3D-BlockGen: Generating Brick-Compatible 3D Objects Using Diffusion Models

Peter A. Massih
EPFL,
Supervisors: Martin Nicolas Everaert, Eric Bezzam
Professor: Sabine Süsstrunk
Contact: peter.abdelmassih@epfl.ch

Abstract

We present 3D-BlockGen, an exploration approach for generating Brick-compatible (ex: LEGO®) 3D models from text descriptions using conditional diffusion models. Our method addresses the time-consuming nature of manual design of 3D models using bricks/parts, by automating the generation step through a multi-stage diffusion pipeline. Working with resolution of $32 \times 32 \times 32$ *(Height, Width, Depth) voxels, we first introduce a shape generation stage that learns occupancy, followed by an RGB color prediction stage that adds colors while maintaining the geometry through the previous step. The system is trained on a subset of the Objaverse dataset and processed through a voxelization process that handles both shape and color information. We demonstrate that our approach can generate 3D structures with detailed color patterns and provide an automated "LegoLization" algorithm that converts the generated voxel into physically constructible LEGO® designs. Our results show the potential of diffusion models in automated 3D content creation, especially for the construction of tangible 3D models with brick requirements.*

1. Introduction

Creating 3D models, such as LEGO®-compatible constructions, is normally a manual and time-intensive task. To automate this process, we present 3D-BlockGen, a pipeline that transforms text to brick usable 3D models using a conditional diffusion model followed by a post processing legolization algorithm. Our approach employs a two-stage diffusion process:

1. Occupancy Generation: The first stage predicts the 3D shape by learning voxel occupancy, creating a structural representation.

2. Color Prediction: The second stage build upon this structure by assigning colors to the voxels ensuring the model matches the true color of the object's shape (ex: a banana should be yellow, or if conditioned on a certain color the color should be shown on the generated 3D model).

The figure below 1 conceptually shows the transformation from a text prompt to a voxelized 3D shape and then to a fully colored 3D voxelized model.

To support further research and reproducibility, we make the dataset [12] and codebase [11] available. The dataset is derived from Objaverse [3] and processed using a custom voxelization pipeline, capturing both geometric and color attributes. Additionally, we introduce a "LegoLization" algorithm that translates voxelized models into realizable LEGO® designs.

In this paper, we detail the dataset preparation process, including voxelization and augmentation, the diffusion model architecture, the inference techniques used for prediction, and the post-processing algorithms for LEGO® generation using pre-defined bricks. Results are evaluated using metrics like Intersection over Union (IoU), F1 and color similarity. Finally, we perform ablation studies on inference parameters and model configurations to identify the factors that affect the quality of generation.

2. Related Work

Recent years have seen significant progress in 3D generative models, specifically in diffusion-based approaches. While early methods relied on GANs [4] and VAEs [8] to operate on various representations like point clouds and meshes, these approaches often struggled with mode collapse (single type of output) and limited details on the 3D model. Recent work has shown diffusion models to be increasingly effective for high-quality 3D generation.

2.1. Diffusion Models for 3D Generation

XCube [19] represents a major advance in high-resolution 3D generation by introducing a hierarchical (coarse to fine generation) voxel latent diffusion model.

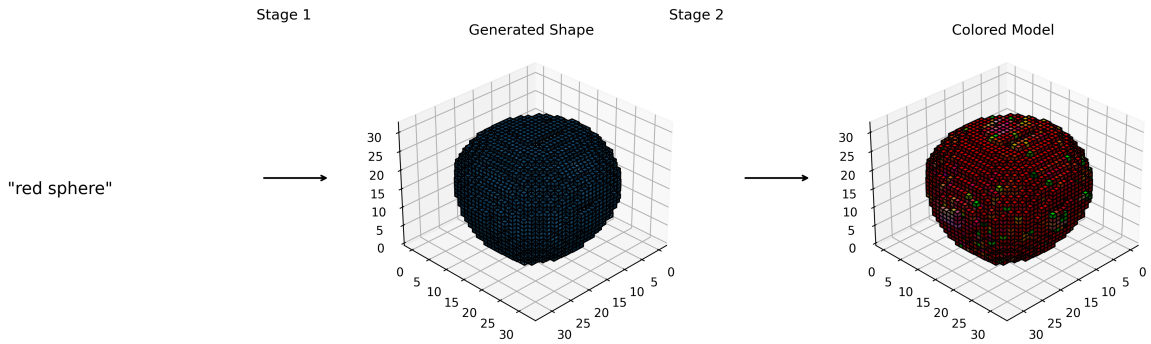

Figure 1. Illustration of the 3D-BlockGen pipeline: from text description to occupancy generation to final colored model. Using guidance scale=20 (for shape and color), no mean initialization and no rotation-augmented sampling

Their method uses a sparse voxel hierarchy that permits generation at resolutions up to 1024^3. While achieving impressive results, their focus on maximum resolution makes physical construction of generated models sometimes challenging.

Taking a different architectural approach, DiT-3D [15] uses pure transformer architectures for 3D shape generation. Building on 2D diffusion transformers, they incorporate 3D positional and patch embeddings to effectively learn complex 3D information.

LION [24] introduces a hierarchical latent point diffusion model that combines global shape representation with point-structured latent space. Their two-level architecture supports shape denoising and text-conditioned generation. But the architecture always output the same number of point clouds.

2.2. Structure and Appearance Integration

DiffRF [16] tackles joint geometry and appearance generation through volumetric radiance fields. By applying diffusion directly to the radiance field representation and introducing a volumetric rendering loss, they achieve high-quality results with an FID score of 15.95 on PhotoShape Chairs dataset [18].

Recent work by Nunes et al. [17] demonstrates the scalability of diffusion models to real-world LiDAR scene completion by formulating the diffusion process for raw point clouds. While their method focuses on geometric detail of the scene, it focuses only on geometry without considering appearance (color).

Our work takes a different approach by separating these concerns through a two-stage diffusion pipeline. In contrast to XCube's hierarchical latent diffusion or DiT-3D's transformer architecture, we operate directly on voxel representations at a fixed resolution (32^3) which is useful for brick-based construction by grouping voxels together. While DiffRF presents generation thanks to volumetric rendering (Radiance fields), our method first establishes the model's structures through occupancy prediction, then handles RGB values while preserving the shape through an alpha mask. This approach maintains control over geometry and color and makes sure outputs can be realized using standard brick systems (as in LEGO®).

3. Background on Diffusion Models

Diffusion models [6, 22] are a class of generative models that map noise to data by means of a gradual noising process (forward) and a corresponding denoising process (reverse). Let $\{x_t\}_{t=0}^{T}$ be a sequence of latent variables of the same dimensionality as the data, where t indexes discrete diffusion time-steps.

3.1. Forward Process (Adding Noise)

We define a schedule of scalars $\{\alpha_t\}_{t=0}^{T}$ in $[0, 1]$, where each α_t represents the *cumulative* noise-scaling factor up to time t. [1] We assume $\alpha_0 = 1$ for $t = 0$. In the *forward* (diffusion) process, a clean data sample x_0 is gradually corrupted over T steps by adding Gaussian noise:

$$x_t = \sqrt{\alpha_t}\, x_0 + \sqrt{1-\alpha_t}\, \epsilon, \quad \epsilon \sim \mathcal{N}(0, I). \tag{1}$$

As t increases, α_t decreases towards 0, causing x_t to become increasingly noisy. By $t = T$, x_T is close to pure noise.

3.2. Reverse Process (DDPM and DDIM)

The *reverse* process seeks to denoise x_T back to x_0. A neural network $\epsilon_\theta(x_t, t)$ is trained (e.g. via a mean-squared error) to predict the noise present in x_t. In the traditional **Denoising Diffusion Probabilistic Models (DDPM)**, a small amount of random noise is sampled at each reverse step. A more general approach, **Denoising Diffusion Implicit Models (DDIM)**, introduces a noise-scaling term σ_t

[1] In many references, α_t is defined step-by-step as $1-\beta_t$, and its product $\prod_{s=1}^{t}(1-\beta_s)$ is denoted $\bar{\alpha}_t$. Here, for simplicity, we call that product directly α_t.

that can be *tuned* to transition between deterministic and stochastic sampling.

We write a unified reverse-update formula (cf. Equation (12) in the DDIM paper [23]), moving from x_t to x_{t-1}:

$$x_{t-1} = \sqrt{\alpha_{t-1}} \underbrace{\left(\frac{x_t - \sqrt{1-\alpha_t}\,\epsilon_\theta(x_t, t)}{\sqrt{\alpha_t}}\right)}_{\text{“predicted } x_0\text{”}} + \underbrace{\sqrt{1-\alpha_{t-1}-\sigma_t^2}\,\epsilon_\theta(x_t, t)}_{\text{“direction pointing to } x_t\text{”}} + \underbrace{\sigma_t\,\epsilon_t}_{\text{random noise}}, \quad (2)$$

$$\epsilon_t \sim \mathcal{N}(0, I).$$

Here, $\sigma_t \geq 0$ is a user-defined schedule that specifies the standard deviation of Gaussian noise added at each reverse step. Different choices for $\{\sigma_t\}$ yield different sampling properties:

- **If $\sigma_t > 0$ for all t,** one recovers a fully *stochastic* reverse process. Choosing σ_t to match the forward-process variance yields the standard DDPM sampler [6].
- **If $\sigma_t = 0$ for all t,** the process is *deterministic*, yielding DDIM sampling [23]. This often allows skipping many intermediate reverse steps (e.g., from $T = 1000$ down to $T = 100$ or 200) with minimal quality loss, which significantly speeds up generation.

Summary We train using the DDPM objective, since it robustly learns to predict noise at all timesteps, and then choose an appropriate schedule $\{\sigma_t\}$ at inference time. This approach lets us flexibly trade off between sample quality, diversity, and speed in a unified framework.

4. Methodology

Our methodology for generating brick-compatible 3D models consists of four key components: data processing, model architecture, inference strategies and post-processing for brick constraint. In the first section, we first detail our data processing pipeline, which forms the foundation for training our diffusion models. We then describe our model architecture and training approach, followed by inference techniques and finally our LegoLization algorithm.

4.1. Data Processing Pipeline

4.1.1 Dataset Collection

We utilize Objaverse, a comprehensive dataset of 3D models with associated metadata. From the full dataset, we initially selected 200,000 models. To ensure efficient processing, we filtered out models larger than 50MB, resulting in approximately 135,000 usable models for our pipeline. We used trimesh [14] in order to extract colors and discretize the 3d-objects. Each model is paired with descriptive text combining its name, category, and associated tags (e.g., ”wooden tray, furniture-home, wooden, substancepainter, 3d, blender” as shown in Figure 2).

4.1.2 Voxelization Pipeline

Our pipeline converts 3D mesh models into a discrete 32^3 voxel representation through two main stages:

1. **Mesh Preprocessing:**
   - Scale mesh to fit in 32^3 grid with one voxel margin
   - Center mesh at origin
   - Fill holes in mesh surface, using the flood fill algorithm
2. **Voxelization Process:**
   - Subdivide triangles using 0.5 voxel unit edge length
   - Round vertex positions to the nearest voxel grid points (e.g., $3.7 \rightarrow 4$)
   - Create binary occupancy grid
   - For colored meshes, compute the average color of vertices within each occupied voxel.

4.1.3 Color Processing

Color information is processed hierarchically based on availability:

1. Vertex Colors: Extract per-vertex color information when available through the mesh’s visual properties. (using UV Mapping)
2. Material Colors: Use the mesh’s material base color when vertex colors are unavailable, applying it uniformly across all vertices

The final voxelized output is stored in one of two formats:

- Models with color normalize RGB values to [0,1] range: 4-channel tensor [4, 32, 32, 32] (RGB + occupancy)
- Models without color: Single-channel tensor [1, 32, 32, 32] (occupancy only)

4.1.4 Data Augmentation

To enhance model robustness, we apply three 90-degree rotations to each model:

- Rotation around X-axis (modifying Y,Z coordinates)
- Rotation around Y-axis (modifying X,Z coordinates)
- Rotation around Z-axis (modifying X,Y coordinates)

Figure 3 shows this augmentation that effectively quadruples our dataset size while preserving shape characteristics.

4.1.5 Dataset Analysis

The dataset contains approximately 135,000 voxelized 3D models, with 40% including color information. The average voxel occupancy is 5%, reflecting the sparsity typical of 3D models. RGB values are balanced, with mean intensities of 0.45 (Red), 0.43 (Green), and 0.40 (Blue). The occupancy heatmap of the middle slice (over the Z-dimension) reveals a spherical structure, with a higher concentration of occupied voxels in the left region, which might be corresponding to the base or lower parts of the models as shown in Figure 4. Data augmentation with 90-degree rotations increases the dataset size to **542,292** models.

4.2. Model Architecture

4.2.1 UNet3D Architecture

At the core of our approach is a modified UNet3D [21] architecture adapted from the Hugging Face Diffusers library. This model serves as our noise predictor ϵ_θ in the diffusion process. While traditional 3D UNets with full 3D convolutions would require vast amounts of memory (more than 100GB VRAM), we employ an efficient architecture that primarily uses 2D operations while maintaining 3D coherence.

The model processes input tensors of shape [B, C, D, H, W] where B is the batch size, C is the number of channels (1 or 4), and D, H, W are the spatial dimensions (all 32 in our case). As shown in Figure 5, our architecture consists of:

1. **Initial Processing**:
   - Input tensor [B, C, 32, 32, 32] is reshaped to [B×32, C, 32, 32]
   - Initial temporal transformer processes frames with both self-attention and text cross-attention
2. **Downsampling Path**: Consists of four blocks that progressively reduce spatial dimensions while increasing feature channels (64 → 128 → 256 → 512). Each CrossAttnDown block (except the last) combines:
   - ResNet2D with time embedding for spatial features
   - Temporal Conv1D operating on 3-frame sliding windows
   - Transformer2D block:
     - Self-attention on flattened features
     - Cross-attention with text embeddings
   - TransformerTemporal block:
     - Self-attention across frames
     - Cross-attention with text embeddings

   The final DownBlock4 contains only ResNet2D and temporal convolution.
3. **Middle Block**: Processes features at the lowest resolution (2×2) with 512 channels, using the same four components as CrossAttn blocks.
4. **Upsampling Path**: Mirror of the downsampling path with skip connections. The first block (UpBlock) contains only ResNet2D and temporal convolution, while the remaining CrossAttnUp blocks match the structure of CrossAttnDown blocks.

This architecture achieves memory efficiency (38GB of VRAM with batch size 4 vs more than 100GB for standard 3D UNets with batch 1) through:

- Treating frames (depth) as batch elements to enable 2D operations
- Using 1D convolutions for local temporal relationships (3-frame windows)
- Combining spatial and temporal attention mechanisms
- Maintaining cross-attention with text throughout for conditioning

The model contains approximately 120M parameters (120,096,449 for 1-channel input/output shape model, 120,099,331 for 4-channel input/3-channel output color model, and 120,099,908 for 4-channel input/output combined model).

4.2.2 Multi-Stage Training and Loss Functions

Our approach implements three distinct training strategies with the UNet3D that predicts noise. For each strategy, we reconstruct the estimated clean sample from the predicted noise using:

$$\hat{x}_0 = \frac{x_t - \sqrt{1-\alpha_t}\epsilon_\theta(x_t, t)}{\sqrt{\alpha_t}} \tag{3}$$

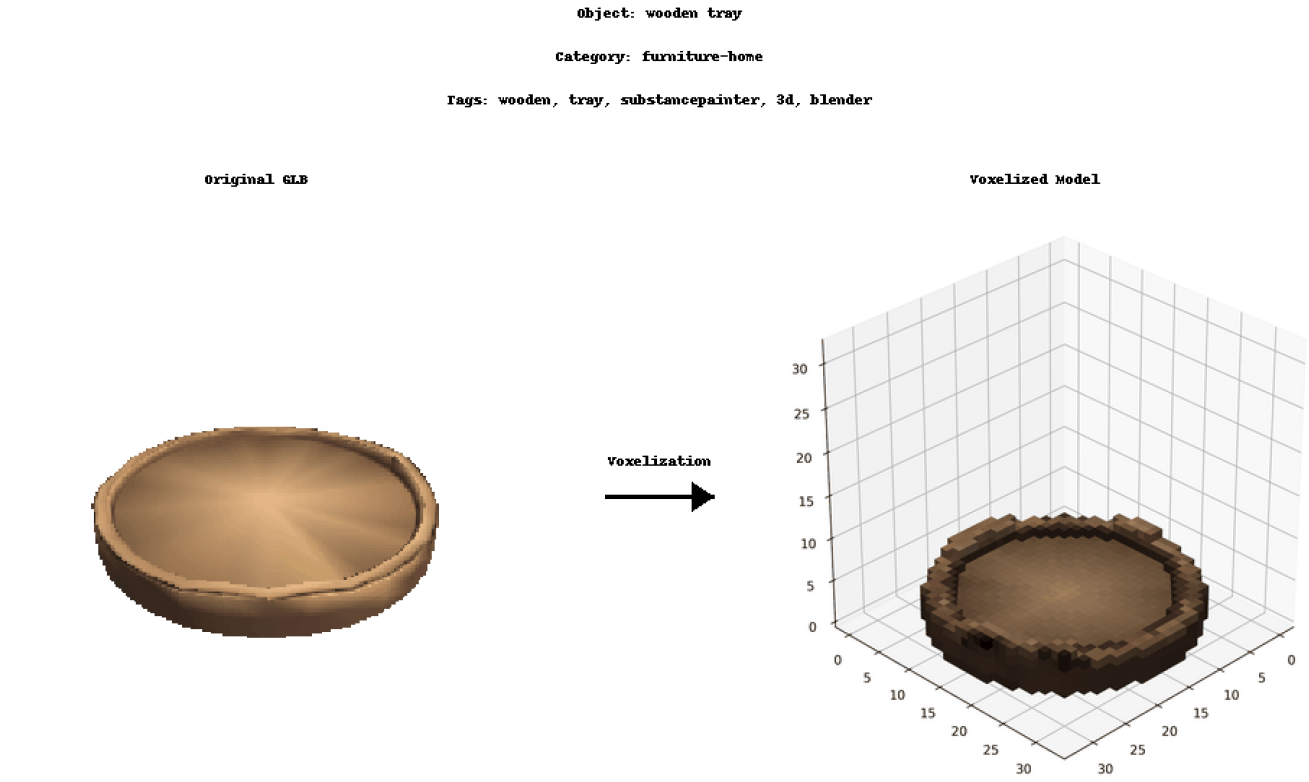

Figure 2. Visualization of our voxelization pipeline. Left: Original GLB model of a wooden tray. Right: Resulting voxelized representation, preserving both geometry and color information.

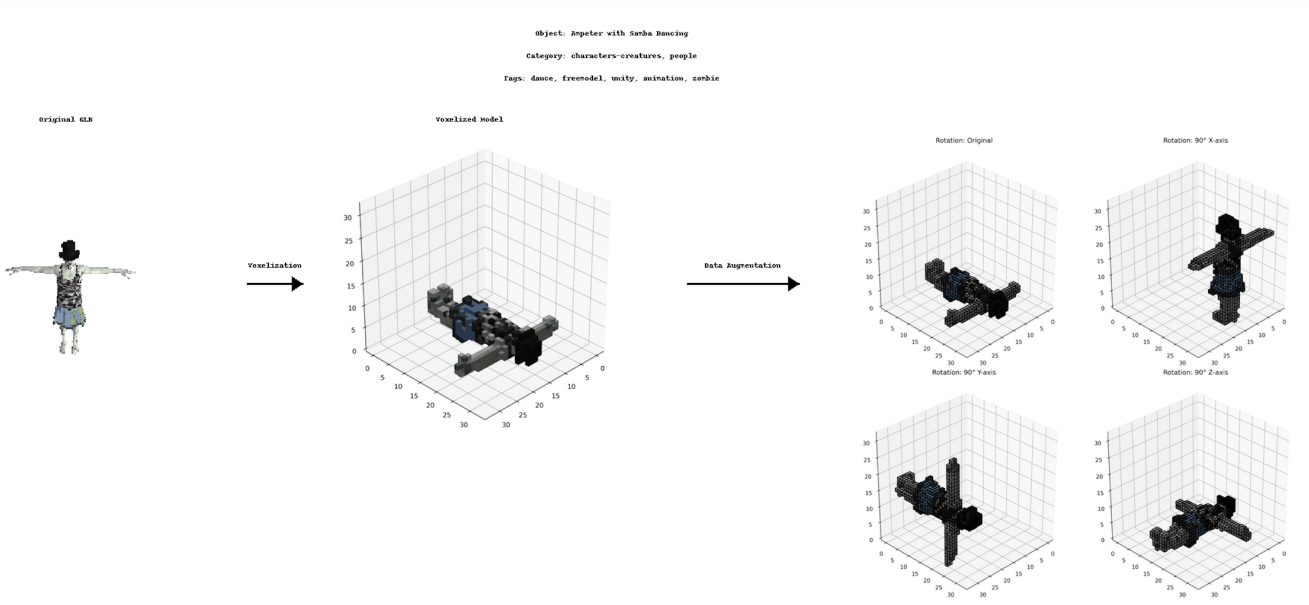

Figure 3. Data augmentation through rotations. From left to right: Original GLB model, initial voxelization, and four orientations of the model (original + three 90-degree rotations around principal axes). Object metadata is preserved across all augmentations.

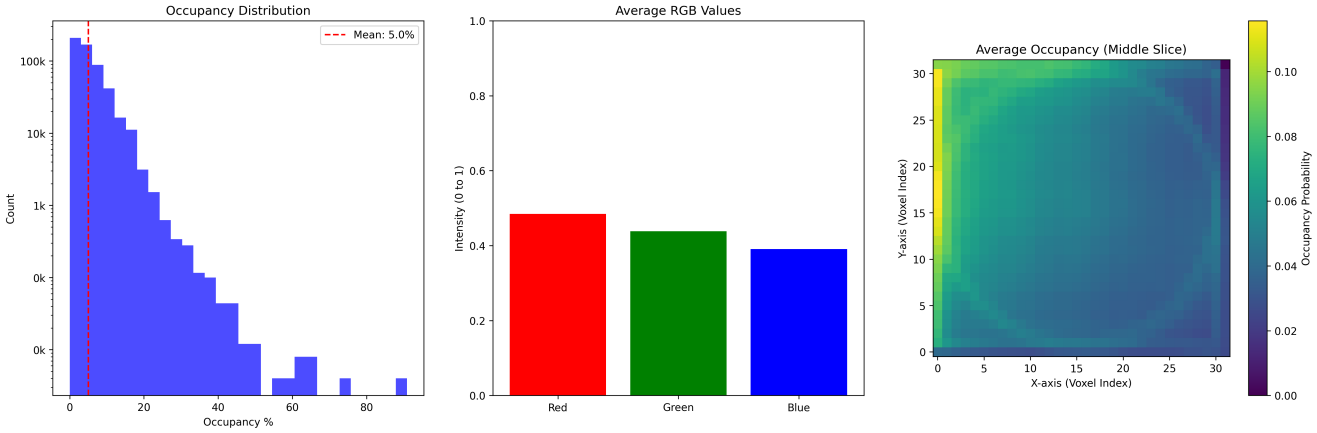

Figure 4. Dataset statistics: (Left) Distribution of voxel occupancy percentages. (Middle) Average RGB values of occupied voxels. (normalized between 0 and 1) (Right) Heatmap of average voxel occupancy for the middle slice along the Z-dimension

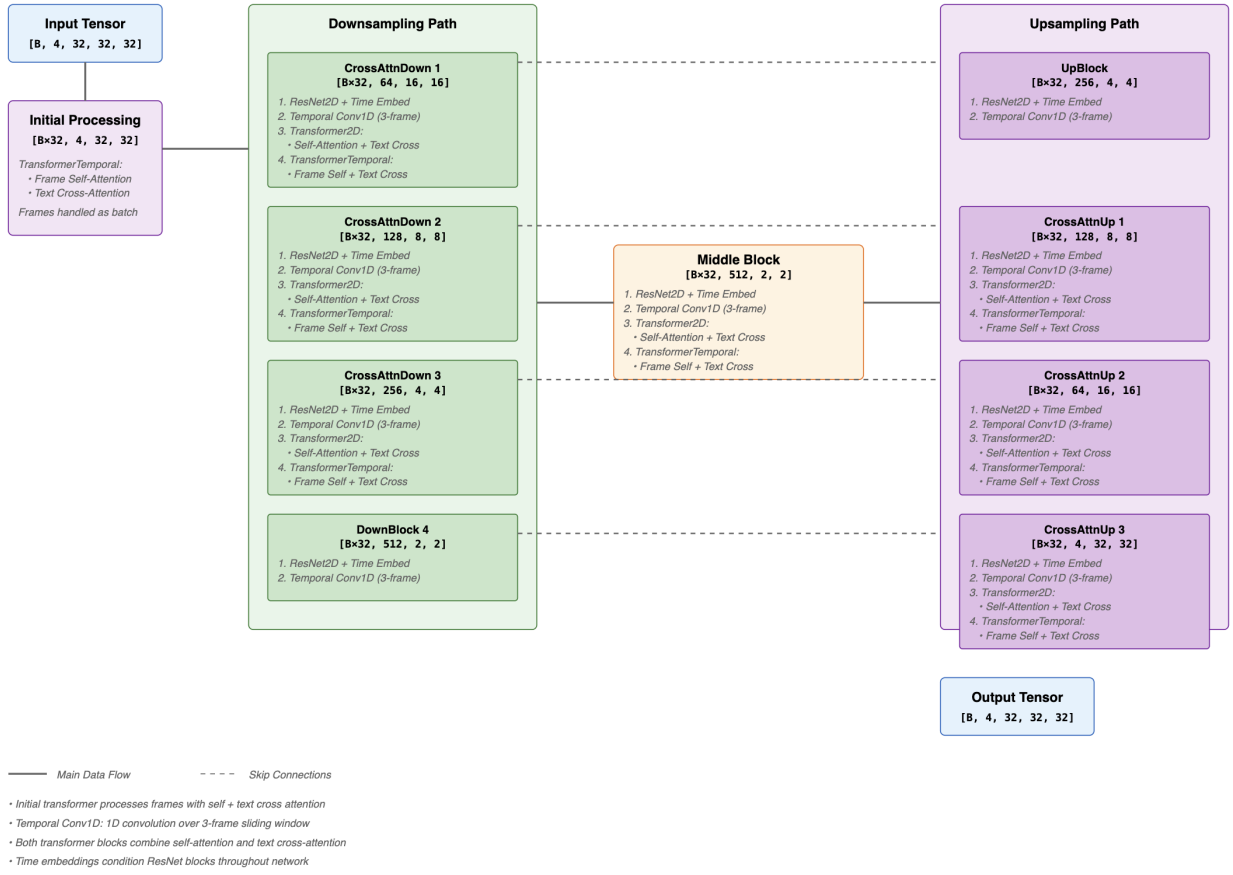

Figure 5. UNet3D architecture overview. Input tensors [B, 4, 32, 32, 32] are transformed to process frames as batch elements [B×32, 4, 32, 32]. Each CrossAttn block combines ResNet2D (with time embedding), temporal convolution (across 3-frame windows), Transformer2D (self-attention + text cross-attention), and TransformerTemporal (self-attention + text cross-attention across frames). Skip connections preserve spatial details through the network.

where $\epsilon_\theta(x_t, t)$ is the predicted noise, x_t is the noisy input at timestep t, and α_t is the cumulative product of alphas

from the noise scheduler.

1. Combined RGBA Mode[2]: For joint shape and color prediction [4, 32, 32, 32], we implement:

a) Simple MSE across all RGBA channels:

$$\mathcal{L}_{\text{combined}}^{\text{simple}} = \|\hat{x}_0 - x_0\|_2^2 \tag{4}$$

b) Weighted RGBA loss:

$$\mathcal{L}_{\text{combined}}^{\text{weighted}} = \lambda_\alpha \mathcal{L}_\alpha + \lambda_{\text{rgb}} \mathcal{L}_{\text{rgb}}, \tag{5}$$

where:

$$\mathcal{L}_\alpha = \|\hat{x}_0^\alpha - x_0^\alpha\|_2^2 \text{ (occupancy loss)} \tag{6}$$

$$\mathcal{L}_{\text{rgb}} = \|x_0^\alpha \cdot (\hat{x}_0^{\text{rgb}} - x_0^{\text{rgb}})\|_2^2 \text{ (color loss)} \tag{7}$$

Here, x_0^α is the binary occupancy channel that masks the RGB loss to focus only on occupied voxels, and $\lambda_\alpha = \lambda_{\text{rgb}} = 1.0$ by default.

2. Two-Stage Pipeline:

Stage 1 (Shape): Predicts occupancy using a single channel [1, 32, 32, 32]:

$$\mathcal{L}_{\text{stage1}} = \|\hat{x}_0 - x_0\|_2^2 \tag{8}$$

Stage 2 (Color): Predicts RGB values [3, 32, 32, 32] using fixed Stage 1 occupancy mask:

$$\mathcal{L}_{\text{stage2}} = \|\alpha_{\text{mask}} \cdot (\hat{x}_0^{\text{rgb}} - x_0^{\text{rgb}})\|_2^2 \tag{9}$$

where α_{mask} is the binary occupancy mask from Stage 1. In Stage 2, the model only predicts noise for RGB channels while using the fixed occupancy mask from Stage 1. All quantities above containing a hat (as in $\hat{x}$) means they represent predictions.

4.2.3 Two-Stage Generation

The two stage generation process operates sequentially:

1. Shape Stage:
   - Generate binary occupancy using shape model
   - Threshold at 0.5 to obtain binary mask m
2. Color Stage:
   - Initialize RGB noise only in occupied voxels: $\mathcal{N}(0, \mathbf{I}) \cdot m$
   - Maintain fixed occupancy from shape stage
   - Predict RGB values while preserving original shape

[2]For visualization (and in our dataset), whenever the occupancy channel is zero (i.e., an empty voxel), the RGB color is also set to zero.

4.2.4 EMA Implementation

For weights stability, we maintain an exponential moving average of model weights:

$$\theta_{\text{EMA}}^{(t)} = \beta \theta_{\text{EMA}}^{(t-1)} + (1 - \beta)\theta^{(t)}, \tag{10}$$

Here, $\theta^{(t)}$ represents the weights of the model at training step t, and $\theta_{\text{EMA}}^{(t)}$ is the exponentially averaged version of the model weights up to step t.

4.3. Inference and post-processing

Our inference pipeline incorporates several techniques to improve generation quality and stability:

4.3.1 Classifier-Free Guidance

Following [7], we adopt a *classifier-free guidance* strategy to balance faithfulness to the text prompt with sample diversity. Concretely, we train a *single* diffusion model that can handle both a "conditional" input prompt (e.g. `"banana"`) and an "unconditional" prompt (i.e. an empty string `""`). At inference time, we perform two forward passes of the noise-prediction network:

$$\epsilon_{\text{cond}}(x_t, t) = \epsilon_\theta\big(x_t, t \mid \text{prompt}\big), \tag{11}$$

$$\epsilon_{\text{uncond}}(x_t, t) = \epsilon_\theta\big(x_t, t \mid \texttt{""}\big). \tag{12}$$

We then blend these predictions via a linear interpolation controlled by the *guidance scale* γ:

$$\epsilon_{\text{guided}}(x_t, t) \;=\; \epsilon_{\text{uncond}}(x_t, t) + \gamma \Big(\epsilon_{\text{cond}}(x_t, t) - \epsilon_{\text{uncond}}(x_t, t)\Big) \tag{13}$$

Intuitively, ϵ_{uncond} captures a general "prior" over 3D shapes, while ϵ_{cond} steers the generation toward the text prompt. As we increase γ, we enforce stronger alignment with the prompt, often producing sharper but less diverse samples.

4.3.2 Rotation-Augmented Sampling

To reduce directional bias and encourage more 3D-consistent generations, we introduce *rotation-augmented sampling*. At each reverse-diffusion step t, we create three rotated copies of x_t:

- $Rot_x(x_t)$: a 90° rotation around the x-axis
- $Rot_y(x_t)$: a 90° rotation around the y-axis
- (The original) x_t with no rotation

We feed each copy into the network (using the same prompt if doing classifier-free guidance) to get three noise predictions. After "unrotating" the rotated predictions back into

the original coordinate frame, we average them:

$$\epsilon_{\text{final}}(x_t, t) = \frac{1}{3}\Big(\epsilon_\theta(x_t, t) + Rot_x^{-1}\big(\epsilon_\theta(Rot_x(x_t), t)\big) + Rot_y^{-1}\big(\epsilon_\theta(Rot_y(x_t), t)\big)\Big). \tag{14}$$

This procedure ensures the final noise is consistent under rotations around the primary axes.

4.3.3 Mean Initialization

We support two initialization strategies:

- Normally we commence with random initialization: Start from Gaussian noise $\mathcal{N}(0, \mathbf{I})$
- Mean initialization: Start with a value of 0.5 in all dimensions and add Gaussian noise $\mathcal{N}(0, \mathbf{I})$ to introduce variability. This is done to provide a balanced starting point for the color stage and to incite the presence of voxels in the shape stage.

For the color stage, these initializations are always masked by the shape stage's binary occupancy to maintain the structure intact.

4.4. LegoLization Algorithm

The final stage of our pipeline converts voxelized models into physically ready for construction LEGO® designs through a brick placement algorithm. The process operates on either single-channel occupancy or four-channel RGB + occupancy.

The algorithm employs a predefined set of standard LEGO® bricks sorted by descending volume (2×4×1 to 1×1×1, where all bricks have a height - z axis - of one in our analysis). Processing layer by layer, it attempts to place the largest possible brick at each unprocessed position. Brick placement follows two key criteria:

- Space availability: Two conditions must be satisfied:
  - All voxels within the brick's dimensions must be part of the target shape (checking alpha/occupancy channel)
  - None of these voxels should be already occupied by a previously placed brick
- Color consistency: RGB values are averaged across all voxels within the brick's dimensions to determine the brick's color

The algorithm prioritizes larger bricks while ensuring complete coverage of the input shape, resulting in a constructible approximation of the original model (see figure 6). While not guaranteeing optimal brick usage or structural stability, this approach provides a practical balance between model fidelity and construction feasibility.

Algorithm 1: LegoLization Algorithm

Input : `voxel_data` of shape $(C \times H \times W \times D)$, where $C \in \{1, 4\}$ (occupancy or RGBA).
`brick_library`: list of brick sizes, in descending volume
Output: `bricks`: list of {size, position, color}

Step 1: Preprocessing
`voxels` ← preprocess_voxels(`voxel_data`);
`occupied` ← zeros(`voxels`[3].shape);
`bricks` ← [];

Step 2: For each layer z **in** $[0..D-1]$
for $z \leftarrow 0$ **to** $D-1$ **do**
  for $x \leftarrow 0$ **to** $H-1$ **do**
    for $y \leftarrow 0$ **to** $W-1$ **do**
      if $voxels[3, x, y, z] > 0 \wedge occupied[x, y, z] == 0$ **then** // check alpha & not occupied
        best_size ← None;
        foreach $size \in brick\_library$ **do**
          if *check_space_available*(`occupied`, (x, y, z), `size`) $\wedge$ *region_is_fully_voxels*(`voxels`[3], (x, y, z), `size`) **then** // largest fitting brick
            best_size ← size; **break**;
        if $best\_size \neq None$ **then**
          brick_color ← get_average_color(`voxels`, (x, y, z), `best_size`);
          `bricks`.append({size = `best_size`, pos = (x, y, z), color = `brick_color`});
          mark_region(`occupied`, (x, y, z), `best_size`);

return `bricks`

5. Results & Analysis

We trained three variants of our model to evaluate different approaches for generating brick-compatible 3D models corresponding to the three stage training in section 4.2.2:

1. **Combined RGBA (Simple):** Joint prediction of shape and color using standard MSE loss.
2. **Combined RGBA (Weighted):** Joint prediction with our weighted RGBA loss function.
3. **Two-Stage Pipeline:** Sequential shape then color prediction.

To determine the most effective approach, we initially trained each model variant for 60,000 steps (equivalent to training on approximately 200,000 images with batch size 4) using identical hardware and optimization settings. Based on these preliminary results, we selected the best-performing model for extended training to 129,000 steps.

All models were trained on a single NVIDIA A100 40GB GPU. Table 1 summarizes the key configurations and hyperparameters used across all experiments:

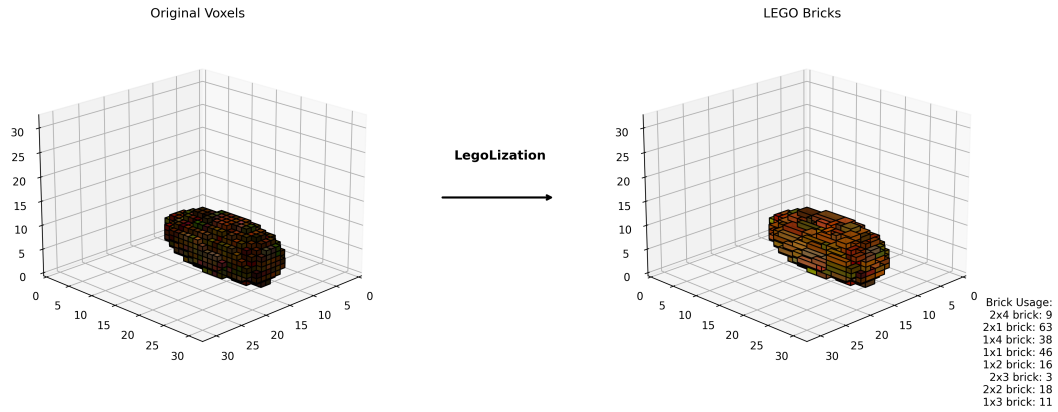

Figure 6. A example of going from model output (voxelized 3d object) to legolized version of the model, prompt was "Riped banana", this model requires 204 bricks: 9 pieces of 2x4x1 brick, 63 pieces of 2x1x1 bricks, 38 pieces of 1x4x1 bricks, 46 pieces of 1x1x1 bricks, 16 pieces of 1x2x1 bricks, 3 pieces of 2x3x1 bricks, 18 pieces of 2x2x1 pieces, 11 pieces of 1x3x1 pieces. Using guidance scale=20 (for shape and color), no mean initialization and no rotation-augmented sampling

Table 1. Training Configuration Details

Parameter	Value	Justification
Hardware	1× A100 40GB	Single GPU training setup
VRAM Usage	38GB	Maximum utilization with safety margin
Batch Size	4	Maximum size fitting in GPU memory
Initial Training	60,000 steps	For model comparison (200k images)
Extended Training	129,000 steps	Full training for best model
Learning Rate	1e-4	Standard for transformer architectures
LR Scheduler	CosineAnnealing	Smooth decay to 1e-6
Optimizer	AdamW	Weight decay (0.01) for regularization
EMA Decay	0.9999	Stability in diffusion training
Gradient Clipping	1.0	Prevent gradient explosions

Model-Specific Configurations:

1. **Combined RGBA (Simple):**
   (a) Input/Output: 4 channels (RGB + Alpha)
   (b) Loss: Standard MSE across all channels
   (c) Initial Training: 60k steps for comparison
2. **Combined RGBA (Weighted):**
   (a) Input/Output: 4 channels (RGB + Alpha)
   (b) Loss: Weighted combination (α = 1.0, RGB = 1.0)
   (c) Initial Training: 60k steps for comparison
3. **Two-Stage Pipeline:**
   (a) Shape Stage:
      i. Input/Output: 1 channel (Occupancy)
      ii. Loss: MSE on occupancy
      iii. Initial Training: 60k steps for comparison
   (b) Color Stage:
      i. Input: 4 channels (RGB + fixed Alpha)
      ii. Output: 3 channels (RGB)
      iii. Loss: MSE on RGB (masked by occupancy)
      iv. Initial Training: 60k steps for comparison

We chose step-based training over epochs due to our large augmented dataset size (515,177 samples for training and 27,115 for testing). For text prompting during training, inspired by OpenAI's DALL-E 3 [1], we employed a balanced prompting strategy:

- 45% detailed prompts (e.g., "wooden tray, furniture-home, wooden, substancepainter")
- 45% simple prompts (e.g., "wooden tray")
- 10% unconditional generation (empty prompt "")

Here is a word cloud visualization of the prompts in our dataset 7

Figure 7. Word cloud visualization of the most frequent terms in our training prompts

This prompting strategy helps the model learn both detailed attribute control and basic shape generation, while the unconditional samples improve the model's base generation capabilities. The initial 60,000 steps provided sufficient data to evaluate model performance while keeping computational requirements manageable. The best performing model was then selected for extended training to 129,000 steps, representing approximately one full epoch over the augmented dataset.

5.1. Model Evaluation and Selection

5.1.1 Quantitative evaluation

To compare the three model variants, we evaluated each model on a selected evaluation dataset, unseen by the model, consisting of 8 different objects: a sawyer fish, a kitchen knife, a red apple, a fighter jet, a traffic cone, a donut, a wooden chair, and a banana. These objects were chosen to represent a range of shapes, colors, and complexities. We assume that each prompt (e.g., "apple fruit, food red, round, shiny, fresh" or "sawyer fish marine life blue, yellow fins, striped") is sufficiently detailed to describe the features of the 3D object in both geometry and color. Each model is evaluated by conditioning only on text and then comparing the predicted 3D model with the corresponding ground-truth shape using three metrics that measures both geometric and color precision:

1. **Shape Metrics:**
   - Intersection over Union (IoU) measures spatial overlap by calculating $\frac{|P \cap T|}{|P \cup T|}$, where P and T are the predicted and target voxel occupancy maps respectively. A value of 1 indicates perfect overlap and 0 indicates no overlap.
   - F1 Score balances precision (correctly predicted occupied voxels / total predicted occupied voxels) and recall (correctly predicted occupied voxels / total true occupied voxels): $2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$
2. **Color Score:** We compute the L2 distance between RGB values, but only for voxels that exist in both prediction and target (the intersection $P \cap T$). This ensures that color comparison is only performed where both shapes agree on occupancy. The metric is normalized to [0,1] where 1 indicates identical colors and 0 indicates maximum difference.

A significant limitation of these metrics is that small shifts or rotations of the predicted object can considerably reduce IoU, F1 and color score, despite the semantics and visual identity being correct. For example, if our model generates a banana that is translated a few voxels away from the expected position or rotated slightly in 3D, IoU and F1 will drop—yet in terms of "banana-ness" and color accuracy, it might be perfectly acceptable. Additionally, our voxel grid is inherently sparse (typically over 95% empty), which leads to numerically small values, due to the high choice of locations, for both shape metrics.

Table 2 presents the comparison results:

Table 2. Model Performance Comparison at 60k Steps Arrows ↑ indicate that higher is better

Model	IoU ↑	F1 ↑	Color Score ↑
Combined RGBA (Simple)	0.0190	0.0368	0.7251
Combined RGBA (Weighted)	0.0416	0.0770	0.6299
Two-Stage Pipeline	**0.0484**	**0.0895**	**0.7875**

Despite these metric limitations, the Two-Stage Pipeline consistently outperforms the other variants across all measures. By separating shape and color prediction, the model can focus on geometric correspondence first (achieving better IoU and F1 scores), and then refine the final appearance without disrupting the already-predicted occupancy. This design prevents color errors from interfering with shape predictions and vice versa.

Based on these results, we selected the Two-Stage Pipeline for extended training to 129,000 steps.

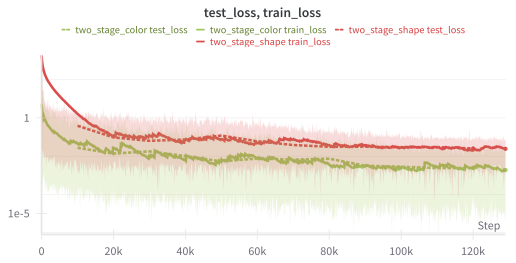

Figure 8. Training and test losses for our two-stage pipeline. Both the shape stage (red lines) and color stage (green lines) show steady convergence with no overfitting. Notably, the color loss converges to a lower value than the shape loss, reflecting the simpler nature of color assignment to a fixed occupancy.

Training Dynamics Figure 8 illustrates the losses for both shape (red lines) and color (green lines) across training steps, including training (solid lines) and test (dashed lines). Despite the more complex spatial uncertainty for shape prediction (e.g., an object can be valid in different positions or orientations), the model's shape loss consistently improves and plateaus around 0.024. In contrast, the color stage converges to an even lower loss (around 0.002), likely due to only assigning RGB values once the shape is already fixed.

5.1.2 Qualitative Evaluation

To complement our quantitative results, we now present a side-by-side visual comparison of the three model variants—Combined RGBA (Simple), Combined RGBA (Weighted), and the Two-Stage Pipeline— at 60k steps and across six prompts: *"A red cube"*, *"A banana"*, *"A blue sphere"*, *"A fish"*, *"A green diamond"*, and *"A tree"*. Figure 9 illustrates each model's output in both 2D slice form (at depth = 16) and reconstructed 3D voxel.

From these visualizations, the Two-Stage Pipeline produces the most coherent shapes and faithful color assignments. For instance, in the "red cube" example, the Two-Stage model cleanly captures a cubic form and a uniform red hue, whereas the Combined RGBA (Simple) often shows noisier edges and color leakage. Similarly, for prompts featuring more organic objects such as a "banana" or a "fish," the Two-Stage model produces smoother silhouettes (e.g., capturing the gentle curvature of the banana) and applies color more accurately (e.g., including a subtle greenish shade near the banana stem).

Interestingly, Combined RGBA (Simple) shows stronger results for the "blue sphere" and the "fish," while Combined RGBA (Weighted) displays correct color for the "red cube" and "banana" but struggles with other shapes. Both Combined RGBA models are subject to slight geometry distortions and inconsistent color patches, likely due to the complexity of predicting occupancy and color simultaneously in a single pass.

Overall, the visual fidelity of the Two-Stage Pipeline confirms the quantitative findings (Table 2), underscoring the advantage of decoupling shape and color prediction to achieve more robust 3D generation.

5.1.3 Diffusion Steps Visualization

In addition to final outputs, we show the intermediate diffusion steps for our Two-Stage Pipeline (shape + color). Figure 10 illustrates how the shape and color evolve at 100-step intervals, highlighting the transition from noise to a coherent 3D structure and eventual color assignment.

5.2. Ablation Studies

We conduct a series of experiments to assess how particular design and inference choices affect the Two-Stage Pipeline's performance. We adopt the same evaluation protocol as in Table 2, measuring Intersection over Union (IoU), F1, and Color Score. Our **baseline** is the fully trained Two-Stage model at 129k steps [3] (IoU: 0.0698, F1: 0.1163, Color Score: 0.7685). This baseline model predictions did not use rotation-aug sampling, mean initialization, DDIM or EMA and used a guidance scale of 20. In each ablation experiment, we keep all parameters identical to the baseline except for the one being tested. Table 3 summarizes the results for each ablation, with **bold** indicating the best score in each column.

[3]This is the model that generated all the figures outside figure 9

Table 3. Ablation study results compared to the baseline Two-Stage model. Arrows (↑) indicate that higher is better.

Ablation	IoU ↑	F1 ↑	Color ↑
Baseline (129k steps)	0.0698	0.1163	0.7685
Finetuned 135k steps	0.0703	0.1170	0.7719
Guidance scale = 10	0.0583	0.1012	0.8280
Guidance scale = 30	**0.0801**	**0.1353**	0.6862
Guidance scale = 50	0.0577	0.0979	0.5300
With rotation-aug sampling	0.0589	0.0970	0.7930
With Mean initialization	0.0661	0.1066	0.7746
DDIM (200 steps)	0.0127	0.0246	0.3347
DDIM (400 steps)	0.0059	0.0116	0.3005
With EMA	0.0394	0.0701	**0.8358**

Fine-Tuning vs Not Fine-Tuning To further improve shape generation, which appears most susceptible to errors, we fine-tune our baseline model (trained to 129k steps) for an additional 6k steps (135k total). This fine-tuning uses a smaller dataset of 11,464 voxelized 3D models (split 90/10 for training and testing) drawn from Objaverse, including categories such as chair, seashell, antenna, shield, snowman, chandelier, gravestone, cone, control, sword, armor, doughnut, banana, ring, fireplug, mushroom, fighter_jet, figurine, and more. We pair these simpler prompts (e.g. "banana," "chair") with a reduced learning rate of 1e-5 and use no data augmentation. Notably, this dataset, which we release publicly [13], overlaps with three of our core evaluation prompts, yielding slight metric improvements. Figure 11 shows an example of the fine-tuned model's output for the prompt "a sword". Thus, targeted fine-tuning can refine structural fidelity without sacrificing color.

Classifier-Free Guidance We tested different guidance scales (10, 30, and 50) relative to the baseline of 20. At a scale of 10, the model focuses less on following the prompt and more on creating general 3D shapes, resulting in smoother objects with better colors (Color Score of 0.8280). However, this can lead to objects breaking into separate pieces, as seen with the knife's blade and handle splitting in Figure 12. On the other hand, higher guidance scales (30, 50) make objects look more sharp and conform to the prompt, improving IoU and F1 in some cases, but can reduce color diversity. This can constrain the shape, as seen by the straight blade at scale 50. Thus, there is a trade-off between shape accuracy and color richness when adjusting the guidance scale (see Table 3).

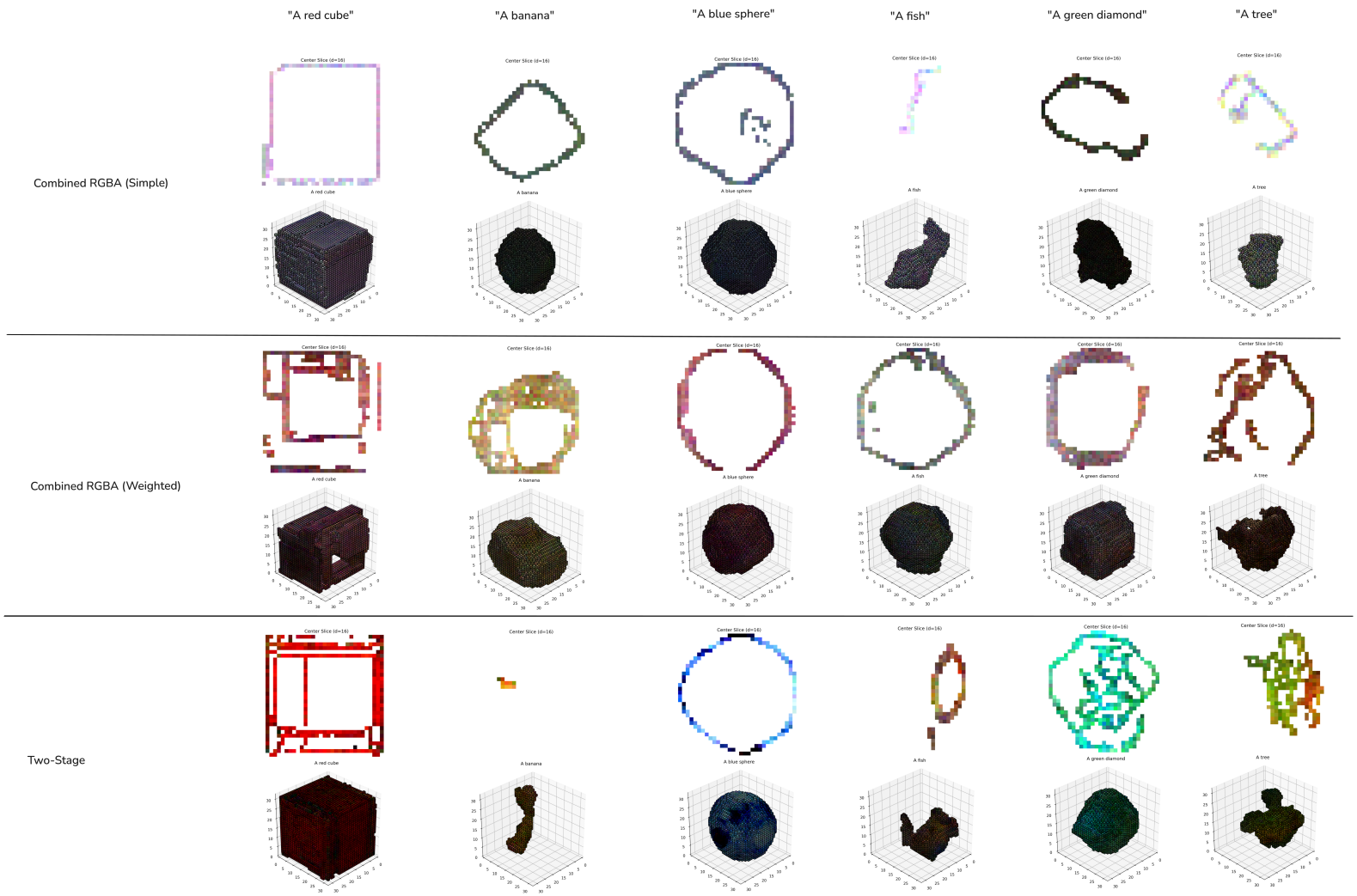

Figure 9. Qualitative comparison of three architectures for six text prompts at 60k steps, guidance scale=20 (for shape and color in two_stage pipeline and for the combined cases), no mean initialization and no rotation-augmented sampling. Each row shows top-down slices of voxel occupancies and their 3D renderings. Results were not cherry picked thus outputted one after the other with no retrying

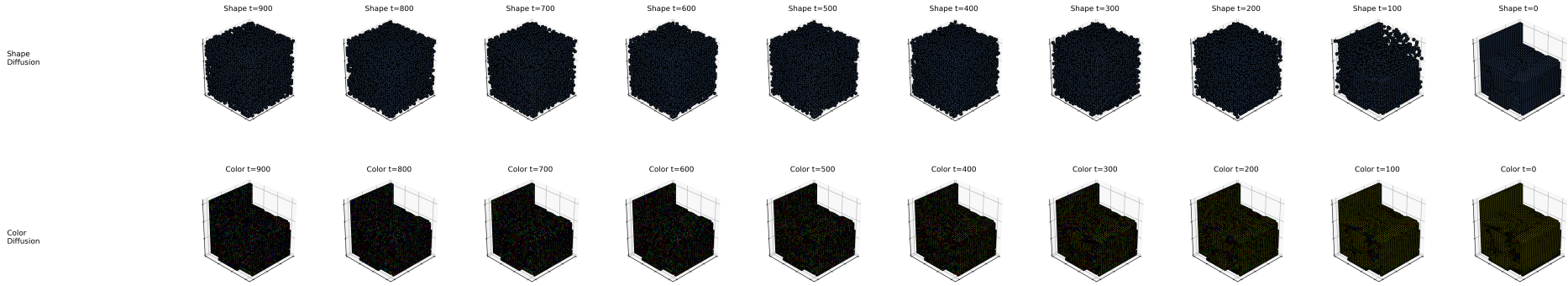

Figure 10. Visualization of our Two-Stage diffusion process at intervals of 50 steps, showing the actual intermediate states during denoising. The top row shows the shape stage gradually refining occupancy from pure noise to the final shape, while the bottom row shows color diffusion on the fixed shape. Using ”Yellow Pyramid” as prompt, guidance scale=20 (for shape and color), no mean initialization and no rotation-augmented sampling. We can see the structure forming and the yellow color towards the end steps. Results are not cherry picked. Best viewed in color with zoom.

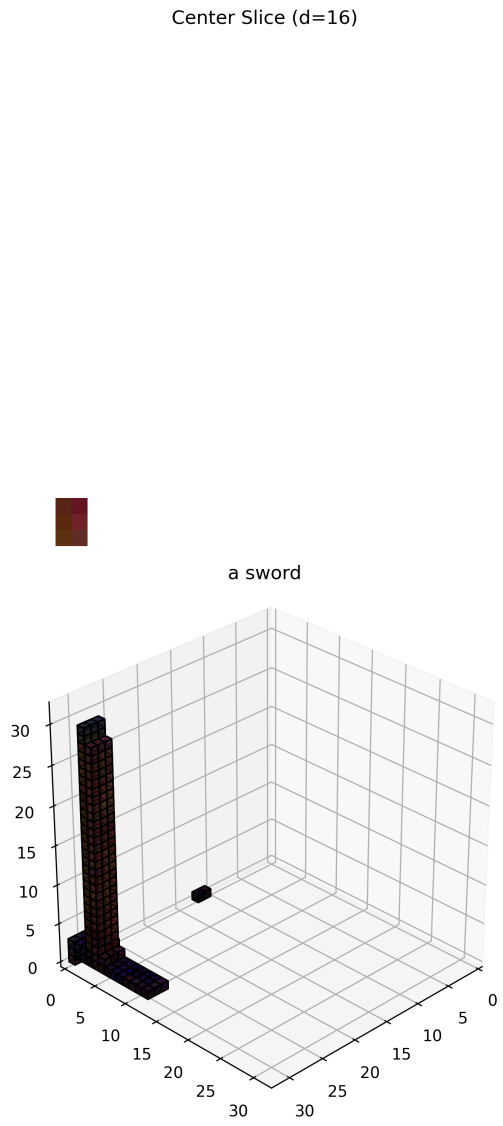

Figure 11. Visualization of the Two-Stage pipeline with fine-tuned shape mode, output with prompt *"a sword"*. Using guidance scale=20, no mean initialization and no rotation-augmented sampling. We can see the handle and sharp part of the sword with a uniform red color. Result not cherry picked.

Rotation-Augmented Sampling Enabling rotation-aug sampling slightly lowers IoU and F1 relative to the baseline, yet it boosts the Color Score to 0.7930. Qualitatively, rotation augmentation often yields more symmetrical shapes and more accurate colors, as illustrated by our *banana* example in Figure 13, where the object appears more spheric and shows a more uniform yellow color. Thus, demonstrating that rotation-augmented sampling enforces more symmetrical shapes.

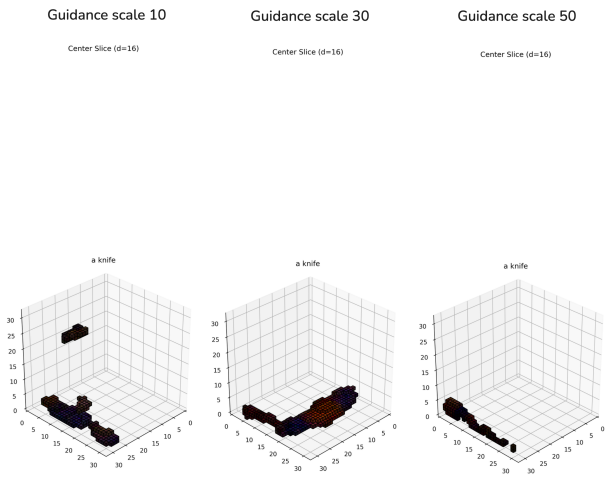

Figure 12. Effect of varying guidance scales (10, 30, 50) on the prompt *"a knife"*. Scale 10 gives a disconnected shape with good color, scale 30 balances shape coherence and color details, and scale 50 forces an overly linear shape with little color variation.

Mean Initialization We replace the default random initialization with a constant 0.5 "mean field" added to the initial noise. As shown in Figure 14, the resulting "banana" example show slightly better yellow shade compared to random initialization. Quantitatively, IoU and F1 dip slightly, while the Color Score increases moderately to 0.7746, indicating a small but measurable improvement in color fidelity. Overall, mean initialization does not drastically change final outputs, but can help with stronger color presence for certain prompts.

DDPM vs. DDIM We compared our standard DDPM method (using 1000 steps) with DDIM using fewer steps - 200 and 400 ($\sigma_t = 0$ for all t). While DDIM is faster since it uses fewer steps and is deterministic, it produces lower quality results across all our metrics (IoU, F1, and color scores). This suggests that DDPM's randomness and longer sampling process helps create better shapes and colors. While using 200 DDIM steps gives slightly better numbers than 400 steps, they look similar visually. If highest quality is sought, DDPM seems to be the better choice, even though it is slower (See figure 15).

EMA vs. No EMA. Using EMA during training *improves* how well the model assigns colors - the Color Score goes up from 0.7685 to 0.8358. However, there's a trade-off: the model becomes slightly worse at getting the shape right, as shown by lower IoU and F1 scores. Looking at a generated basketball as an example in Figure 16, the model gets the brown coloring correct, but leaves some small holes in the ball's shape. Whether this trade-off is acceptable depends

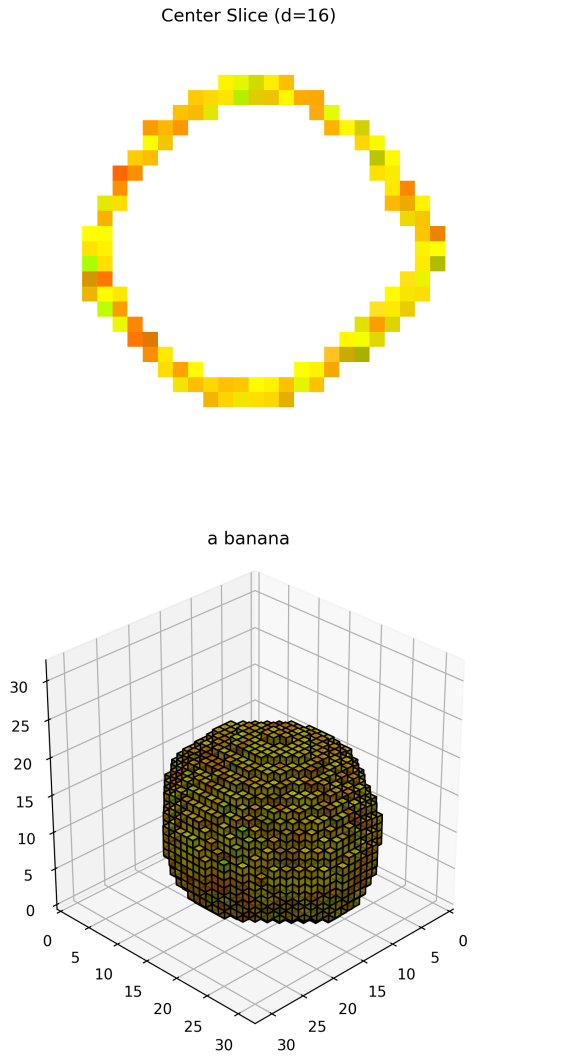

Figure 13. Effect of rotation-aug sampling on the prompt *"a banana"*. Compared to the baseline, the shape is more spherical, and colors appear more consistent. Guidance scale = 20, no mean initialization, with rotation-augmented sampling. Result not cherry picked

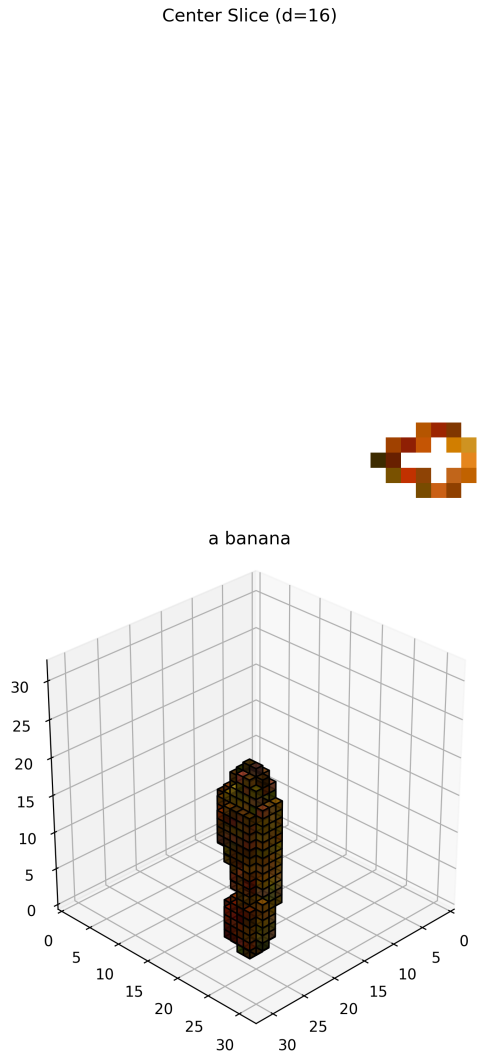

Figure 14. Effect of mean initialization on the prompt *"a banana"*. Guidance scale = 20, no EMA, no rotation-aug sampling, with min initialization. The banana's shade is more pronounced, though the shape remains somewhat similar more round but with no curvature. Result not cherry picked.

on whether accurate colors are more important than perfect shapes for the use case.

Overall, these experiments show that two factors had the biggest impact on model performance: fine-tuning the model and using moderate guidance scales around 20 or 30. EMA improved color consistency but slightly impacted shape quality likely due to more stable training. The other techniques we tested: rotation augmentation sampling, mean initialization, and DDIM- had smaller effects. Thus, the choice would come down to trade-offs between quality, color accuracy and inference speed.

6. Conclusion and Future Work

In this work, we presented 3D-BlockGen, a two-stage diffusion model for generating brick-compatible 3D objects from text descriptions. We showed that separating shape and color, leads to a more robust generation compared to combined (RGB + occupancy) prediction techniques. Our ablation studies, revealed that guidance scale and fine-tuning choices impact generation quality the most, while inference techniques like rotation-augmented sampling and mean initialization offer smaller but useful improvements depending on use case.

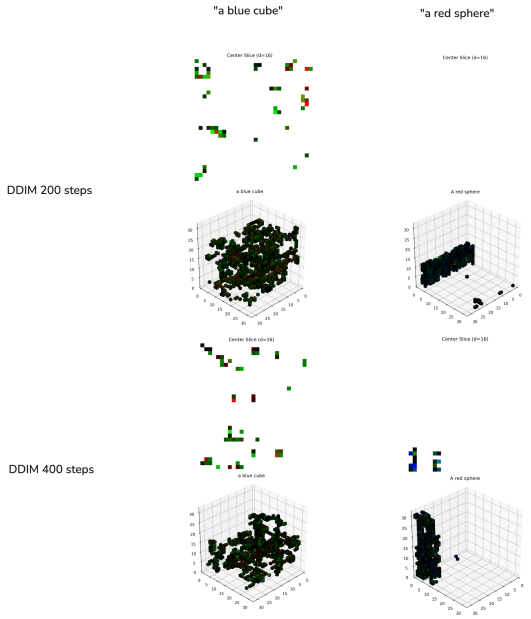

Figure 15. Visualization of the Two-Stage pipeline with DDIM 200 steps and 400 steps (rows) output with prompt "a blue cube" and "a red sphere" (columns) demonstrating the need for DDPM in generation of high-quality output. Using guidance scale=20, no mean initialization and no rotation-augmented sampling

Limitations Our current approach faces some limitations. First, hard prompts involving multiple objects or dynamic scenes (e.g., "Grey dog jumping on his owner") fail to produce interesting results. This limitation is due to the training data, which consists in majority of single objects. Second, our voxel-based representation, while convenient, imposes a rigid grid structure that doesn't reflect the true flexibility of physical brick construction - most LEGO® models aren't build with perfect 90-degree alignments. Third, our sparse voxel representation (typically 95% empty) makes training challenging due to the curse of dimensionality in 3D space.

Future Directions We would like to explore using Zero123-XL's [9] image-to-3D model to first generate high-quality 3D objects that could become input to into our LegoLization algorithm. Moreover, we could resort to Stable Diffusion [20] by working in its latent space, thus reducing memory compared to our current full-resolution voxel method.

A different direction would be to treat bricks similar to points in a point cloud diffusion model [10]. Instead of predicting voxel occupancy, the model would operate directly on brick parameters with shape $[batch\_size \times max\_bricks \times channels]$, where channels encode position, orientation, and brick type. This could handle non-grid brick placements and match natural construction.

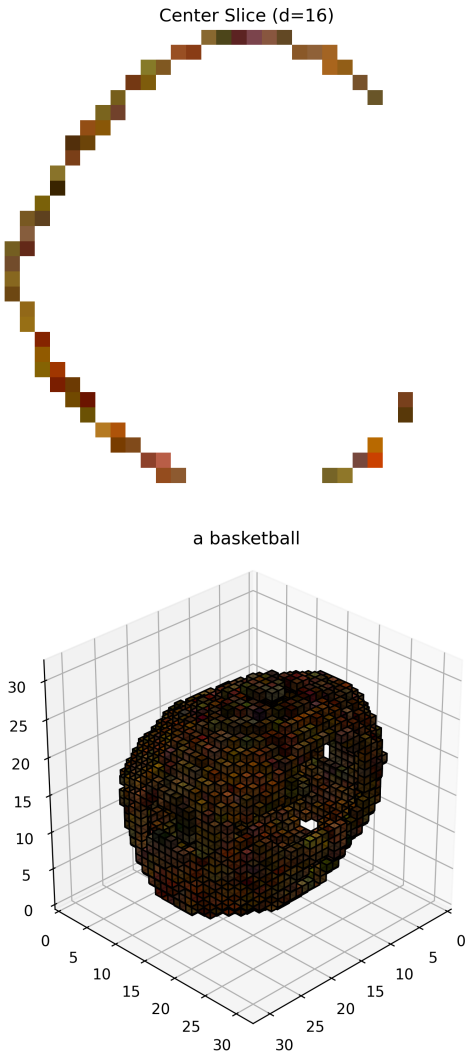

Figure 16. Output from the Two-Stage pipeline *with* EMA for the prompt *"a basketball"*. Guidance scale = 20, no rotation-aug sampling, no mean initialization. The colors appear more cohesive, though the shape contains a few holes likely due to smoother occupancy updates. Result not cherry picked.

We could also leverage LDRAW [2], an open standard for LEGO CAD programs that allow the user to create virtual LEGO models and scenes. By treating LDRAW as a language for Large Language Models, we could generate building instructions. However, we need to be careful when tokenizing due to the format relying heavily on numbers to encode bricks and positions.

To better evaluate our results, we propose adapting Fréchet Inception Distance (FID) [5] for our evaluation of 3D voxels. Unlike our current metrics, FID could better capture structural similarities and do not suffer from rotations or shifts.

Finally, we plan to create a higher quality dataset with

detailed, specific prompts for fine-tuning our models such as "a bright yellow banana with a slight green stem". We are also interested in exploring hierarchical approaches like XCube's sparse voxel representation, which could enable generation at much higher resolutions (up to 1024^3) while maintaining memory efficiency.

These improvements, and better handling of sparse voxels, could lead to more efficient systems for generating constructible 3D models.

References

[1] James Betker, Gabriel Goh, Li Jing, Tim Brooks, Jianfeng Wang, Linjie Li, Long Ouyang, Juntao Zhuang, Joyce Lee, Yufei Guo, Wesam Manassra, Prafulla Dhariwal, Casey Chu, Yunxin Jiao, and Aditya Ramesh. Improving image generation with better captions. https://cdn.openai.com/papers/dall-e-3.pdf, 2023. OpenAI Preprint. 9

[2] LDraw.org Community. LDraw: An open standard for LEGO CAD programs. https://www.ldraw.org/, 2025. Accessed: January 10, 2025. 16

[3] 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. *arXiv preprint arXiv:2212.08051*, 2022. 1

[4] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014. 1

[5] 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. In *Advances in Neural Information Processing Systems (NIPS)*, pages 6626–6637, 2017. 16

[6] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 2, 3

[7] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance, 2022. 7

[8] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022. 1

[9] Ruoshi Liu, Rundi Wu, Basile Van Hoorick, Pavel Tokmakov, Sergey Zakharov, and Carl Vondrick. Zero-1-to-3: Zero-shot one image to 3d object, 2023. 16

[10] Shitong Luo and Wei Hu. Diffusion probabilistic models for 3d point cloud generation, 2021. 16

[11] Peter A. Massih. 3d-blockgen: Codebase for text-to-3d voxel generation, 2024. https://github.com/PeterAMassih/3D-BlockGen. 1

[12] Peter A. Massih. Blockgen-3d: A large-scale dataset for text-to-3d voxel generation, 2024. https://huggingface.co/datasets/PeterAM4/blockgen-3d. 1

[13] Peter A. Massih. Blockgen-3d: A large-scale dataset for text-to-3d voxel generation- finetuned version, 2024. https://huggingface.co/datasets/PeterAM4/blockgen-3d-finetune. 11

[14] Mikedh. Trimesh [computer software], 2019. Retrieved from https://github.com/mikedh/trimesh. 3

[15] Shentong Mo, Enze Xie, Ruihang Chu, Lanqing Hong, Matthias Nießner, and Zhenguo Li. Dit-3d: Exploring plain diffusion transformers for 3d shape generation. *arXiv preprint arXiv: 2307.01831*, 2023. 2

[16] Norman Müller, Yawar Siddiqui, Lorenzo Porzi, Samuel Rota Bulo, Peter Kontschieder, and Matthias Nießner. Diffrf: Rendering-guided 3d radiance field diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4328–4338, 2023. 2

[17] Lucas Nunes, Louis Wiesmann, Rodrigo Marcuzzi, Xieyuanli Chen, Jens Behley, and Cyrill Stachniss. Temporal consistent 3d lidar representation learning for semantic perception in autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. 2

[18] Keunhong Park, Konstantinos Rematas, Ali Farhadi, and Steven M. Seitz. Photoshape: Photorealistic materials for large-scale shape collections. *ACM Trans. Graph.*, 37(6), Nov. 2018. 2

[19] Xuanchi Ren, Jiahui Huang, Xiaohui Zeng, Ken Museth, Sanja Fidler, and Francis Williams. Xcube: Large-scale 3d generative modeling using sparse voxel hierarchies, 2024. 1

[20] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022. 16

[21] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015. 4

[22] Jascha Sohl-Dickstein, Eric A Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep Unsuper-

vised Learning using Nonequilibrium Thermodynamics. *arXiv preprint arXiv:1503.03585*, 2015. 2

[23] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models, 2020. 3

[24] Xiaohui Zeng, Arash Vahdat, Francis Williams, Zan Gojcic, Or Litany, Sanja Fidler, and Karsten Kreis. Lion: Latent point diffusion models for 3d shape generation, 2022. 2