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

Aadarsh Sahoo
Computing and Mathematical Sciences
California Institute of Technology
asahoo@caltech.edu

Vansh Tibrewal
Computing and Mathematical Sciences
California Institute of Technology
vansh@caltech.edu

Georgia Gkioxari
Computing and Mathematical Sciences
California Institute of Technology
georgia@caltech.edu

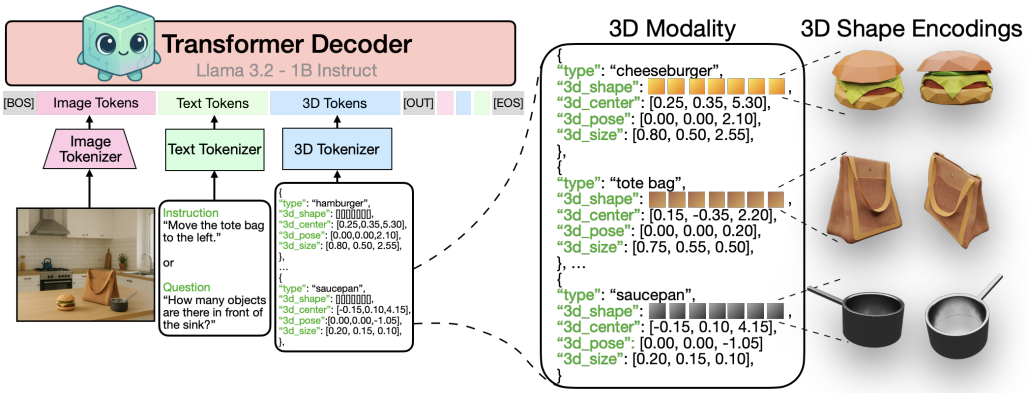


Figure 1: Kyvo: a decoder-only transformer aligns a structured 3D modality with language and vision. This 3D modality represents scenes as lists of objects, each defined by its 3D shape, type, 3D position, pose and size parameters. Kyvo unifies the token space of images, text, and 3D to enable a variety of complex visual 3D tasks.

Abstract

Creating machines capable of understanding the world in 3D is essential in assisting designers that build and edit 3D environments and robots navigating and interacting within a three-dimensional space. Inspired by advances in language and image modeling, we investigate the potential of autoregressive models for a new modality: structured 3D scenes. To this end, we propose a unified LLM framework that aligns language, images, and 3D scenes and provide a detailed "cookbook" outlining critical design choices for achieving optimal training and performance addressing key questions related to data representation, modality-specific objectives, and more. We evaluate performance across four core 3D tasks – rendering, recognition, instruction-following, and question-answering – and four 3D datasets, synthetic and real-world. We extend our approach to reconstruct complex 3D object shapes by enriching our 3D modality with quantized shape encodings, and show our model's effectiveness on real-world 3D object recognition tasks. Project webpage: https://glab-caltech.github.io/kyvo/.

Preprint. Under review.

1 Introduction

Large language models (LLMs) that fuse text and images have unlocked unprecedented visual capabilities, such as visual captioning and text-guided image generation. Inspired by this success, we explore a third modality – structured 3D scenes – and show how aligning it with text and images allows LLMs to tackle a new suite of visual tasks in 3D, such as 3D reconstruction from a single image and 3D-conditioned image generation.

Our structured 3D modality encodes a scene as a list of its objects, where every object is specified by its 3D shape (*e.g.*, 3D mesh, 3DGS), type, 3D position, 3D size and 3D pose, shown in Fig. 1. This modality captures aspects of the physical world that are not directly conveyed through language or images alone. Additionally, it slots naturally into the unified token space shared by language and vision through an object-by-object tokenization scheme that integrates seamlessly with image and text tokens, so any modality can serve as input or output. As a result, it supports a broad range of tasks in 3D, such as image generation conditioned on the 3D scene structure (3D $\rightarrow$ image), predicting 3D objects, their shapes and locations from a single image (image $\rightarrow$ 3D), and language-guided object-centric 3D editing (3D + image + text $\rightarrow$ 3D + image) – all within the same model design – shown in Fig. 2. These capabilities can reshape workflows. Robots can parse an image into a 3D scene composed of 3D objects, while designers can create complex scenes of objects through language in one forward pass of an LLM instead of wrestling with hard-to-use software like Blender.

But, how does one design and train an LLM that aligns this structured 3D modality with image and text? While there is extensive work on language-only [1, 13, 36] or vision-and-language models [37, 20, 6], there is limited work on training with an additional structured 3D modality. We explore the vast design space and evaluate the impact of design choices in architecture, training strategies, data representation, modality-specific objectives, and more. Our testbed consists of four core 3D tasks – image generation, recognition, instruction-following, and question-answering – and four challenging datasets, both synthetic and real-world. Through an extensive "cookbook" of rigorously validated empirical findings we hope to provide guidelines to design 3D-aligned multi-modal LLMs. We then extend our findings to demonstrate the effectiveness of our unified LLM framework, Kyvo (Greek for "3D cube"), to single-view 3D reconstruction of complex object geometries via a tokenization technique for 3D shape encodings. Finally, we apply our model to real-world domains of indoor and outdoor scenes for the task of image-based 3D object recognition, where we show that our model can outperform prior vision specialists designed for the task.

2 Related Work

LLMs. The success of LLMs is attributed to the scalability of transformers [40] taking the form of encoder-decoder [34, 10] and most popular decoder-only [1, 13, 36] models. LLMs showcase the effectiveness of autoregressive formulations in achieving task generalization, a finding we extend to 3D structured representations.

Vision-Language Models (VLMs). Early VLMs [30, 17, 3] pioneered learning from image-text pairs, focusing on zero-shot classification, cross-modal retrieval, and question-answering. With the advent of LLMs, modern VLMs [22, 8, 23, 26, 41, 43] expand these capabilities, leveraging internet-scale training data and improved alignment strategies. Several works [20, 6, 37] conduct vision-centric analysis and investigate key design choices for training VLMs. However, they limit images only as input. Others [35, 46, 38] explore early fusion and diffusion for images along with text to support image generation.

LLMs & 3D. Prior work integrates LLMs with various 3D formats for tasks like VQA and semantic grounding [25]. Some [24, 9, 21] extend VLMs to 3D reasoning via depth-aware image queries. Others [15, 33, 45, 27, 28, 19, 31] embed 3D data (*e.g.*, point clouds, NeRFs) with CLIP-style features to support free-form QA. Specifically, 3D-LLM [15] inputs holistic 3D scene point clouds (obtained from scanning techniques or structure-from-motion) and trains a language model for the task of language captioning and QA. In contrast, our model, via its structured 3D modality, decomposes scenes into 3D objects, enabling direct alignment with language and vision. This allows broader tasks like 3D shape and pose prediction from a single image and image generation from object-centric 3D scene input. SceneScript [4] uses an autoregressive decoder over videos and holistic scene point

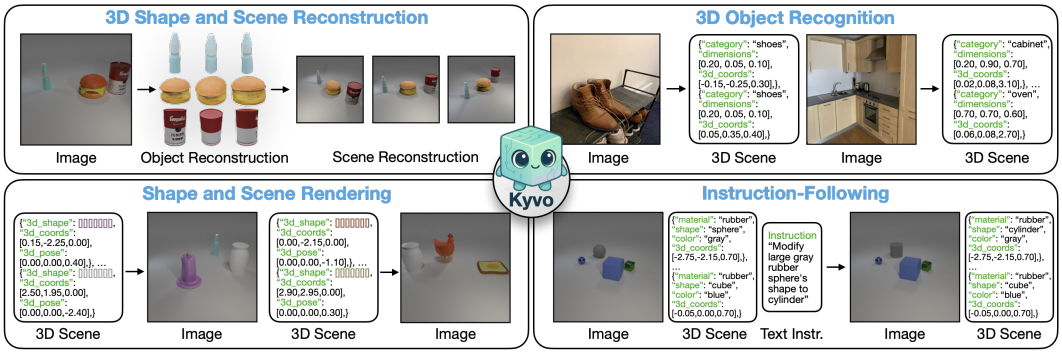


Figure 2: **3D task examples** with Kyvo's unified autoregressive framework using a structured 3D modality. (1) *3D shape and scene reconstruction:* From a single input image, Kyvo reconstructs individual objects with accurate geometry and spatial relationships. (2) *3D object recognition:* Given an input image, Kyvo identifies objects and predicts their 3D positions in real-world scenes. (3) *Shape and scene rendering:* Kyvo generates semantically consistent images from structured 3D scene inputs. (4) *Instruction-Following:* Given an image, 3D scene and text instruction, Kyvo produces coherent modifications to both image and the 3D representation.

clouds to predict 3D bounding boxes. In contrast, our method focuses on aligning images with structured 3D scene and object representations.

3 Building our model token-by-token

We present our approach, Kyvo, that aligns the 3D modality with text and images. The result is a unified multimodal LLM framework than can perform a range of visual 3D tasks, as shown in Fig. 2.

Sec. 3.1 outlines our experimental setup, including tasks and datasets. Sec. 3.2 presents a bottom-up analysis of key design choices – covering data representation, sequence design, and scaling – and addresses challenges in modality-specific tokenization and optimization. We summarize empirical insights and adopt optimal configurations for each subsequent experiment. Finally, we explore generalization to complex shapes (Sec. 3.3), 3D shape representations (Sec. 3.4), and real-world recognition (Sec. 3.5).

3.1 Setup: tasks and datasets

We design and train an autoregressive model aligning 3D with images and language, capable of performing complex 3D tasks.

Tasks. We focus on four core 3D tasks: image generation from 3D scene specifications (rendering), 3D object recognition from a single image, instruction-following 3D editing and question-answering.

Rendering: 3D $\rightarrow$ Image. Given a 3D scene described by our structured modality (object types, shapes, locations, poses), the model generates a corresponding image. This task tests whether rendering – typically requiring tools (*e.g.*, Blender) and complex processes (*e.g.*, ray tracing, rasterization) – can be reframed as feed-forward next-token prediction.

Recognition: Image $\rightarrow$ 3D. This is a dual of rendering. Given an image, the model predicts the underlying 3D scene structure, including object shapes, types, positions and poses.

Instruction-Following: $(\text{Image}, \text{3D}, \text{Text}_\text{I}) \rightarrow (\text{Image}, \text{3D})$. We define a set of tasks to manipulate and modify 3D scenes given a text instruction. These include: (1) *modifying the appearance of objects*, (2) *adding new objects*, (3) *removing objects*, and (4) *moving an object to a desired location*. For each subtask, we craft templated natural language instructions to evaluate the model's ability to follow instructions in 3D, *e.g.* "Remove the small red mug behind the large yellow metal bottle".

Question-Answering: $(\text{Image}, \text{3D}, \text{Text}_\text{Q}) \rightarrow \text{Text}_\text{A}$. We generate question-answer pairs for our 3D scenes using templated questions. The model is tasked to predict the answer in natural language, given the 3D scene, an image and the query, *e.g.* "Is there a green matte thing of the same size as the blue glass vase?". We provide more details of our instruction and QA dataset in Appendix A.

All four tasks require spatial reasoning and grounding. Current VLMs struggle with 3D and geometry prediction, and both rendering and recognition from a single image remain challenging due to issues like object misplacement and hallucinations. Failure cases are shown in Appendix F.

Datasets. We consider four datasets for our experiments: CLEVR [18] features scenes of simple shapes in varying layouts; ObjaWorld features complex objects of any geometry; Objectron [2] and ARKitScenes [5] comprising real-world indoor and outdoor scenes of various object types.

Our choice of datasets reflects varying complexity and practical use cases. We use CLEVR and design ObjaWorld to simulate the workflow of a design studio that composes and edits 3D scenes with 3D assets. Our autoregressive model provides an alternative to specialized 3D software by enabling image generation, scene inference, and editing in a single forward pass of an LLM, leveraging our structured 3D modality. ObjaWorld showcases our model's ability to generalize to complex shapes, supporting both asset-level geometry prediction and full scene reconstruction. Finally, Objectron and ARKitScenes demonstrate real-world applicability in robotics, AR, and assistive technologies.

Evaluation. Per our task definitions, Kyvo's output can be of either modality: text, image, or 3D.

Recognition. Here, the model outputs 3D scenes, evaluated using the Jaccard Index ($J = \frac{tp}{tp+fp+fn}$), which compares predicted and ground-truth objects based on matching attributes (type, size, color, material) and spatial proximity within a threshold τ. True positives (tp) are matched objects within τ; false positives (fp) are unmatched predictions; and false negatives (fn) are missed ground-truth objects. This metric captures both object identification and spatial accuracy, penalizing omissions and hallucinations. We report the average Jaccard Index over $\tau \in \{0.05, 0.10, 0.15, 0.20, 0.25\}$.

Question-Answering. We report answer accuracy based on exact match between predicted and ground-truth text (typically 1–2 tokens). A random baseline yields 0.359 accuracy, while a frequency-based baseline reaches 0.430. See Appendix C.3 for more details.

Rendering. In the rendering task, images are predicted from 3D scene inputs. Standard image metrics (L2, SSIM[42], FID [14], PSNR) fall short, as they don't capture object placement and attribute accuracy in the generated images. Instead, we rely on human evaluations, where annotators rank model outputs (anonymized) against the ground-truth scene, and we report the mean rank. While L2 and SSIM trends align with human judgment, we report human evaluations as they directly assess scene correctness. More details are provided in Appendix C.2.

Instruction-Following. Here, the model predicts both images and 3D scenes. We report the Jaccard Index for 3D outputs, as this is the primary focus of our work.

3.2 Cookbook

Our model, Kyvo, builds on the decoder-only Llama-3.2-1B-Instruct [13], originally designed for text-only tasks, and extends it with modality-specific tokenizers for images and the structured 3D modality, along with modified input embeddings and output projections (Fig. 1). We evaluate key architectural choices – each with significant performance impact – which we hope offer practical insights to guide future multimodal LLM development in the 3D domain.

We begin by exploring how to represent and tokenize our structured 3D modality using CLEVR, which contains 3–10 object scenes with varied layouts. Each scene is described by object type, color, material, size, and 3D position, and rendered in Blender to generate ground truth for all four tasks – totaling $120{,}000$ scenes. This setup helps establish best practices for integrating 3D into a unified LLM framework. We then extend these findings to complex geometries and shape encodings in ObjaWorld (Sec. 3.3 & Sec. 3.4) and validate generalization to real-world scenes in Objectron and ARKitScenes (Sec. 3.5).

3.2.1 What is the optimal representation for modalities?

Our model handles three modalities – images, text, and structured 3D scenes – by converting each into token sequences for autoregressive modeling. Below, we outline tokenization strategies for each and evaluate optimal design choices.

Text. We employ an off-the-shelf text tokenizer from Llama-3.2 [13] with a $128{,}000$ size vocabulary. Text instructions, questions, and answers are tokenized and enclosed within special, learnable tokens `[TEXT-START]` and `[TEXT-END]`.

Images. Our framework addresses tasks involving images as both inputs and outputs. To enable image generation within an autoregressive paradigm, we adopt discrete image representations using VQGAN [12]. This approach maps continuous image features to discrete tokens through a learned codebook. Specifically, we fine-tune a pre-trained VQGAN model to optimize the codebook for our visual domain. Image tokens are enclosed within special, learnable tokens `[IMAGE-START]` and `[IMAGE-END]` to form a sequence.

3D scene tokenization. We parse 3D scenes, containing 3D object information in the form of structured lists, to construct sequence of tokens. We encode object attributes – size, color, material, shape, and location – by introducing special, learnable tokens: `[SIZE]`, `[COLOR]`, `[MATERIAL]`, `[SHAPE]`, and `[LOCATION]`. Advanced attributes such as shape representations and pose are introduced later to address more complex tasks. Each object sequence is enclosed by `[OBJECT-START]` and `[OBJECT-END]`, and the full scene sequence is wrapped by `[SCENE-START]` and `[SCENE-END]`. An example scene with two objects is shown below.

```
[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]
```

The 3D location and orientation of an object are key attributes for understanding 3D spatial relationships and performing grounded 3D actions based on instructions. Thus, accurately encoding coordinates is essential. However, LLMs are inefficient at handling numbers [29, 32]. To mitigate this, we encode each object's x, y, z coordinates separately as individual tokens, allowing the model to learn distinct embeddings for each coordinate. We discretize the coordinate values using equally spaced bins based on a chosen granularity. We find this granularity to be decisive for performance – if it is too coarse, spatial inaccuracies arise; if it is too fine, it leads to an exponential increase in tokens, with fewer training samples per bin and thus difficulty in learning. Table 1 shows the effect of granularity across the four tasks on CLEVR. We find that a granularity of 0.05 outperforms the coarser 0.5 and the finer 0.005. Fig. 3 compares image generation for the rendering task on the test set across varying levels of granularity.

Granularity	Rendering(↓)	Recognition(↑)	Instruction(↑)	QA(↑)
0.005	1.380	0.5707	0.8643	**0.5185**
0.05	**1.200**	**0.9212**	**0.8666**	0.4980
0.5	2.020	0.2352	0.2427	0.4730

Table 1: **Effect of granularity.** A value of 0.05 yields the best overall performance on CLEVR.

Furthermore, our tokenization approach substantially improves efficiency by reducing sequence length compared to standard text tokenizers, which typically fragment floating-point values (e.g., "0.000" becomes "0", ".", "000"). We achieve a mean sequence length reduction from 271.4 tokens using the standard Llama-3.2 tokenizer to 93.2 tokens – a $2.91\times$ compression ratio. These efficiency gains directly translate to decreased memory requirements and faster training and inference times. The final vocabulary comprises $129{,}415$ tokens covering all modalities.

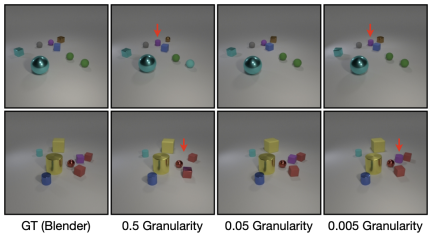


Figure 3: **Effect of Granularity.** A 0.05 granularity more accurately captures object locations and shapes.

3.2.2 What matters in input sequence design?

We combine three modalities to construct the input sequence to the model. How should we order the modalities? And how should we encode numbers? We discuss our findings for input sequence design.

Sine-cosine encoding of numbers. While we independently tokenized coordinate values, naively learning embeddings for these tokens may fail to capture the inherent ordering of the numbers (*e.g.*, 2 is between 1 and 3). We investigate whether augmenting the learned embeddings with sine-cosine encodings can enforce numeric ordering relationships. Specifically, we evaluate three encoding strategies: (1) fixed sine-cosine encodings, (2) learned embeddings initialized from scratch, and (3) a hybrid approach where embeddings are learned but augmented with sine-cosine encodings. Fig. 4 plots the effect of these encoding strategies on the recognition task with varying training

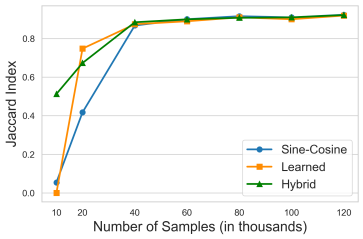
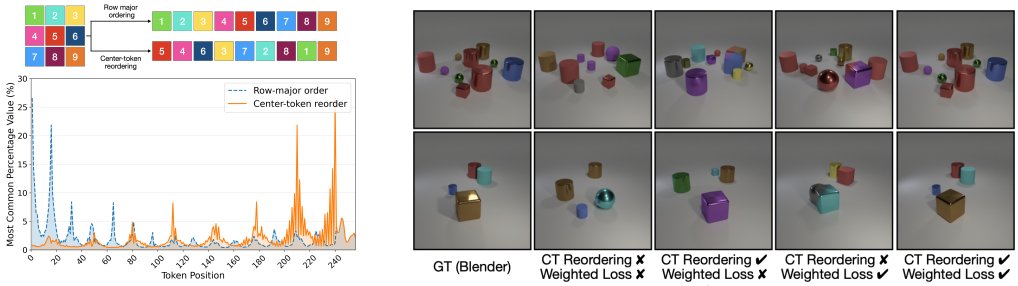


Figure 4: Hybrid number encodings are robust across training data scales.



(a) (b)

Figure 5: (a) **Top:** Schematic representation of center token reordering. **Bottom:** We convert CLEVR images into 256-token sequences and analyze token frequency. Over 25% of images share the same first token, causing biased predictions. Center token reordering significantly reduces this issue. (b) Inaccurate first-token predictions can cause catastrophic object- and scene-level diversions in autoregressive generations.

data sizes. While all methods perform on par in the high data regime, standalone fixed sine-cosine and learned embeddings collapse with low data. Consequently, we adopt the hybrid approach as it demonstrates robustness across data regimes. We show the performance of these encoding strategies across all four tasks in Appendix E.

Should the image or 3D come first? We investigate the impact of the ordering between the image and 3D modality for instruction-following and QA tasks. Our experiments reveal that placing the image before the 3D sequence leads to better performance compared to the reverse order. Specifically, we achieve an accuracy of 0.8666 with the sequence $(I, \text{3D}, T_I) \rightarrow (I, \text{3D})$ compared to 0.8350 with $(\text{3D}, I, T_I) \rightarrow (\text{3D}, I)$. Moreover, we obtain an accuracy of 0.4980 with $(I, \text{3D}, T_Q) \rightarrow T_A$ compared to 0.4720 with $(\text{3D}, I, T_Q) \rightarrow T_A$. This improvement could be attributed to the 3D tokens attending to the entirety of the preceding image tokens, enabling better conditioning and performance.

3.2.3 What matters in output sequence design?

We outline the important design decisions concerning the output sequence and objectives.

Initial token prediction matters. The next-token prediction scheme caused challenges in generating reliable image outputs during inference. Despite achieving low training loss, the model's decoded (next-token) predictions during inference deviated from expected outputs. At a high level, the issue stems from predicting rich outputs (images) from less informative conditions (3D specifications).

Further investigation revealed the issue: the first token. During inference, the model decodes sequentially, with the first token guiding the output. For CLEVR images, this token, representing the top-left corner, was biased toward a few codes – see Fig. 5a (blue dash plot) – due to CLEVR's uniform gray background. This caused overfitting and during inference a wrongly predicted first token caused the decoding to diverge. This finding is not just applicable to CLEVR but to any image set with a more uniform background, often the case in graphic and studio design. Moreover, this trend is even evident with real-world images which we show in Appendix E.

To mitigate this issue, we incorporate a *center-token reordering* scheme to balance the token distribution at the sequence's starting position. As shown in Fig. 5a (top), the sequence begins with the center token of the image, followed by alternating hops to the left and right until the image is covered. This reordering strategy alleviates the issue since the first token now captures a representative part of the scene, instead of an uninformative background patch. This is shown in Fig. 5a and by qualitative examples in Fig. 5b.

CT reorder	Weighted loss	Rendering(↓)
✗	✗	2.66
✓	✗	3.56
✗	✓	2.78
✓	✓	**1.00**

Table 2: Effect of center-token reordering and weighted loss.

Token-specific loss weighting. We apply a *weighted loss* during training by assigning a higher weight (10.0) to the loss for the first five tokens of the output image sequence. This enforces a stronger constraint to correctly predict the initial tokens, which proved critical for autoregressive decoding.

Table 2 shows the impact of center-token reordering and weighted loss for the rendering task. Best performance is achieved when both are combined, also shown qualitatively in Fig. 5b.

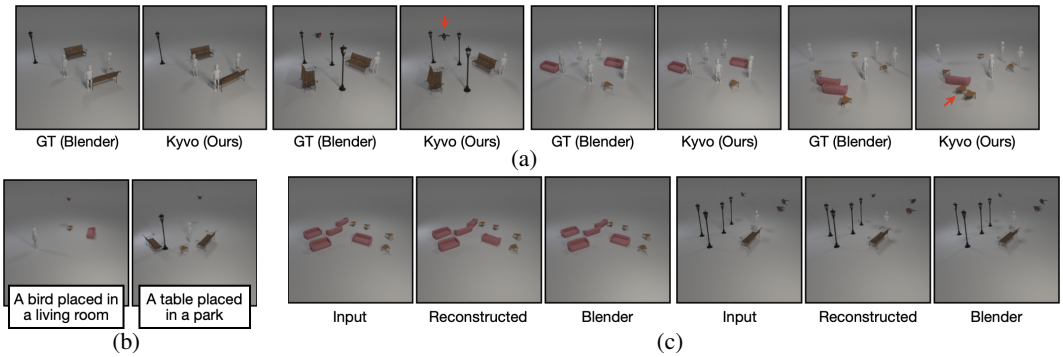


Figure 6: (a) **Rendering examples.** The model predicts images autoregressively from 3D inputs. Errors include pose mispredictions, *e.g.*, the bird, and missing objects, *e.g.* the sofa. (b) **Novel scene configurations.** (c) **Chaining tasks.** Our recognition model predicts the 3D scene representation for an input image, which is then visualized through both our rendering model and Blender.

3.3 Generalization to complex object shapes

Above, we validated the effectiveness of the structured 3D modality and developed a cookbook of findings for optimally training unified autoregressive models. We now investigate generalization to complex shapes using ObjaWorld.

ObjaWorld features scenes composed of Objaverse assets [11] arranged in diverse layouts. In this section, we focus on two scene categories: park scenes (*person, bench, lamppost, bird*) and living room scenes (*person, sofa, coffee table*), featuring realistic textured objects with varied geometry positioned at varying locations and heights. We generate $50{,}000$ park and $50{,}000$ living room scenes to create a training set of $100{,}000$ samples and train rendering and recognition models. Our test set consists of $2{,}000$ park and living room scenes of novel object configurations (locations and poses).

In the recognition task, the model receives a novel scene image and predicts object types, 3D locations, and poses. Kyvo achieves a Jaccard Index of 0.6415, notably lower than its CLEVR performance (0.9212, Table 1), which reflects the greater complexity of ObjaWorld scenes. By comparison, Llama3.2-V, prompted with in-context examples to output the same object attributes, performs near zero – failing to accurately predict (x, y, z) coordinates.

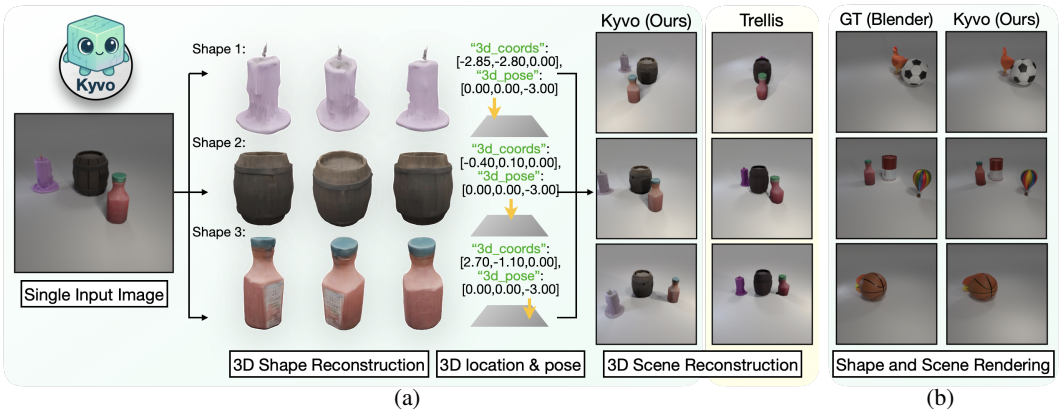
In the rendering task, the model takes a structured list of 3D objects – types, locations, and poses – and generates a corresponding image. This requires linking object types and poses to visual appearance and placing them correctly in 3D space. Fig. 6a compares model outputs to ground-truth Blender renderings, showing accurate object types, counts, positions, and poses, though some errors occur in fine pose alignment.

Generalizing to novel scene configurations. ObjaWorld's training scenes contain scene-specific objects (*e.g.*, *bird, bench, lamppost* appear only in park scenes). We test on out-of-distribution inputs, such as placing park objects in a living room. Fig. 6b shows our model's ability to generalize to such novel scenarios for the rendering task.

Chaining tasks. We show that chaining our recognition (image $\rightarrow$ 3D) and rendering (3D $\rightarrow$ image) models enables image reconstruction, providing a test of model robustness. Fig. 6c shows two examples: the input image, the reconstruction from our chained models, and a Blender rendering of the predicted 3D scene. The Blender view confirms accurate recognition of object types, poses, and positions, while our reconstructed images closely match the input, with only minor pose errors (*e.g.*, the sofa in the first example).

3.4 Extending the 3D modality for explicit shape representations

In Sections 3.2 & 3.3, we demonstrated the effectiveness of our structured 3D modality to align 3D information with images and text. To this end, we focused on type-specific object descriptions (*e.g.*, cube, sphere, cylinder in CLEVR & person, bench, bird *etc.* in ObjaWorld). We now extend our 3D modality to describe objects with their geometry using 3D shape representations. This requires the model to autoregressively encode and reconstruct geometric details of each object while



(a) (b)

Figure 7: (a) **Unified shape and scene reconstruction example.** Given a single input image, Kyvo predicts shape sequences and reconstructs individual objects (candle, barrel, bottle) 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. (b) **Shape and scene rendering examples.** Given the structured 3D modality as input, Kyvo renders images with consistent object appearance and spatial relationships.

simultaneously performing scene-level recognition and placement – a substantially more challenging task as it significantly increases the complexity of the prediction and input space.

3.4.1 Vector-quantized 3D shapes

To enable autoregressive models to handle explicit 3D shapes, we require compact sequences encoding geometry and appearance. We adopt Trellis [44] structured latents (slats), which represent objects as sparse voxels $z = \{(z_i, p_i)\}_{i=1}^{L}$, where each p_i indexes an active voxel in an N^3 grid and $z_i \in \mathbb{R}^8$ encodes local features. Despite their sparsity ($L \ll N^3$, typically $L \approx$ 20k for $N = 64$), slats pose challenges for autoregressive modeling: predicting both voxel positions and embeddings, and handling long sequences ($L \approx$ 20k is intractable during training and decoding)

To address this, we train a 3D VQ-VAE [39] on slats, compressing the $64^3 \times 8$ into a dense $8^3 \times 256$ representation, then vector-quantize it with a 8192-token codebook. Each object is thus represented by just 512 discrete tokens – a $\sim 40\times$ reduction – making autoregressive modeling tractable while preserving essential geometric information. These tokens are added to our unified vocabulary (final size: 137,607) and integrated into our structured 3D modality for joint training and inference. See Appendix D.3 for more details. With this compressed 3D shape representation, our structured 3D modality can integrate 3D shape encodings – denoted by `[SHAPE]` special tokens:

`[SCENE-START]` `[OBJECT-START]` `[SHAPE]` $<v_1^1, v_2^1, \ldots, v_{512}^1>$ `[LOCATION]` `-0.15 1.05 0.00`
`[POSE]` `0.00 0.00 3.00` `[OBJECT-END]` `[OBJECT-START]` `[SHAPE]` $<v_1^2, v_2^2, \ldots, v_{512}^2>$
`[LOCATION]` `0.25 2.10 0.00` `[POSE]` `0.00 0.00 -2.05` `[OBJECT-END]` `[SCENE-END]`

3.4.2 Unified 3D shape and scene understanding

We show that Kyvo, with the shape-augmented 3D modality, can reconstruct and render complex 3D objects and scenes. We build a variant of ObjaWorld with complex objects (*e.g.*, barrel, fire balloon, chicken) from Objaverse [11], and train recognition and rendering models on 100k image-scene pairs.

The recognition task requires the model to: (1) reconstruct full 3D geometry per object, even under occlusion, and (2) infer each object's 3D position and pose – all from a *single image*. Fig. 7a shows results on unseen scenes, where our model accurately recovers object geometries and spatial layouts via our structured, object-centric 3D modality. In contrast, Trellis [44], a diffusion-based image-to-3D method, trained on Objaverse, that models the scene as a whole, often produces distorted shapes (*e.g.*, deformed bottle) and misaligned layouts (*e.g.*, linearly arranged objects).

In the rendering task, the model inputs the structured 3D modality with embedded shape sequences and outputs the corresponding image. It must map shapes to appearance and position objects accurately by location and pose. Fig. 7b compares our model's outputs with Blender renderings. The model reliably captures object shapes and spatial relationships, with minor distortions in challenging cases (*e.g.*, the occluded cheeseburger behind the basketball in the third image).

Qualitative scaling behavior. Fig. 8 shows that with limited data, the rendering model captures coarse layouts but struggles with object geometry and positioning. As training data increases, object appearance and spatial accuracy improve, yielding more consistent renderings.

Figure 8: **Qualitative Scaling Behavior.** Training on limited data produces amorphous color blobs lacking semantic coherence. Increasing training data progressively improves geometric precision and spatial relationships.

3.5 Generalization to real-world recognition

We now evaluate our model's effectiveness for 3D object recognition in real-world scenes. We conduct experiments on two challenging real-world datasets: Objectron [2], which features indoor and outdoor scenes of diverse object categories (*e.g.*, bicycle, camera, car, cereal box); and ARKitScenes [5], featuring complex indoor environments of many object categories (*e.g.*, bathtub, bed, cabinet, chair). ARKitScenes presents additional complexity due to its scene density and ground truth annotation noise. Following the train-test splits by [7], we train a recognition model to detect objects by type and predict their 3D center coordinates and size dimensions in metric space. During training, we augment the data with random horizontal flipping. Table 3 reports the Jaccard Index of our Kyvo and a state-of-the-art 3D object detector Cube R-CNN [7] – we apply a 0.05 confidence threshold to their predictions, as recommended by the authors. Kyvo significantly outperforms Cube R-CNN with two backbone variants on Objectron and performs on par on the more challenging ARKitScenes dataset. This result demonstrates the potential of a general autoregressive framework that aligns images with the 3D modality.

Model	Objectron	ARKitScenes
Cube R-CNN (ResNet-34)	0.3276	0.2043
Cube R-CNN (DLA-34)	0.4012	0.2208
Kyvo (Ours)	0.4784	0.2118

Table 3: Jaccard-accuracy comparison with Cube R-CNN.

3.6 Analysis and observations

We explore the effects of training strategies and model backbones for Kyvo.

Training recipes. Table 4 compares three approaches for model adaptation: training from scratch, LoRA [16], and full fine-tuning (FFT). FFT from pre-trained language-only weights yields superior performance even when adapting to image and 3D modalities unseen during pre-training – suggesting effective cross-modal transfer with limited domain-specific data. Notably, LoRA performs worse despite its established efficacy for text-only adaptation, indicating limitations when incorporating entirely new modalities.

Recipe	Rendering(↓)	Recognition(↑)	Instruction(↑)	QA(↑)
Scratch	1.36	0.6265	0.7744	0.4645
LoRA	1.82	0.8684	**0.8680**	0.3950
FFT	**1.26**	**0.9212**	0.8666	**0.4980**

Table 4: **Training recipe.** From scratch, LoRA vs FFT.

Instruction-tuned backbones and model sizes. Instruction-tuned backbones outperform or are on par with non-instruction-tuned ones across all tasks, as shown in Table 5. However, increasing the model size from 1B to 3B parameters provides no significant gains, with question-answering performance declining, indicating that the 1B model sufficiently captures our dataset's complexity while avoiding overfitting.

Backbone	Rendering(↓)	Recognition(↑)	Instruction(↑)	QA(↑)
Llama-3.2-1B	1.38	0.8948	0.8674	0.4490
Llama-3.2-1B-Instruct	1.28	**0.9212**	0.8666	**0.4980**
Llama-3.2-3B-Instruct	**1.18**	0.8626	**0.8763**	0.2345

Table 5: **Effect of backbone** varying size and instruction-tuning.

4 Conclusion & Limitations

We introduce Kyvo, a autoregressive model that aligns structured 3D with language and vision to support a broad range of 3D visual tasks. Our empirical "cookbook", based on training **307 models**, outlines effective design choices, and we demonstrate how our 3D modality scales to complex shape encodings. We will release code and data.

Limitations: We cover limitations through scaling behavior, qualitative and quantitative results. A key challenge is the limited availability of 3D data. While we show that strong performance and within-domain generalization can be achieved with relatively modest training samples, achieving cross-domain generalization demands larger datasets, which are not readily available in 3D domains. A promising direction is extending Kyvo to handle mixed training data when 3D is not always available as a paired modality. This could enable generalization to new domains, even in the absence of 3D information.

Acknowledgments

We thank Damiano Marsili, Raphi Kang, Ilona Demler, and Ziqi Ma for their valuable feedback. Aadarsh is supported by the Kortschak Scholarship. Georgia is supported by the Powell Foundation, Meta, Google and Amazon.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Adel Ahmadyan, Liangkai Zhang, Artsiom Ablavatski, Jianing Wei, and Matthias Grundmann. Objectron: A large scale dataset of object-centric videos in the wild with pose annotations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7822–7831, 2021.

[3] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35:23716–23736, 2022.

[4] Armen Avetisyan, Christopher Xie, Henry Howard-Jenkins, Tsun-Yi Yang, Samir Aroudj, Suvam Patra, Fuyang Zhang, Duncan Frost, Luke Holland, Campbell Orme, et al. Scenescript: Reconstructing scenes with an autoregressive structured language model. In *ECCV*, 2024.

[5] Gilad Baruch, Zhuoyuan Chen, Afshin Dehghan, Tal Dimry, Yuri Feigin, Peter Fu, Thomas Gebauer, Brandon Joffe, Daniel Kurz, Arik Schwartz, and Elad Shulman. ARKitscenes - a diverse real-world dataset for 3d indoor scene understanding using mobile RGB-d data. In *NeurIPS Datasets and Benchmarks Track (Round 1)*, 2021.

[6] Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, et al. Paligemma: A versatile 3b vlm for transfer. *arXiv preprint arXiv:2407.07726*, 2024.

[7] 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.

[8] Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, et al. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 24185–24198, 2024.

[9] An-Chieh Cheng, Hongxu Yin, Yang Fu, Qiushan Guo, Ruihan Yang, Jan Kautz, Xiaolong Wang, and Sifei Liu. Spatialrgpt: Grounded spatial reasoning in vision language model. *arXiv preprint arXiv:2406.01584*, 2024.

[10] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024.

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

[12] 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.

[13] 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.

[14] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[15] Yining Hong, Haoyu Zhen, Peihao Chen, Shuhong Zheng, Yilun Du, Zhenfang Chen, and Chuang Gan. 3d-llm: Injecting the 3d world into large language models. *Advances in Neural Information Processing Systems*, 2023.

[16] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[17] Chao Jia, Yinfei Yang, Ye Xia, Yi-Ting Chen, Zarana Parekh, Hieu Pham, Quoc Le, Yun-Hsuan Sung, Zhen Li, and Tom Duerig. Scaling up visual and vision-language representation learning with noisy text supervision. In *International conference on machine learning*, pages 4904–4916. PMLR, 2021.

[18] 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.

[19] Justin Kerr, Chung Min Kim, Ken Goldberg, Angjoo Kanazawa, and Matthew Tancik. Lerf: Language embedded radiance fields. In *ICCV*, 2023.

[20] Hugo Laurençon, Léo Tronchon, Matthieu Cord, and Victor Sanh. What matters when building vision-language models? *arXiv preprint arXiv:2405.02246*, 2024.

[21] Yuan-Hong Liao, Rafid Mahmood, Sanja Fidler, and David Acuna. Reasoning paths with reference objects elicit quantitative spatial reasoning in large vision-language models. *arXiv preprint arXiv:2409.09788*, 2024.

[22] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36, 2024.

[23] Haoyu Lu, Wen Liu, Bo Zhang, Bingxuan Wang, Kai Dong, Bo Liu, Jingxiang Sun, Tongzheng Ren, Zhuoshu Li, Hao Yang, et al. Deepseek-vl: towards real-world vision-language understanding. *arXiv preprint arXiv:2403.05525*, 2024.

[24] Chenyang Ma, Kai Lu, Ta-Ying Cheng, Niki Trigoni, and Andrew Markham. Spatialpin: Enhancing spatial reasoning capabilities of vision-language models through prompting and interacting 3d priors. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[25] Xianzheng Ma, Yash Bhalgat, Brandon Smart, Shuai Chen, Xinghui Li, Jian Ding, Jindong Gu, Dave Zhenyu Chen, Songyou Peng, Jia-Wang Bian, et al. When llms step into the 3d world: A survey and meta-analysis of 3d tasks via multi-modal large language models. *arXiv preprint arXiv:2405.10255*, 2024.

[26] Yiyang Ma, Xingchao Liu, Xiaokang Chen, Wen Liu, Chengyue Wu, Zhiyu Wu, Zizheng Pan, Zhenda Xie, Haowei Zhang, Liang Zhao, et al. Janusflow: Harmonizing autoregression and rectified flow for unified multimodal understanding and generation. *arXiv preprint arXiv:2411.07975*, 2024.

[27] Ziqi Ma, Yisong Yue, and Georgia Gkioxari. Find any part in 3d. *arXiv preprint arXiv:2411.13550*, 2024.

[28] Damiano Marsili, Rohun Agrawal, Yisong Yue, and Georgia Gkioxari. Visual agentic ai for spatial reasoning with a dynamic api. In *CVPR*, 2025.

[29] Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*, 2024.

[30] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[31] William Shen, Ge Yang, Alan Yu, Jansen Wong, Leslie Pack Kaelbling, and Phillip Isola. Distilled feature fields enable few-shot language-guided manipulation. *arXiv preprint arXiv:2308.07931*, 2023.

[32] Aaditya K Singh and DJ Strouse. Tokenization counts: the impact of tokenization on arithmetic in frontier llms. *arXiv preprint arXiv:2402.14903*, 2024.

[33] Yuan Tang, Xu Han, Xianzhi Li, Qiao Yu, Yixue Hao, Long Hu, and Min Chen. Minigpt-3d: Efficiently aligning 3d point clouds with large language models using 2d priors. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 6617–6626, 2024.

[34] Yi Tay, Mostafa Dehghani, Vinh Q Tran, Xavier Garcia, Jason Wei, Xuezhi Wang, Hyung Won Chung, Siamak Shakeri, Dara Bahri, Tal Schuster, et al. Ul2: Unifying language learning paradigms. *arXiv preprint arXiv:2205.05131*, 2022.

[35] Chameleon Team. Chameleon: Mixed-modal early-fusion foundation models. *arXiv preprint arXiv:2405.09818*, 2024.

[36] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[37] Shengbang Tong, Ellis Brown, Penghao Wu, Sanghyun Woo, Manoj Middepogu, Sai Charitha Akula, Jihan Yang, Shusheng Yang, Adithya Iyer, Xichen Pan, et al. Cambrian-1: A fully open, vision-centric exploration of multimodal llms. *arXiv preprint arXiv:2406.16860*, 2024.

[38] Shengbang Tong, David Fan, Jiachen Zhu, Yunyang Xiong, Xinlei Chen, Koustuv Sinha, Michael Rabbat, Yann LeCun, Saining Xie, and Zhuang Liu. Metamorph: Multimodal understanding and generation via instruction tuning. *arXiv preprint arXiv:2412.14164*, 2024.

[39] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

[40] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[41] Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024.

[42] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[43] Zhiyu Wu, Xiaokang Chen, Zizheng Pan, Xingchao Liu, Wen Liu, Damai Dai, Huazuo Gao, Yiyang Ma, Chengyue Wu, Bingxuan Wang, et al. Deepseek-vl2: Mixture-of-experts vision-language models for advanced multimodal understanding. *arXiv preprint arXiv:2412.10302*, 2024.

[44] 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.

[45] Runsen Xu, Xiaolong Wang, Tai Wang, Yilun Chen, Jiangmiao Pang, and Dahua Lin. Pointllm: Empowering large language models to understand point clouds. In *ECCV*, 2024.

[46] Chunting Zhou, Lili Yu, Arun Babu, Kushal Tirumala, Michihiro Yasunaga, Leonid Shamis, Jacob Kahn, Xuezhe Ma, Luke Zettlemoyer, and Omer Levy. Transfusion: Predict the next token and diffuse images with one multi-modal model. *arXiv preprint arXiv:2408.11039*, 2024.

Appendix Contents

A Dataset details 15

A.1 Data generation 15

A.2 3D scene serialization 16

A.3 Tokenization 16

A.4 Task sequence formation 17

B Additional qualitative examples 18

C Evaluation 20

C.1 3D scenes 20

C.2 Images 21

C.3 Text 24

D Additional implementation details 25

D.1 Image VQGAN architecture 25

D.2 Encoding of numbers 26

D.3 3D VQ-VAE training 27

D.4 Compute resources and time 28

E Additional experiments and observations 29

F Failure cases 30

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 [18] 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 6.

Question-answering. For question-answer pair generation, we use the question generation engine by [18] 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.

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

ObjaWorld. We extend the CLEVR framework to support 3D Blender assets from Objaverse [11], 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 object shapes (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 extension of the 3D modality for explicit shape representations (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 [7] 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 [7] and use the provided dataset splits and extract object category labels, 3D center coordinates, and dimensions from annotations to generate image-3D scene pairs.

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 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 [13] 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 3.4.1 of the main paper. We provide more details on training the 3D VQ-VAE in Section D.3 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 all tasks across the four datasets.

CLEVR:

Rendering: 3D → 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 → 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: $(\text{Image}, \text{3D}, \text{Text}_\text{I}) \rightarrow (\text{Image}, \text{3D})$

```
[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: (Image, 3D, Text_Q) $\rightarrow$ 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] <$v_1^1, v_2^1, \ldots, v_{512}^1$> [LOCATION] 2.10 0.15 -0.75
[POSE] 0.00 0.00 -2.10 [OBJECT-END] [OBJECT-START] [SHAPE] <$v_1^2, v_2^2, \ldots, v_{512}^2$>
[LOCATION] -1.25 2.85 0.05 [POSE] 0.00 0.00 1.85
[OBJECT-END] [OBJECT-START] [SHAPE] <$v_1^3, v_2^3, \ldots, v_{512}^3$> [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] <$v_1^1, v_2^1, \ldots, v_{512}^1$>
[LOCATION] 1.15 -2.85 0.40 [POSE] 0.00 0.00 -1.75
[OBJECT-END] [OBJECT-START] [SHAPE] <$v_1^2, v_2^2, \ldots, v_{512}^2$> [LOCATION] -0.35 2.10 -0.50
[POSE] 0.00 0.00 2.45 [OBJECT-END] [OBJECT-START] [SHAPE] <$v_1^3, v_2^3, \ldots, v_{512}^3$>
[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.

Rendering. We present qualitative results for the rendering task on CLEVR and ObjaWorld with complex shapes in Figures 9 and 10, respectively. Results on ObjaWorld with explicit shape representations are shown in Figure 11. 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 11 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 9, the model fails to predict a small red cube positioned at the front. Similarly, in the last example of Figure 10, the model mispredicts the bird's pose, causing it to face the wrong direction. In the third column of comparisons in Figure 11, 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 12 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 ground truth JSON in the figure. To evaluate recognition performance, we match the predicted

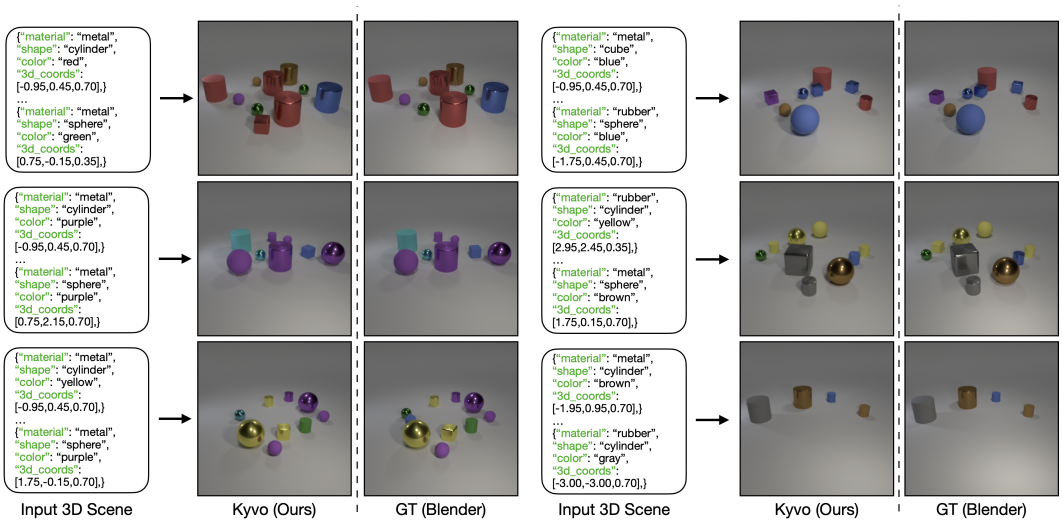


Figure 9: **Rendering examples for CLEVR.** Example image generations for the rendering task on CLEVR. The model takes a 3D scene as input and produces a corresponding image. Additionally, we show the ground-truth image rendered using Blender.

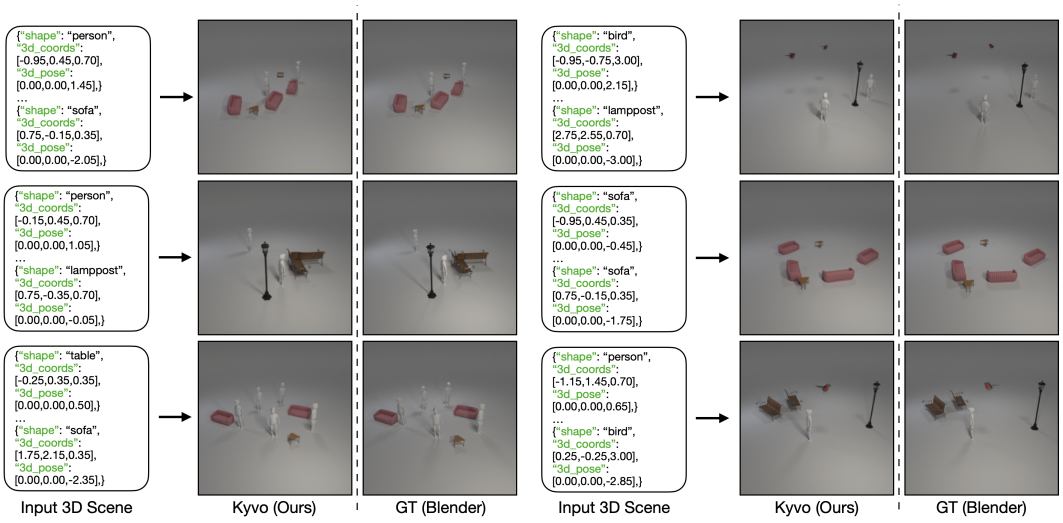


Figure 10: **Rendering examples for ObjaWorld.** Example image generations for the rendering task on ObjaWorld with complex shapes. The model takes a 3D scene as input and produces a corresponding image. Additionally, we show the ground-truth image rendered using Blender.

Figure 11: **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.

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 using colored numbers in Figure 12, 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 13 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 13, Kyvo accurately recovers object geometries and spatial layouts via our structured, object-centric 3D modality. In contrast, Trellis [44] 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 14, 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 modalities.

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:

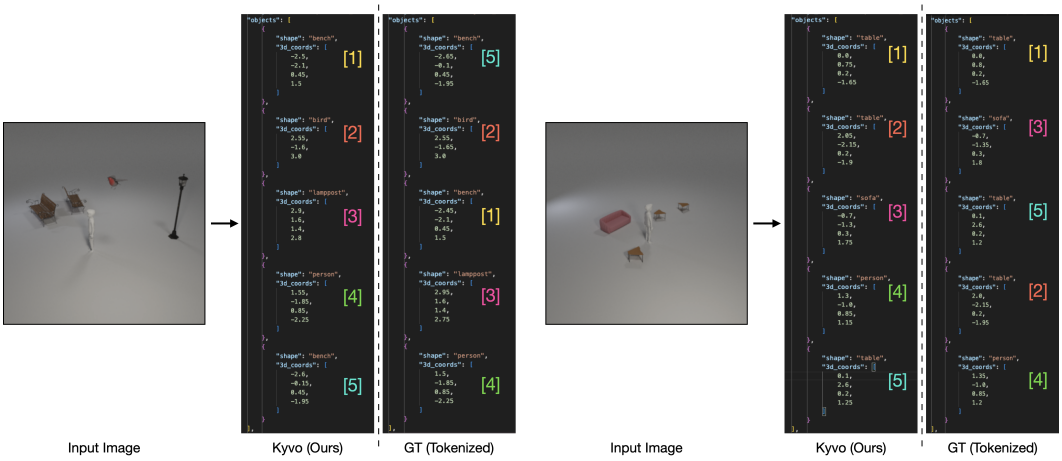


Figure 12: **Recognition examples for ObjaWorld.** Two example predictions from the recognition task on ObjaWorld. The colored numbers indicate object matching between the predicted and ground-truth scenes, based on the criteria for Jaccard Index as defined in Algorithm 1. Note that the fourth number in the list is the azimuth pose value, this format of prediction saves sequence length.

- 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: $\tau \in \{1.25, 1.50, 1.75, 2.00, 2.25\}$

Lower τ values impose stricter spatial constraints, requiring predicted objects to be closer to ground-truth positions. Figure 15 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 7, main paper; Figure 13). 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 16. 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

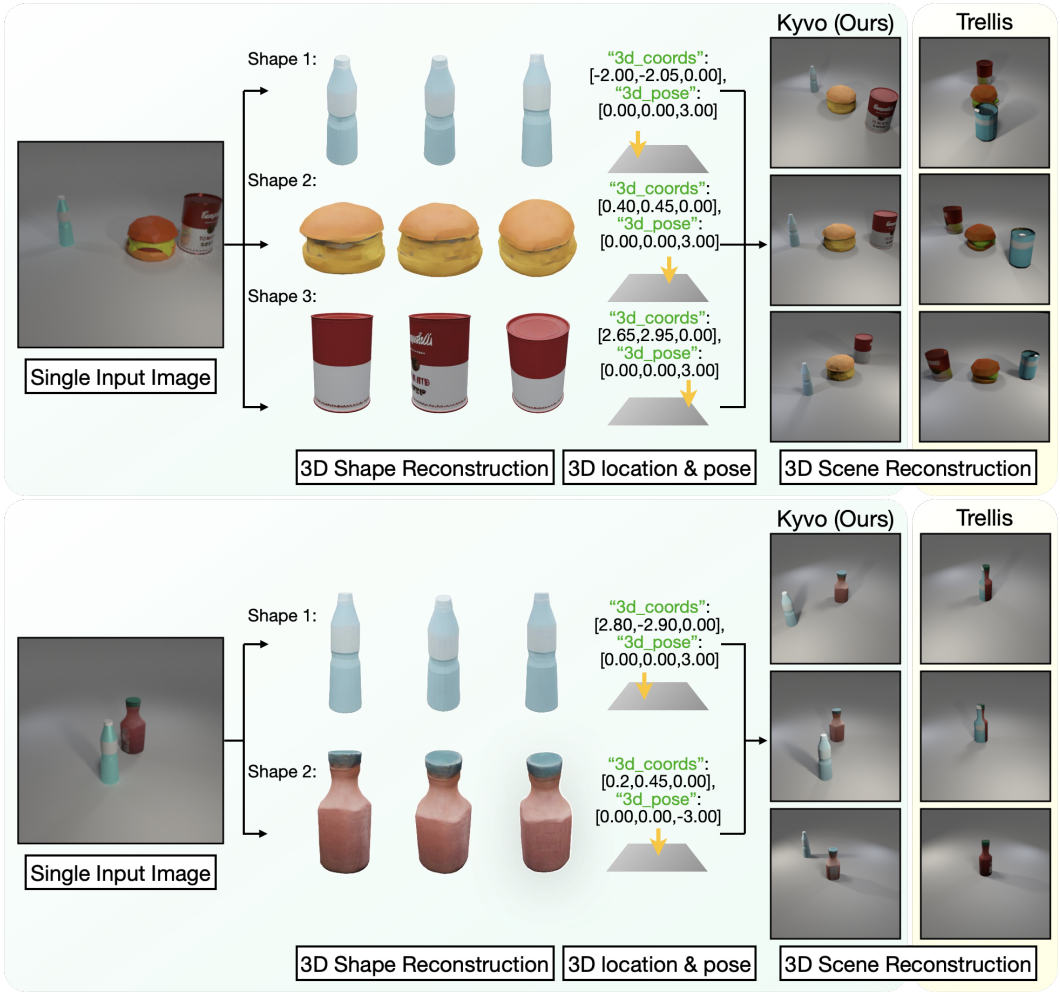


Figure 13: **Unified shape and scene reconstruction examples.** Given a single input image, Kyvo predicts shape sequences and reconstructs individual objects (bottle, cheeseburger, *etc.*) along with their 3D locations and poses via our structured 3D modality, effectively reconstructing the 3D scene with consistent spatial relations between the objects, visualized using Blender.

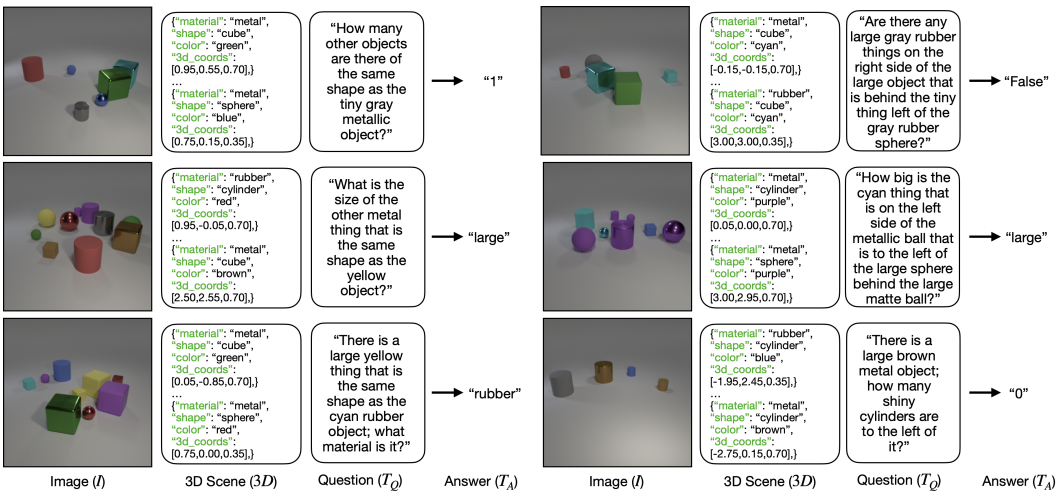


Figure 14: **Question-answering examples for CLEVR.** Example cases from the question-answering task on CLEVR. The model takes an image, a 3D scene, and a question as input to generate the corresponding answer.

Algorithm 1 Compute Jaccard Index

Input: GTS (list of ground-truth scenes), PS (list of predicted scenes), distance threshold τ
Output: Average JaccardIndex
Initialize `JaccardIndex` = 0
for (G, P) in zip(GTS, PS) **do**
  `GObjs` ← G.objects
  `PObjs` ← P.objects
  `matchedFlags` ← Boolean array of length |`GObjs`|, initialized to `False`
  Initialize `TP` = 0, `FP` = 0, `FN` = 0
  for `p` in `PObjs` **do**
    `foundMatch` ← `False`
    for $j = 1$ to |`GObjs`| **do**
      if $\neg$`matchedFlags`$[j]$ and `attributesMatch`(`p`, `GObjs`$[j]$) and `dist`(`p.coords`, `GObjs`$[j]$`.coords`) $< \tau$ **then**
        `matchedFlags`$[j]$ ← `True`
        `foundMatch` ← `True`
        `TP` ← `TP` + 1
        break
      end if
    end for
    if `foundMatch` = `False` **then**
      `FP` ← `FP` + 1
    end if
  end for
  `FN` ← `FN` + count(`matchedFlags` = `False`)
  `JaccardIndex` += $\frac{\texttt{TP}}{\texttt{TP+FP+FN}}$
end for
`JaccardIndexAvg` ← `JaccardIndex`/|PS|
return `JaccardIndexAvg`

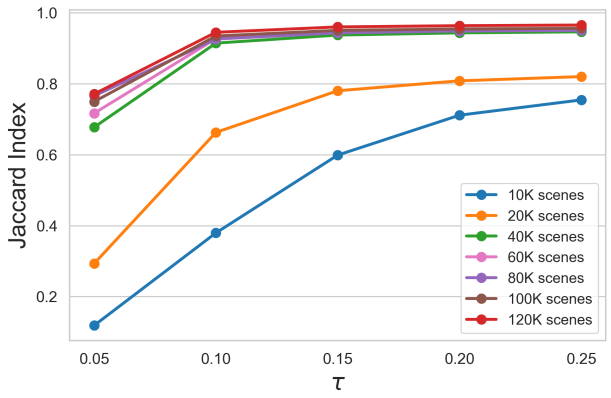


Figure 15: **Effect of τ.** The plot shows the impact of τ on the Jaccard Index for models trained with increasing amounts of training data on CLEVR for the recognition task. The drop in Jaccard Index with decreasing τ is more pronounced for models trained on smaller datasets. Higher-performing models demonstrate greater robustness to changes in τ.

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 16 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 2 of the main 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 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 7.

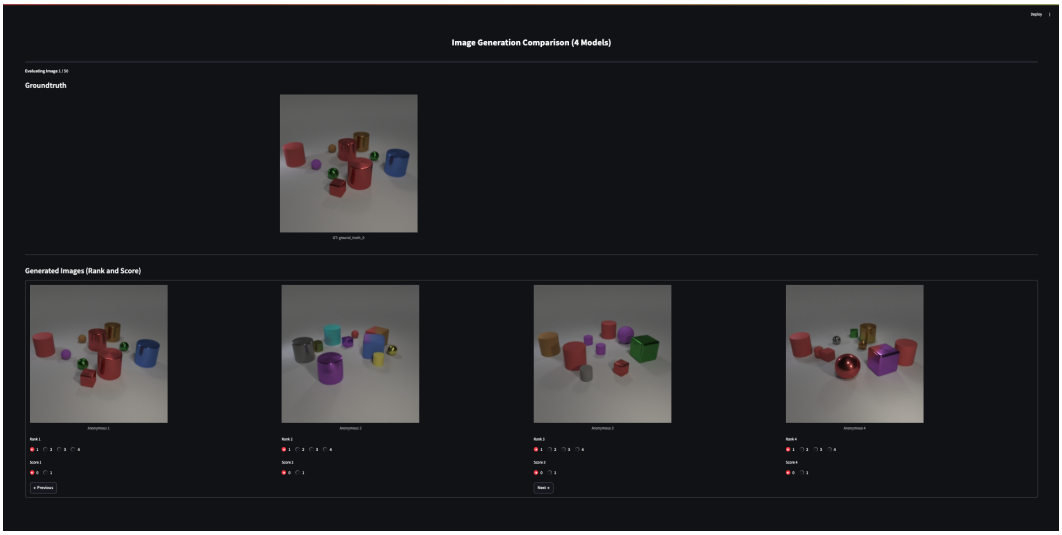


Figure 16: **Human Evaluation User Interface.** A snapshot of the user interface used for human evaluation of generated images. This example is taken from an experiment analyzing the effect of center-token reordering and weighted loss, comparing four models. The results of this experiment are presented in Table 2 of the main paper.

SSIM and L2-loss. Evaluation of the generated images requires careful assessment of the objects and their attributes in the scene. For example, consider the example predictions in Figure 17. 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 7. While the values show similar trends to human evals, we report human evals in the main paper as they directly assess image correctness.

C.3 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

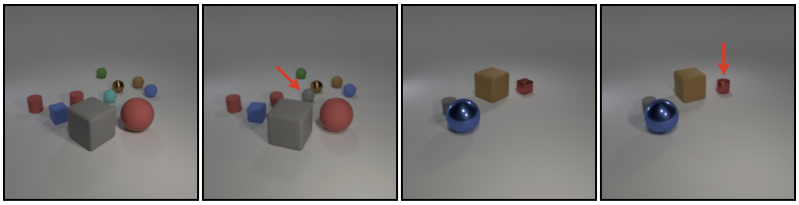


Figure 17: Object-level nuances are challenging for image metrics, like SSIM and L2-loss, to capture, prompting the need for human evaluation. The predicted image is incorrect in both cases but differs only subtly from the groundtruth.

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 7: **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.

involve text words describing attributes like "small","green", "metal", *etc.*. Table 8 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 8), 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. Figure 18 illustrates the complete network architecture, with output shapes shown for each layer given an input of shape (1, 3, 256, 256). Our VQGAN configuration uses a 1024-entry codebook with 256-dimensional embeddings, trained on 256×256

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

resolution images. 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 dataset, we initialize the VQGAN model with ImageNet pretrained weights from [12], 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.

```
VQModel                                  [1, 3, 256, 256]          17,481,793
├─Encoder: 1-1                          [1, 256, 16, 16]          --
│    └─Conv2d: 2-1                      [1, 128, 256, 256]        3,584
│    └─ModuleList: 2-2                  --                        --
│    │    └─Module: 3-1                 --                        738,944
│    │    └─Module: 3-2                 --                        738,944
│    │    └─Module: 3-3                 --                        2,690,304
│    │    └─Module: 3-4                 --                        2,952,448
│    │    └─Module: 3-5                 --                        10,498,048
│    └─Module: 2-3                      --                        --
│    │    └─ResnetBlock: 3-6            [1, 512, 16, 16]          4,721,664
│    │    └─AttnBlock: 3-7              [1, 512, 16, 16]          1,051,648
│    │    └─ResnetBlock: 3-8            [1, 512, 16, 16]          4,721,664
│    └─GroupNorm: 2-4                   [1, 512, 16, 16]          1,024
│    └─Conv2d: 2-5                      [1, 256, 16, 16]          1,179,904
├─Conv2d: 1-2                           [1, 256, 16, 16]          65,792
├─VectorQuantizer2: 1-3                 [1, 256, 16, 16]          --
│    └─Embedding: 2-6                   [256, 256]                262,144
├─Conv2d: 1-4                           [1, 256, 16, 16]          65,792
├─Decoder: 1-5                          [1, 3, 256, 256]          --
│    └─Conv2d: 2-7                      [1, 512, 16, 16]          1,180,160
│    └─Module: 2-8                      --                        --
│    │    └─ResnetBlock: 3-9            [1, 512, 16, 16]          4,721,664
│    │    └─AttnBlock: 3-10             [1, 512, 16, 16]          1,051,648
│    │    └─ResnetBlock: 3-11           [1, 512, 16, 16]          4,721,664
│    └─ModuleList: 2-9                  --                        --
│    │    └─Module: 3-12                --                        19,679,744
│    │    └─Module: 3-13                --                        4,855,296
│    │    └─Module: 3-14                --                        4,133,632
│    │    └─Module: 3-15                --                        1,215,232
│    │    └─Module: 3-16                --                        887,040
│    └─GroupNorm: 2-10                  [1, 128, 256, 256]        256
│    └─Conv2d: 2-11                     [1, 3, 256, 256]          3,459
```

Figure 18: **VQGAN Architecture.** We show the detailed architecture of the VQGAN model that we use to train a domain-specific codebook. We show the output shapes for an input shape (1, 3, 256, 256).

D.2 Encoding of numbers

In Section 3.2.2 (Figure 4), 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 n^{th} number is encoded using an embedding of dimension d. First, we obtain the sine-cosine encoding as follows:

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

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.3 3D VQ-VAE training

In Section 3.4.1 of the main paper we propose to compress the structured latents (SLAT) representation proposed by Trellis [44] into a sequence of discrete tokens to represent 3D shapes in the autoregressive framework. Although a SLAT already contains only $\sim$20k sparse voxels (in a 64^3 grid, with vectors of dimension 8 at active positions), the resulting sequence is still an order of magnitude longer than what current language-style models handle comfortably. Moreover, predicting the active voxel locations along with the latent embeddings is a challenge in the next-token prediction framework. We train a 3D VQ–VAE and disentangle these aspects in three 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 $8^3 \times 256$ latent volume, and (iii) vector-quantizes every latent with an 8192-entry codebook. We provide more details for training the 3D VQ-VAE below.

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 256-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 100k steps on 5000 Sketchfab assets from Objaverse (identifiers released with code). We use an AdamW optimizer (β_1=0.9, β_2=0.999) with a constant learning-rate of 3×10^{-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, $z \in \mathbb{R}^{8^3 \times 256}$ denotes the 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 $\hat{I}$. 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.

Qualitative reconstructions. Figure 19 demonstrates reconstruction quality across three conditions: ground-truth assets, Trellis-only reconstructions (using solely the frozen encoder-decoder), and our 3D VQ-VAE reconstructions on held-out assets. Note that while the test assets were seen during Trellis training, they remain unseen by our VQ-VAE components. Despite compression through discrete quantization, our reconstructions preserve essential structural and textural characteristics. While not pixel-perfect, the fidelity proves sufficient for downstream autoregressive modeling tasks, successfully balancing compression efficiency with perceptual quality necessary for sequence generation.

Codebook usage. Figure 20 shows codebook utilization across 5000 training assets (log-scale, decreasing order, unused codes omitted). The heavy-tailed distribution indicates effective learning: frequent codes capture common 3D primitives while rare codes encode geometric variations. Active

(a) Groundtruth (b) Reconstructions w/ Trellis (c) Reconstructions w/ 3D VQ-VAE

Figure 19: **Qualitative reconstructions.** The figure compares (a) ground-truth 3D assets, (b) reconstructions from the frozen Trellis encoder–decoder, and (c) our 3D VQ-VAE reconstructions on the same held-out assets. Even after latent compression, the VQ-VAE preserves key geometry and texture; residual artifacts are minor and do not hinder downstream autoregressive modeling.

utilization of a substantial codebook fraction demonstrates appropriate vocabulary size of 8192 for 3D asset diversity.

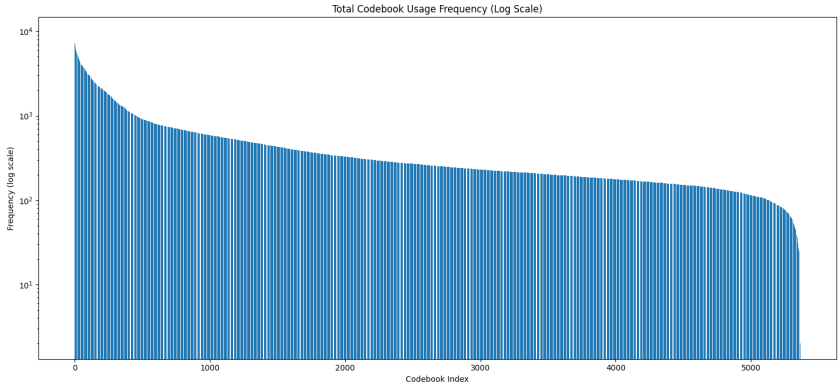


Figure 20: **Codebook usage.** Histogram of usage counts (log scale) for the 8192 latent codes on 5000 assets used for training (unused codes are omitted). The heavy-tailed distribution reveals a compact core of frequently reused codes that capture ubiquitous 3D primitives, while the long tail represents rarer geometric variations. In total, $\approx 70\%$ of the vocabulary is exercised at least once, indicating that the codebook is neither under- nor over-parameterised for the diversity of real-world assets.

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 [44] code.

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 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 21). 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 22 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.

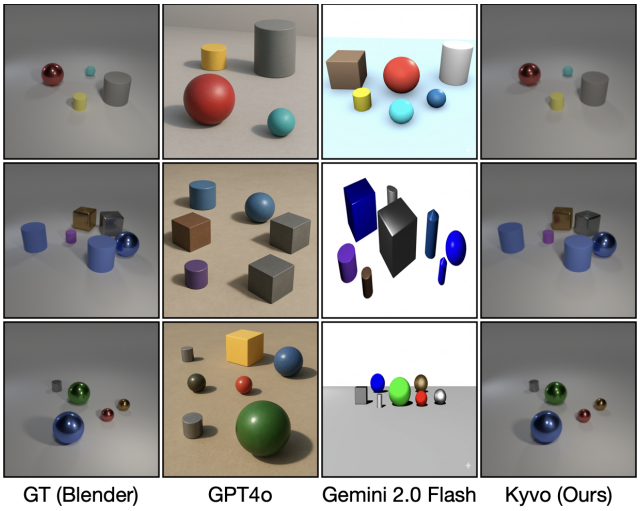


Figure 21: **Failure of modern-day LLMs on rendering task.** Each row depicts one test scene described by our 3D structured modality; columns show the ground-truth Blender render (GT) and images produced by GPT4o, Gemini 2.0 Flash, and Kyvo (ours). GPT4o and Gemini frequently hallucinate extra objects, omit specified ones, or displace shapes, violating fundamental xyz and relational constraints.

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 9: **Encoding strategies for numbers.** Performance of each strategy across the four tasks.

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 4 of the main paper, with implementation details provided in Section D.2. While Figure 4 assessed the performance of these strategies across data regimes, Table 9 summarizes the performance of these encoding strategies across all four tasks for the largest training set size.

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

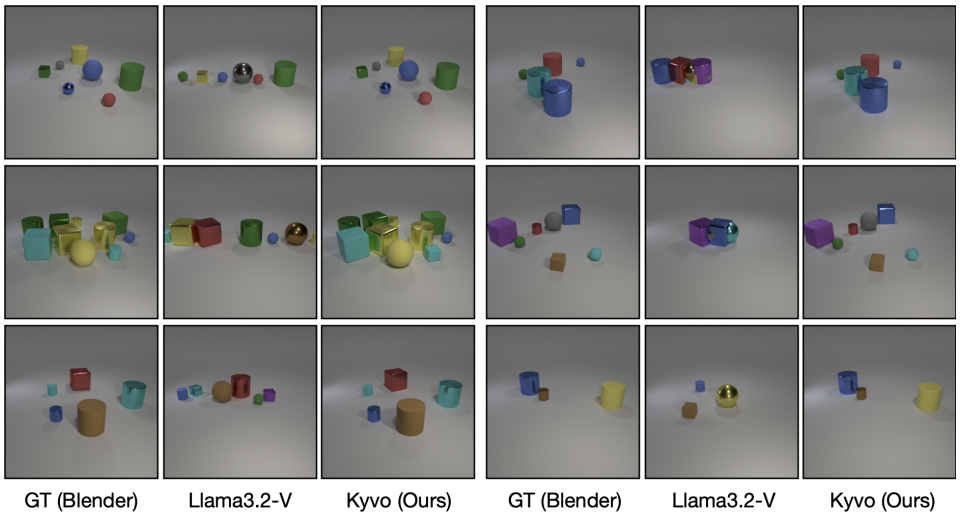
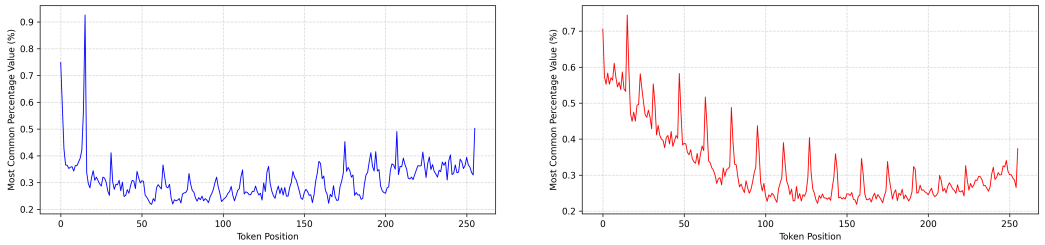


Figure 22: **Failure of Llama3.2-V on recognition task.** For six randomly selected scenes, we render (from left to right) the ground-truth scene, the recognition prediction scene by Llama 3.2-Vision, and the predicted scene by Kyvo (ours). Llama 3.2-Vision frequently collapses objects toward the centre 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.

substantial bias attributed to the uniform gray backgrounds prevalent in synthetic scenes. Interestingly, in Figure 23 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 23: 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 24 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.

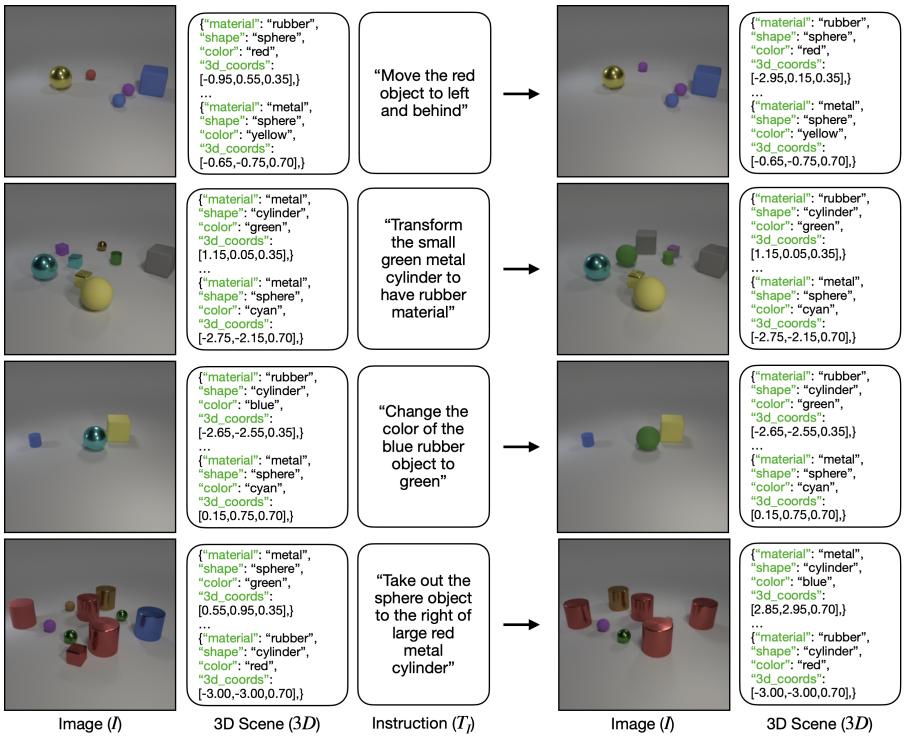


Figure 24: **Instruction-following failure cases for the image modality on CLEVR.**. As observed, the images generated by the model do not accurately reflect the intended modifications based on the input image and text instruction. On the other hand, the output 3D scenes are correct, meaning that our 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.

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.