-START][SIZE]small[COLOR]brown
[MATERIAL]rubber[SHAPE]sphere
[LOCATION]-2.50 0.30 0.35
[OBJECT-END][OBJECT-START][SIZE]large
[COLOR]blue[MATERIAL]metal[SHAPE]cylinder
[LOCATION]2.95 1.60 0.70
[OBJECT-END][OBJECT-START][SIZE]large
[COLOR]red[MATERIAL]metal[SHAPE]cylinder
[LOCATION]-0.85 0.60 0.70
[OBJECT-END][OBJECT-START][SIZE]small
[COLOR]green[MATERIAL]metal[SHAPE]sphere
[LOCATION]0.85 1.85 0.35
[OBJECT-END][SCENE-END][EOS]
```

Instruction-Following: (Image, 3D, Text_I) $\rightarrow$ (Image, 3D)

```
[BOS][IMAGE-START]<image-tokens>
[IMAGE-END][SCENE-START][OBJECT-START][SIZE]
small[COLOR]brown[MATERIAL]rubber
[SHAPE]sphere[LOCATION]-2.50 0.30 0.35
[OBJECT-END][OBJECT-START][SIZE]large
[COLOR]blue[MATERIAL]metal[SHAPE]cylinder
[LOCATION]2.95 1.60 0.70
[OBJECT-END][OBJECT-START][SIZE]large
[COLOR]red[MATERIAL]metal[SHAPE]cylinder
[LOCATION]-0.85 0.60 0.70
[OBJECT-END][OBJECT-START][SIZE]small
[COLOR]green[MATERIAL]metal[SHAPE]sphere
[LOCATION]0.85 1.85 0.35
[OBJECT-END][SCENE-END][TEXT-START]Change
the brown object to have purple color
[TEXT-END]
[OUTPUT-SEP][IMAGE-START]<image-tokens>
[IMAGE-END][SCENE-START][OBJECT-START][SIZE]
small[COLOR]purple[MATERIAL]rubber
[SHAPE]sphere[LOCATION]-2.50 0.30 0.35
[OBJECT-END][OBJECT-START][SIZE]large
[COLOR]blue[MATERIAL]metal[SHAPE]cylinder
[LOCATION]2.95 1.60 0.70
[OBJECT-END][OBJECT-START][SIZE]large
[COLOR]red[MATERIAL]metal[SHAPE]cylinder
[LOCATION]-0.85 0.60 0.70
[OBJECT-END][OBJECT-START][SIZE]small
[COLOR]green[MATERIAL]metal[SHAPE]sphere
[LOCATION]0.85 1.85 0.35
[OBJECT-END][SCENE-END][EOS]
```

Question-Answering: $(\text{Image}, \text{3D}, \text{Text}_\text{Q}) \rightarrow \text{Text}_\text{A}$

```
[BOS][IMAGE-START]<image-tokens>
[IMAGE-END][SCENE-START][OBJECT-START][SIZE]
small[COLOR]brown[MATERIAL]rubber
[SHAPE]sphere[LOCATION]-2.50 0.30 0.35
[OBJECT-END][OBJECT-START][SIZE]large
[COLOR]blue[MATERIAL]metal[SHAPE]cylinder
[LOCATION]2.95 1.60 0.70
[OBJECT-END][OBJECT-START][SIZE]large
[COLOR]red[MATERIAL]metal[SHAPE]cylinder
[LOCATION]-0.85 0.60 0.70
[OBJECT-END][OBJECT-START][SIZE]small
[COLOR]green[MATERIAL]metal[SHAPE]sphere
[LOCATION]0.85 1.85 0.35
[OBJECT-END][SCENE-END][TEXT-START]What
size is the rubber sphere?
[TEXT-END][OUTPUT-SEP][TEXT-START]small
[TEXT-END][EOS]
```

Similar structure is followed for **ObjaWorld** (*Rendering* and *Recognition*), **Objectron** (*Recognition*), and **ARKitScenes** (*Recognition*). For **ObjaWorld**, when we encode the explicit shape representations, the complete sequences look like the following.

ObjaWorld: (with explicit shape representations)

Rendering: 3D $\rightarrow$ Image

```
[BOS][SCENE-START][OBJECT-START][SHAPE]<v^1_1,
v^1_2,...,v^1_512>[LOCATION]2.10 0.15 -0.75
[POSE]0.00 0.00 -2.10
[OBJECT-END][OBJECT-START][SHAPE]<v^2_1,v^2_2,...,
v^2_512>[LOCATION]-1.25 2.85 0.05[POSE]0.00
0.00 1.85[OBJECT-END][OBJECT-START][SHAPE]<
v^3_1,v^3_2,...,v^3_512>[LOCATION]0.45 -2.35 -1.50
[POSE]0.00 0.00 0.60
[OBJECT-END][SCENE-END][OUTPUT-SEP]
[IMAGE-START]<image-tokens>[IMAGE-END][EOS]
```

Recognition: Image $\rightarrow$ 3D

```
[BOS][IMAGE-START]<image-tokens>
[IMAGE-END][OUTPUT-SEP][SCENE-START]
[OBJECT-START][SHAPE]<v^1_1,v^1_2,...,v^1_512>
[LOCATION]1.15 -2.85 0.40[POSE]0.00 0.00
-1.75[OBJECT-END][OBJECT-START][SHAPE]<v^2_1,
v^2_2,...,v^2_512>[LOCATION]-0.35 2.10 -0.50
[POSE]0.00 0.00 2.45
[OBJECT-END][OBJECT-START][SHAPE]<v^3_1,v^3_2,...,
v^3_512>[LOCATION]2.75 0.15 -1.60[POSE]0.00
0.00 -0.25[OBJECT-END][SCENE-END][EOS]
```

B. Additional qualitative examples

In this section, we provide additional qualitative examples.

3D Tokenization. We present additional qualitative results for our 3D tokenization scheme. Fig. 1 provides additional examples of how the 3D tokens are effective for reconstructions, as shown in Figure 4 (b) in the paper. Fig. 2 provides additional examples of how the 3D tokens are effective for decoding, as shown in Figure 4 (c) in the paper.

Rendering. We present qualitative results for the rendering task on CLEVR and ObjaWorld with complex shapes in Figures 3 and 4, respectively. Results on ObjaWorld with explicit shape representations are shown in Figure 5. The model takes only the structured 3D scene representation as input and predicts the corresponding image tokens. These tokens are then decoded using the VQGAN decoder, which reconstructs the image by mapping them from token-space to pixel-space. For each example, we also provide the ground truth image rendered using Blender, allowing direct comparison with the model-generated output. As observed, the model effectively captures the 3D scene structure based solely on the JSON input and accurately synthesizes the corresponding image. Notably, Figure 5 demonstrates the model's ability to integrate multiple information sources: it successfully maps spatial arrangements from JSON scene descriptions and visual properties from asset sequences to generate coherent, realistic pixel-space representations.

However, certain failure cases highlight the model's limitations. For instance, in the first example of Figure 3, the model fails to predict a small red cube positioned at the front. Similarly, in the last example of Figure 4, the model mispredicts the bird's pose, causing it to face the wrong direction.

GT | Kyvo 3D Tokenizer | GT | Kyvo 3D Tokenizer

Figure 1. **Reconstruction examples for Trellis-based 3D VQ-VAE.** VQ-VAE reconstructions, shown from multiple views, compared to ground truth for unseen Objaverse assets.

Figure 2. **Decoding examples for Trellis-based 3D VQ-VAE on unseen Objaverse assets.** (a) Reconstruction of 3D shape encoding using Llama 3.2 as decoder. (b) Rendering from 3D shape encoding using Llama 3.2 as decoder.

In the third column of comparisons in Figure 5, some distortions in the shapes and poses can be seen as well. Despite these occasional errors, the overall rendering quality demonstrates strong spatial understanding and scene reconstruction capabilities.

Recognition. In Figure 6 we show example recognition results on the ObjaWorld dataset with complex shapes. The model takes the image (tokenized) as input and outputs the sequential 3D representation. We then parse the predicted 3D scene into JSON format and display it alongside the

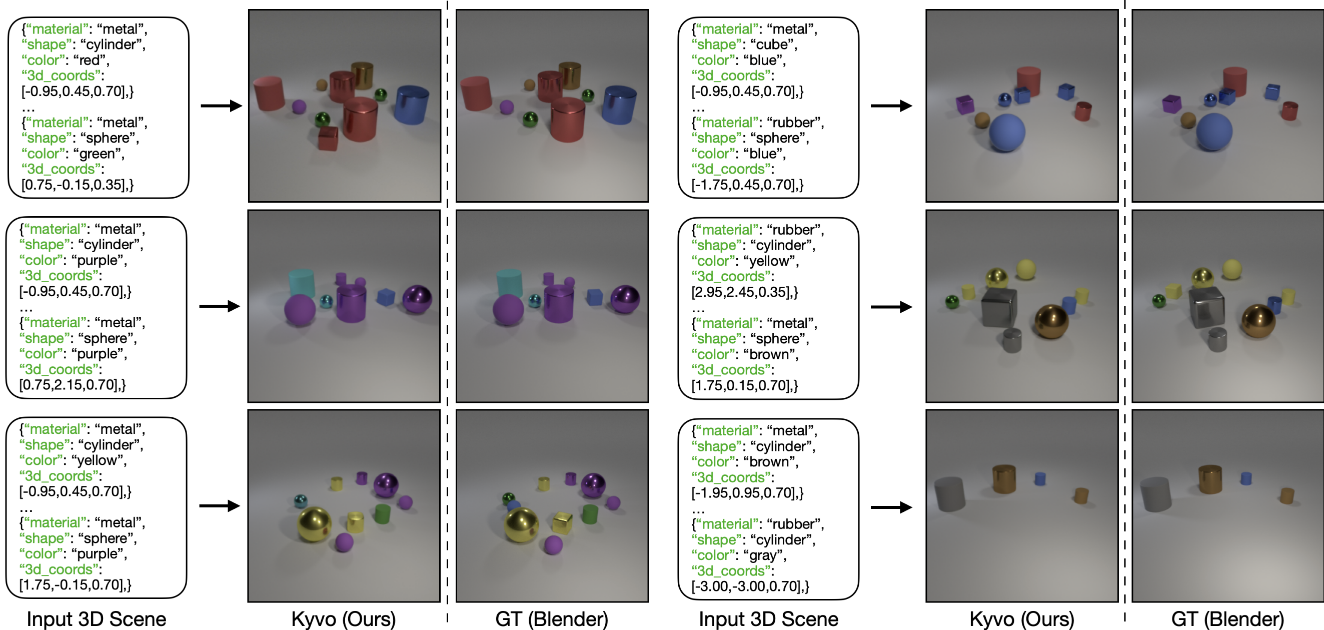


Figure 3. **Rendering examples for CLEVR.** Example image generations for the rendering task on CLEVR. The model takes a 3D scene as input and produces a corresponding image. Additionally, we show the ground-truth image rendered using Blender.

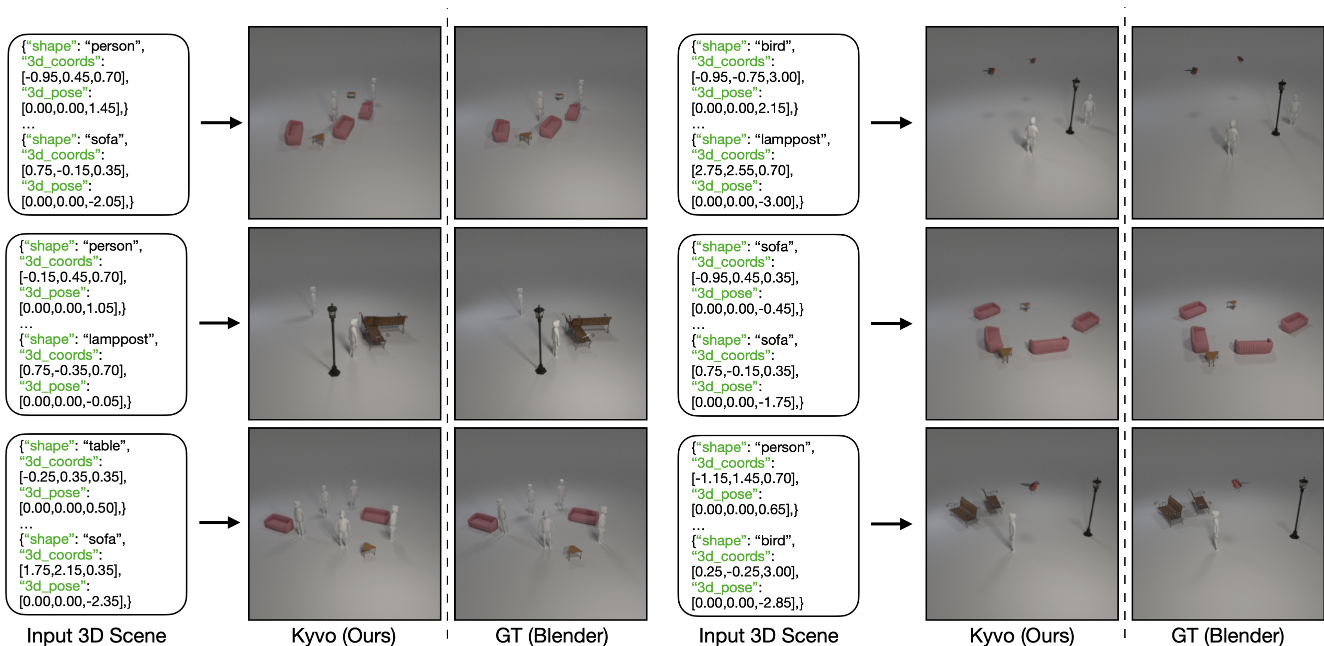


Figure 4. **Rendering examples for ObjaWorld.** Example image generations for the rendering task on ObjaWorld with complex shapes. The model takes a 3D scene as input and produces a corresponding image. Additionally, we show the ground-truth image rendered using Blender.

ground truth JSON in the figure. To evaluate recognition performance, we match the predicted objects with ground truth objects to compute the Jaccard Index (see Algorithm 1). This matching is based on attribute similarity and spatial location criteria. Additional details on the Jaccard Index calculation are provided in Section C. The matching results, visualized

Figure 5. **Rendering examples for ObjaWorld with explicit shape encodings.** Example image generations for the rendering task on ObjaWorld. The model takes a 3D scene with embedded shape encodings as input and produces a corresponding image. Additionally, we show the ground-truth image rendered using Blender.

using colored numbers in Figure 6, illustrate the model's strong ability to accurately predict object attributes and their 3D spatial locations almost accurately. While minor spatial deviations may occur, the model effectively reconstructs the structured 3D scene from image input.

In Figure 7 we show examples of recognition task on ObjaWorld with explicit shape representations on unseen scenes. Specifically, the model takes a single images as input and reconstructs the full 3D geometry per object and infers each object's 3D position and pose to accurately reconstruct the 3D scene. As can be seen from Figure 7, Kyvo accurately recovers object geometries and spatial layouts via our structured, object-centric 3D modality. In contrast, Trellis [6] often merges two objects into one (e.g. the second example) or hallucinates shapes (e.g., blue can in the first example) and misaligned layouts.

Question-answering. In Figure 8, we present qualitative examples from the question-answering task on the CLEVR dataset. The figure showcases model predictions across a diverse set of question types, including binary (True/False) responses, categorical answers such as object sizes ("small" or "large"), and numerical values. These examples highlight the model's ability to understand and reason about structured 3D scenes, demonstrating accurate comprehension of spatial relationships, object attributes, and numerical reasoning.

C. Evaluation

In this section we provide more details on the evaluation strategies that we adopt. We discuss the evaluation method for the three primary modalities as well as for the 3D shape encodings.

C.1. 3D scenes

We evaluate predicted 3D scenes using the Jaccard Index, computed via Algorithm 1. Objects are matched between predicted and ground-truth scenes based on attribute similarity and spatial proximity.

Matching criteria by dataset:

- CLEVR: Match shape, size, color, and material
- ObjaWorld: Match shape with pose constraint (predicted pose within ± 0.15 radians)
- Objectron: Match category with dimension constraint (mean absolute error ≤ 0.05)
- ARKitScenes: Match category with dimension constraint (mean absolute error ≤ 1.00)

Spatial proximity thresholds (τ): We average Jaccard Index across multiple threshold values:

- CLEVR, ObjaWorld, Objectron: $\tau \in \{0.05, 0.10, 0.15, 0.20, 0.25\}$
- ARKitScenes: $\tau \in \{1.25, 1.50, 1.75, 2.00, 2.25\}$

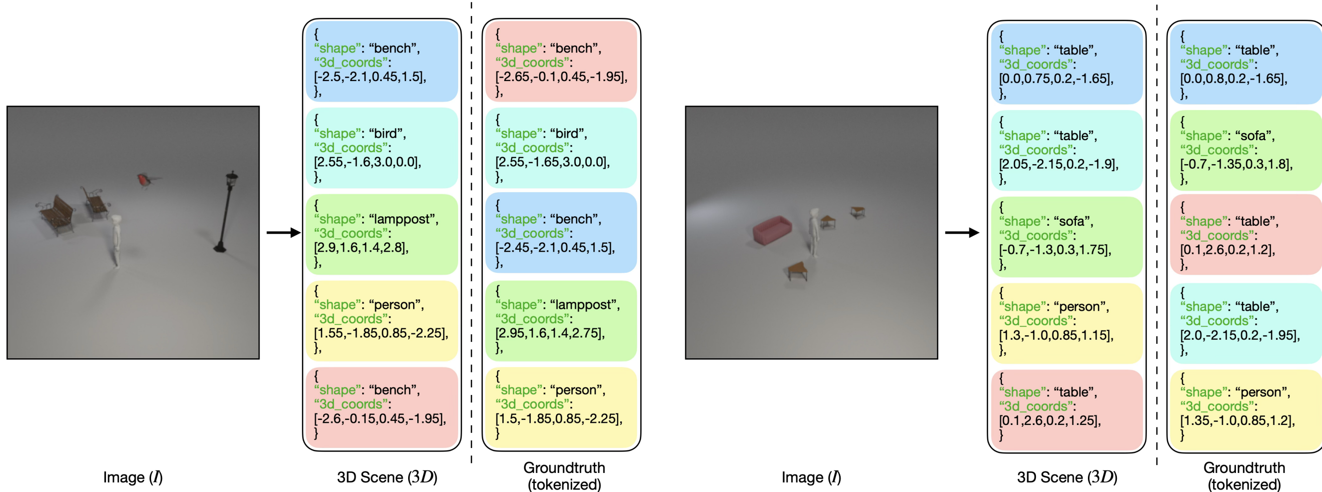


Figure 6. **Recognition examples for ObjaWorld.** Two example predictions from the recognition task on ObjaWorld. The colored boxes indicate object matching between the predicted and ground-truth scenes, based on the criteria for Jaccard Index as defined in Algorithm 1. Note that the fourth number in the list is the azimuth pose value, this format of prediction saves sequence length.

Lower τ values impose stricter spatial constraints, requiring predicted objects to be closer to ground-truth positions. Figure 9 illustrates how τ affects Jaccard Index on CLEVR across different training data sizes.

Comparison with Trellis. We compare our unified shape and scene reconstruction against Trellis (Figure 9, main paper; Figure 7). Trellis reconstructs scenes by inputting an image to a rectified flow transformer (DiT) that generates a scene-level SLAT representation, which is subsequently decoded into a single holistic 3DGS. In contrast, Kyvo employs a different approach through two key distinctions: (1) *Scene decomposition*: Kyvo decomposes scenes into constituent objects, each parameterized by shape, 3D location, and orientation within our structured 3D modality. (2) *Quantized representation*: While both methods utilize SLAT representations for object shapes, Kyvo vector-quantizes these representations via our 3D VQ-VAE, to slot naturally into Kyvo's autoregressive generation framework. This decomposition enables Kyvo to achieve precise reconstruction of individual objects while simultaneously inferring their 3D spatial locations and relationships. Moreover, we obtained an average Jaccard Index of 0.666 averaged over $\tau \in \{0.50, 0.75, 1.00, 1.25, 1.50\}$ for this recognition model. We observed that the model seems to have a relatively harder time predicting the location coordinates when explicit shape sequences are involved as compared to when the shapes are identified using a word token like in CLEVR.

C.2. Images

Human Evaluation. As discussed in the main paper, we rely on human evaluations to assess image generations by the model, as object-level nuances often go beyond the scope of quantitative metrics. To facilitate this evaluation, we designed a user interface, a snapshot of which is shown in Figure 10. For each comparison involving N models, the interface presents users with N generated images alongside the ground truth image, all displayed in a shuffled and anonymized order. This ensures that users remain unaware of which model generated each image, mitigating potential biases in evaluation. In each comparison, users are asked to assign both a *score* and a *rank* to every generated image based on its visual fidelity and alignment with the ground truth. Figure 10 shows a snapshot of the evaluation of images from the experiment where we studied the effect of center-token reordering and weighted loss on model generation involving four models (results reported in Table 4 of the main paper). There were *two* annotators across all human evaluation results reported in the paper.

Score: The score takes a binary value of "0" or "1" and signifies the complete correctness of an image generation. The user is asked to provide a score of "1" only if the user believes that all the objects in the scene were accurately generated and accurately placed spatially. If the generated image has any differences with the groundtruth, *e.g.* a cube was not generated correctly, the user provides a score of "0". The score value is independent of any other model involved in the comparison and solely depends on the model under consideration and the groundtruth.

Rank: The rank takes a value from {"1", "2", ..., "N"}. The user is expected to rank the N images from "1" to "N" by comparing the generation quality among them. If the user is unable to decide between two images, then we allow equal

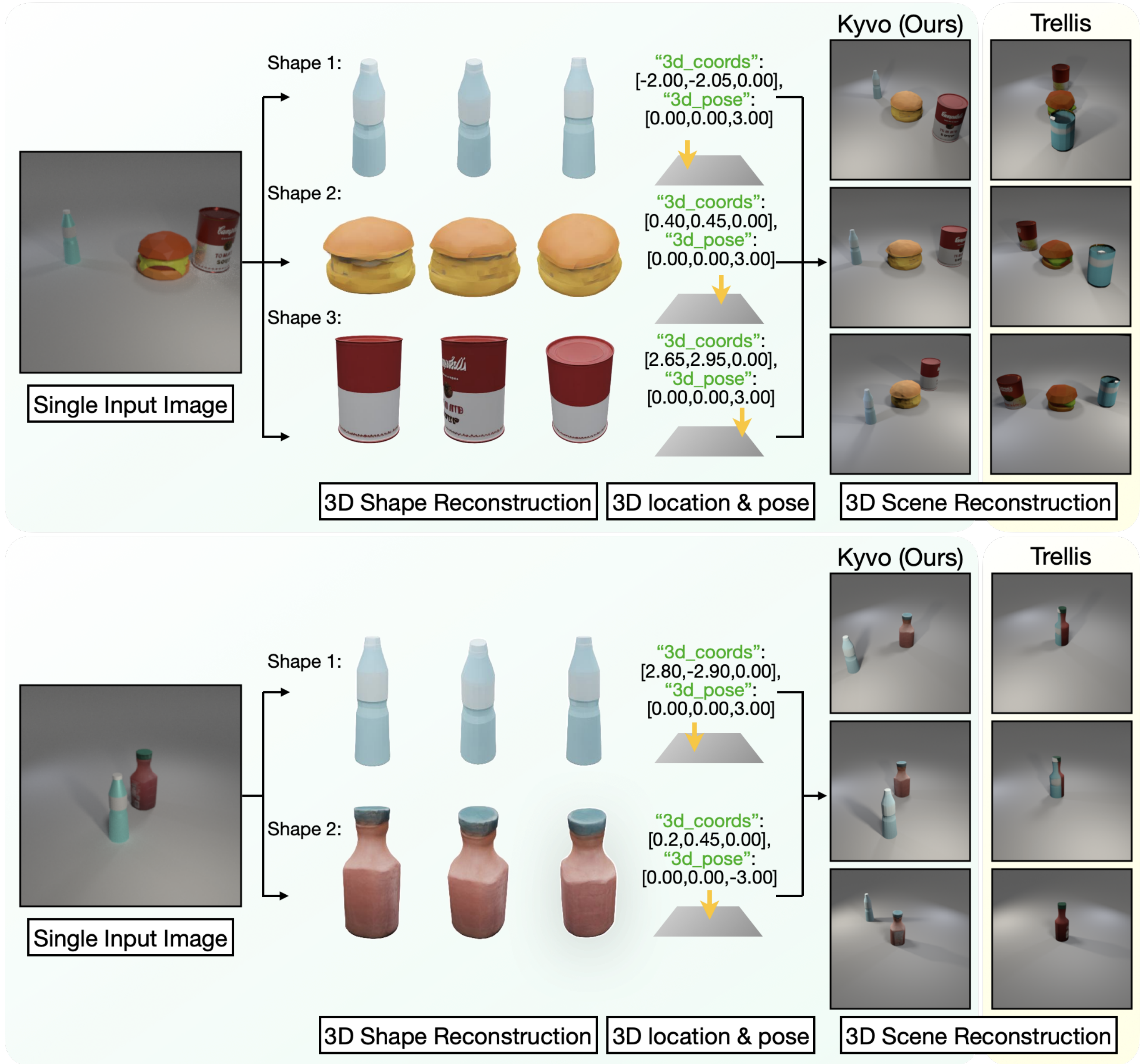


Figure 7. **Unified shape and scene reconstruction examples.** Given a single input image, Kyvo predicts shape sequences and reconstructs individual objects (bottle, cheeseburger, *etc.*) along with their 3D locations and poses via our structured 3D modality, effectively reconstructing the 3D scene with consistent spatial relations between the objects, visualized using Blender.

rank assignment to more than one image, *e.g.* if two models equally perform for a given scene, they get the same rank value.

We consider a test set of 50 scenes and use the user interface for scoring and ranking the generations. Specifically, let's say we want to have an evaluation on the effect of granularity, then we consider the image generated by the models for the 50 scenes and score and rank them as discussed above.

In the main paper, we report the *mean rank* (the lower, the better) over the 50 images in the tables. In addition to the *mean rank*, we obtain the *mean score* (the higher, the better) for every model. We also compute the *winning rate* (the higher, the better) that is defined as the fraction of times a given model was assigned a rank of 1. We report the *mean score* and *winning rate* for all the models in Table 2.

SSIM and L2-loss. Evaluation of the generated images

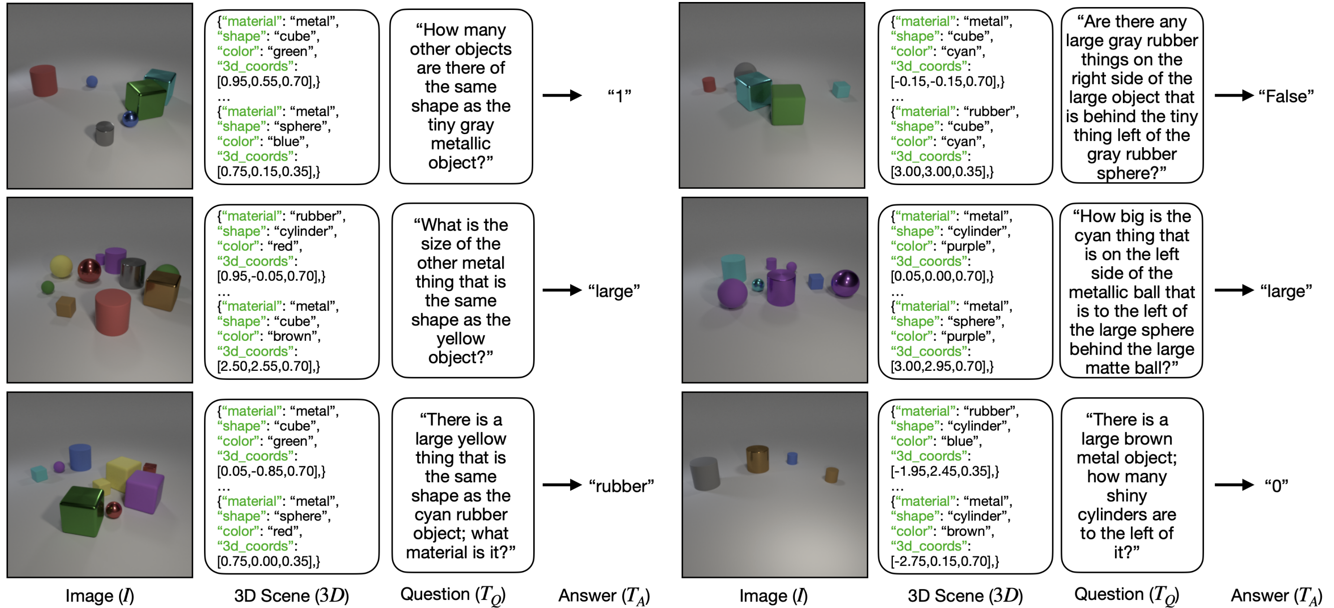


Figure 8. **Question-answering examples for CLEVR.** Example cases from the question-answering task on CLEVR. The model takes an image, a 3D scene, and a question as input to generate the corresponding answer.

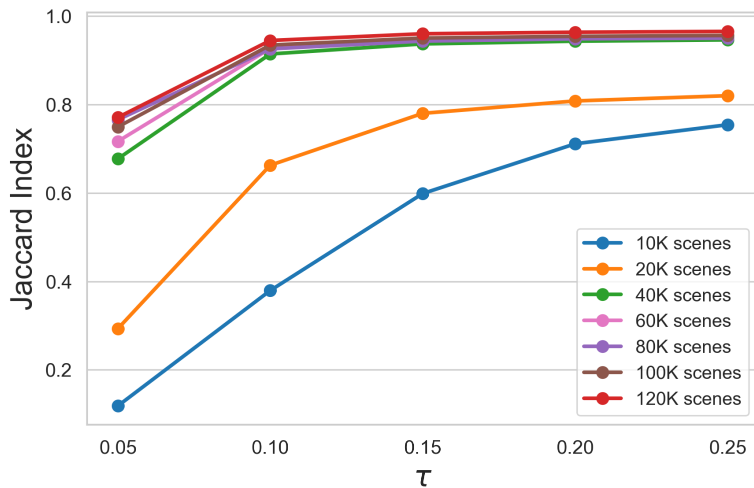


Figure 9. **Effect of τ.** The plot shows the impact of τ on the Jaccard Index for models trained with increasing amounts of training data on CLEVR for the recognition task. The drop in Jaccard Index with decreasing τ is more pronounced for models trained on smaller datasets. Higher-performing models demonstrate greater robustness to changes in τ.

requires careful assessment of the objects and their attributes in the scene. For example, consider the example predictions in Figure 11. For both cases, the predicted image is incorrect and minutely differs from the groundtruth. For the first case, the small cyan sphere gets an incorrect gray color, while in the second case, the small cube gets mispredicted as a cylinder. Quantitative metrics like SSIM and L2-loss fail to capture these subtle differences between the predicted and the groundtruth images that occupy a very small pixel region, leading us to qualitative human evaluations. However, for experimental completeness, we computed the SSIM and pixel-wise L2-loss for all the models and reported them in Table 2. While the values show similar trends to human evals, we report human evals in the main paper as they directly assess image correctness.

Algorithm 1 Compute Jaccard Index

```
1:  Input: GTS (list of ground-truth scenes), PS (list of predicted scenes), distance threshold τ
2:  Output: Average JaccardIndex
3:  Initialize JaccardIndex = 0
4:  for (G, P) in zip(GTS, PS) do
5:    GObjs ← G.objects
6:    PObjs ← P.objects
7:    matchedFlags ← Boolean array of length |GObjs|, initialized to False
8:    Initialize TP = 0, FP = 0, FN = 0
9:    for p in PObjs do
10:     foundMatch ← False
11:     for j = 1 to |GObjs| do
12:       if ¬matchedFlags[j] and attributesMatch(p, GObjs[j]) and dist(p.coords, GObjs[j].coords) <
          τ then
13:         matchedFlags[j] ← True
14:         foundMatch ← True
15:         TP ← TP + 1
16:         break
17:       end if
18:     end for
19:     if foundMatch = False then
20:       FP ← FP + 1
21:     end if
22:   end for
23:   FN ← FN + count(matchedFlags = False)
24:   JaccardIndex+ = TP / (TP+FP+FN)
25: end for
26: JaccardIndexAvg ← JaccardIndex/|PS|
27: return JaccardIndexAvg
```

C.3. 3D assets

For the same reasons as above, we use human evaluations to assess the quality of 3D assets when it comes to the 3D tokenizer. In Table 1 in the main paper, we use human evaluations to quantify the effect of auxiliary reconstruction losses on the reconstruction quality of the 3D VQ-VAE, and report mean rank. In Table 2 in the main paper, we use human evaluations to compare the reconstruction quality of our Trellis-based 3D VQ-VAE with that of SAR3D, and report mean rank. In both cases, the human evaluation is facilitated using an anonymized, shuffled interface as described above, and only relative rank is assessed and reported.

C.4. Text

Among the four tasks we consider, only question-answering produces text output. The question templates used in CLEVR cover a diverse range of answer types. Some questions require binary (True/False) responses, others expect numerical values ranging from 0 to 10, while some answers involve text words describing attributes like "small","green", "metal", *etc.*. Table 3 provides a breakdown of the different question types in the training set.

Using these distributions, we establish two baseline accuracies on the test set: random and frequency. For the random baseline, we predict answers uniformly at random for each question type, yielding a mean accuracy of 0.359 with a standard deviation of 0.009 over 100 runs. For the frequency baseline, we predict the most common answer for each question type based on its distribution in the training data (as shown in Table 3), achieving a test accuracy of 0.430.

D. Additional implementation details

D.1. Image VQGAN architecture

In this section, we provide additional implementation details for the VQGAN architecture employed in our experiments to represent images. We adopt the standard VQGAN encoder-decoder architecture from [3] with a spatial compression factor of $16\times$ (256×256 image resolution to 16×16 latent resolution) and a codebook of 1024 entries with embedding dimension 256, totaling $\sim$ 17.5M parameters. The encoder produces quantized embeddings of shape (1, 256, 16, 16), yielding $16 \times 16 = 256$ discrete tokens per image that correspond to learned codebook entries. For each

Comparison	**Metrics**				
Granularity	Mean Rank (↓)	Winning Rate (↑)	Mean Score (↑)	SSIM (↑)	L2-loss (↓)
0.005	1.38	0.70	0.74	0.9527	0.0021
0.05	1.20	0.82	0.80	0.9505	0.0010
0.5	2.02	0.28	0.52	0.9030	0.0010
Number Encoding	Mean Rank (↓)	Winning Rate (↑)	Mean Score (↑)	SSIM (↑)	L2-loss (↓)
Fixed Sine-Cosine	1.44	0.64	0.64	0.9525	0.0010
Learned	1.58	0.60	0.60	0.9487	0.0011
Fixed Sine-Cosine + Learned	1.28	0.78	0.78	0.9505	0.0010
CT Reordering, Weighted Loss	Mean Rank (↓)	Winning Rate (↑)	Mean Score (↑)	SSIM (↑)	L2-loss (↓)
✗, ✗	2.66	0.00	0.00	0.8575	0.0049
✓, ✗	3.56	0.00	0.00	0.8410	0.0061
✗, ✓	2.78	0.00	0.00	0.8668	0.0044
✓, ✓	1.00	1.00	0.84	0.9505	0.0010
Recipe	Mean Rank (↓)	Winning Rate (↑)	Mean Score (↑)	SSIM (↑)	L2-loss (↓)
Scratch	1.36	0.68	0.72	0.9515	0.0010
LoRA	1.82	0.48	0.56	0.9507	0.0010
FFT	1.26	0.8	0.82	0.9505	0.0010
Backbone	Mean Rank (↓)	Winning Rate (↑)	Mean Score (↑)	SSIM (↑)	L2-loss (↓)
Llama-3.2-1B	1.38	0.76	0.78	0.9521	0.0010
Llama-3.2-1B-Instruct	1.28	0.72	0.80	0.9505	0.0010
Llama-3.2-3B-Instruct	1.18	0.84	0.86	0.9539	0.0010

Table 2. **Image Metrics.** Comparison of mean rank, winning rate, and mean score from human evaluation across all models for the CLEVR rendering task. Additionally, we provide SSIM and pixel-wise L2 loss values for each model.

Question type	**# Questions**	**Majority answer**
True/False	8,080	False
Number (0–10)	4,401	0
Shape	1,840	cylinder
Color	1,919	cyan
Material	1,840	metal
Size	1,920	small
Baseline accuracy (test set)		
Random (100 runs)	0.359 ± 0.009	
Frequency	0.430	

Table 3. **Question-Answering Data.** Statistics of various question types in the training dataset and baseline accuracies on the test set.

dataset, we initialize the VQGAN model with ImageNet pretrained weights from [3], then fine-tune on the respective training set for 100 epochs. This fine-tuned model serves as both the encoder for converting images to token sequences and the decoder for reconstructing images from predicted tokens during evaluation.

D.2. 3D VQ-VAE training

In Section 3.2.1 of the main paper we describe how we train a 3D VQ-VAE to compress the $64^3 \times 8$ slats to a dense $8^3 \times 128$ latent. Here, we provide some additional details on the training. A slat contains $\sim$ 20k sparse voxels (in a 64^3 grid, with vectors of dimension 8 at active positions). We train the 3D VQ-VAE with these slats in 3 steps: (i) it *densifies* the sparse tensor onto a $64^3 \times 8$ grid, (ii) encodes that grid with a 3-D convolutional U-Net down to an $8^3{\times}128$ latent volume, and (iii) vector-quantizes every latent with an 8192-entry codebook.

Architecture. The original Trellis SLAT encoder and decoder remain frozen and provide supervision. Sparse latents (z_i, p_i) are first rasterised into a zero-initialised $64^3{\times}8$ grid; using a learnable padding vector instead of zeros showed no measurable gain and is therefore omitted. The dense grid is processed by a 3D U-Net that downsamples as $64^3 \to 32^3 \to 16^3 \to 8^3$ with channel widths $(32, 128, 512, 1024)$. Each 8^3 cell outputs a 128-dim vector, quantized to the nearest of 8192 code vectors updated by exponential moving average (τ=0.99). Gradients flow to the encoder through the straight-through estimator.

Optimization. We train for 200k steps on $\sim$ 168k Sketchfab assets from Objaverse (further details provided in Sec. A.1) with a batch size of 8. We use an AdamW optimizer (β_1=0.9, β_2=0.999) with a constant learning-rate of $3{\times}10^{-4}$, no weight decay, mixed precision, and adaptive gradient clipping.

The training objective combines four terms:

$$\mathcal{L} = \underbrace{\|x - \hat{x}\|_2^2}_{\text{dense-SLAT recon}} + \beta \underbrace{\| z - \mathrm{sg}[e]\|_2^2}_{\text{commit}} + \lambda_{\text{KL}}\, D_{\text{KL}} + \gamma\, \mathcal{L}_{\text{render}} \tag{1}$$

where $x \in \mathbb{R}^{64^3 \times 8}$ is the rasterised SLAT voxel, $\hat{x}$ its VQ–VAE reconstruction, $\mathrm{sg}[\cdot]$ denotes the stop-gradient operator. In the commitment term, $z \in \mathbb{R}^{8^3 \times 128}$ denotes the

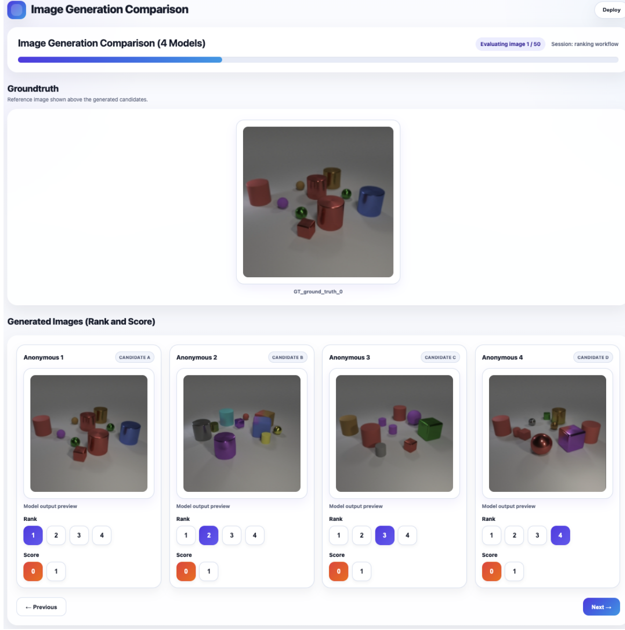


Figure 10. **Human Evaluation User Interface.** A snapshot of the user interface used for human evaluation of generated images. This example is taken from an experiment analyzing the effect of center-token reordering and weighted loss, comparing four models. The results of this experiment are presented in Table 2 of the main paper.

pre-quantization output of the encoder, while e is the corresponding vector-quantized output selected from the codebook. The $\mathcal{L}_{\text{render}}$ follows TRELLIS:

$$\mathcal{L}_{\text{render}} = \mathcal{L}_1(I, \hat{I}) + 0.2\Big(1 - \text{SSIM}(I, \hat{I})\Big) + 0.2\,\text{LPIPS}(I, \hat{I}),$$

computed between images rendered from ground-truth images I and rendering from Gaussian reconstructions $\hat{I}$. This is the auxiliary pixel-space reconstruction loss discussed in the main paper. We set $\beta = 0.25,\ \lambda_{\text{KL}} = 10^{-6},\ \gamma = 0.1$.

The resulting codebook is used to represent objects using $8^3 = 512$ discrete tokens, enabling unified processing with structured 3D representations for autoregressive modeling. We employ bi-directional attention within shape token sequences for full intra-sequence connectivity when training the LLM.

Codebook usage. Figure 12 shows codebook utilization across 5000 training assets (log-scale, decreasing order). The heavy-tailed distribution indicates effective learning: frequent codes capture common 3D primitives while rare codes encode geometric variations. Active utilization of the whole codebook indicates it is not over-parameterised. Additionally, increasing codebook size did not help empirically indicating it is not under-parameterised.

The full training script and code for 3D VQ-VAE are included in the `code.zip` file. The code was built on top of Trellis [6] code.

D.3. Encoding of numbers

In Section 3.2.2 (Figure 6), we showed the robustness of a hybrid approach that we used for encoding numbers, where the embeddings are learned but are augmented with sine-cosine encodings. Here, we provide more details on how we implement the encodings. Specifically, if we have N numbers to encode, then the nth number is encoded using an embedding of dimension d. First, we obtain the sine-cosine encoding as follows:

$$NE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right), \tag{1}$$

$$NE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \tag{2}$$

where i indexes the embedding dimensions. On top of this, we incorporate a learned embedding layer, implemented as `nn.Embedding` in PyTorch, of the same dimensionality d. The final representation for each number is obtained by summing its learned embedding with the corresponding static sine-cosine encoding. This hybrid approach allows the model to leverage both the flexibility of learned embeddings and the structured inductive bias introduced by the sine-cosine encoding for numbers.

D.4. Compute resources and time

Experiments were executed on a single Ubuntu 22.04.5 server equipped with 8×NVIDIA A100–SXM4 GPUs (80 GB each, driver 570.124, CUDA 12.8), dual-socket AMD EPYC 7763 CPUs (256 threads) and 2 TB RAM. The software stack comprised Python 3.11 and PyTorch 2.4.1 built against CUDA 12.1 and cuDNN 9.1. Each model was trained on a single GPU, with an average training throughput of $\sim$ 8,800 tokens sec^{-1} per GPU for all language model training. The LLM experiments were implemented using the torchtune PyTorch library, providing a unified framework for fine-tuning language models. We employed PagedAdamW8bit optimizer with learning rate 1×10^{-4}, batch size 32, and trained for 10 epochs using bfloat16 precision and used the cross-entropy loss function.

The code and configs are included in the `code.zip` file. The code was built on top of the torchtune code.

E. Additional experiments and observations

Failure of modern-day LLMs. To demonstrate the complexity of our 3D tasks, we evaluated several state-of-the-art large language models on simple CLEVR scenes. While CLEVR scenes appear visually simple, the tasks they address are complex. For instance, Google's Gemini-Pro and

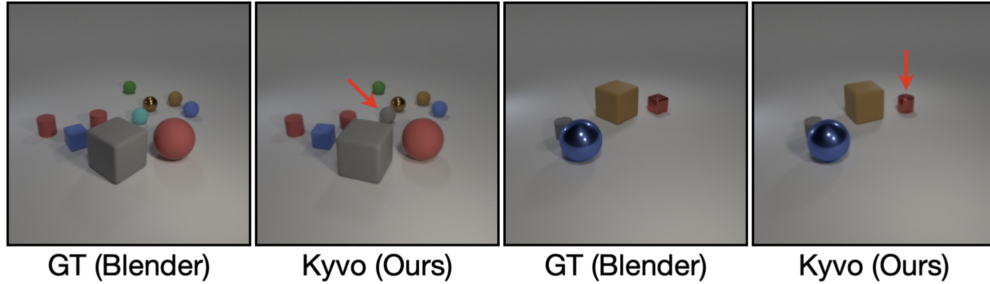


Figure 11. Object-level nuances are challenging for image metrics, like SSIM and L2-loss, to capture, prompting the need for human evaluation. The predicted image is incorrect in both cases but differs only subtly from the groundtruth.

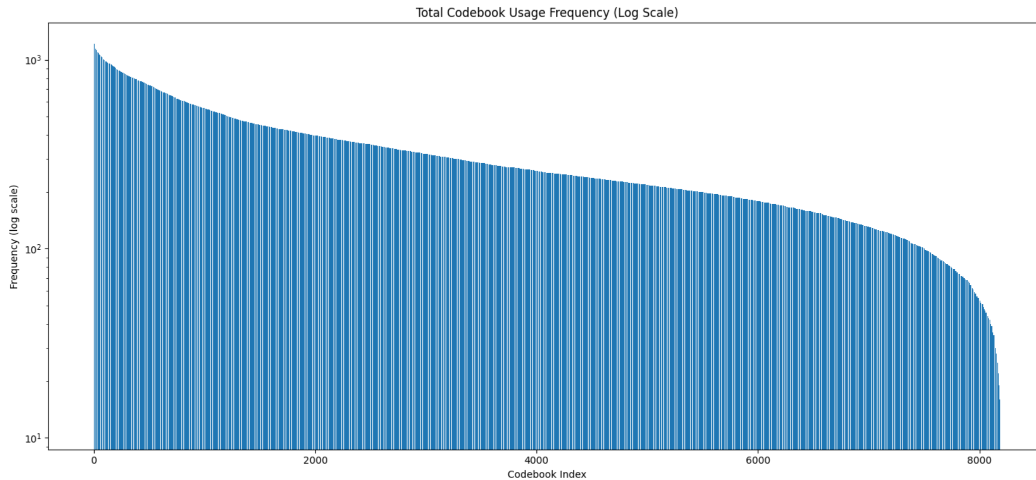


Figure 12. **Codebook usage.** Histogram of usage counts (log scale) for the 8192 latent codes on 5000 assets used for training. The heavy-tailed distribution reveals a compact core of frequently reused codes that capture ubiquitous 3D primitives, while the long tail represents rarer geometric variations. In total, all of the vocabulary is exercised at least once, indicating that the codebook is not over-parameterised. Additionally, increasing codebook size did not help empirically indicating it is not under-parameterised.

OpenAI's latest GPT4o, when prompted to generate an image with CLEVR's 3D scene specifications, produce wrong images – failing to adhere to relative object positions and often hallucinating new objects (see Figure 14). For the recognition task, Meta's state-of-the-art VLM, Llama3.2-Vision (from the same family of backbones as ours), achieves near-zero performance. Figure 15 visualizes Llama3.2-Vision's predictions rendered in Blender alongside ground truth scenes and the predictions from Kyvo recognition model. As can be observed, it fails to accurately predict xyz coordinates, despite the simplicity of CLEVR scenes and its objects. These failures demonstrate that 3D spatial reasoning remains a significant challenge for internet-scale trained models, even within controlled synthetic environments, underscoring the difficulty of the tasks we address.

Number encodings. As detailed in Section 3.2.2 of the main paper, we employ a hybrid approach for number encoding that combines learned representations with sine-cosine positional encodings. We demonstrated the robustness of this hybrid encoding strategy across varying data regimes in Figure 6 of the main paper, with implementation details provided in Section D.3. While Figure 6 assessed the performance of these strategies across data regimes, Table 4 summarizes the performance of these encoding strategies across all four tasks for the largest training set size.

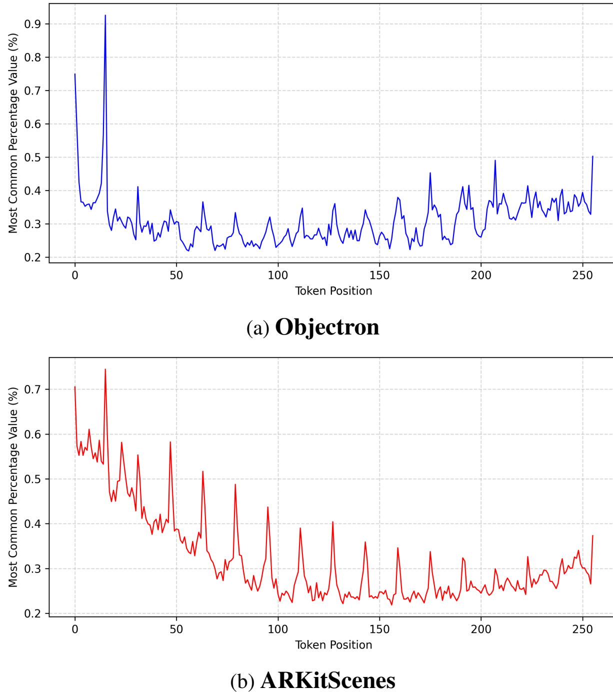
Number Encoding	Rendering(↓)	Recognition(↑)	Instruction(↑)	QA(↑)
Fixed Sine–Cosine	1.44	**0.9229**	**0.8678**	0.4845
Learned	1.58	0.9185	0.8572	0.4680
Fixed Sine–Cosine + Learned	**1.28**	0.9212	0.8666	**0.4980**

Table 4. **Encoding strategies for numbers.** Performance of each strategy across the four tasks.

Embedding dimension projector. We use a projector to

embed the VQGAN codes and the 3D VQ-VAE codes into the same dimensional space as the Llama embeddings, creating a unified sequence of dimensions. A simple linear layer without biases proves sufficient for this task. We tried more complex alternatives, such as a two-layer MLP with ReLU activation, but did not show any notable performance improvements.

Token visualization plot. As discussed in Section 3.2.3 of the main paper, our analysis of 256-token image sequences revealed that over 25% of CLEVR images shared identical first tokens, creating a substantial bias attributed to the uniform gray backgrounds prevalent in synthetic scenes. Interestingly, in Figure 13 we demonstrate that similar positional biases emerge in the Objectron and ARKitScenes datasets, which contain images of real-world 3D scenes, though with notably reduced magnitude. These plots illustrate the percentage of images sharing the most frequent token value at each sequence position.



(a) **Objectron**

(b) **ARKitScenes**

Figure 13. For 256-token image sequences, the plots show the percentage of images that have the most common token at each position in Objectron (left, blue) and ARKitScenes (right, red). Early positions repeat more often, so there is still some bias, but it is weaker than in synthetic CLEVR scenes.

3D scene conditioning in QA. To assess the impact of 3D scene data on question-answering, we train a model excluding 3D scene inputs, i.e., $(\mathrm{I}, \mathrm{T_Q}) \rightarrow \mathrm{T_A}$. This model achieves a Jaccard accuracy of 0.4465, compared to 0.4980 for the $(\mathrm{I}, \mathrm{3D}, \mathrm{T_Q}) \rightarrow \mathrm{T_A}$ model, demonstrating the critical role of 3D semantic information for accurate answers.

F. Failure cases

In this section, we discuss a failure case of Kyvo and potential areas for improvement. Among the four tasks used in our experiments for CLEVR, *Instruction-Following* presents the highest complexity. This task requires the model to process three modalities – 3D scenes, images, and text instructions – as input and generate both a modified image and a modified 3D scene. This requires precise comprehension of the text instruction and accurate application of the specified modifications across both the image and 3D scene sequences.

Our experiments indicate that while Kyvo effectively predicts the modified 3D scene sequence, it struggles with image sequence modifications. In the main paper, we report Jaccard Index values for the instruction-following task, demonstrating the model's effectiveness in handling 3D scenes. Additionally, Figure 16 presents qualitative examples of the model's image outputs. Although the predicted images are somewhat close to the groundtruth, the model often fails to accurately modify the scene within the image sequence. For instance, in the first example, the red sphere was incorrectly assigned a purple color and was moved behind but not to the left as instructed.

While this highlights areas for improvement, an interesting direction for future work is exploring whether decomposing complex tasks like instruction-following into sequences of simpler tasks can enhance performance. For example, instead of predicting both images and 3D scenes simultaneously, the task could be divided into two stages: first, predicting the modified 3D scene, and then using a rendering model to generate the corresponding image.

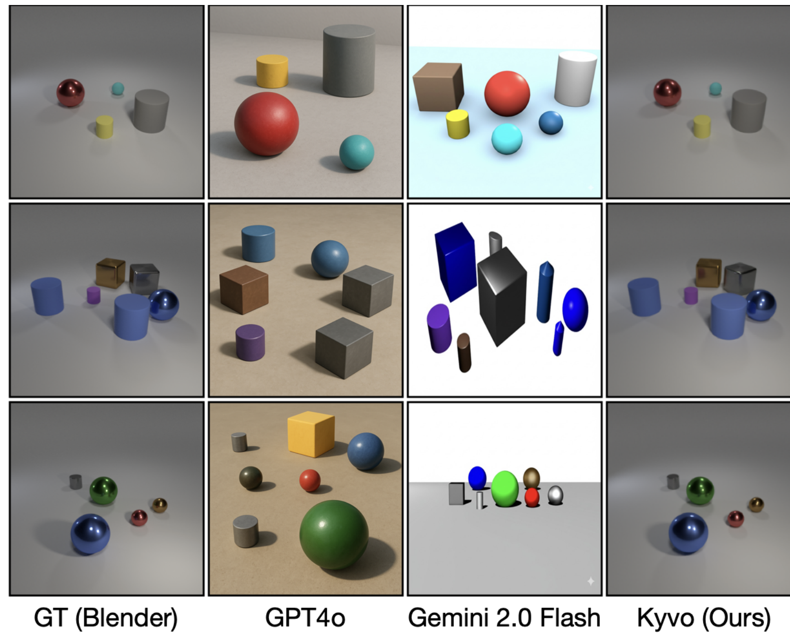


Figure 14. **Failure of modern-day LLMs on rendering task.** Each row depicts one test scene described by our 3D structured modality; columns show the ground-truth Blender render (GT) and images produced by GPT4o, Gemini 2.0 Flash, and Kyvo (ours). GPT4o and Gemini frequently hallucinate extra objects, omit specified ones, or displace shapes, violating fundamental xyz and relational constraints.

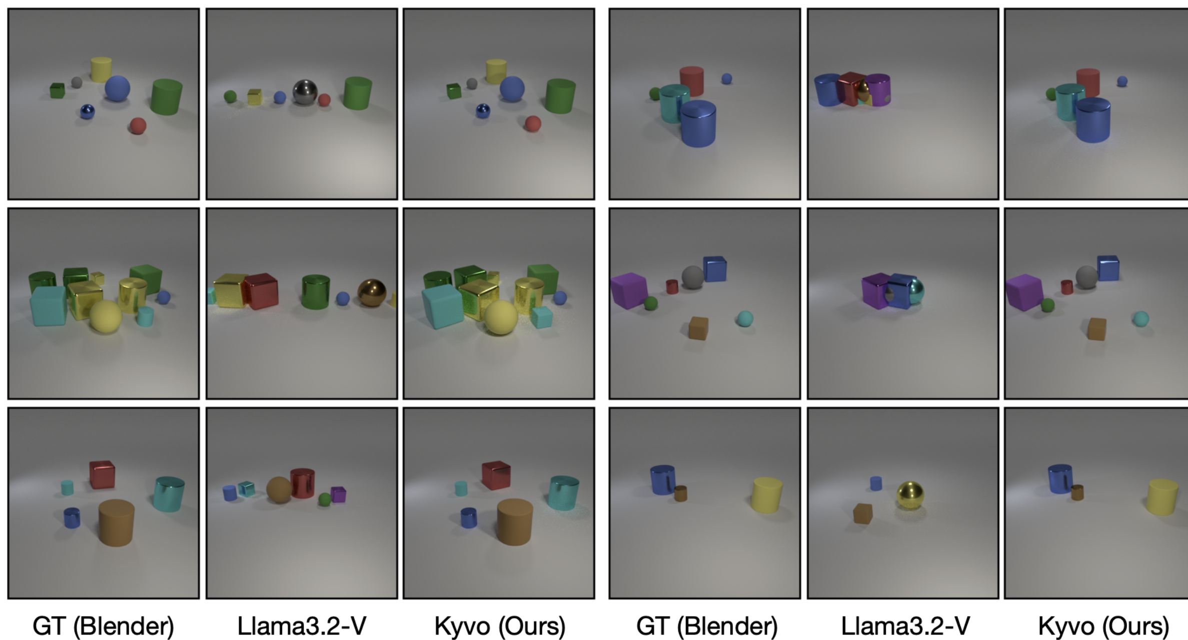


Figure 15. **Failure of Llama3.2-V on recognition task.** For six randomly selected scenes, we render (from left to right) the ground-truth scene, the recognition prediction scene by Llama 3.2-Vision, and the predicted scene by Kyvo (ours). Llama 3.2-Vision frequently collapses objects toward the centre or misplaces them entirely, failing to recover the true spatial layout even in this simple synthetic setting. Moreover, it also misidentifies some objects in the scene.

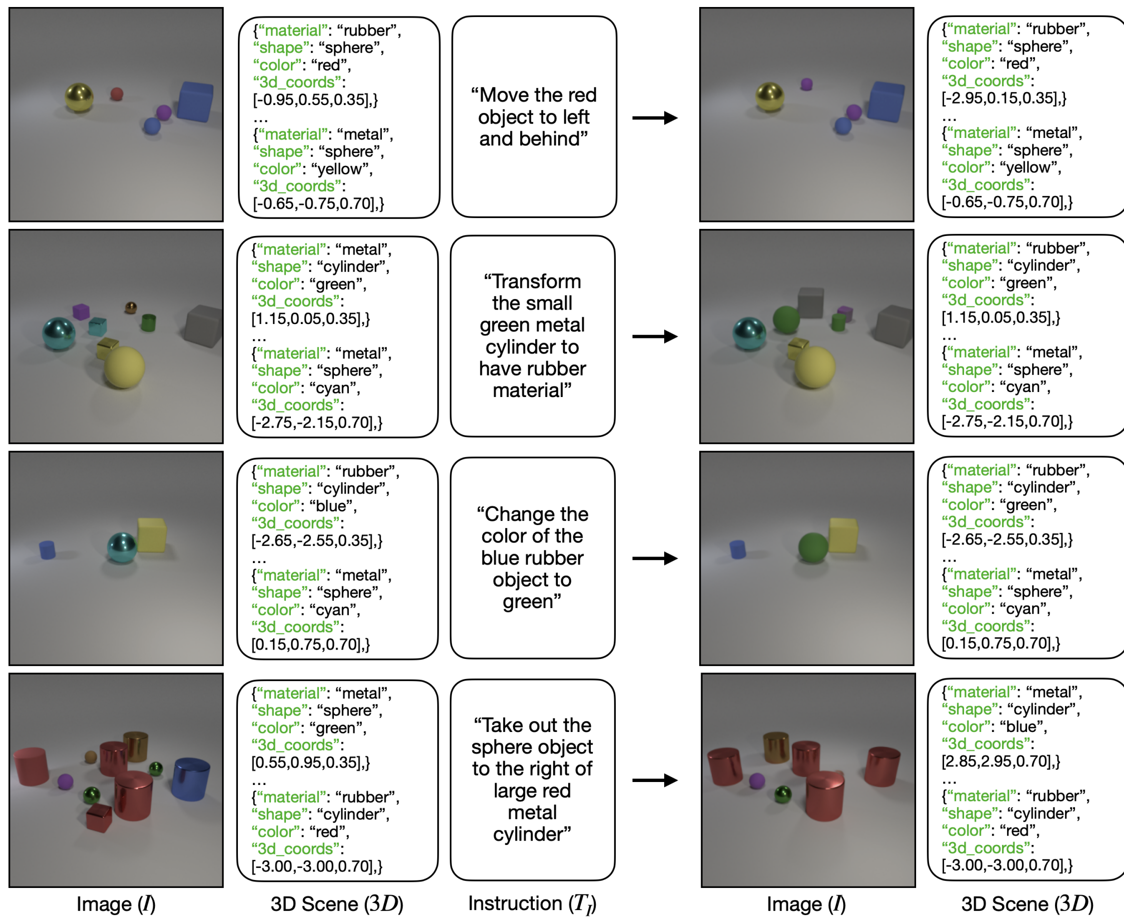


Figure 16. **Instruction-following failure cases for the image modality on CLEVR.**. As observed, the images generated by the model do not accurately reflect the intended modifications based on the input image and text instruction. On the other hand, the output 3D scenes are correct, meaning that our Kyvo accurately modified them based on the instructions. This suggests that a better avenue for predicting instruction-modified images is by task decomposition: first predict the modified 3D scene and then render the final image.

References

[1] Garrick Brazil, Abhinav Kumar, Julian Straub, Nikhila Ravi, Justin Johnson, and Georgia Gkioxari. Omni3d: A large benchmark and model for 3d object detection in the wild. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13154–13164, 2023. 2

[2] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13142–13153, 2023. 2

[3] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12873–12883, 2021. 12, 13

[4] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, and (additional authors not shown). The llama 3 herd of models, 2024. 3

[5] Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2901–2910, 2017. 1, 2

[6] Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong, and Jiaolong Yang. Structured 3d latents for scalable and versatile 3d generation. In *CVPR*, 2025. 8, 14