# SuperVoxelGPT: Adaptive and Ordered 3D Tokenization for Autoregressive Shape Generation

Yuan Li[1,2], Congyi Zhang[1†], Xifeng Gao[2†], and Xiaohu Guo[1†]

[1] University of Texas at Dallas, USA
[2] LightSpeed Studios, Tencent America, USA
{Li.Yuan, congyi.zhang, xguo}@utdallas.edu,
johnyuanli@global.tencent.com, gxf.xisha@gmail.com
[†] Corresponding authors.

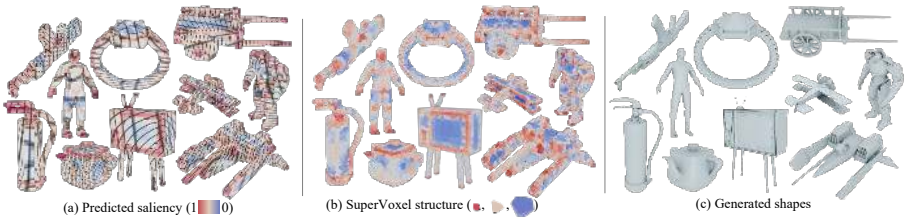



**Fig. 1:** We propose a new two-stage MLLM framework for high-resolution 3D generation. The model first predicts a saliency map (a), which is then transformed into our SuperVoxel structure via a customized 3D CVT (b), enabling the extraction of compact and ordered tokens. Finally, our autoregressive model, SuperVoxelGPT, generates high-resolution 3D geometry based on this representation (c).

**Abstract.** Autoregressive multimodal large language models (MLLMs) enable 3D generation but struggle to scale to high-resolution shapes due to inadequate 3D tokenizations. Compact set-based representations discard deterministic spatial ordering, leading to ambiguous sequence prediction, while uniform or octree-based voxel grids preserve ordering at the cost of severe redundancy and excessively long sequences. This structural trade-off limits stable and efficient autoregressive 3D generation. We present **SuperVoxelGPT**, a representation-first framework that resolves this tension through *adaptive and deterministically ordered supervoxel tokenization*. Given a prompt, we first predict a coarse geometric saliency distribution and construct a shape-adaptive supervoxel partition using saliency-guided centroidal Voronoi tessellation, allocating fine-grained cells to complex regions and larger cells to smooth regions. Conditioned on this prompt and ordered supervoxel layout, we introduce a SuperVoxelVAE and fine-tune a pretrained MLLM to autoregressively generate supervoxel tokens. Experiments using Trellis-500K data show that SuperVoxelGPT reduces token sequence length to **12.8%** of uniform voxel tokenization while achieving state-of-the-art generation quality and an average **10**× speedup over prior methods.

**Keywords:** Super Voxel · Multi-modality · Autoregressive Models · GPT · 3D Generation

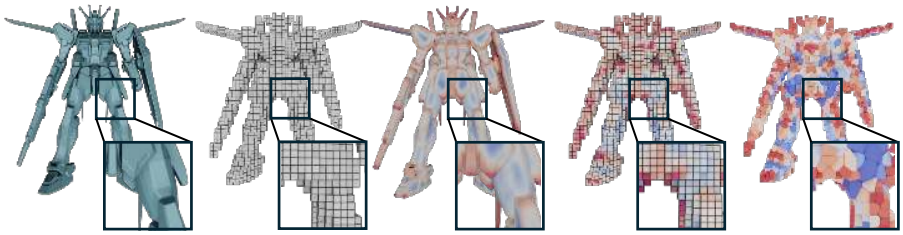



**Fig. 2:** (a) Original mesh, (b) Uniform voxel structure, (c) Mesh saliency, (d) Volume saliency (e) Supervoxel structure

## 1 Introduction

In recent years, Multimodal Large Language Models (MLLMs) [45] have achieved remarkable success across a wide range of generative tasks [1]. Built upon pre-trained large language models and fine-tuned on multimodal data, modern MLLMs can jointly reason over text and images, advancing text-to-video [37], image-to-video [23], and text-to-action generation [47], exhibiting both state-of-the-art generation quality and speed. In contrast to diffusion-based methods that typically require many iterative denoising steps at training and testing time, autoregressive MLLMs, can generate outputs in several left-to-right passes, often enabling faster inference while maintaining competitive quality [30, 35].

Despite these advances, extending MLLMs to high-resolution 3D generation remains challenging. A central reason is that existing 3D tokenizations are poorly matched to autoregressive modeling, typically falling into two extremes, either overly compact [46] or overly redundant [40, 43], which makes it difficult to achieve both stable sequence modeling and efficient high-resolution generation.

On one end, *compact set-based* tokenizations [4, 17, 46] maximize information density per token by discarding explicit spatial ordering. To achieve such high compactness, these methods typically encode the entire geometric shape into a holistic set of latent tokens simultaneously, rather than assigning each token to a specific, localized spatial region. Consequently, the resulting tokens form an unordered collection that lacks a deterministic sequence. This structure is poorly suited to autoregressive modeling, as the absence of a natural order makes sequence prediction ambiguous and weakens the ability of MLLMs to learn stable long-range dependencies. Furthermore, applying quantization on such highly compressed representations inevitably incurs substantial shape information loss, degrading generation quality.

At the other end, *voxel-based* tokenizations [22, 40, 43] represent the opposite extreme. By discretizing space into a uniform 3D grid and assigning tokens densely to capture local geometry (Fig 2, b), they can preserve high-resolution details with a natural traversal order. However, real shapes exhibit highly non-uniform geometric complexity (Fig 2, c), and uniform grids inevitably waste a large fraction of tokens on smooth or planar regions. This leads to prohibitively long sequences, which undermines inference efficiency and makes autoregressive training harder to scale, less stable, and more prone to degraded generaliza-

tion [13, 24]. Recently, several methods [15, 42] have recognized that such long sequences harm both efficiency and training stability, and mitigate this issue by simplifying shapes to a thin layer of surface-intersecting voxels or introducing an additional super-resolution stage to generate the final detailed geometry. However, such simplifications are still not thorough enough: these methods continue to rely on fixed-sized volume tokens to represent shapes, and thus still suffer from inefficiency and instability of redundant token allocation.

To preserve both the efficiency and generation quality of MLLM-based 3D generation, we argue that a new 3D representation explicitly designed for autoregressive MLLM-based 3D generation is required. And such a representation should satisfy two properties: (1) a *deterministic sequential structure* that consistently aligns token order with the shape [39], and (2) *adaptive capacity allocation* that concentrates tokens where geometry is complex while avoiding waste where geometry is simple [19, 33]. In this way, the tailored representation can remain compact and sequential, making it suitable for autoregressive prediction.

Motivated by these observations, we propose a new representation based on *SuperVoxels*. SuperVoxels partition space into variable-sized cells (Fig 2, e), naturally enabling adaptive token allocation: detailed regions can be covered by many small cells, while smooth regions can be represented by fewer large ones [10, 20]. Moreover, since each supervoxel is associated with an explicit spatial location, we can impose a deterministic ordering over supervoxels, yielding an autoregressive-friendly 3D token sequence that is both compact and spatially grounded. Compared with unordered set-based tokenizations, this provides a well-defined generation order; compared with uniform or octree-based voxel grids, it substantially reduces redundancy while preserving high-frequency geometric details.

Building on this representation, we introduce **SuperVoxelGPT**, a two-stage framework for multimodal 3D generation that decouples *where to allocate tokens* from *what geometry to generate*.

Our main contributions are summarized as follows:

- **SuperVoxelGPT Framework.** We propose a two-stage MLLM framework for high-resolution 3D generation: (i) a prompt-conditioned adaptive supervoxel partition to allocate tokens by geometric complexity, and (ii) efficient autoregressive generation of supervoxel tokens with Jacobi decoding [30].
- **SuperVoxelVAE Representation.** We introduce a supervoxel-based 3D tokenizer that produces a compact, deterministically ordered token sequence aligned with 3D space, making it well suited to stable and efficient autoregressive modeling. Its plug-and-play design allows it to be integrated into any existing sparse voxel VAE to compress token sequences without modifying backbone architectures.
- **Saliency-driven 3D CVT for SuperVoxels.** We develop a customized 3D CVT variant to generate well-shaped, variable-sized supervoxels for shape-adaptive tokenization. Experiments show that our approach reduces sequence length to 12.8% of uniform voxel tokenization, achieves the state-of-the-art performance, and delivers a 10× average speedup.

## 2 Related Works

*3D Representation and Tokenization.* In recent learning-based 3D generation, popular 3D tokenization schemes broadly fall into two categories: set-based tokenizations and voxel-based tokenizations.

Set-based methods encode shapes as a compact set of latent tokens, treating geometry as an unordered collection of descriptors. For instance, Shape-2-VecSet [46] embeds sampled surface points and normals into a vector set using a Transformer and supervises decoding with surrounding signed distance field (SDF) values. Subsequent works such as Dora [4] and Hunyuan3D 2.1 [17] improve sampling strategies by emphasizing salient regions to better preserve fine details. Recently, EfficientTokenizer [8] further improves the efficiency of vector-set-based tokenizations by using an octree-based adaptive tokenization. While vector-set tokenizations are compact and avoid token redundancy, they commonly exhibit two limitations. First, they are commonly implemented with full self-attention in prior works, which hampers scaling token count for high-resolution geometry. Second, by modeling tokens as an unordered set, they discard deterministic ordering, which introduces permutation ambiguity and makes them less suitable for autoregressive sequence modeling.

Voxel-based tokenizations lie at the opposite end. These methods preserve explicit spatial structure by discretizing space into voxel grids and assigning features to occupied cells. XCube [29] replaces the global VecSet in 3DShape2VecSet with voxel-aligned SDF and normal features, improving detail preservation. TRELLIS [43] and Direct3D-S2 [40] adopt two-stage strategies to avoid generating empty voxels, focusing computation on shape-intersecting regions. TripoSF [12], Sparc3D [22], and TRELLIS2 [42] further scale this paradigm to higher resolutions. However, treating each grid cell as an individual token ignores the highly non-uniform distribution of geometric complexity, wasting many tokens on smooth regions and incurring excessive computation.

In contrast, our SuperVoxel-based representation achieves adaptive compactness while preserving a deterministic spatially grounded ordering, making it better suited to autoregressive MLLM-based generation.

*3D MLLMs for 3D Generation.* Recent works have explored MLLM-based 3D generation by converting 3D geometry into discrete tokens and autoregressively predicting them. Existing methods fall into two paradigms.

*Token-by-token* methods, such as MeshGPT [31], MeshAnythingV2 [5], PivotMesh [32], EdgeRunner [34], LLaMA-Mesh [38], and BrickGPT [27], iteratively predict the next token conditioned on all preceding tokens. However, 3D sequences often reach tens of thousands of tokens, where strictly sequential decoding erases the efficiency advantage over diffusion methods [8]. Moreover, the variable and unknown sequence length prevents the application of parallel decoding strategies such as Jacobi decoding [30].

*Sequence-to-sequence* methods recognize the inefficiency of single-token autoregression and instead adopt sequence-to-sequence autoregression to address

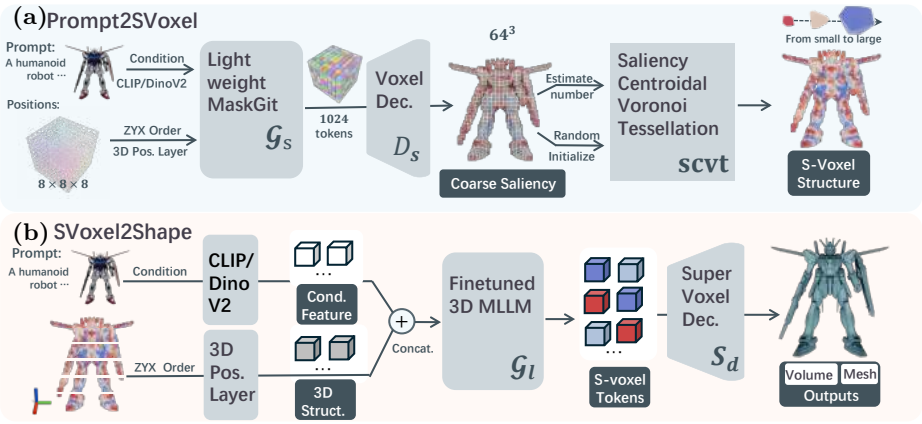



**Fig. 3:** Overview of SuperVoxelGPT. SuperVoxelGPT employs two stages for 3D asset generation: (a) Prompt-to-supervoxel structure. In the Prompt-to-supervoxel stage, we first predict the coarse saliency distribution of a shape via a lightweight MaskGIT model, then use Saliency-guided Centroidal Voronoi Tessellation to partition the space into adaptive supervoxels based on the saliency distribution. (b) Supervoxel-to-shape stage, we fine-tune an MLLM to generate the supervoxel token sequence guided by the multi-modal prompts and supervoxel positions, and decode the tokens into the final 3D shape.

the efficiency bottleneck [35]. For example, OctGPT [39] and Sar3D [6] reformulate 3D generation as autoregression from one resolution to the next. However, such approaches typically require uniform resolution levels and explicit inter-resolution correspondences, forcing uniform token allocation and thus producing overly long sequences, which undermines the efficiency gains that sequence-to-sequence methods are designed to achieve.

Our SuperVoxelGPT employs a two-stage design to solve the efficiency bottleneck. In the first stage, we adopt MaskGIT [2,3] to predict the coarse supervoxel structure efficiently, thereby determining the output sequence length a priori. In the second stage, since the sequence structure is already known, we directly apply Jacobi decoding [30] to parallelize the autoregressive generation of the entire supervoxel token sequence, achieving significant speedup of token-by-token autoregression.

## 3 Method

As shown in Fig. 3, SuperVoxelGPT consists of two stages: (1) a Prompt-to-supervoxel structure stage and (2) a supervoxel-to-shape generation stage. In the first stage, given a text or image prompt, we predict the geometric complexity distribution of the target shape and partition the 3D space into multiple supervoxels according to the complexity concentration, yielding an adaptive supervoxel structure. In the second stage, we fine-tune an MLLM to autoregressively generate the supervoxel token sequence guided by the multimodal

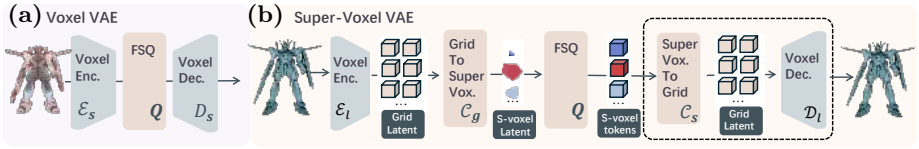



**Fig. 4:** Overview of two VAE structures. (a) Saliency VQ-VAE, (b) SuperVoxelVAE.

prompt and supervoxel positions, and then decode the tokens into the final 3D shape. We describe the Prompt-to-supervoxel structure stage in Sec. 3.1 and the supervoxel-to-shape generation stage in Sec. 3.2.

### 3.1 Prompt-to-Supervoxel Structure Stage

This stage predicts where to allocate tokens based on the geometric complexity distribution of the target shape. Given a text or image prompt, we first estimate a coarse saliency distribution indicating regions of varying geometric detail, then partition the 3D space into adaptive supervoxels—allocating more supervoxels to high-complexity regions and fewer to low-complexity regions. This stage comprises two modules: (1) *Prompt-to-Saliency Volume*, which maps the input prompt to a dense 3D saliency volume encoding the target shape's occupancy and geometric complexity distribution, and (2) *Saliency-guided Centroidal Voronoi Tessellation (SCVT)*, which generates a supervoxel structure adapted to the predicted complexity distribution.

**Prompt-to-Saliency Volume** This module predicts a 3D saliency volume from a text or image prompt using two lightweight components: a Saliency VQ-VAE that encodes dense saliency volumes into 1024 tokens, and a prompt-conditioned MaskGIT model that generates these latent embeddings from the input prompt. We compute ground-truth saliency using spectral mesh saliency [33] and apply max pooling to produce $64^3$ dense saliency volumes. At training time, we train a 3-layer Saliency VQ-VAE to convert saliency volumes into 1024 tokens, and a MaskGIT [2] to generate the token sequence conditioned on the input prompt. At inference time, the MaskGIT generates the token sequence with 12 iterations as suggested in [2], and the decoder reconstructs the saliency volume from the generated tokens.

**Saliency VQ-VAE.** Our Saliency VQ-VAE inherits the encoder and decoder from TRELLIS2 [42] with 3 layers, and adds an FSQ layer in the middle (as shown in Fig. 4(a)). In this way, a $64^3$ saliency volume is encoded into $8 \times 8 \times 8 \times 2 = 1024$ tokens.

Compared to the VAEs in TRELLIS2 [42], our VQ-VAE differs in two aspects: (1) we add a FSQ layer for quantization [25], and (2) we predict not only the occupancy mask but also the saliency values at occupied positions. Prior methods use the last-layer features solely to predict occupancy, while we split the last-layer features into two halves: the first half predicts the occupancy mask, and the second half predicts the saliency values. We inherit the training loss from

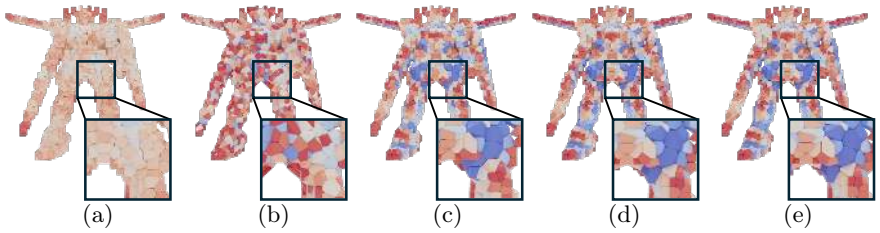



**Fig. 6:** Effect of saliency-driven CVT over GPU iterations: (a) 0, (b) 10, (c) 20, (d) 30, and (e) 40 iterations. The supervoxel partition adapts to geometric complexity, with denser cells near detailed features and sparser cells in smooth regions.

TRELLIS2 [42] and add an additional L1 loss to supervise the saliency values. In this way, a saliency volume is mapped into 1024 tokens.

**Saliency Volume MaskGIT.** To avoid extensive autoregressive and denoising steps, we adopt the MaskGIT architecture with 24 layers to generate 1024 tokens in only 12 iterations, as suggested in [2]. We inherit the original MaskGIT architecture and replace the conditioning inputs with text CLIP features and image DINOv2 features.

**Saliency-guided Centroidal Voronoi Tessellation** The second module converts the predicted saliency volume into an adaptive supervoxel partition using Centroidal Voronoi Tessellation (CVT) [9, 20]. Our goal is to define a piecewise-linear target size field from saliency values to final supervoxel sizes: in geometrically complex regions (high saliency), we maintain unit-size supervoxels with a 1:1 correspondence to grid cells, while in flat or smooth regions (low saliency), we use larger supervoxels of size $K$ to achieve a compact supervoxel structure. Since CVT is a well-established algorithm with theoretical convergence guarantees and only requires a density field as input, we carefully design a two-stage mapping from saliency to size field to density field, ensuring that the converged result meets our compression objective.

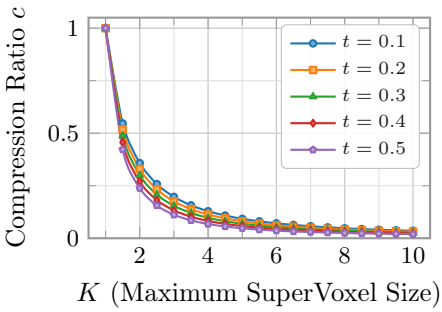



**Fig. 5:** Compression ratio $c$ as a function of max supervoxel size $K$ for different saliency thresholds $t$, computed from statistics over our training dataset. The monotonic decrease enables efficient parameter selection for target compression ratios.

*Saliency-to-Supervoxel Size Field.* We first design a piecewise-linear mapping from saliency $x$ to target supervoxel size $f(x)$, which determines the desired partition that CVT should converge to. Let $x_i \in [0, 1]$ denote the saliency value at voxel $i$, $t$ is a saliency pivot that separates low-importance regions from salient

regions, and $K$ the supervoxel size in low-saliency regions (where $x_i < t$). We define the supervoxel size field $f(x; K, t)$ as:

$$f(x; K, t) = \begin{cases} K & \text{if } x < t \\ \frac{x-t}{1-t} \cdot (1 - K) + K & \text{if } x \geq t \end{cases} \tag{1}$$

where $f(x)$ represents the target supervoxel size at saliency $x$. This piecewise linear function ensures that the highest saliency regions ($x = 1$) maintain unit size ($f(1) = 1$), while low-saliency regions have supervoxel size $K$. The parameter $K$ controls the compression aggressiveness—larger $K$ means coarser supervoxels in smooth regions, freeing up more tokens for geometric details.

*Supervoxel Size Field to Density Field.* CVT requires a density field $d(x)$ as input. To make CVT converge to our target size field $f(x)$, we derive the corresponding density: the CVT cell linear size satisfies $f \propto d^{-1/5}$ [10], so the density function is:

$$d(x) \propto \frac{1}{f(x; K, t)^5} \tag{2}$$

With this density field, CVT converges to supervoxels matching our target size distribution.

*Compression Ratio Control.* A supervoxel with linear size $f$ occupies approximately $f^3$ voxels, so each voxel contributes $1/f^3$ to the total token count. The estimated number of supervoxels for given parameters $(K, t)$ is:

$$N(K, t) = \sum_{i \in V} \frac{1}{f(x_i; K, t)^3} \tag{3}$$

where $x_i$ is the saliency value at voxel $i$. We define compression ratio w.r.t. the baseline voxel-tokenization tokens at the same resolution, i.e., $c = N(K, t)/N_{\text{voxels}}$. Since $N(K, t)$ monotonically decreases with $K$ and $t$, we can explore this relationship empirically. As shown in Fig. 5, increasing $K$ and $t$ progressively coarsens the non-salient regions first and eventually compresses the entire volume, producing a spectrum of compression ratios. Increasing $K$ first yields a rapid decrease in the compression ratio by merging low-saliency regions into larger supervoxels, but the gain gradually saturates once these regions have been largely coarsened; further increasing $K$ then provides limited additional compression and may degrade salient geometry. Thus, $K = 4$ offers a good trade-off between compactness and fidelity. Similarly, increasing $t$ suppresses more low-saliency noises, improving compression but risking the removal of genuinely salient structures when set too high. We find $t = 0.1$ to be a balanced choice. The parameter sensitivity analysis in the supplementary material further validates this parameter selection. Based on these observations, we select $K = 4$ and $t = 0.1$ to achieve our target compression ratio of $c = 12.8\%$.

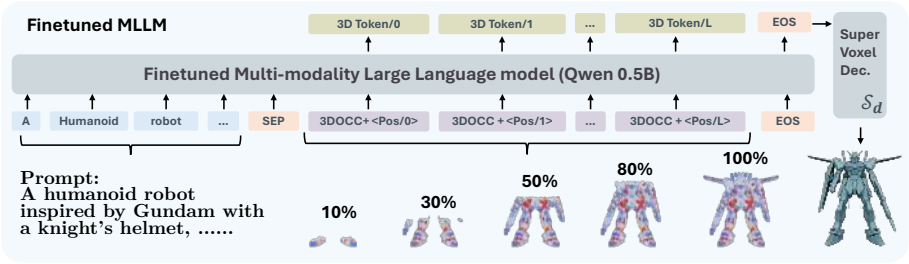



**Fig. 7:** Architecture of the MLLM generation module. The MLLM autoregressively generates the token sequence conditioned on the multimodal prompt, which is then decoded into the final 3D shape.

*CVT Generation Pipeline.* With the density field $d(x)$ and parameters $(K, t)$, we partition the 3D domain into supervoxels through three steps:

1. **Initialization:** Compute the number of supervoxels $N$ using Eq. (3), and initialize $N$ seeds by uniform sampling from occupied voxels.
2. **Lloyd's Iteration:** Iteratively update seed positions using weighted centroids:
$$\mathbf{s}_i^{(k+1)} = \frac{\sum_{\mathbf{x}\in V_i} \mathbf{x} \cdot d(\mathbf{x})}{\sum_{\mathbf{x}\in V_i} d(\mathbf{x})} \tag{4}$$
where $V_i = \{\mathbf{x} : \|\mathbf{x} - \mathbf{s}_i\| \leq \|\mathbf{x} - \mathbf{s}_j\|, \forall j \neq i\}$ is the Voronoi cell of seed $\mathbf{s}_i$, and $d(\mathbf{x})$ is the density at position $\mathbf{x}$. As shown in Fig. 6, after 40 iterations the supervoxels concentrate in geometrically complex regions.
3. **Ordering:** Sort supervoxel centers by Z-Y-X coordinates to obtain a deterministic sequence $\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_N\}$, enabling autoregressive generation.

In this way, we obtain a compact supervoxel structure that is well-shaped and adapted to the geometric complexity distribution of the target shape.

### 3.2 Supervoxel-to-Shape Generation

Given a text or image prompt and the supervoxel structure from the previous stage, this stage generates the corresponding 3D shapes. It consists of two components: (1) *SuperVoxelVAE*, which learns to encode local geometry into discrete supervoxel tokens, and (2) a fine-tuned *MLLM*, which autoregressively generates the token sequence conditioned on the multimodal prompt. During training, we first train SuperVoxelVAE to learn the shape-to-token mapping, then fine-tune Qwen2.5-0.5B to learn the prompt-to-token mapping. At inference time, the MLLM generates 3D tokens from the input prompt, which are decoded into geometry by the SuperVoxelVAE decoder.

**SuperVoxelVAE** SuperVoxelVAE encodes the local 3D geometry within each supervoxel into a compact discrete token. As shown in Fig. 3, it builds upon the

sparse grid convolution architecture from prior methods [40, 42] and introduces three additional layers:

- **Grid-to-Supervoxel Layer:** Compresses grid-level latent features into supervoxel level representations.
- **FSQ Layer:** Applies Finite Scalar Quantization [25] to convert continuous supervoxel latents into discrete tokens.
- **Supervoxel-to-Grid Layer:** Maps supervoxel latents back to grid structure for decoding.

Since these three layers operate solely on the interface between grid latents and supervoxel centers, they are agnostic to the specific sparse convolution backbone. This plug-and-play design allows our supervoxel tokenization to be readily integrated into any existing sparse voxel VAE [40, 42, 43], compressing their token sequences without modifying their encoder-decoder architectures or training losses.

We adopt KNN Cross Attention from PointTransformerV2 [41] for the grid-supervoxel conversion. Taking the grid-to-supervoxel layer as an example: given grid latents $\mathbf{G}_f$ from sparse convolution, we first initialize supervoxel features $\mathbf{V}_f$ via nearest neighbor search between grid coordinates $\mathbf{G}_c$ and supervoxel centers $\mathbf{V}_c$. Then, KNN Cross Attention refines these features into $\mathbf{V}'_f$ by aggregating information from neighboring grid cells. The decoder uses a symmetric supervoxel-to-grid layer for reconstruction. Since our architecture is built upon TRELLIS2 with these three plug-in modules, we directly inherit the loss functions and training paradigm from TRELLIS2 [42].

**MLLM-based Token Generation** With the supervoxel structure $\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_N\}$ defined, we formulate 3D generation as an autoregressive next-token prediction task. We fine-tune Qwen2.5-0.5B [14] to autoregressively predict supervoxel tokens conditioned on both the multimodal prompt and supervoxel positions. Since the sequence length $N$ is already determined by Stage 1, we can directly apply Jacobi decoding [30] at inference time to parallelize token generation.

The input sequence concatenates prompt embeddings with token embeddings augmented by spatial information:

$$\mathbf{E}_{\text{input}} = [\mathbf{E}_{\text{prompt}}; \mathbf{E}_{\text{token}} + \text{SA}^2(\text{FourierPE}(\mathbf{p}_1, \ldots, \mathbf{p}_N))] \tag{5}$$

where $\mathbf{E}_{\text{prompt}}$ is the prompt embeddings (from text tokens or image features), $\mathbf{E}_{\text{token}}$ represents the token embeddings, $\text{FourierPE}(\cdot)$ computes Fourier positional embeddings [36] from supervoxel coordinates, and $\text{SA}^2(\cdot)$ denotes a 2-layer self-attention module that processes the positional embeddings before adding them to the token embeddings. This allows the model to capture inter-position relationships and leverage both semantic and spatial information.

*Autoregressive Training.* We train the model with the standard next-token prediction objective. Given the token sequence $(t_1, t_2, \ldots, t_N)$ ordered by the supervoxel structure, the model learns to predict each token conditioned on all

preceding tokens, the prompt $\mathcal{T}$, and the supervoxel positions:

$$\mathcal{L}_{\text{MLLM}} = -\sum_{i=1}^{N} \log p(t_i | t_{<i}, \mathcal{T}, \mathbf{p}_1, \ldots, \mathbf{p}_N) \tag{6}$$

*Jacobi Decoding.* At inference time, standard autoregressive decoding generates tokens one by one, requiring $N$ sequential forward passes. Since our Stage 1 already determines the sequence length $N$, we apply Jacobi decoding [30] to accelerate inference. Jacobi decoding initializes all $N$ token positions with random predictions and iteratively refines them in parallel: at each iteration, every position is updated simultaneously by predicting $t_i$ conditioned on the current estimates of $t_{<i}$. When consecutive tokens converge to a fixed point, they are accepted together, effectively generating multiple tokens per iteration. The decoded tokens are then processed by SuperVoxelVAE decoder to reconstruct the final 3D shape.

## 4 Experiments

We evaluate SuperVoxelGPT on both text-to-3D and image-to-3D generation tasks. We first describe the experimental setup (Sec. 4.1), then compare with state-of-the-art methods (Sec. 4.2), and finally discuss limitations (Sec. 4.3). Ablation studies, parameter sensitivity analysis, discussion, and more quantitative results are provided in the supplementary material.

### 4.1 Experimental Setup

*Datasets.* We select a subset from the Trellis-500K dataset [43], using 10,000 shapes for training and 1,000 for testing. We filter out meshes with poor quality by removing those whose non-manifold faces exceed 20% of the total surface area, as such meshes produce unreliable ground-truth saliency and supervoxel structures.

*Evaluation Metrics.* We evaluate generation quality from four aspects: geometry, semantics, surface detail, and efficiency. For geometric accuracy, we use L2 Chamfer Distance ($\text{CD}_{L2}$) and PSNR on rendered multi-view normal images. For semantic alignment, we compute ULIP-2 Similarity, which measures the cosine similarity between generated and ground-truth shapes in a unified 3D-language embedding space. For surface detail quality, we report the area-weighted sum of mean curvature as Mean Curvature Sum (MCS). For efficiency, we report inference time (seconds per shape) and the number of tokens generated. The calculation definitions of these metrics can be found in the supplementary material.

**Table 1:** Quantitative comparison on text-to-3D generation. ↓ indicates lower is better, ↑ indicates higher is better. Best results are in **bold**, second best are underlined. All methods report averages over the test set.

| Method | $CD_{L2}$ ↓ | PSNR ↑ | ULIP-2 Sim. ↑ | MCS ↑ | Time (s) ↓ | Resolution | Tokens |
|---|---|---|---|---|---|---|---|
| BrickGPT [27] | 0.0851 | 12.99 | 0.199 | 15.39 | 215.5 | $20^3$ | 179 |
| OctGPT [39] | 0.0797 | 19.80 | 0.329 | 11.74 | 165.8 | $256^3$ | 47,783 |
| **Ours** | **0.0134** | **32.05** | **0.469** | **44.19** | **4.48** | $1024^3$ | **1,065** |

**Table 2:** Quantitative comparison on image-to-3D generation. Best results are in **bold**, second best are underlined. Resolution and token counts are reported for each method.

| Method | $CD_{L2}$ ↓ | PSNR ↑ | ULIP-2 Sim. ↑ | MCS ↑ | Time (s) ↓ | Resolution | Tokens |
|---|---|---|---|---|---|---|---|
| CraftsMan3D [21] | 0.0252 | 22.90 | 0.448 | 37.85 | 7.92 | $512^3$ | 2,048 |
| TRELLIS [43] | 0.0182 | 27.28 | 0.464 | 21.64 | 9.03 | $256^3$ | 8,147 |
| Direct3D-S2 [40] | 0.0168 | 26.14 | 0.461 | 33.17 | 137.2 | $1024^3$ | 55,420 |
| TRELLIS2 [42] | 0.0123 | **32.19** | **0.475** | **44.24** | 34.25 | $1024^3$ | 7,303 |
| **Ours** | **0.0122** | 32.08 | 0.474 | 44.21 | **4.60** | $1024^3$ | **1,048** |

*Implementation Details.* Our method consists of two generative modules (MaskGIT and fine-tuned MLLM) and two VAEs (Saliency VQ-VAE and SuperVoxelVAE). The Saliency Volume MaskGIT follows the MaskGIT architecture with 24 layers conditioned on text CLIP features and image DINOv2 features, generating tokens in 12 iterations at inference. The MLLM is fine-tuned from Qwen2.5-0.5B [14] with the standard next-token prediction objective and applies Jacobi decoding [30] at inference. Since the number of supervoxels is known from Stage 1, we set the Jacobi context window to cover all supervoxel tokens so that all tokens can be updated in parallel, and cap the iterations to 30. The Saliency VQ-VAE uses 3 encoder/decoder layers with an FSQ layer (num_levels $= [9, 9, 5, 5]$, num_quantizer $= 2$), encoding each $64^3$ saliency volume into 1024 tokens. The SuperVoxelVAE is built upon TRELLIS2 [42] with plug-in grid-to-supervoxel, FSQ, and supervoxel-to-grid layers. We set the KNN neighborhood size for grid-supervoxel conversion to 16 and the FSQ codebook levels to $[9, 9, 5, 5, 5]$. We discretize the density field on a $256^3$ grid (upsampled from $64^3$) to approximate the CVT energy and weighted centroids. All trainable modules are trained for 300 epochs with a learning rate of $5 \times 10^{-5}$ and 5,000 warm-up steps on 8×H100 GPUs (AMD EPYC 9334 32-Core Processor). All inference time measurements are conducted on a single NVIDIA RTX 4090.

*Baselines.* For text-to-3D, we compare with BrickGPT [27] and OctGPT [39]. For image-to-3D, we compare with CraftsMan3D [21], TRELLIS [43], Direct3D-S2 [40], and TRELLIS2 [42]. Detailed parameter settings for all baselines can be found in the supplementary material.


**Fig. 8:** Qualitative comparison on image-to-3D generation. Given an input image, we compare the generated 3D shapes from CraftsMan3D, TRELLIS, Direct3D-S2, TRELLIS2, and our method (SV, i.e., supervoxel structure).

### 4.2 Comparison with State-of-the-Art Methods

Tab. 1 and Tab. 2 present quantitative comparisons on the text-to-3D and image-to-3D generation tasks, respectively.

*Generation Quality.* Our method achieves generation quality on par with the best existing methods across both text-to-3D and image-to-3D tasks in terms of geometric accuracy, semantic alignment, and surface detail. Taking the image-to-3D task as an example (Tab. 2), our method, TRELLIS2, and Direct3D-S2 all operate at $1024^3$ resolution, which enables richer geometric detail and stronger generation performance compared to lower-resolution methods such as CraftsMan3D ($512^3$) and TRELLIS ($256^3$). Moreover, since our SuperVoxelVAE inherits the multi-view normal rendering loss from TRELLIS2 [42], the generated shapes are sharper and more geometrically faithful compared to Direct3D-S2. As shown in Tab. 2, our method closely matches TRELLIS2 across all quality metrics, with only a marginal gap in multi-view normal PSNR ($-0.11$ dB). A

minor limitation is that regions with saliency below $t = 0.1$ receive a much lower token budget due to aggressive coarsening, which may wash out very fine-grained structures—*e.g.*, the ring-shaped grooves on the pipe in Fig. 8(e) and the spoke grooves on the wheel hub in Fig. 8(f). Lowering $t$ could alleviate this issue but would reduce the compression ratio and make the method more susceptible to saliency-prediction noises. This explains why our method is slightly inferior to TRELLIS2 on particularly fine details.

For the text-to-3D task (Tab. 1), our method also demonstrates strong performance. Compared to BrickGPT and OctGPT, which are limited to low-resolution generation and constrained to specific shape categories (*e.g.*, LEGO structures), our method supports general-purpose shapes and scales to $1024^3$ resolution, achieving substantially better results across all metrics.

*Generation Efficiency.* Our method achieves a significant speedup over existing approaches. Compared to the average inference time of all image-to-3D baselines (47.1s), our method reduces inference time by 10×, requiring only 4.60s to generate a $1024^3$-resolution shape. The inference time breaks down as follows: CVT takes 1.209s, the SuperVoxelVAE decoder takes 1.147s, the MLLM takes 1.202s, and the MaskGIT and Saliency VQ-VAE decoder contribute 0.805s and 0.237s, respectively. We attribute this speedup to two factors. First, our token generation is highly parallel: the MaskGIT natively generates all saliency tokens in parallel, and the MLLM leverages Jacobi decoding to generate multiple shape tokens per iteration once the supervoxel count is known. This avoids the sequential bottleneck of generating one token at a time. Second, our supervoxel tokenization produces significantly fewer tokens than competing methods (*e.g.*, 1,048 *vs*. 55,420 for Direct3D-S2 and 47,783 for OctGPT), directly translating into faster autoregressive generation.

### 4.3 Limitations

While SuperVoxelGPT achieves strong performance with compact token sequences, several limitations remain:

- **Saliency prediction dependency:** The quality of the final shape depends on the accuracy of the first-stage saliency prediction. As illustrated in Fig. 8(c), since saliency values below $t = 0.1$ are heavily coarsened, our method is insensitive to structures with very low geometric saliency, such as the mesh-like perforations on the shoe. Because these regions are not identified as salient, only sparse tokens are allocated to reconstruct them, causing such fine structures to be smoothed over.
- **Degradation on detail-rich shapes:** For shapes where fine details are uniformly distributed across the surface, the saliency-guided allocation degenerates towards a near-uniform distribution, reducing the compression advantage.
- **Geometry-only generation:** our supervoxel-based compression is currently designed for geometry and does not extend to texture modeling; thus, our

framework cannot generate or represent surface textures or material properties. In addition, our method is trained on only 10,000 shapes from the Trellis-500K dataset, which may limit its generalization compared to methods trained on larger datasets.

## 5 Conclusion

We presented SuperVoxelGPT, a representation-first framework for high-resolution 3D generation that mitigates the fundamental tension between sequence compactness and geometric fidelity in autoregressive 3D generation. Our key insight is to decouple *where to allocate tokens* from *what geometry to generate*. Our experiments demonstrate that SuperVoxelGPT compresses the token sequence to 12.8% of uniform voxel tokenization while achieving generation quality on par with state-of-the-art methods and a 10× inference speedup, validating that adaptive tokenization is a promising direction for scalable and efficient 3D generation. Moreover, the plug-and-play nature of our supervoxel layers makes them readily applicable to other sparse voxel VAE frameworks, offering a general-purpose compression module for autoregressive 3D generation.

## References

1. Caffagni, D., Cocchi, F., Barsellotti, L., Moratelli, N., Sarto, S., Baraldi, L., Cornia, M., Cucchiara, R.: The revolution of multimodal large language models: A survey. Findings of the association for computational linguistics: ACL 2024 pp. 13590–13618 (2024)
2. Chang, H., Zhang, H., Jiang, L., Liu, C., Freeman, W.T.: Maskgit: Masked generative image transformer. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 11315–11325 (2022)
3. Chen, J., Zhu, L., Hu, Z., Qian, S., Chen, Y., Wang, X., Lee, G.H.: Mar-3d: Progressive masked auto-regressor for high-resolution 3d generation. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 11083–11092 (2025)
4. Chen, R., Zhang, J., Liang, Y., Luo, G., Li, W., Liu, J., Li, X., Long, X., Feng, J., Tan, P.: Dora: Sampling and benchmarking for 3d shape variational auto-encoders. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 16251–16261 (2025)
5. Chen, Y., Wang, Y., Luo, Y., Wang, Z., Chen, Z., Zhu, J., Zhang, C., Lin, G.: Meshanything v2: Artist-created mesh generation with adjacent mesh tokenization. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 13922–13931 (2025)
6. Chen, Y., Lan, Y., Zhou, S., Wang, T., Pan, X.: Sar3d: Autoregressive 3d object generation and understanding via multi-scale 3d vqvae. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 28371–28382 (2025)
7. Delétang, G., Ruoss, A., Duquenne, P.A., Catt, E., Genewein, T., Mattern, C., Grau-Moya, J., Wenliang, L.K., Aitchison, M., Orseau, L., et al.: Language modeling is compression. In: International Conference on Learning Representations. pp. 14165–14181 (2024)

8. Deng, K., Liu, H.T.D., Zhu, Y., Sun, X., Shang, C., Bhat, K.S., Ramanan, D., Zhu, J.Y., Agrawala, M., Zhou, T.: Efficient autoregressive shape generation via octree-based adaptive tokenization. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 11685–11696 (2025)
9. Du, Q., Faber, V., Gunzburger, M.: Centroidal voronoi tessellations: Applications and algorithms. SIAM review **41**(4), 637–676 (1999)
10. Du, Q., Gunzburger, M., Ju, L.: Advances in studies and applications of centroidal voronoi tessellations. Numerical Mathematics: Theory, Methods and Applications **3**(2), 119–142 (2010)
11. Fan, H., Su, H., Guibas, L.J.: A point set generation network for 3d object reconstruction from a single image. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 605–613 (2017)
12. He, X., Zou, Z.X., Chen, C.H., Guo, Y.C., Liang, D., Yuan, C., Ouyang, W., Cao, Y.P., Li, Y.: Sparseflex: High-resolution and arbitrary-topology 3d shape modeling. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 14822–14833 (2025)
13. Hou, W., Zhou, L., Hu, H.Y., Chen, Y., You, Y.Z., Qi, X.L.: How focused are llms? a quantitative study via repetitive deterministic prediction tasks. arXiv preprint arXiv:2511.00763 (2025)
14. Hui, B., Yang, J., Cui, Z., Yang, J., Liu, D., Zhang, L., Liu, T., Zhang, J., Yu, B., Lu, K., et al.: Qwen2.5-coder technical report. arXiv preprint arXiv:2409.12186 (2024)
15. Jia, T., Yan, D., Hao, D., Li, Y., Zhang, K., He, X., Li, L., Wang, Y., Chen, J., Jiang, L., et al.: Ultrashape 1.0: High-fidelity 3d shape generation via scalable geometric refinement. arXiv preprint arXiv:2512.21185 (2025)
16. Jiang, Z., Yang, M., Tsirlin, M., Tang, R., Dai, Y., Lin, J.: "low-resource" text classification: A parameter-free classification method with compressors. In: Findings of the Association for Computational Linguistics: ACL 2023. pp. 6810–6828 (2023)
17. Lai, Z., Zhao, Y., Liu, H., Zhao, Z., Lin, Q., Shi, H., Yang, X., Yang, M., Yang, S., Feng, Y., et al.: Hunyuan3d 2.5: Towards high-fidelity 3d assets generation with ultimate details. arXiv preprint arXiv:2506.16504 (2025)
18. Lan, Y., Zhou, S., Lyu, Z., Hong, F., Yang, S., Dai, B., Pan, X., Loy, C.C.: Gaussiananything: Interactive point cloud flow matching for 3d generation. In: International Conference on Learning Representations. pp. 97419–97443 (2025)
19. Lee, C.H., Varshney, A., Jacobs, D.W.: Mesh saliency. In: ACM SIGGRAPH 2005 Papers, pp. 659–666 (2005)
20. Lévy, B., Liu, Y.: Lp centroidal voronoi tessellation and its applications. ACM Transactions on Graphics (TOG) **29**(4), 1–11 (2010)
21. Li, W., Liu, J., Yan, H., Chen, R., Liang, Y., Chen, X., Tan, P., Long, X.: Craftsman3d: High-fidelity mesh generation with 3d native diffusion and interactive geometry refiner. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 5307–5317 (2025)
22. Li, Z., Wang, Y., Zheng, H., Luo, Y., Wen, B.: Sparc3d: Sparse representation and construction for high-resolution 3d shapes modeling. Advances in Neural Information Processing Systems **38**, 118582–118600 (2026)
23. Liu, P., Ren, X., Liu, F., Xie, Q., Zheng, Q., Zhang, Y., Lu, H., Yang, Y.: Dynamic-i2v: Exploring image-to-video generation models via multimodal llm. arXiv preprint arXiv:2505.19901 (2025)
24. McCoy, R.T., Yao, S., Friedman, D., Hardy, M.D., Griffiths, T.L.: Embers of autoregression show how large language models are shaped by the problem they

are trained to solve. Proceedings of the National Academy of Sciences **121**(41), e2322420121 (2024)
25. Mentzer, F., Minnen, D., Agustsson, E., Tschannen, M.: Finite scalar quantization: Vq-vae made simple. In: International Conference on Learning Representations. pp. 51772–51783 (2024)
26. Meyer, M., Desbrun, M., Schröder, P., Barr, A.H.: Discrete differential-geometry operators for triangulated 2-manifolds. In: Visualization and mathematics III, pp. 35–57. Springer (2003)
27. Pun, A., Deng, K., Liu, R., Ramanan, D., Liu, C., Zhu, J.Y.: Generating physically stable and buildable brick structures from text. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 14798–14809 (2025)
28. Ravi, N., Reizenstein, J., Novotny, D., Gordon, T., Lo, W.Y., Johnson, J., Gkioxari, G.: Accelerating 3d deep learning with pytorch3d. arXiv preprint arXiv:2007.08501 (2020)
29. Ren, X., Huang, J., Zeng, X., Museth, K., Fidler, S., Williams, F.: Xcube: Large-scale 3d generative modeling using sparse voxel hierarchies. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 4209–4219 (2024)
30. Santilli, A., Severino, S., Postolache, E., Maiorca, V., Mancusi, M., Marin, R., Rodolà, E.: Accelerating transformer inference for translation via parallel decoding. In: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 12336–12355 (2023)
31. Siddiqui, Y., Alliegro, A., Artemov, A., Tommasi, T., Sirigatti, D., Rosov, V., Dai, A., Nießner, M.: Meshgpt: Generating triangle meshes with decoder-only transformers. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 19615–19625 (2024)
32. Song, G., Zhao, Z., Weng, H., Zeng, J., Jia, R., Gao, S.: Mesh silksong: Auto-regressive mesh generation as weaving silk. arXiv preprint arXiv:2507.02477 (2025)
33. Song, R., Liu, Y., Martin, R.R., Rosin, P.L.: Mesh saliency via spectral processing. ACM Transactions On Graphics (TOG) **33**(1), 1–17 (2014)
34. Tang, J., Li, M., Hao, Z., Liu, X., Zeng, G., Liu, M.Y., Zhang, Q.: Edgerunner: Auto-regressive auto-encoder for artistic mesh generation. In: International Conference on Learning Representations. pp. 35913–35934 (2025)
35. Tian, K., Jiang, Y., Yuan, Z., Peng, B., Wang, L.: Visual autoregressive modeling: Scalable image generation via next-scale prediction. Advances in neural information processing systems **37**, 84839–84865 (2024)
36. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
37. Wang, Z., Wang, L., Zhao, Z., Wu, M., Lyu, C., Li, H., Cai, D., Zhou, L., Shi, S., Tu, Z.: Gpt4video: A unified multimodal large language model for instruction-followed understanding and safety-aware generation. In: Proceedings of the 32nd ACM International Conference on Multimedia. pp. 3907–3916 (2024)
38. Wang, Z., Lorraine, J., Wang, Y., Su, H., Zhu, J., Fidler, S., Zeng, X.: Llama-mesh: Unifying 3d mesh generation with language models. arXiv preprint arXiv:2411.09595 (2024)
39. Wei, S.T., Wang, R.H., Zhou, C.Z., Chen, B., Wang, P.S.: Octgpt: Octree-based multiscale autoregressive models for 3d shape generation. In: Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers. pp. 1–11 (2025)

40. Wu, S., Lin, Y., Zhang, F., Zeng, Y., Yang, Y., Qian, J., Zhu, S., Cao, X., Torr, P., Yao, Y., et al.: Direct3d-s2: Gigascale 3d generation made easy with spatial sparse attention. Advances in Neural Information Processing Systems **38**, 170778–170804 (2026)
41. Wu, X., Lao, Y., Jiang, L., Liu, X., Zhao, H.: Point transformer v2: Grouped vector attention and partition-based pooling. Advances in Neural Information Processing Systems **35**, 33330–33342 (2022)
42. Xiang, J., Chen, X., Xu, S., Wang, R., Lv, Z., Deng, Y., Zhu, H., Dong, Y., Zhao, H., Yuan, N.J., et al.: Native and compact structured latents for 3d generation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 14419–14429 (2026)
43. Xiang, J., Lv, Z., Xu, S., Deng, Y., Wang, R., Zhang, B., Chen, D., Tong, X., Yang, J.: Structured 3d latents for scalable and versatile 3d generation. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 21469–21480 (2025)
44. Xue, L., Yu, N., Zhang, S., Panagopoulou, A., Li, J., Martín-Martín, R., Wu, J., Xiong, C., Xu, R., Niebles, J.C., et al.: Ulip-2: Towards scalable multimodal pre-training for 3d understanding. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 27091–27101 (2024)
45. Yin, S., Fu, C., Zhao, S., Li, K., Sun, X., Xu, T., Chen, E.: A survey on multimodal large language models. National Science Review **11**(12), nwae403 (2024)
46. Zhang, B., Tang, J., Niessner, M., Wonka, P.: 3dshape2vecset: A 3d shape representation for neural fields and generative diffusion models. ACM Transactions On Graphics (TOG) **42**(4), 1–16 (2023)
47. Zhang, Z., Shi, Y., Yang, L., Ni, S., Ye, Q., Wang, J.: Openhoi: Open-world hand-object interaction synthesis with multimodal large language model. Advances in Neural Information Processing Systems **38**, 166582–166612 (2026)
48. Zheng, J., Tan, T.S.: Computing centroidal voronoi tessellation using the gpu. In: Symposium on interactive 3D graphics and games. pp. 1–9 (2020)
49. Zhou, Q.Y., Park, J., Koltun, V.: Fast global registration. In: European conference on computer vision. pp. 766–782. Springer (2016)
50. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Transactions on information theory **23**(3), 337–343 (1977)

# Supplementary Material

## A Metrics Calculation

We provide detailed definitions of the evaluation metrics used in the main paper. We first describe the shape alignment procedure (Appendix A.1), which is a prerequisite for most metrics, and then define each metric in detail (Appendix A.2).

### A.1 Alignment

Given a generated shape $\mathcal{A}$ and the corresponding ground-truth shape $\mathcal{B}$, we apply all $2^3 = 8$ axis-flip combinations along the $x$, $y$, and $z$ coordinate axes to $\mathcal{A}$, producing eight candidate orientations $\{\mathcal{A}_k\}_{k=1}^{8}$. For each candidate, we uniformly sample 100,000 points from both $\mathcal{A}_k$ and $\mathcal{B}$, and register $\mathcal{A}_k$ to $\mathcal{B}$ using Fast Global Registration (FGR) [49] for coarse alignment followed by Iterative Closest Point (ICP) for refinement. We then compute the $L_2$ Chamfer Distance between each aligned pair and retain only the shape $\mathcal{A}^*$ that yields the smallest Chamfer Distance. All subsequent shape-comparison metrics are evaluated using $\mathcal{A}^*$ and $\mathcal{B}$.

### A.2 Metrics

*$L_2$ Chamfer Distance ($\mathrm{CD}_{L_2}$).* We uniformly sample 100,000 points from the surfaces of the aligned generated mesh $\mathcal{A}^*$ and the ground-truth mesh $\mathcal{B}$, yielding point sets $P$ and $Q$, respectively. The $L_2$ Chamfer Distance [11] is defined as:

$$\mathrm{CD}_{L_2}(P,Q) = \frac{1}{|P|}\sum_{\mathbf{p}\in P}\min_{\mathbf{q}\in Q}\|\mathbf{p}-\mathbf{q}\|_2^2 \;+\; \frac{1}{|Q|}\sum_{\mathbf{q}\in Q}\min_{\mathbf{p}\in P}\|\mathbf{q}-\mathbf{p}\|_2^2, \tag{A1}$$

where the two terms measure the average squared nearest-neighbor distance from $P$ to $Q$ and from $Q$ to $P$, respectively. Lower $\mathrm{CD}_{L_2}$ indicates better geometric reconstruction quality.

*PSNR on Multi-View Normal Maps.* We render multi-view normal maps from 24 random viewpoints for both the aligned generated and ground-truth shapes using the PyTorch3D rasterizer [28], with the camera radius set to 2 and the field of view (FoV) set to 40°. The Peak Signal-to-Noise Ratio (PSNR) is computed per view and averaged:

$$\mathrm{PSNR} = \frac{1}{V}\sum_{v=1}^{V} 10\cdot\log_{10}\left(\frac{\mathrm{MAX}^2}{\mathrm{MSE}_v}\right), \tag{A2}$$

where $V$ is the number of rendered views, MAX is the maximum possible pixel value, and $\mathrm{MSE}_v$ denotes the mean squared error between the normal-map renderings of view $v$. Higher PSNR indicates better geometric surface fidelity.

*ULIP-2 Similarity.* ULIP-2 Similarity [44] measures the semantic alignment between a generated shape and its reference description in a unified vision-language-3D embedding space. We extract ULIP-2 feature embeddings $\mathbf{f}_{\text{gen}}$ and $\mathbf{f}_{\text{ref}}$ from the generated shape and the reference, respectively, and compute their cosine similarity:

$$\text{ULIP-2 Sim.} = \frac{\mathbf{f}_{\text{gen}} \cdot \mathbf{f}_{\text{ref}}}{\|\mathbf{f}_{\text{gen}}\| \, \|\mathbf{f}_{\text{ref}}\|}. \tag{A3}$$

Higher ULIP-2 Similarity indicates that the generated shape is more semantically consistent with the reference.

*Mean Curvature Sum (MCS).* MCS quantifies the geometric richness and surface detail of a generated mesh without requiring a ground-truth reference. We first normalize the mesh to a unit bounding box centered at the origin and compute the mean curvature $H_i$ at each vertex $i$ via the cotangent Laplacian [26]. The MCS is then computed as the area-weighted average of absolute mean curvatures over all vertices:

$$\text{MCS} = \frac{\sum_{i=1}^{N} A_i \, |H_i|}{\sum_{i=1}^{N} A_i}, \tag{A4}$$

where $N$ is the number of vertices and $A_i$ is the barycentric area associated with vertex $i$. Higher MCS indicates richer surface detail and more geometric complexity.

*Token and Time Efficiency.* For all methods, we report the average runtime over 1,000 cases on an RTX 4090. For each method, we count the number of tokens required to generate the shape at the highest resolution or fidelity level. For example, for OctGPT, we use the number of tokens corresponding to the highest-resolution mesh from the final regression step.

*Global Codebook Utilization (GCU).* GCU measures the fraction of the entire codebook that is actively used across the full dataset. Given a codebook of size $C$ and $N$ tokenized shapes, let $\mathcal{U} = \bigcup_{n=1}^{N} \{s_1^{(n)}, \ldots, s_{L_n}^{(n)}\}$ denote the set of all distinct codebook indices observed across all shapes. GCU is defined as:

$$\text{GCU} = \frac{|\mathcal{U}|}{C}. \tag{A5}$$

A higher GCU indicates more effective utilization of the codebook capacity, while a low GCU signals codebook collapse where large portions of the codebook remain unused.

*Average Per-Shape Codebook Utilization (APCU).* APCU captures how diversely each individual shape utilizes the codebook. For shape $n$ with token sequence $\mathbf{s}^{(n)} = (s_1, \ldots, s_{L_n})$, let $\mathcal{U}^{(n)} = \{s_1^{(n)}, \ldots, s_{L_n}^{(n)}\}$ denote the set of distinct codebook indices used by that shape. APCU is defined as the ratio of distinct indices

to the sequence length, averaged over all shapes:

$$\text{APCU} = \frac{1}{N}\sum_{n=1}^{N}\frac{|\mathcal{U}^{(n)}|}{L_n}. \tag{A6}$$

A higher APCU indicates that each shape uses a richer variety of codebook entries, whereas a low APCU suggests that individual shapes are dominated by a small number of repeated codes.

*Spatial Ordering Compressibility Gap ($\Delta_{gzip}$).* To quantify whether spatial sorting induces learnable sequential structure in the token sequence, we measure the compressibility gain of sorted sequences over randomly permuted ones using gzip [50], following recent work that employs general-purpose compressors as non-parametric entropy estimators [7,16]. For each shape $n$ with spatially sorted token sequence $\mathbf{s}^{(n)} = (s_1^{(n)}, \ldots, s_{L_n}^{(n)})$, we compute the gzip compression rate (in bits per token):

$$R^{(n)} = \frac{8 \cdot |\texttt{gzip}(\mathbf{s}^{(n)})|}{L_n}, \tag{A7}$$

where $|\texttt{gzip}(\cdot)|$ denotes the compressed size in bytes. We similarly compute the average compression rate over $K$ random permutations $\pi_k$ of the same token *set*:

$$\bar{R}_{\text{rand}}^{(n)} = \frac{1}{K}\sum_{k=1}^{K}\frac{8 \cdot |\texttt{gzip}(\pi_k(\mathbf{s}^{(n)}))|}{L_n}. \tag{A8}$$

The Spatial Ordering Compressibility Gap is defined as:

$$\Delta_{\text{gzip}} = \frac{1}{N}\sum_{n=1}^{N}\left(\bar{R}_{\text{rand}}^{(n)} - R^{(n)}\right). \tag{A9}$$

A larger $\Delta_{\text{gzip}}$ indicates that spatial sorting produces more compressible sequences than random orderings, implying the existence of exploitable sequential structure for autoregressive modeling. A $\Delta_{\text{gzip}} \approx 0$ suggests that the token sequence exhibits little spatial structure, indicating that autoregressive modeling may not benefit from the chosen ordering.

## B Baseline Settings

We evaluate all baselines using their official implementations and default configurations unless otherwise noted.

*BrickGPT [27].* BrickGPT fine-tunes Llama-3.2-1B-Instruct with LoRA ($r$=32, $\alpha$=16) to autoregressively generate LEGO-style brick assemblies at $20^3$ resolution from text prompts. We cap both rejection sampling and physics-informed rollback at 50 iterations to balance generation quality and efficiency.

*OctGPT [39].* OctGPT adopts octree-based tokenization at $256^3$ resolution and generates shapes through next-scale prediction. We employ the publicly released checkpoint pretrained on Objaverse for text-conditioned generation with an octree depth of 8. The iteration counts per depth level (depths 3–6) are set to [64, 128, 128, 256], with starting temperatures of [1.0, 1.2, 0.5, 0.5], respectively.

*CraftsMan3D [21].* We adopt the CraftsMan3D variant equipped with DoraVAE, which encodes shapes into 2048 latent tokens for improved geometric detail. Meshes are extracted at $512^3$ resolution with 50 sampling steps and a classifier-free guidance (CFG) scale of 7.5.

*TRELLIS [43].* We use the official `TRELLIS-image-large` checkpoint with its default configuration. TRELLIS follows a two-stage flow matching pipeline: sparse structure generation (25 steps, CFG scale 7.5) followed by structured latent generation (25 steps, CFG scale 3.0), with meshes decoded from 3D Gaussian splatting representations.

*Direct3D-S2 [40].* Direct3D-S2 employs sparse voxel SDF representations with a three-stage cascaded diffusion pipeline operating at up to $1024^3$ resolution. We use 50, 30, and 15 DDIM sampling steps for the dense, $512^3$, and $1024^3$ stages, respectively, with a CFG scale of 7.5 for all stages.

*TRELLIS2 [42].* We use the official `TRELLIS.2-4B` checkpoint with the single-stage 1024 pipeline. TRELLIS2 represents shapes as O-Voxel sparse voxels with 16× spatial downsampling, first generating a sparse structure at $64^3$ latent resolution and then producing shape latents at $1024^3$. All stages employ 50 Flow Euler sampling steps with a CFG scale of 7.5.

*SuperVoxelGPT.* We describe the data processing pipeline and architecture settings of SuperVoxelGPT.

*Data Processing.* We adopt the same rendering setup as TRELLIS2 for multi-view image generation and use the captions from Trellis-500K as multimodal prompts. Unlike TRELLIS2, whose first stage predicts only binary occupancy, our Stage 1 additionally predicts per-voxel *saliency*—a measure of local geometric complexity. Specifically, we compute mesh saliency on the original surface and

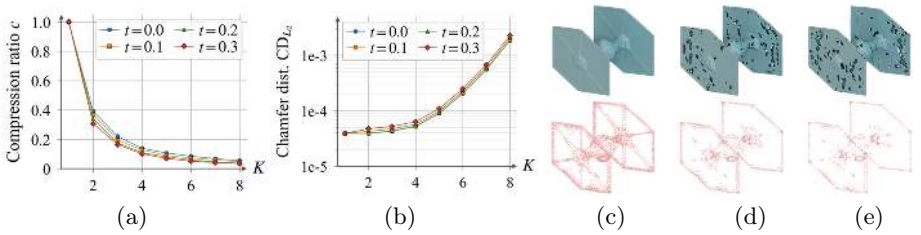



**Fig. C1:** Parameter sensitivity analysis on 1,000 test shapes. (a) Compression ratio $c$ and (b) encode-decode $\mathrm{CD}_{L_2}$ for varying $(K, t)$; lower is better. (c)–(e) show the generated shape (top) and the corresponding supervoxel centroids (bottom) at ($K$=4, $t$=0.1), ($K$=6, $t$=0.1), and ($K$=8, $t$=0.1), respectively.

assign each occupied voxel the maximum saliency value among its interior mesh vertices. The resulting $64^3$ saliency volume is upsampled to $256^3$ via cubic interpolation, from which supervoxel centers are computed through GPU-accelerated Centroidal Voronoi Tessellation (CVT) [48]. For Stage 2, we follow the same output representation as TRELLIS2, using its O-Voxel algorithm to compute ground-truth shape labels and decode the generated structure into meshes.

*Architecture.* Our framework employs two VQ-VAE architectures. The *Saliency VQ-VAE* compresses a $64^3$ saliency volume into $8^3 \times 2 = 1{,}024$ tokens via three downsampling stages and two residual quantizers. The *SuperVoxelVAE* inherits the VAE backbone of TRELLIS2 but introduces plug-in Grid-to-Supervoxel (knn neighborhood size 16), FSQ (quantization levels [9, 9, 5, 5, 5]), and Supervoxel-to-Grid (knn neighborhood size 16) layers that encode shape geometry into one token per supervoxel. During training of both VQ-VAE models, we randomly overwrite 0–5% of tokens with random indices to ensure robustness against noise. To maximize inference speed, we adopt MaskGIT for parallel prediction of all 1,024 saliency tokens in Stage 1, which is among the fastest generation algorithms available. Moreover, since saliency volume prediction provides the sequence length for Stage 2, we directly apply Jacobi decoding implementation [30] to generate the supervoxel token sequence in parallel.

## C Parameter Sensitivity Analysis

As a token compression method, our saliency-based CVT relies on two important parameters to control the compression ratio. Specifically, $K \in [1, 64]$ controls the maximum cell size in low-saliency regions and therefore determines the upper bound of local compression, while $t \in [0, 1]$ is the saliency threshold used to suppress small saliency-estimation noise. In this section, we focus on how $K$ and $t$ affect both compression ratio and reconstruction quality. We sweep $K \in \{1, \ldots, 8\}$ and $t \in \{0, 0.1, 0.2, 0.3\}$, retrain and evaluate 32 SuperVoxelVAE variants, and report compression and encode-decode reconstruction performance on 1,000 test shapes in Fig. C1.

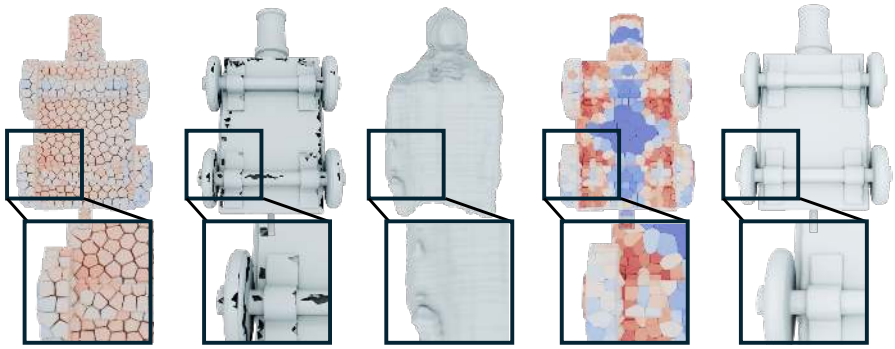



**Fig. C2:** Qualitative ablation results. (a) Uniform CVT supervoxel partition (Model 2). (b) Generation result with uniform CVT. (c) Generation result from CraftsMan3D plus FSQ layer (Model 3). (d) Our saliency-guided CVT supervoxel partition. (e) Our result. The color definition of saliency is the same as Fig.1 in the main paper.

The results show a clear trade-off between compactness and reconstruction fidelity. Larger $K$ and $t$ produce fewer supervoxel cells and thus stronger compression, but they also increase reconstruction error because more fine-grained geometric regions are merged into coarse cells. The setting $K$=4 lies near the knee of the compression curve while preserving reconstruction quality: it benefits from the sweet-spot where non-uniform detail distribution yields a rapid reduction in token count, while avoiding the over-compression of salient regions observed with larger values such as $K$=6 or $K$=8. Meanwhile, $t$=0.1 provides additional compression with nearly unchanged reconstruction quality compared with $t$=0. We therefore use ($K$=4, $t$=0.1) as the default setting in our experiments.

**Table D1:** Ablation on decoding strategy. Token-by-token decoding (Model 1) and Jacobi decoding (Ours) achieve comparable generation quality and token prediction accuracy, while Jacobi decoding is significantly faster. Token Acc. measures the percentage of predicted tokens that match the ground-truth token indices produced by SuperVoxelVAE encoding.

| Decoding Strategy | $CD_{L2}$ ↓ | PSNR ↑ | ULIP-2 Sim. ↑ | MCS ↑ | Token Acc. (%) ↑ | Infer Time (s) ↓ |
|---|---|---|---|---|---|---|
| Token-by-Token (Model 1) | **0.0122** | **32.09** | **0.47** | **44.21** | **97.80** | 47.21 |
| Jacobi Decoding (Ours) | **0.0122** | 32.08 | **0.47** | **44.21** | 97.50 | **1.20** |

**Table D2:** Ablation on CVT strategy at the same compression ratio. Saliency-guided CVT (Ours) significantly outperforms uniform CVT (Model 2) by concentrating tokens in geometrically complex regions.

| CVT Strategy | $CD_{L2}$ ↓ | PSNR ↑ | ULIP-2 Sim. ↑ | MCS ↑ |
|---|---|---|---|---|
| Uniform CVT (Model 2) | 0.0157 | 29.09 | 0.469 | 40.13 |
| Saliency CVT (Ours) | **0.0122** | **32.08** | **0.474** | **44.21** |

# D Ablation Studies

Our method introduces three key design choices: a two-stage MLLM framework, saliency-guided CVT for adaptive supervoxel construction, and a KNN-based grid-to-supervoxel conversion in SuperVoxelVAE. We ablate each component to validate its effectiveness. All ablation experiments are evaluated on the image-to-3D task using the same 1,000-shape test set. Fig. C2 provides a qualitative comparison, and Tabs. D1 to D3 report quantitative results.

*Two-Stage Design: Jacobi Decoding vs. Token-by-Token Decoding.* Following TRELLIS2 [42], we adopt a two-stage MLLM framework. The primary motivation is that the first stage provides a coarse-grained prediction of the supervoxel structure, which determines the output sequence length and 3D structure for the second stage and thereby enables parallel inference via Jacobi decoding [30]. To validate this design, we construct **Model 1** by removing Jacobi decoding and reverting to standard token-by-token autoregressive inference with KV cache. On the 1,000-shape test set, token-by-token decoding requires an average of 47.21 s for the MLLM stage alone, whereas our Jacobi decoding completes in only 1.20 s. As shown in Tab. D1, the two decoding strategies achieve nearly identical generation quality in the final reconstruction task. This is because we inject noise into the token indices when training the VQ-VAE, which ensures the decoder is robust to imperfect sequence predictions. As a result, Jacobi decoding achieves comparable performance to token-by-token decoding while greatly improving inference speed.

*Saliency-Guided CVT vs. Uniform CVT.* Saliency-guided CVT is a core design of our framework, determining *where* to allocate more tokens for geometrically complex regions while avoiding token waste on smooth surfaces. To validate its

**Table D3:** Ablation on grid-to-supervoxel encoding. We compare our spatially localized encoding method against CraftsMan3D [21] with FSQ quantization (Model 3). GCU, APCU, and $\Delta_{\text{gzip}}$ measure codebook utilization and spatial ordering compressibility (see Appendix A).

| Encoding Strategy | $CD_{L2}$ ↓ | PSNR ↑ | ULIP-2 Sim. ↑ | MCS ↑ | GCU (%) ↑ | APCU ↑ | $\Delta_{\text{gzip}}$ ↑ |
|---|---|---|---|---|---|---|---|
| CraftsMan3D + FSQ (Model 3) | 0.0402 | 20.91 | 0.364 | 21.73 | 99.97 | 0.5649 | 0.0060 |
| Ours | **0.0122** | **32.08** | **0.474** | **44.21** | **100.00** | **0.7727** | **0.1254** |

importance, we construct **Model 2**: we keep the same compression ratio but run CVT on a *uniform* saliency volume (*i.e.*, setting all saliency values to a constant), producing uniformly distributed supervoxel centers (Fig. C2(a)). We then retrain both the SuperVoxelVAE and the MLLM on this uniform partition from scratch. As shown in Tab. D2, replacing saliency-guided CVT with uniform CVT leads to a notable drop in generation quality, particularly for shapes with highly non-uniform geometric complexity (Fig. C2(a,b) vs. (d,e)). This confirms our core insight: geometrically complex regions require a denser token allocation to faithfully preserve fine-grained structures, and saliency-guided partitioning is essential for achieving this under a fixed token budget.

*SuperVoxel-Based Compression vs. Set-based Compression.* SuperVoxelVAE compresses 3D information through a spatially localized pipeline: sparse convolutions first encode the occupancy grid into grid features, KNN Cross Attention [41] ($K$=16) then aggregates neighboring grid features into supervoxel features, and finally FSQ quantizes the supervoxel features into discrete tokens. Crucially, the entire process operates locally—each grid cell only interacts with its neighboring supervoxel centers rather than the entire shape—thereby preserving the correspondence between each token and its spatial position. In contrast, set-based methods such as CraftsMan3D [21] employ global cross-attention to compress shape information into a set of continuous, unordered latent features. This holistic encoding destroys the token–position correspondence, *i.e.*, the sequential structure of tokens is lost, rendering such representations unsuitable for position-ordered autoregressive generation. To validate this analysis, we construct **Model 3** using the same MLLM architecture as ours, but replacing our SuperVoxelVAE with the set-based encoder-decoder from CraftsMan3D [21] and inserting an identical FSQ layer (levels $[9, 9, 5, 5, 5]$) at the bottleneck to quantize the latent features into discrete tokens. We then retrain both the VAE and the MLLM from scratch under the same training configuration. As shown in Tab. D3 and Fig. C2(c), our supervoxel-based encoding significantly outperforms the set-based alternative across all metrics, confirming that the preservation of spatial locality is essential for autoregressive 3D generation.

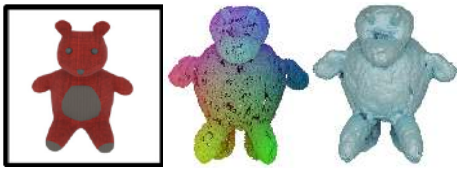
**Table E4:** Maximum training cost of representative methods. "–" means the training time is not reported by the corresponding method.

| Method | TRELLIS [43] | TRELLIS2 [42] | Direct3D-S2 [40] | OctGPT [39] | BrickGPT [27] | Ours |
|---|---|---|---|---|---|---|
| GPU | 64×A100 | 48×H100 | 8×A100 | 8×4090 | 8×A6000 | **8×H100** |
| Time | >7–14 d | – | > 14 d | 7 d | 12 h | **5 d** |

## E Discussion

We note that concurrent Gaussian-based methods [18] have demonstrated strong performance in novel-view synthesis and compact 3D representation. We regard these methods as a parallel direction to our mesh-generation setting rather than direct substitutes. Gaussian representations are primarily optimized for differentiable rendering and view synthesis, and are not specifically designed for extracting clean, high-quality meshes with reliable topology. We therefore examine this issue under our mesh evaluation protocol. We also discuss the training scalability of our method.

*Gaussian Splatting-based methods.* We additionally evaluate GaussianAnything [18], a Gaussian Splatting (GS)-based image-to-shape method, on our image-to-shape test set. Since our target output is an explicit mesh, we convert its predicted Gaussians to meshes through their TSDF reconstruction method before applying the same evaluation protocol. As shown in Fig. E3, the reconstructed meshes exhibit noisy surfaces and weak topology: GaussianAnything obtains $CD_{L_2}$=0.0471, PSNR=20.33, ULIP-2 Sim.=0.363, and an average runtime of 57.92 s, whereas our method obtains $CD_{L_2}$=0.0122, PSNR=32.08, ULIP-2 Sim.=0.474, and 4.60 s on the same test set. These results support our view that GS-based representations are complementary to mesh-generation methods: methods based on this representation can provide texture generation, which is beyond the scope of our method, but heavy surface noise and numerous holes remain a major challenge for them. In the future, exploring token-reduction methods that can support both meshes and textures is an important research direction.



(a) (b) (c)

**Fig. E3:** GaussianAnything outputs on an image-to-shape example. From left to right: (a) the input image, (b) the predicted Gaussian centers, and (c) the extracted (TSDF-reconstructed) mesh.

*Training scalability.* As shown in Tab. E4, SuperVoxelGPT trains with a moderate computational budget among high-resolution 3D generation methods: it uses 8×H100 GPUs for 5 days with our current settings. Although OctGPT and

BrickGPT report lower training cost, they are restricted to specific domains, *i.e.*, specified shape categories or LEGO structures, respectively. In contrast, SuperVoxelGPT targets general high-resolution mesh generation, and its super-voxel tokenizer substantially shortens the 3D token sequence that the generative model needs to learn. This compact tokenization reduces the training burden of the MLLM stage and makes scaling to larger MLLMs and larger datasets more practical. We regard such large-scale training as complementary to our core contribution and leave it as an important direction for future work.

## F Additional Qualitative Results

Fig. F4 presents additional qualitative comparisons on the text-to-3D generation task. We compare our method against BrickGPT [27] ($20^3$ resolution) and OctGPT [39] ($256^3$ resolution) across 10 diverse text prompts with ground-truth (GT) references. BrickGPT produces coarse LEGO-style assemblies that lack geometric detail, while OctGPT generates smoother but still limited shapes. Our method generates significantly more detailed and geometrically faithful shapes at $1024^3$ resolution, closely matching the ground truth.

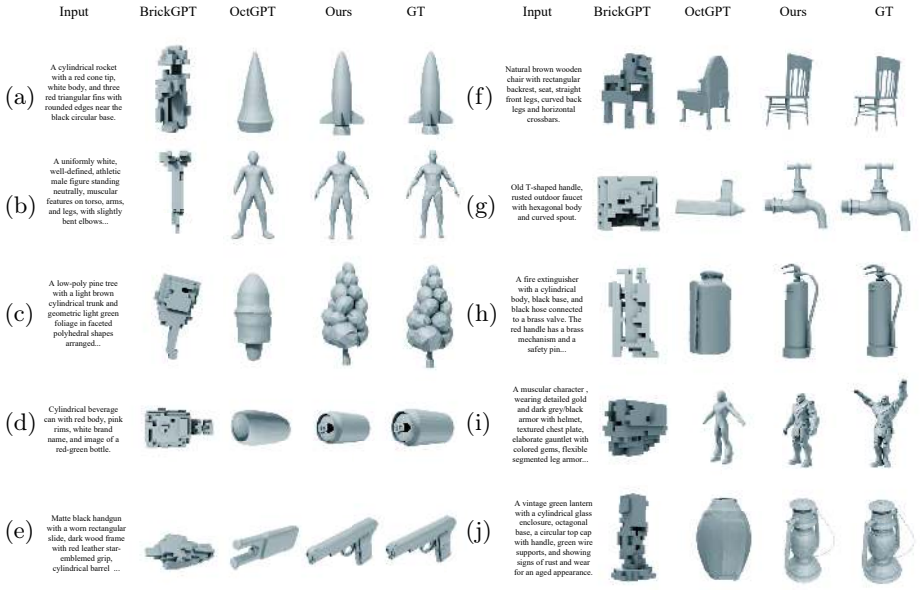



**Fig. F4:** Qualitative comparison on text-to-3D generation. (a)–(j) represent 10 different cases. Each case shows the input text, results from BrickGPT [27] ($20^3$), OctGPT [39] ($256^3$), our SuperVoxelGPT ($1024^3$), and the ground truth.

Fig. F5 shows additional image-to-3D generation results from our method.

**Fig. F5:** Additional image-to-3D results. Each triplet shows, from left to right: the input image, the predicted supervoxel structure, and the decoded 3D mesh. The color definition of saliency is the same as Fig.1 in the main paper.