Split-and-scale Latent 3D Representations

Dorian Chan **Muhammed Kocabas** **Xiaoming Zhao**
Oncel Tuzel **Jen-Hao Rick Chang**
Apple

Abstract

Latent 3D representations have significantly expanded the capabilities of modern shape modeling, enabling compact encoding of 3D scenes and supporting powerful 3D generative models. Despite this progress, current representations struggle to capture the fine details of complex objects, fundamentally limited by the capacity of their latents. However, scaling latent capacity at training time is computationally expensive and often infeasible under memory limits, and naively increasing the number of latents at inference fails to make effective use of the added capacity. In this paper, we address both challenges with *split-and-scale*, a method that effectively expands latent capacity at inference while keeping training cost tractable. Our key observation is that by representing 3D scenes as continuous functions of 3D coordinates, a single pretrained tokenizer can be applied at inference to arbitrarily small sub-regions of a scene, allocating the same latent capacity to a smaller spatial extent and thus capturing far higher detail. We validate this insight by conducting extensive ablations to study the trade-offs between compute, memory, latent capacity, and reconstruction quality, demonstrating our inference-time scaling method achieves quality competitive with much larger models that exceed single-GPU memory limits. We further demonstrate that split-and-scale supports high-quality generative modeling, training an autoregressive 3D generator that performs competitively with state-of-the-art baselines.

1 Introduction

Latent representations have become central to modern 3D shape modeling [52, 6, 48]. By encoding geometry and appearance into a compact set of tokens, they support a wide range of tasks, from shape reconstruction to 3D generation [55, 5, 47, 54, 9, 26, 21, 19, 8, 46, 30].

Capturing the fine details of real-world objects, however, demands far more representational capacity than current latent representations typically provide. For example, consider building a latent representation for a small town, with buildings and trees and signs (Figure 1). Clearly, very large encodings are likely necessary to accurately capture all the detail in such a scene. How can we scale the latent representations of today to handle such a scenario? Since capacity in these token-based architectures is governed by the number of latent tokens, the standard recipe is to scale up the token count. This recipe is far from free: computation grows quadratically and memory linearly with the number of tokens, and existing mitigations like curriculum learning [46] or sophisticated attention patterns [5] only partially help — GPU memory ultimately caps the trainable latent size (*e.g.*, $\sim$8192 tokens for LiTo on an 80GB H100). While certain architectures (*e.g.*, Perceiver-style encoders [52, 5, 30, 21]) permit larger token counts at inference than at training, in practice the resulting quality gains saturate quickly and can even reverse when scaled too far (Figure 3).

In this paper, we propose *split-and-scale* (Figure 1), a technique that enables effective inference-time scaling of latent capacity. Our key observation is that a latent 3D representation parameterized by continuous coordinates can be applied at any spatial scale: the same encoder operates on a full scene or any sub-region, after appropriate normalization of the input coordinates. Critically, normalizing

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

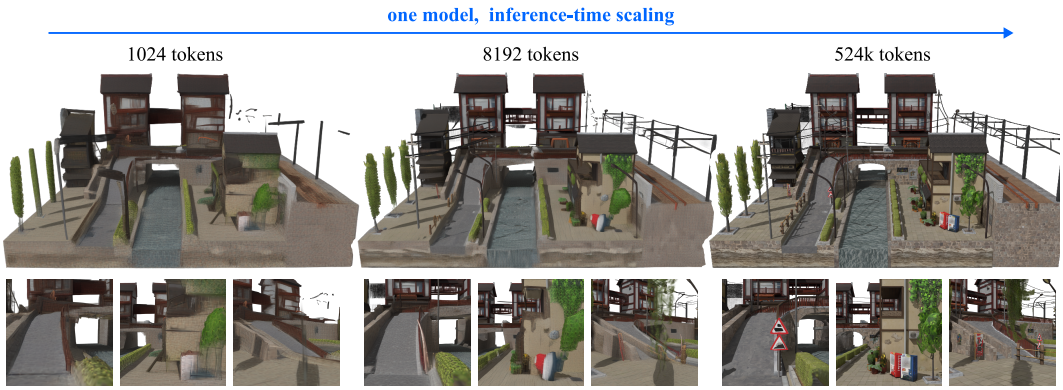


Figure 1: **Split-and-scale** reuses the *same* tokenizer across spatial scales to effectively increase latent capacity at *inference* time. Recommend viewing the figure on monitor, zoomed in. Asset credit [11].

a sub-region of width $1/n$ back to 1 lowers the content frequency seen by the encoder by a factor of n. The same network thus represents detail at $n\times$ finer spatial resolution within each sub-region, and produces proportionally more total tokens across the full scene — effectively increasing latent capacity with no change to the network or its weights. In practice, because 3D scenes are dominated by surfaces, only $O(n^2)$ of the n^3 sub-regions are non-empty — so the total token count grows with surface area rather than volume, keeping inference cost tractable even at fine subdivisions.

Beyond inference, split-and-scale also enables efficient training. Because the encoder operates on normalized sub-regions, it can be trained primarily at coarser subdivisions and still generalize to finer ones at inference, as we will demonstrate empirically. Furthermore, we encode and decode each sub-region independently while computing the reconstruction loss on the assembled full scene. Independent encoding lets us backpropagate gradients through only a small subset of sub-regions per iteration, sharply reducing compute and memory; the full-scene loss simultaneously teaches the network to handle inter-region boundaries. Together, these properties let split-and-scale train comfortably within a single 80GB GPU (*e.g.*, A100 and H100), even at token counts that would otherwise be infeasible to train under standard architectures.

Through extensive reconstruction experiments, we show that split-and-scale consistently outperforms naive inference-time scaling and approaches the quality of much larger baselines that cannot be trained within a single 80GB GPU. Detailed ablations further analyze the design choices behind split-and-scale and characterize the trade-offs between latent capacity, training compute, memory, runtime, and reconstruction quality. We additionally demonstrate the representation's broader applicability by training an autoregressive 3D generative model on top of split-and-scale latents.

In summary, our main contributions are:

- We propose *split-and-scale*, a latent 3D representation that reuses a single tokenizer across spatial scales — enabling memory- and compute-efficient training while supporting much higher-detail reconstruction at inference.
- Through thorough ablation studies, we map the trade-offs between latent capacity, compute, memory, and reconstruction quality, and show that split-and-scale consistently outperforms naive inference-time scaling of the underlying tokenizer.
- We show a potential use case of the representation by training an autoregressive 3D generative model on top of split-and-scale latents.

2 Related work

In recent years, a significant body of work has started to explore the landscape of 3D latent representations. For instance, in the context of geometry, a number of approaches explore the encoding of signed-distance functions or occupancy fields [52, 53, 25, 14, 54, 32, 17], often over some form of sparse grid structure for efficiency [45, 46, 38, 26, 21]. Other works attempt to model 3D surfaces as probability distributions, leveraging generative architectures to sample from them [4, 6, 49].

In recent work, latent representations for both geometry and appearance have seen more traction. A number of approaches have explored how to encode both meshes and their textures. For example, many geometry-only representations have been extended with multi-view texture generation pipelines to add material to generated assets [53, 54, 25, 50]; however, such approaches do not leverage any potential compression in terms of shared information between geometry and appearance. To tackle this aspect, recent work has explored more joint encodings by baking the geometry and material of an input mesh into an intermediate primitive before latent compression. 3DTopia-XL [10] proposed a PrimX primitive that locally encodes both SDF and material parameters, on top of which they train a variational autoencoder (VAE) [20] for further compression. TRELLIS-2 [47] proposed a dual contouring-like O-Voxel representation that they then train a VAE upon. While these mesh-based approaches often produce high-quality, plausible assets, they typically fail to reconstruct the appearance of input objects even under ground truth lighting, as they rely on simplified material models that do not capture the complexities of not only real objects but also many synthetic assets. Furthermore, generative models trained upon these representations do not output the surrounding lighting. This makes such models impractical for tasks like digital tourism or digital memory preservation, where a user would need to guess the lighting that best reproduces their target scene.

Concurrently, another line of research has explored directly encoding multi-view RGBD images as a more accessible proxy of both geometry and appearance. TRELLIS [48] proposed encoding voxelized DINOv2 features [36] into a structured latent representation, where features are encoded over a sparse grid — the representation is then supervised via renderings of decoded 3D Gaussians. SAM 3D [8] built a generation pipeline for a TRELLIS-style representation with a sophisticated post-training process in order to maximize generation quality, improve input image fidelity and handle occlusions. To improve visual realism, LiTo [5] proposed using the surface light field as the input encoder feature, and decoding to view-dependent Gaussians. Relative to TRELLIS, this allowed their method to produce much better highlights and reflections, and also enabled the training of a generative model with much stronger input image fidelity thanks to the continuous nature of the surface light field input. Velox [30] extended the LiTo architecture to 4D scenes, demonstrating strong performance in tracking and generation tasks. Broadly speaking, our work is mostly orthogonal to these directions for improving latent 3D representations, and can potentially be used to scale any of them in a computationally efficient manner. In this paper, we chose to build off LiTo [5] as a base thanks to its strong performance in reconstruction and generation tasks — we discuss more details of how we leverage it in the next section.

More related to our work, a few approaches have begun exploring how objects can be generated in a part-based fashion [43, 9, 27, 15], typically by using different sets of latent tokens for different components. However, these approaches do not consider scaling the overall resolution of the entire object like we do, and require part-based annotations that may limit available 3D training data. Perhaps most similar to our approach, a few world-building approaches have adapted object-level tokenizers and generators, e.g., TRELLIS [48], by applying them at the block level and combining the results to compose entire scenes [13, 56, 51, 22]. While analogous in principle, unlike our work, they are focused on scene generation, and must handle seams between different blocks in ad hoc fashions (*e.g.*, via RePaint [29] or SDEdit [31]) because they reuse pre-existing object tokenizers. In contrast, our work focuses on scaling the quality of object representations by building a tiling-friendly tokenizer and generative model from ground up. Ultimately, we believe our pipelines and trained models may be useful for future such world-building approaches.

3 Method

3.1 Preliminary: LiTo [5]

While split-and-scale can potentially be applied to various existing methods, we focus on LiTo [5], a recent 3D latent representation for geometry and (view-dependent) appearance, as it takes a point cloud with continuous coordinates as input. Given a set of points $\mathcal{X} = \{(\mathbf{x}_i, \mathbf{f}_i)\}_{i=1\ldots n}$, where $\mathbf{x}_i \in \mathbb{R}^3$ is the 3D coordinate and $\mathbf{f}_i \in \mathbb{R}^c$ is the associated feature containing color and view direction, a Perceiver encoder [18, 52] $\mathcal{E}$ takes $\mathcal{X}$ as keys and values and subsamples $\mathcal{X}$ as the initial queries. It then outputs the latent representation $\mathbf{s}$. The latent representation $\mathbf{s}$ is composed of a set of tokens, $\{s_i \in \mathbb{R}^d\}_{i=1\ldots m}$, and can be decoded by different decoders into point clouds and Gaussian

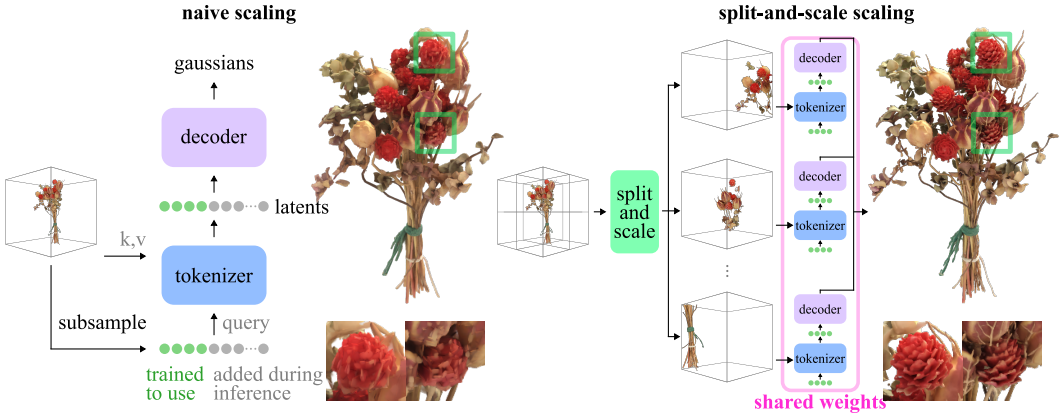


Figure 2: **Naive scaling *vs*. split-and-scale.** (Left) Naive scaling increases number of latents by using more queries to the tokenizer. As the tokenizer is not trained with the higher number of queries, the quality improvement is limited. (Right) Split-and-scale encodes/decodes subregions of the scene with the same tokenizer/decoder and the designed number of queries, effectively increasing latent capacity. Asset credit [42].

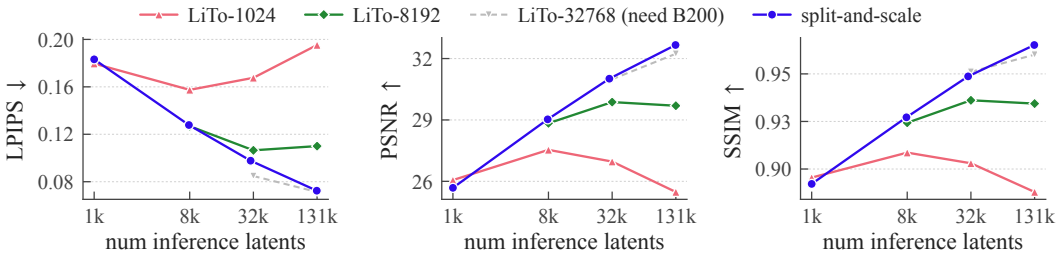


Figure 3: **Quantitative comparison versus naive token count scaling.** We train a split-and-scale model that outputs 1024 latents per voxel. We compare with LiTo-1024, LiTo-8192, LiTo-32768 that use 1024, 8192, 32768 tokens during training, respectively. We also evaluate these models with naive inference-time scaling. Note that training LiTo-32768 requires larger memory than all other methods, and its training takes 2.7× more time. All models are trained to the same number of iterations. As shown, the performance of the proposed method (blue curve) increases consistently as we increase number of tokens during inference, whereas naive scaling quickly hits bottleneck.

splats. At a high level, we have:

$$\{(\mathbf{x}_i, f_i)\}_i \xrightarrow{\mathcal{E}} \mathbf{s} \xrightarrow{\mathcal{D}} \mathbf{y}, \tag{1}$$

where $\mathbf{y}$ can be a point cloud, a mesh, or Gaussian splats. $\mathbf{s}$ is supervised via 1) novel view renderings from the decoded Gaussians and 2) points sampled from a flow-matching decoder [6], given a dataset of RGBD images rendered and points sampled from a collection of synthetic objects. Mesh and voxel occupancy decoders are later trained with a frozen encoder, similar to TRELLIS [48].

To scale LiTo to represent more complex objects, one potential solution is to simply increase the number of latent tokens m (by increasing the number of queries subsampled from $\mathcal{X}$). However, it is often infeasible during *training*, as the compute grows quadratically due to the use of attention and memory linearly (see Table 1). Thereby, existing works [53, 25] often increase number of latents during *inference* by sampling more queries from the input point cloud. However, because the tokenizer is not trained with additional number of queries, the improvement is limited and can negatively impact the quality if used aggressively, as shown in Figure 3.

3.2 Split-and-scale 3D Representations

Can we effectively increase the number of latents during inference while maintaining training efficiency? Our strategy is to keep encoder operating exactly as at training — with the same number

Table 1: **Training compute comparison versus naive token count scaling.** Our split-and-scale recipe induces a similar computational and memory cost to an 8192 token version of LiTo [5], and can be comfortably trained on an 80GB GPU unlike a 32768 token LiTo. Runtime is computed over a single forward + backward pass. FLOPs are computed assuming $2\times$ forward cost for backward.

Method	Peak memory (GB)	Avg runtime (s)	Avg GFLOPs	LPIPS (32768 inf. tokens)
LiTo (8192 tokens)	68.223	1.177	45.51	0.106
Split-and-scale ($\approx$ 14k train. tokens)	73.514	1.322	49.71	0.091
LiTo (32768 tokens)	145.582	3.541	306.23	0.084

of queries m — and invoke the tokenizer multiple times, once per sub-region of the scene (see Figure 2). Concretely, we subdivide the scene (ranging from -1 to 1) into n^3 non-overlapping voxels of size $\frac{2}{n}$ and encode each independently: $s_{\text{split-scale}}[i] = \mathcal{E}(\mathcal{V}_i)$, where $\mathcal{V}_i$ is the content of the ith voxel. To decode, we similarly run the decoder independently for each voxel, and then compute losses over the re-assembled full scene. For efficiency, we skip empty voxels, so the total number of latents grows directly with the number of occupied voxels — due to the surface-dominated nature of 3D scenes, only $\mathcal{O}(n^2)$ of the n^3 voxels are occupied in practice. Output resolution (*e.g.*, number of Gaussians produced by the decoder) also grows correspondingly with n.

However, splitting alone is not enough: as reported in Table 4a ("no scaling"), LPIPS degrades sharply from 0.107 to 0.161 at $n = 4$. The cause is a frequency mismatch. A voxel of width $\frac{1}{n}$ packs the same amount of geometric and textural detail into a spatial extent n times smaller, so the encoder is fed content at substantially higher spatial frequencies than any it was exposed to during training. As n grows, the input distribution drifts out-of-domain and quality collapses.

We correct this mismatch via the space-frequency duality of the Fourier transform: dilating a signal in space by a factor of n compresses its spectrum by the same factor. We therefore shift and rescale each voxel back to the original scene extent $[-1, 1]^3$ with a normalization $\mathcal{M}_i$ before encoding, and invert it at the decoder:

$$s_{\text{split-scale}}[i] = \mathcal{E}(\mathcal{M}_i(\mathcal{V}_i)), \qquad \mathcal{D}_{\text{split-scale}}(s_{\text{split-scale}}) = \bigcup_i \mathcal{M}_i^{-1}(\mathcal{D}(s_{\text{split-scale}}[i]))\,. \tag{2}$$

Note that the transformation $\mathcal{M}_i$ only depends on the size and location of the voxel, not its content. Under this rescaling, every voxel presents the encoder with content whose frequency range matches what it saw at training, regardless of voxel size.

Interestingly, the same network captures $n\times$ finer detail inside each voxel while reusing its pretrained weights, effectively expanding total latent capacity at inference without any retraining. By choosing n, we directly control the level of detail.

3.2.1 Computational strengths of split-and-scale

Strictly speaking, a traditional train-time-scaling approach for n tokens (*i.e.*, directly training a tokenizer to use an increased number of tokens) should outperform a split-and-scale representation at n tokens. Under split-and-scale, tokens are partitioned across subregions and restricted to intra-region attention, while tokens in a traditional approach can attend globally across the entire object allowing for more potential compression. However, when implemented appropriately, the proposed method has important computational strengths, as we will discuss below:

Coarse-scale training. As the encoder and decoder functions are fixed across all scales, they can be trained at coarser scales only, but still generalize to finer subdivisions at inference. The smaller number of latent tokens at coarser scales allows for much faster training and reduced memory usage. In practice, slight distributional differences can be introduced due to subdividing the scene. First, at $n > 1$, geometry can cross different voxels, and the latent encoder/decoder must therefore be able to handle inter-voxel boundaries. Second, if the available number of points is fixed for an object regardless of n, the density of $\mathcal{X}$ can change with different scales, *i.e.*, splitting an object into n^3 voxels also splits input points into $1/n^3$ per voxel. Both can be easily mitigated by training with some iterations of $n \geq 2$ and training coarser levels with a random number of input points.

Voxel subset training. The encoder/decoder are applied independently to each subvoxel of the object. Thus, rather than training the entire object as would be required under standard token scaling,

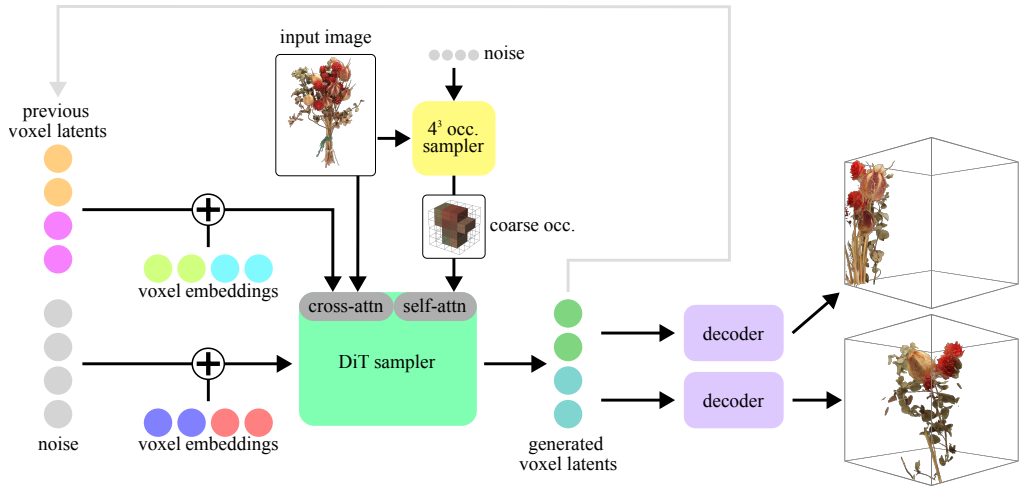


Figure 4: **Visualization of our proposed generation procedure.** To reduce memory requirements, we propose autoregressive voxel-based generation with flow matching-based sampling and cross-attention conditioning. Given an input image, we start by first generating coarse 4^3 occupancy. For the occupied voxels, we then autoregressively sample voxel latents conditioned on previously-generated latents, until all voxels are filled. Asset credit [42].

we can train just a subset of voxels, allowing for significantly reduced memory usage even when using many latent tokens. Note the forward pass still includes all active voxels (but without storing their activations), as we only have RGB renderings of full objects (not individual voxels).

Implementation and ablation. We build off LiTo to implement our encoder and decoder; see Section A.2 for details. Following this above regimen, training comfortably fits within the memory limits of an 80GB GPU (Table 1) — an important threshold given their relative availability (*e.g.*, A100 and H100 GPUs) compared to GPUs with larger memory sizes. Simultaneously, compute time and FLOP usage remain comparable to a standard LiTo architecture. However, performance scales well even up to 131072 total latent tokens ($n \approx 8$), as visualized in Figure 3.

We ablate our design choices in downscaled experiments in Table 4a, where we train tokenizers without various components of our final recipe for 44k iterations. Voxel normalization $\mathcal{M}$ and training with both $n = 1$ and $n = 2$ can be seen to be crucial for performance, while training with $n = 4$ and random point counts yield smaller improvements.

3.3 Autoregressive generation

While split-and-scale provides a computationally-efficient avenue for training latent representations with high number of latents, training a generative model for these larger latents is not straightforward. For instance, $n = 4$ and 1024 tokens per voxel translates to approximately 32768 total tokens on average — in our experiments, naively scaling the LiTo diffusion transformer to generate 32768 tokens induces a peak training memory usage of nearly 160 GBs, far outpacing the capacity of most commonly available GPUs. To tackle this challenge, we propose an autoregressive voxel-based formulation with flow matching-based sampling [24] and cross-attention conditioning, as visualized in Figure 4. In particular, we train a flow-matching model [28] based on a diffusion transformer [37] that models the following distribution: $P(\mathbf{v}_{\text{target}} \mid \mathbf{v}_{\text{context}}, O, I)$, where $\mathbf{v}_{\text{target}}$ denotes the latents of voxels to be generated, $\mathbf{v}_{\text{context}}$ the already-generated voxel latents, O the n^3 coarse occupancy (as empty voxels are not reconstructed in split-and-scale), and I the input image. We use self attention between noisy $\mathbf{v}_{\text{target}}$ tokens with cross attention to $\mathbf{v}_{\text{context}}$, mitigating the memory costs of larger latent generation. To apply other conditioning, we use cross attention with I and self attention with the fewer token O.

Implementation and ablation. We train this autoregressive model for split-and-scale latents from $n = 4$ with 1024 tokens per voxel, using randomly selected context and target voxels. To minimize

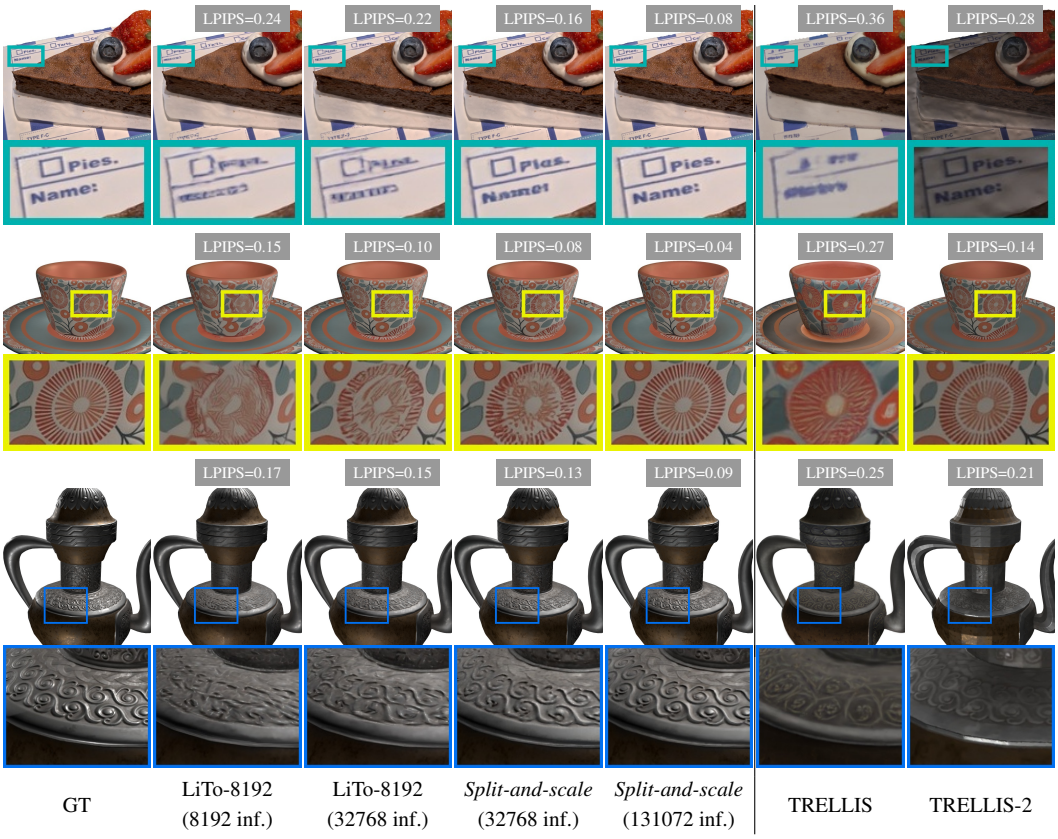


Figure 5: **Qualitative reconstruction comparison.** For comparison, we visualize a LiTo model trained with 8192 tokens ("LiTo-8192"), that can be trained under similar compute limits as our split-and-scale model. Note that the metrics for TRELLIS and TRELLIS-2 are quite poor due to color shift. Asset credits [35, 33, 2].

the overall number of autoregressive steps, we sample 8 voxels at a time, resulting in 4 flow-matching samplings per object on average. We train this model for 450k iterations with a batch size of 256. Note that training this model can again be performed on H100 80GB GPUs — please see the appendix for more details on our network architecture, which mostly follows that of LiTo [5].

In Table 4b, to evaluate our proposed autoregressive method, we also implement a naive approach that directly generates $n = 4$ split-and-scale latents with a DiT, but with reduced hidden dimension such that the model can be trained on an 80GB GPU. Our autoregressive approach can be seen to comfortably outperform this baseline when both models have been trained for 135k iterations. For an equivalent amount of training time, our autoregressive model can also be seen to be competitive with a DiT trained to directly generate 32768 token LiTo latents, even though neither the representation nor the generative model fit within 80GB memory.

We also ablate potential inference voxel sampling orders in Table 4b. Generating voxels in random orders versus in raster order away from the camera seems to behave comparably — we use raster order for the results shown in this paper. To leverage this model in practice, we also train a small flow-matching model that generates the $4 \times 4 \times 4$ occupancy of an object, conditioned on an input image. We ablate the impact of this additional generation step on overall performance in Table 4b.

4 Results

4.1 Reconstruction

Dataset. We build our split-and-scale representation off TRELLIS's [48] training subset of Objaverse-XL [12], which we split into 10% validation, 10% test and 80% training sets follow-

Table 2: **Quantitative reconstruction comparison with other baselines.**

Method	Infer. latent size	LPIPS ↓	Chamfer (mesh) ↓	Chamfer (flow) ↓
LiTo-8192	8192×32	0.124	-	2.87×10^{-3}
LiTo-8192	32768×32	0.106	-	2.74×10^{-3}
TRELLIS	$\approx 20\text{k} \times 11$	0.230	1.28×10^{-2}	-
TRELLIS-2	$\approx 10\text{k} \times 32$	0.184	1.53×10^{-2}	-
Split-and-scale	8192×32	0.124	1.70×10^{-3}	2.46×10^{-3}
Split-and-scale	32768×32	0.091	1.57×10^{-3}	2.09×10^{-3}

ing LiTo [5]. For quantitative evaluation, we randomly select 200 objects from the 1000 test set objects with highest aesthetic score as computed by TRELLIS.[1] We follow a simplified LiTo pipeline and render 1036×1036 resolution images under random lighting. To get the input point cloud, we backproject 150 views rendered on a sphere and randomly select 16 million points. We supervise the Gaussian decoder with 100 images rendered from random views.

Results. For quantitative evaluation, we first compare our split-and-scale model with baseline LiTo models [5] trained for a variety of different latent counts, under both equal iteration (Figure 3) and equal compute time (Figure 9(a)) settings. In both cases, it can be seen that for an equivalent total latent count, split-and-scale is competitive with a LiTo model trained with that total latent count. Additionally, split-and-scale outperforms the comparable LiTo model when scaled to use more tokens, *e.g.*, split-and-scale with $4 \times 4 \times 4$ voxels (≈ 32768 tokens) performs better than a LiTo model trained to use 8192 tokens, even when that LiTo model is applied with more latents at inference. For completeness, we also include a 32768 token variant of LiTo (LiTo-32768), that notably cannot be trained on an H100 unlike the other methods shown. Nevertheless, split-and-scale still performs comparably. We also evaluate the geometry quality via Chamfer distance between ground truth point clouds and point clouds sampled from the flow-matching decoder of LiTo and split-and-scale. As shown in Figure 8, point clouds sampled from split-and-scale can be seen to significantly outperform those sampled from comparable LiTo models.

In Table 2, we compare with other baselines more broadly (we include our LiTo implementation trained with 8192 tokens, *i.e.*, "LiTo-8192", for completeness). To compare with the geometry of mesh-based approaches, *e.g.* TRELLIS [48] or TRELLIS-2 [47], we also train a mesh decoder from our split-and-scale representation (network architecture discussed in appendix). Our approach can be seen to be broadly competitive across both appearance and geometry.

For qualitative evaluation, in Figure 5 and the supplemental website, we visualize the results of split-and-scale relative to LiTo-8192 as well as recent baselines [48, 47]. As can be seen, split-and-scale performs well in regions of high detail compared to a computationally-similar LiTo model trained with 8192 tokens. Simultaneously, it lacks the color drift issues of TRELLIS [48] and TRELLIS-2 [47][2]. We also show a comparison with the upper-bound LiTo-32768 in Figure 10.

4.2 Generation

Dataset. We use the same dataset as reconstruction, but we render additional conditioning views at random distances away from the object, with field-of-view set such that the object takes up the majority of the frame. We use two sets of cameras for evaluation. To capture the global structure of generated objects, we use "horizon" cameras sampled from a circle centered around the object, with a fixed field-of-view that captures the entire object silhouette. To measure the quality of finer details, we also use random poses that are randomly placed closer to the object than the horizon views. Our quantitative metrics are computed over the 2000 test set objects with highest aesthetic scores.

Results. Figure 6 shows example results from our proposed autoregressive generation. More results can be found in the supplemental website. In general, we observe that our proposed method generates output assets with finer details than a 8192-token LiTo model ("LiTo-8192"), thanks to the increased capacity of the latent space. We also compare to generations from TRELLIS [48] and

[1] This means our test set is TRELLIS's and TRELLIS-2's training set.

[2] This color shift occurs despite rendering TRELLIS-2's textured mesh outputs with ground truth lighting (Section A.3).

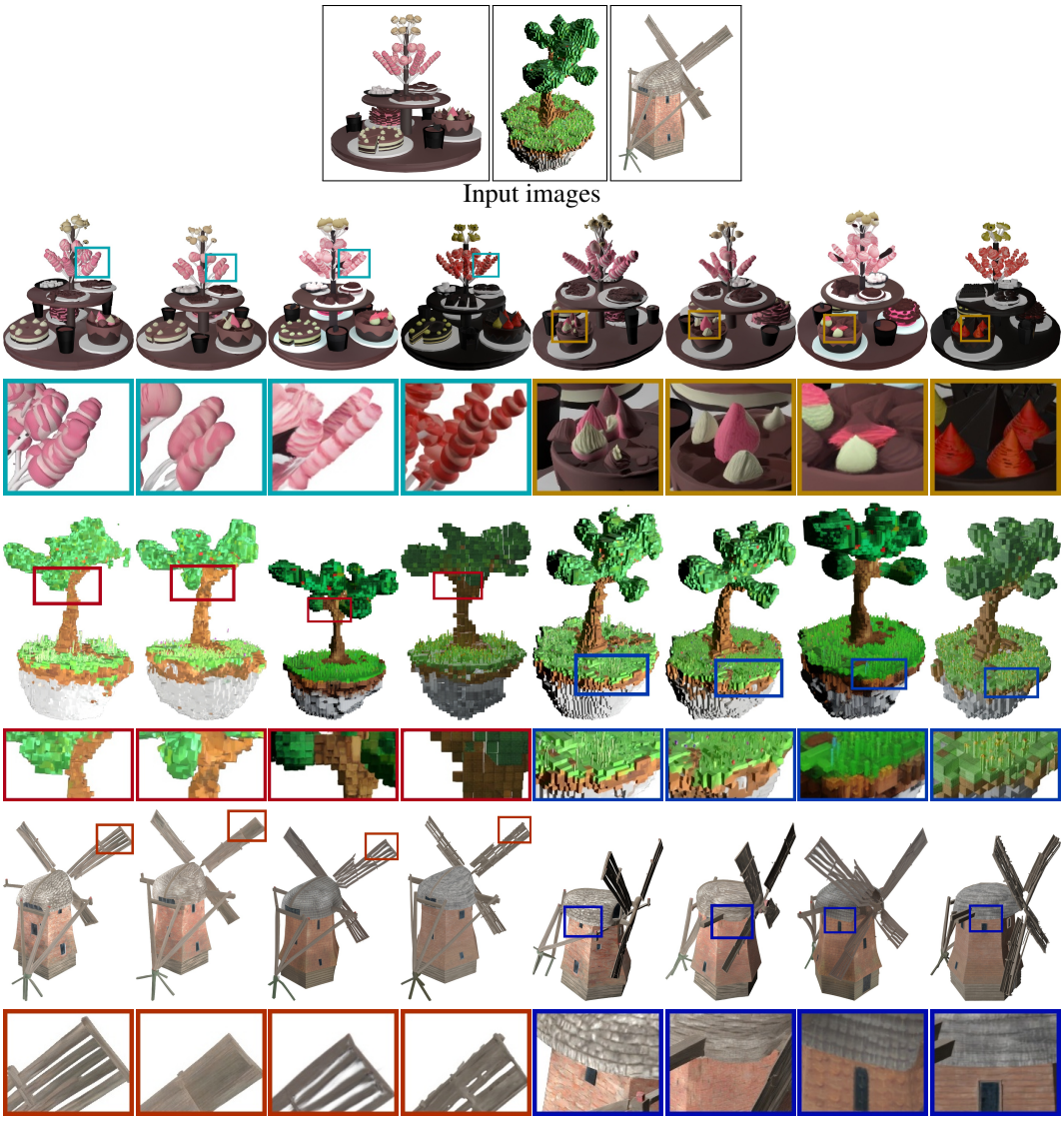


Figure 6: **Qualitative generation comparison.** A generative model trained on split-and-scale can reproduce fine details. In contrast, competitor methods suffer from either poor detail or poor material estimation. Asset credits [44, 40, 3].

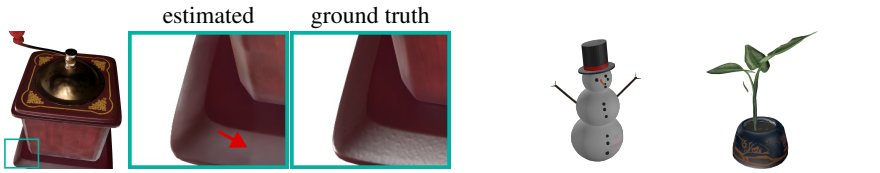
TRELLIS-2 [47][3] — our method seems to produce much less color shift relative to the input images compared to both methods. However, as shown in Figure 7(b), we sometimes observe structural flaws in the output assets from our generative model, *e.g.*, duplicated features or asymmetry. We believe such degradations are perhaps expected due to our autoregressive formulation, where later voxel generations cannot revise earlier ones.

For quantitative evaluation, Table 3 shows the performance of our proposed autoregressive voxel generation method compared to existing 3D generation approaches. Our model can be seen to outperform TRELLIS and TRELLIS-2 baselines, while being competitive with LiTo-8192. In particular, our method seems to be roughly on par with LiTo-8192 in terms of input view metrics, while it begins to outperform for novel views.

[3]We render TRELLIS-2's textured mesh outputs with ground truth lighting (Section A.3).

Table 3: **Quantitative generation comparison.** CFG scale is 3.

	Input view		Novel view (horizon)		Novel view (random)	
Method	**FID**↓	**KID**$_{10^{-3}}$↓	**FID**↓	**KID**$_{10^{-3}}$↓	**FID**↓	**KID**$_{10^{-3}}$↓
LiTo-8192	16.78	0.42	6.33	0.97	7.95	1.77
TRELLIS	38.09	3.94	10.32	2.17	15.35	5.02
TRELLIS-2	42.84	3.76	10.39	1.81	14.71	3.79
Split-and-scale	17.98	0.23	6.26	0.83	7.70	1.83



(a) Reconstruction seams (b) Autoregressive structural flaws

Figure 7: **Failure cases. (a)** In tokenization, because each voxel is independently processed, subtle seams can appear in areas with significant view-dependent appearance. **(b)** Because our proposed generation model is autoregressive, sometimes it can experience the structural flaws common to such models. This manifests in the form of repeated geometry or poor symmetry. Asset credits [1, 41, 34].

5 Limitations and future work

While split-and-scale is a promising approach for scalable latent representations, it is not without its limitations. First, because every subregion is encoded and decoded independently, subtle seam artifacts can arise, particularly in areas with significant view dependence (Figure 7(a)). Visually, these artifacts appear as highlights that are subtly cut off at voxel boundaries. They can be mitigated in a compute-efficient way by training a decoder that conditions on neighboring latents. Second, the current implementation of split-and-scale subdivides objects into voxels of equal size for simplicity, regardless of the content complexity. While our proposed adaptive latent allocation (see Section A.2) can help mitigate any information asymmetry across different voxels, this regular division fundamentally increases the complexity of the distribution the split-and-scale tokenizer must be able to encode. Intelligent divisions, such as K-d tree or part-based approaches, could improve performance, but may introduce extra complexity for generative modeling or limit available training data. Third, in our current split-and-scale implementation, the number of output Gaussians grows with the subdivision level (*i.e.*, each decoder run produces a full-sized set of splats). For a large n, this can generate too many Gaussians to render efficiently in practice. A smarter implementation where the decoder can output a variable number of splats depending on n would mitigate this problem.

Our training recipe for autoregressive generation is also quite simplistic, and can be further improved. For example, a common challenge with autoregressive models is that later generations cannot revise earlier ones, occasionally leading to global structural flaws (Figure 7(b)). One scenario that occasionally appears is where the object looks reasonable from the input view, but occluded regions show slight asymmetries or floating geometry relative to the input. A more complex design that enables the autoregressive model to generate global structures early in the sampling sequence could help remedy such issues, *e.g.*, following recent work in fractal [23] and discrete diffusion models. Another common challenge with such models is the requisite teacher forcing during training. Adapting recent techniques from other domains that attempt to deal with this training-inference gap [16, 7] could improve quality.

Acknowledgments

We thank Byeongjoo Ahn, Raviteja Vemulapalli and Anagh Malik for their detailed feedback. All funding for this work came from Apple.

Author contributions

All authors were involved in project planning and progress discussions.

Scope. Rick Chang and Oncel Tuzel set the overarching direction of the work.

Method. Rick Chang formulated the main split-and-scale methodology, and Dorian Chan formalized the computational benefits. Dorian Chan conceived the autoregressive generation architecture. Muhammed Kocabas, Xiaoming Zhao and Oncel Tuzel provided idea, design and architecture feedback.

Writing. Dorian Chan wrote the initial draft of the paper, which Rick Chang and Muhammed Kocabas later refined. Oncel Tuzel critiqued the manuscript.

Model training. Dorian Chan led model training for both the tokenizer and the generative model. Rick Chang and Xiaoming Zhao developed the original training framework.

Experiments. Dorian Chan developed the evaluation framework for all ablations. Muhammed Kocabas implemented the comparisons with TRELLIS and TRELLIS-2. Dorian Chan and Rick Chang created all qualitative figures. Dorian Chan created the website.

References

[1] 3Dji. Coffee grinder, 2025. Licensed under Creative Commons Attribution.

[2] andrea.notarstefano. Tibetan teapot, 2020. Licensed under Creative Commons Attribution.

[3] bobrik00. Windmill, 2020. Licensed under Creative Commons Attribution.

[4] Ruojin Cai, Guandao Yang, Hadar Averbuch-Elor, Zekun Hao, Serge Belongie, Noah Snavely, and Bharath Hariharan. Learning gradient fields for shape generation. In *ECCV*, pages 364–381. Springer, 2020.

[5] Jen-Hao Chang, Xiaoming Zhao, Dorian Chan, and Oncel Tuzel. Lito: Surface light field tokenization. In *International Conference on Learning Representations*, volume 2026, pages 100644–100671, 2026.

[6] Jen-Hao Rick Chang, Yuyang Wang, Miguel Angel Bautista Martin, Jiatao Gu, Xiaoming Zhao, Josh Susskind, and Oncel Tuzel. 3d shape tokenization via latent flow matching. *arXiv preprint arXiv:2412.15618*, 2024.

[7] Boyuan Chen, Diego Martí Monsó, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. *Advances in Neural Information Processing Systems*, 37:24081–24125, 2024.

[8] Xingyu Chen, Fu-Jen Chu, Pierre Gleize, Kevin J Liang, Alexander Sax, Hao Tang, Weiyao Wang, Michelle Guo, Thibaut Hardin, Xiang Li, et al. SAM 3d: 3dfy anything in images. *arXiv preprint arXiv:2511.16624*, 2025.

[9] Yiwen Chen, Zhihao Li, Yikai Wang, Hu Zhang, Qin Li, Chi Zhang, and Guosheng Lin. Ultra3d: Efficient and high-fidelity 3d generation with part attention. *arXiv preprint arXiv:2507.17745*, 2025.

[10] Zhaoxi Chen, Jiaxiang Tang, Yuhao Dong, Ziang Cao, Fangzhou Hong, Yushi Lan, Tengfei Wang, Haozhe Xie, Tong Wu, Shunsuke Saito, et al. 3dtopia-xl: Scaling high-quality 3d asset generation via primitive diffusion. In *CVPR*, pages 26576–26586, 2025.

[11] Sooi Cherchye. City scene tokyo, 2016. Licensed under Creative Commons Attribution.

[12] Matt Deitke, Ruoshi Liu, Matthew Wallingford, Huong Ngo, Oscar Michel, Aditya Kusupati, Alan Fan, Christian Laforte, Vikram Voleti, Samir Yitzhak Gadre, et al. Objaverse-xl: A universe of 10m+ 3d objects. *NeurIPS*, 36:35799–35813, 2023.

[13] Paul Engstler, Aleksandar Shtedritski, Iro Laina, Christian Rupprecht, and Andrea Vedaldi. Syncity: Training-free generation of 3d worlds. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 27585–27595. IEEE, 2025.

[14] Xianglong He, Zi-Xin Zou, Chia-Hao Chen, Yuan-Chen Guo, Ding Liang, Chun Yuan, Wanli Ouyang, Yan-Pei Cao, and Yangguang Li. Sparseflex: High-resolution and arbitrary-topology 3d shape modeling. *arXiv preprint arXiv:2503.21732*, 2025.

[15] Xufan He, Yushuang Wu, Xiaoyang Guo, Chongjie Ye, Jiaqing Zhou, Tianlei Hu, Xiaoguang Han, and Dong Du. Unipart: Part-level 3d generation with unified 3d geom-seg latents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 34227–34236, 2026.

[16] Xun Huang, Zhengqi Li, Guande He, Mingyuan Zhou, and Eli Shechtman. Self forcing: Bridging the train-test gap in autoregressive video diffusion. *arXiv preprint arXiv:2506.08009*, 2025.

[17] Ka-Hei Hui, Aditya Sanghi, Arianna Rampini, Kamal Rahimi Malekshan, Zhengzhe Liu, Hooman Shayani, and Chi-Wing Fu. Make-a-shape: a ten-million-scale 3d shape model. In *ICML*, 2024.

[18] Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, et al. Perceiver IO: a General Architecture for Structured Inputs & Outputs. In *ICLR*, 2022.

[19] Tanghui Jia, Dongyu Yan, Dehao Hao, Yang Li, Kaiyi Zhang, Xianyi He, Lanjiong Li, Yuhan Wang, Jinnan Chen, Lutao Jiang, et al. UltraShape 1.0: High-fidelity 3d shape generation via scalable geometric refinement. *arXiv preprint arXiv:2512.21185*, 2025.

[20] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[21] Zeqiang Lai, Yunfei Zhao, Zibo Zhao, Haolin Liu, Qingxiang Lin, Jingwei Huang, Chunchao Guo, and Xiangyu Yue. LATTICE: Democratize high-fidelity 3d generation at scale. *arXiv preprint arXiv:2512.03052*, 2025.

[22] Sikuang Li, Chen Yang, Jiemin Fang, Taoran Yi, Jia Lu, Jiazhong Cen, Lingxi Xie, Wei Shen, and Qi Tian. Worldgrow: Generating infinite 3d world. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 6433–6441, 2026.

[23] Tianhong Li, Qinyi Sun, Lijie Fan, and Kaiming He. Fractal generative models. *arXiv preprint arXiv:2502.17437*, 2025.

[24] Tianhong Li, Yonglong Tian, He Li, Mingyang Deng, and Kaiming He. Autoregressive image generation without vector quantization. *Advances in Neural Information Processing Systems*, 37:56424–56445, 2024.

[25] Yangguang Li, Zi-Xin Zou, Zexiang Liu, Dehu Wang, Yuan Liang, Zhipeng Yu, Xingchao Liu, Yuan-Chen Guo, Ding Liang, Wanli Ouyang, et al. TripoSG: High-Fidelity 3D Shape Synthesis using Large-Scale Rectified Flow Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2025.

[26] Zhihao Li, Yufei Wang, Heliang Zheng, Yihao Luo, and Bihan Wen. Sparc3d: Sparse representation and construction for high-resolution 3d shapes modeling. *arXiv preprint arXiv:2505.14521*, 2025.

[27] Yuchen Lin, Chenguo Lin, Panwang Pan, Honglei Yan, Feng Yiqiang, Yadong Mu, and Katerina Fragkiadaki. Partcrafter: Structured 3d mesh generation via compositional latent diffusion transformers. *Advances in Neural Information Processing Systems*, 38:35387–35415, 2025.

[28] Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

[29] Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. Repaint: Inpainting using denoising diffusion probabilistic models. In *2022 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pages 11451–11461. IEEE, 2022.

[30] Anagh Malik, Dorian Chan, Xiaoming Zhao, David B Lindell, Oncel Tuzel, and Jen-Hao Rick Chang. Velox: Learning representations of 4d geometry and appearance. *arXiv preprint arXiv:2605.04527*, 2026.

[31] Chenlin Meng, Yutong He, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. Sdedit: Guided image synthesis and editing with stochastic differential equations. *arXiv preprint arXiv:2108.01073*, 2021.

[32] Quan Meng, Lei Li, Matthias Nießner, and Angela Dai. LT3SD: Latent trees for 3d scene diffusion. In *CVPR*, pages 650–660, 2025.

[33] MrsM. Tazza e piattino, 2020. Licensed under Creative Commons Attribution-NonCommercial.

[34] Nikkip. Tropical pot plant, 2021. Licensed under Creative Commons Attribution.

[35] okajima. [chocolate cake], 2019. Licensed under Creative Commons Attribution.

[36] Maxime Oquab, Timothée Darcet, Theo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Russell Howes, Po-Yao Huang, Hu Xu, Vasu Sharma, Shang-Wen Li, Wojciech Galuba, Mike Rabbat, Mido Assran, Nicolas Ballas, Gabriel Synnaeve, Ishan Misra, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

[37] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4195–4205, 2023.

[38] Xuanchi Ren, Jiahui Huang, Xiaohui Zeng, Ken Museth, Sanja Fidler, and Francis Williams. XCube: Large-Scale 3D Generative Modeling using Sparse Voxel Hierarchies. In *CVPR*, 2024.

[39] Tianchang Shen, Jacob Munkberg, Jon Hasselgren, Kangxue Yin, Zian Wang, Wenzheng Chen, Zan Gojcic, Sanja Fidler, Nicholas Sharp, and Jun Gao. Flexible isosurface extraction for gradient-based mesh optimization. *ACM Transactions on Graphics (ToG)*, 42(4):1–16, 2023.

[40] slasmir. Procedural voxel tree, 2020. Licensed under Creative Commons Attribution.

[41] Sohaibakr. Snowman, 2019. Licensed under Creative Commons Attribution.

[42] Second Studio. Dry flower, 2022. Licensed under Creative Commons Attribution.

[43] Jiaxiang Tang, Ruijie Lu, Max Li, Zekun Hao, Xuan Li, Fangyin Wei, Shuran Song, Gang Zeng, Ming-Yu Liu, and Tsung-Yi Lin. Efficient part-level 3d object generation via dual volume packing. *Advances in Neural Information Processing Systems*, 38:27115–27137, 2025.

[44] wribeast. Day 2/ sweet, 2020. Licensed under Creative Commons Attribution.

[45] Shuang Wu, Youtian Lin, Feihu Zhang, Yifei Zeng, Jingxi Xu, Philip Torr, Xun Cao, and Yao Yao. Direct3D: Scalable Image-to-3D Generation via 3D Latent Diffusion Transformer. In *NeurIPS*, 2024.

[46] Shuang Wu, Youtian Lin, Feihu Zhang, Yifei Zeng, Yikang Yang, Yajie Bao, Jiachen Qian, Siyu Zhu, Xun Cao, Philip Torr, et al. Direct3d-s2: Gigascale 3d generation made easy with spatial sparse attention. *arXiv preprint arXiv:2505.17412*, 2025.

[47] Jianfeng Xiang, Xiaoxue Chen, Sicheng Xu, Ruicheng Wang, Zelong Lv, Yu Deng, Hongyuan Zhu, Yue Dong, Hao Zhao, Nicholas Jing Yuan, and Jiaolong Yang. Native and compact structured latents for 3d generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14419–14429, 2026.

[48] Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong, and Jiaolong Yang. Structured 3d latents for scalable and versatile 3d generation. In *CVPR*, pages 21469–21480, 2025.

[49] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. PointFlow: 3D point cloud generation with continuous normalizing flows. In *ICCV*, 2019.

[50] Jiayu Yang, Taizhang Shang, Weixuan Sun, Xibin Song, Ziang Chen, Senbo Wang, Shenzhou Chen, Weizhe Liu, Hongdong Li, and Pan Ji. Pandora3D: A Comprehensive Framework for High-Quality 3D Shape and Texture Generation. *arXiv preprint arXiv:2502.14247*, 2025.

[51] Seungwoo Yoon, Jinmo Kim, and Jaesik Park. Extend3d: Town-scale 3d generation. *arXiv preprint arXiv:2603.29387*, 2026.

[52] Biao Zhang, Jiapeng Tang, Matthias Nießner, and Peter Wonka. 3DShape2VecSet: A 3D Shape Representation for Neural Fields and Generative Diffusion Models. *TOG*, 2023.

[53] Longwen Zhang, Ziyu Wang, Qixuan Zhang, Qiwei Qiu, Anqi Pang, Haoran Jiang, Wei Yang, Lan Xu, and Jingyi Yu. CLAY: A Controllable Large-scale Generative Model for Creating High-quality 3D Assets. *TOG*, 2024.

[54] Zibo Zhao, Zeqiang Lai, Qingxiang Lin, Yunfei Zhao, Haolin Liu, Shuhui Yang, Yifei Feng, Mingxin Yang, Sheng Zhang, Xianghui Yang, et al. Hunyuan3D 2.0: Scaling Diffusion Models for High Resolution Textured 3D Assets Generation. *arXiv preprint arXiv:2501.12202*, 2025.

[55] Zibo Zhao, Wen Liu, Xin Chen, Xianfang Zeng, Rui Wang, Pei Cheng, BIN FU, Tao Chen, Gang YU, and Shenghua Gao. Michelangelo: Conditional 3D Shape Generation based on Shape-Image-Text Aligned Latent Representation. In *NeurIPS*, 2023.

[56] Kaizhi Zheng, Ruijian Zhang, Jing Gu, Jie Yang, and Xin Eric Wang. Constructing a 3d town from a single image. *arXiv preprint arXiv:2505.15765*, 2025.

A Appendix

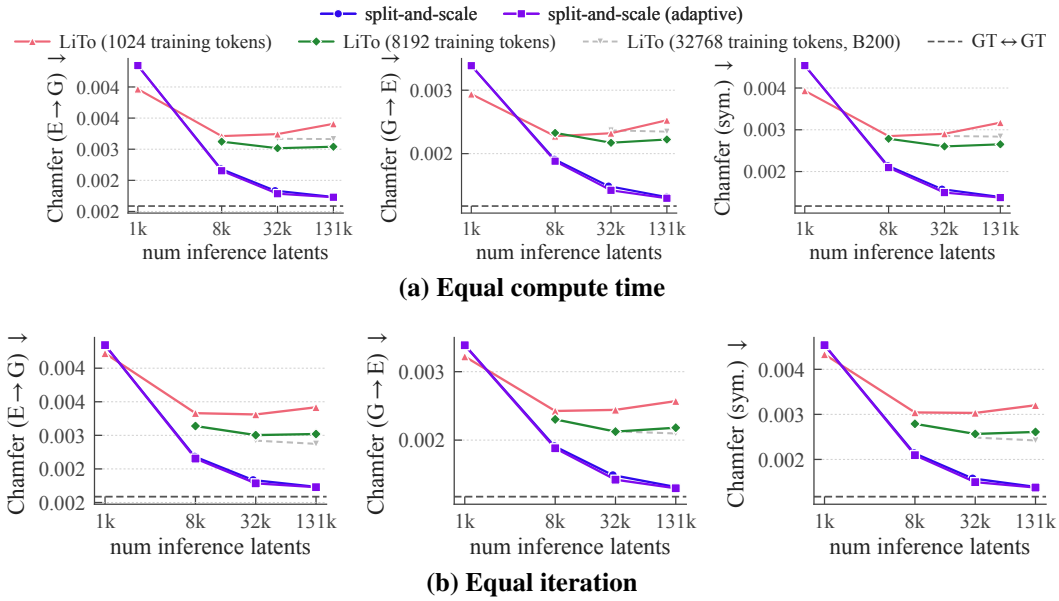


Figure 8: **Quantitative geometry comparison versus naive token count scaling.** We compare the quality of sampled point clouds from split-and-scale versus comparable LiTo models [5].

Table 4: **Ablation study.** We ablate our proposed designs for reconstruction as well as generation. In (b), both LiTo-32768 variants cannot be trained on an 80GB GPU, for both the representation and generative model.

	$n=4$, ≈ 32768 tokens		$n=8$, ≈ 131072 tokens	
Recipe	**LPIPS**	**Chamfer**	**LPIPS**	**Chamfer**
Full recipe	0.107	2.19×10^{-3}	0.081	2.05×10^{-3}
no random point count	0.110	2.23×10^{-3}	0.092	2.07×10^{-3}
no scaling	0.161	2.92×10^{-3}	0.164	2.84×10^{-3}
Training scales:				
Only $n=1$ and $n=2$ training	0.113	2.21×10^{-3}	0.088	2.06×10^{-3}
Only $n=1$ training	0.145	3.34×10^{-3}	0.133	2.71×10^{-3}
Only $n=4$ training	0.135	2.29×10^{-3}	0.101	2.08×10^{-3}

(a) Reconstruction ablation (44k iterations)

	Input view		Novel view (horizon)		Novel view (random)	
	FID↓	**KID↓**	**FID↓**	**KID↓**	**FID↓**	**KID↓**
Architecture (135k iterations):						
Autoregressive	21.56	6.34×10^{-4}	8.63	1.81×10^{-3}	11.67	3.80×10^{-3}
Direct with smaller DiT	27.67	7.35×10^{-4}	21.89	8.65×10^{-3}	32.63	1.58×10^{-2}
LiTo-32768 DiT, equal time	21.95	7.35×10^{-4}	8.76	1.62×10^{-3}	11.38	3.44×10^{-3}
LiTo-32768 DiT, equal iter	18.50	2.30×10^{-4}	6.53	8.29×10^{-4}	7.95	1.77×10^{-3}
Generation order:						
Random	18.07	2.70×10^{-4}	6.18	8.26×10^{-4}	7.89	1.94×10^{-3}
Raster	17.87	2.45×10^{-4}	6.21	8.21×10^{-4}	7.72	1.81×10^{-3}
Occupancy impact:						
GT occupancy	17.95	2.48×10^{-4}	6.15	7.78×10^{-4}	7.75	1.83×10^{-3}
Generated occupancy	17.98	2.26×10^{-4}	6.26	8.29×10^{-4}	7.70	1.83×10^{-3}

(b) Generation ablation

A.1 Additional experiments

Table 4 details our ablation studies for both our tokenizer architecture as well as our generative model, as already discussed in the main manuscript. In the reconstruction ablation, one interesting point is that $n=4$-only training underperforms training with only $n=1$ and $n=2$. We believe the reasons to be twofold. First, coarse-level training may improve the quality of fine details; in particular, since we split the scene and scale each voxel back to the full range ($[-1,1]$) and encode each voxel separately, each rescaled voxel of $n=4$ will contain more smoothed geometry compared to $n=1$ and $n=2$. For example, a voxel in $n=1$ may contain a chair, but a voxel in $n=4$ might only contain the seat back (which has lower geometry frequency). This means the training data in $n=4$ is of lower frequency than $n=1$ or $n=2$. Thus, training with $n=1$ gives the highest content frequency, which is important for handling any downstream scenarios with extremely detailed objects. To handle boundaries between voxels, it is important to train with $n=2$ and $n=4$, as done in the full recipe. A second potential reason is compute-related implementation tradeoffs: specifically, in order to fit training within 80GB GPU memory (e.g., A100 or H100 GPU) as well as increase speed, we detach all but a single voxel at every iteration when $n=4$ (see Section A.2). In contrast, we can jointly train 8 voxels at a time when $n=2$, resulting in a stronger training signal. This difference in trainable voxels across different n can be attributed to the larger number of Gaussian activations that must be stored for backprop when more voxels are used, even when they are detached. This effect is validated in a small-scale experiment in Table 5.

Figure 8 and Figure 9 are extensions to Figure 3. Figure 9 shows the results under an equal compute setting, where LiTo-1024 has been trained to 200k iterations, LiTo-8192 to 100k iterations, LiTo-32768 to 34k iterations and split-and-scale to the standard 90k iterations. Split-and-scale can be seen to be very strong. Figure 8 explores the scaling performance of split-and-scale versus naive token scaling in the context of geometry only — split-and-scale can be seen to significantly outperform the standard approach, in both equal compute and equal iteration settings. Both figures also show the impact of adaptive allocation, as will be discussed in Section A.2.

In Figure 10, we also show an additional qualitative comparison of split-and-scale with 32768 inference tokens, and the upper-bound LiTo-32768 which cannot be trained in an 80GB GPU (Table 1). Split-and-scale remains competitive in visual performance.

To validate the importance of our multiscale training recipe, we also test whether the pretrained TRELLIS [48] tokenizer can be used in a subdivision, tile-based fashion out of the box with $n = 2$ and $n = 4$. Specifically, as the base TRELLIS uses a 64^3 grid, for $n = 2$, we construct a 128^3 grid and reconstruct individual 64^3 subgrids using TRELLIS. While the level of fine detail can be seen to qualitatively improve, large blur artifacts appear along object and voxel boundaries that worsen with n. Introducing overlapping reconstructions across different voxels does not seem to mitigate these issues and in fact worsens performance — blurry tile seams actually cover larger portions of the object than if they were not overlapped. This shows that tiling-aware training, as we developed in this paper, is important for scaling latent sizes as we have proposed.

A.2 Split-and-scale implementation details

Latent representation recipe. Our work builds off the LiTo representation [5], with a Perceiver encoder, a Gaussian decoder, and a flow matching-based point-cloud decoder [6]. We randomly select a grid size of $n = 1$, 2 or 4 at every iteration; we train for a total of 90k iterations. When $n = 4$ (≈ 32 active voxels on average), we detach all but one voxel. During training, we select anywhere between 2^{10}-2^{20} total input points, and randomly select a latent size between 256-4096 tokens per voxel. For supervision, we decode Gaussians for every voxel, render them jointly, and compute PSNR and LPIPS losses against the ground truth. We also apply a flow-matching loss through the point-cloud decoder [6] for every voxel. During inference we use 1024 tokens per voxel, and a maximum of 1 million points per voxel.

One potential downside of naive split-and-scale is that a fixed m latent tokens are used per voxel, when some voxels may contain more details and content than others. To tackle this, we can also train the encoder to handle a variable number of latent tokens, and at inference select the number of tokens per voxel according to some metric, *e.g.*, the number of input points per voxel. The improvement afforded by this adaptive allocation is shown in Figure 9. Unless otherwise noted, we use this variant for all reconstruction results shown in the paper. For generation, we do not use this adaptation to avoid any additional prerequisite generation outside coarse $4 \times 4 \times 4$ occupancy.

A.2.1 Network architectures

Our work is built off LiTo [5], and primarily inherits the same encoder/decoder/generative architectures from their work. Please refer to their paper for detailed architecture diagrams. Here, we summarize how each one works, and any modifications we have made.

Encoder: A Perceiver encoder receives a surface light field as input, which it subsamples to use as queries. Each query cross attends to only the points that it is the closest query to — this configuration can be interpreted as a sort of 3D patchification. Instead of the voxel-based attention pattern leveraged by LiTo for computational efficiency, we instead use standard full attention, as our per-voxel latent sizes are small enough for it to not become a bottleneck.

Point-cloud decoder: A noisy input point is positionally encoded and then passed into a Transformer architecture, where it only cross attends to the corresponding voxel latent (there is no self attention). The model conditions on the flow-matching timestep via gating and layernorm parameters predicted by a small MLP.

Gaussian decoder: A Perceiver encoder receives 64^3 coarse occupancy as input, cross attending to the latent of the corresponding voxel. It decodes a feature for every coarse location, each of which

is then processed by an MLP to produce 64 Gaussians. Note that this decoder runs once for every occupied voxel, *i.e.*, the number of Gaussians scales with the number of occupied voxels and therefore latent capacity.

Voxel decoder: A Perceiver architecture is used to decode a sparse structure TRELLIS latent. A pretrained TRELLIS sparse-structure decoder is then used to convert this latent to a full occupancy grid. To train this voxel decoder, at every iteration we select a random voxel, compute a split-and-scale latent for it and then decode a single sparse structure latent — we do not predict sparse structure latents for all voxels as we found such a procedure to be too memory inefficient for 80GB GPU training. We use this voxel decoder to decode Gaussians from generated latents.

Mesh decoder: A Perceiver takes in coarse occupancy for queries (like the Gaussian decoder) and outputs low resolution shape features, which are then upsampled by sparse convolutional layers to Flexicube parameters [14, 39]. We follow the split-and-scale recipe described in the main paper to train this mesh decoder, *i.e.*, we pick a random n, decode up to n^3 sets of Flexicube parameters, construct a mesh from their union, and compute losses on rendered depth and silhouettes. We do not compute losses on rendered normal, because we observe that the normals and depths of synthetic geometry do not often align (*e.g.*, bump or normal maps).

Generative model: Like LiTo, our work leverages a standard Diffusion Transformer architecture, but to sample in an autoregressive fashion. Unlike LiTo, we do not use a positional embedding for each noisy token, but instead a "voxel" embedding on which voxel each token should correspond to — we use a sinusoidal positional embedding of the voxel center, with 32 frequencies sampled from 2^0 to 2^{12}. We use a hidden dimension of 1152, with 16 self attention and cross attention heads. Time conditioning is implemented via adaLN-Zero [37]. We input in $\mathbf{v}_{\text{context}}$ and I via cross attention, where the image is preprocessed with DINOv2-large [36] and a learnable patchification layer, following LiTo [5]. I and $\mathbf{v}_{\text{context}}$ are given separate key-value and output projections, and their output cross attentions are summed. O is input via self attention with a single positionally-encoded learnable token per active voxel. To generate O, we use the same flow-matching architecture as above, with O represented as 4^3 binary tokens — we threshold with 0.5 to obtain the active voxels. We train this occupancy generator for 100k iterations.

A.3 Baseline implementation Details

LiTo [5]: We retrain LiTo variants with the same training data as split-and-scale. We primarily reuse the exact Perceiver-IO architectures used by LiTo for the encoder and decoder — we also use the same architectures for our split-and-scale encoder/decoder for fair comparison. When training variants with different latent counts, we also adjust the resolution of the occupancy grid that is input to the Gaussian decoder, in order to match the equivalent occupancy grid resolution used by split-and-scale for a particular latent count, *i.e.*, 1024 tokens $\rightarrow 64^3$, 8192 tokens $\rightarrow 128^3$, 32768 tokens $\rightarrow 256^3$. We use ground truth occupancy as input to the Gaussian decoder for reconstruction results, and the output of a voxel decoder for generation results.

For the generative model, we use the same DiT architecture as LiTo [5], except without a per-token zero-initialized positional embedding to aid with set generation. We also use 1036×1036 image inputs, rather than the 518×518 used in the original paper for fairness. We train this model for a total of 240k iterations — note that our AR model, at 450k iterations, has only been trained for roughly half the total object data of this LiTo model, as it is trained to generate only a subset of a given sample at every iteration.

TRELLIS [48]: We use the publicly released TRELLIS pipelines for image-to-3D and the tokenizer. For the tokenizer, TRELLIS expects a sparse 64^3 occupancy grid plus per-view DINO features. We voxelize the ground-truth point cloud directly into TRELLIS's 64^3 format. The DINO pool consumes 150 dense input views (*i.e.*, the same views used to generate the point cloud). We evaluate the results using TRELLIS' own Gaussian decoder. We render the output gaussians at our target cameras using TRELLIS's own splat rasterizer for a fair comparison against split-and-scale renders.

For image-to-3D, TRELLIS produces results in its own canonical frame ($[-0.5, 0.5]^3$, y-up), which does not align with our input-view coordinate system; we account for this with a fixed similarity transform (axis flip and $\times 2$ scale) before rendering. This rigid alignment is exact for the tokenizer

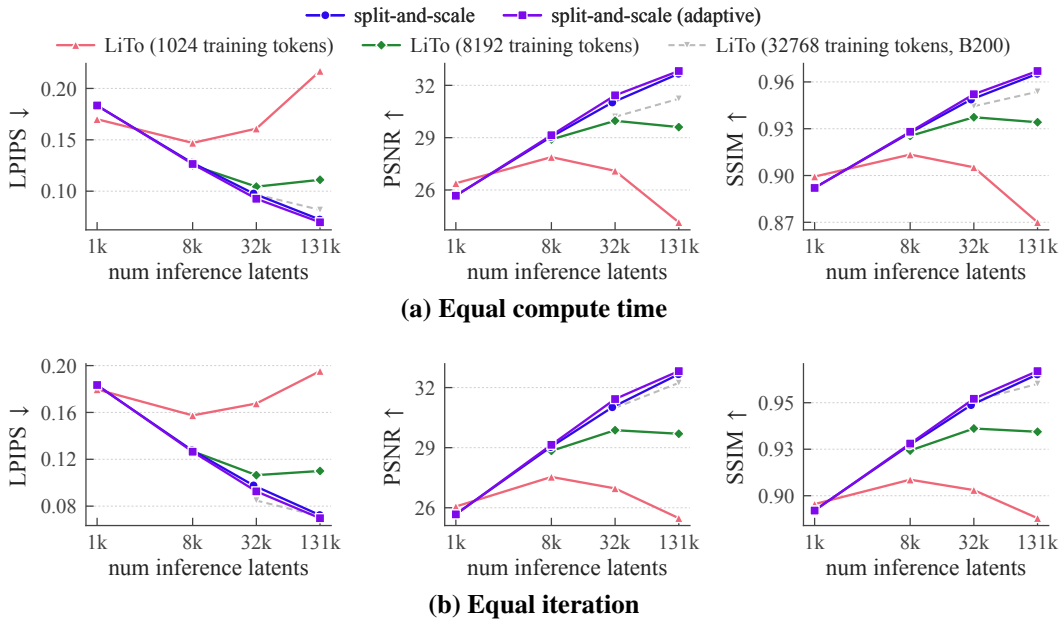


Figure 9: **Quantitative comparison versus naive token count scaling.** Following the recipe described in Section 3.2, we train a split-and-scale model that outputs 1024 latents per voxel at inference. For comparison, we train 1024 and 8192 token LiTo models [5] that use a similar amount of memory/compute during training. In both (a) equal compute time and (b) equal iteration scenarios, split-and-scale can be seen to scale with token count far better than the baseline models. Although it cannot be trained within the memory limits of an 80GB GPU, we include a 32768 token variant of LiTo [5] as an upper-bound for performance; split-and-scale is still competitive.

(which is conditioned on the GT geometry), but for the generative model the orientation of the asset relative to the camera is not constrained by the conditioning image. We therefore only report quantitative gen-numbers using set-level metrics (FID/KID) that are invariant to such a global rotation. For qualitative results, we manually realign each gen output.

TRELLIS-2 [47]: We use the publicly released checkpoints of TRELLIS-2. The tokenizer has the spatial downsample factor of 16 and 32 latent channels per active voxel. We run the tokenizer at 1024^3 voxel resolution, which after the $f16$ downsample yields a 64^3 sparse latent grid populated only at voxels intersected by the surface.

For image-to-3D, we run TRELLIS-2 with the 1024^3 voxel resolution. Because TRELLIS-2 represents appearance through PBR materials, evaluating it against our Gaussian renderings requires re-rendering all TRELLIS-2 outputs under the same lighting as our GT. We therefore re-render every reconstructed/generated GLB through Blender Cycles, using the exact per-sample HDRI light that were used to render our GT views. In practice TRELLIS-2's material-light disentanglement is not perfect: even with the GT lighting, the decoded base-color/metallic/roughness channels often produce appearance noticeably different from our diffuse GT, which inflates TRELLIS-2's pixel-space and FID/KID numbers in ways that are not entirely attributable to the tokenizer/generator itself.

As with TRELLIS, TRELLIS-2's gen output lives in its own canonical frame; we apply a fixed rotation/scale alignment for the qualitative figures, and rely on rotation-invariant set metrics (FID/KID) for the quantitative gen comparison. All renders use 1036×1036 resolution to match split-and-scale's GT.

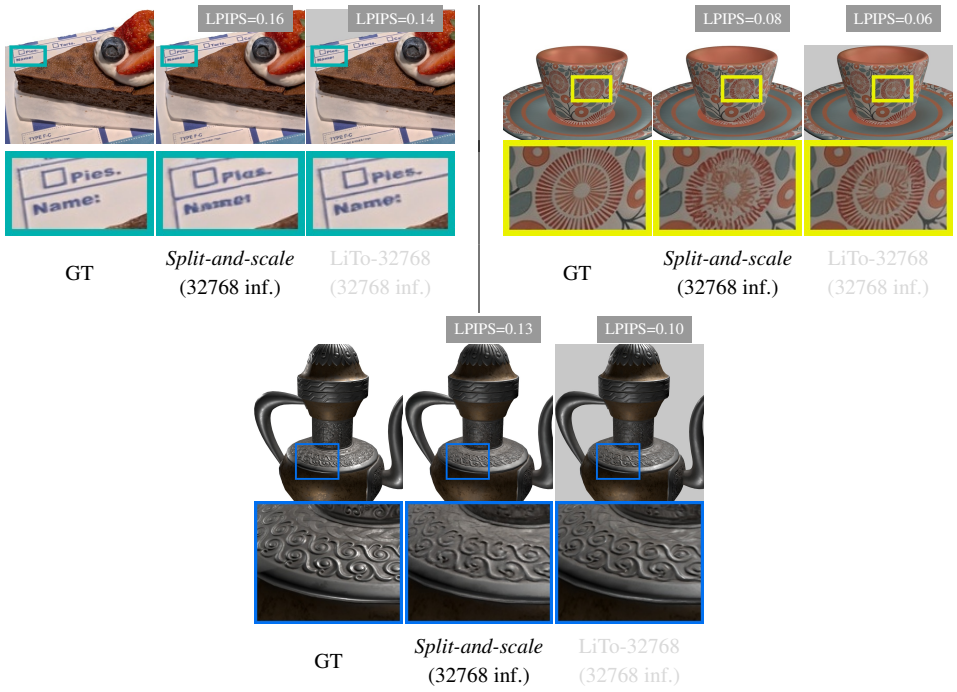


Figure 10: **Upper-bound qualitative reconstruction comparison.** Building off Figure 5, although it cannot be trained within the memory limits of an 80GB GPU, we show a 32768 token variant of LiTo ("LiTo-32768") as an upper-bound for performance; split-and-scale is still competitive. Asset credits [35, 33, 2].

Table 5: **Few-iteration training scale ablation.** We evaluate the impact of training at $n = 4$ without detaching any voxels. While this makes training infeasible on most standard 80GB GPUs, it provides the best performance.

Training recipe	**Fits 80 GB?**	$n = 4$ **LPIPS**	$n = 4$ **PSNR**	$n = 8$ **LPIPS**	$n = 8$ **PSNR**
Our recipe	yes	0.129	29.58	0.102	31.26
Only $n = 1$ training	yes	0.168	27.60	0.158	28.82
Only $n = 1$ and $n = 2$ training	yes	0.131	29.61	0.106	31.37
Only $n = 4$ training, detach	yes	0.161	27.17	0.128	28.98
Only $n = 4$ training, no detach	no	0.111	29.72	0.080	31.69

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

Table 6: **TRELLIS tiling experiment.** We evaluate whether the vanilla TRELLIS [48] tokenizer can be applied in a similar tile-based fashion as split-and-scale. Quantitatively, performance can be seen to drop with the amount of tiling, even with overlapping reconstructions across different voxels.

Method	**LPIPS**	**PSNR**
original TRELLIS [48]	0.212	20.86
TRELLIS, $n = 2$	0.254	19.63
TRELLIS, $n = 2$ + overlapping	0.284	18.69
TRELLIS, $n = 4$	0.295	17.86

Justification: Claims (i) inference-time latent-capacity scaling with a single pretrained tokenizer, (ii) competitive reconstruction quality versus much larger baselines that exceed single-GPU memory limits, and (iii) applicability to generative modeling via an autoregressive 3D generator — each of which is verified by the reconstruction ablations and the generation experiments.

Guidelines:

- The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [Yes]

Justification: Section 5 ("Limitations and Future Work") explicitly discusses the main limitations

Guidelines:

- The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not include theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper describes the method formally, documents all training, architecture, dataset details, enabling reproduction of the main results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

Justification: Code and trained model checkpoints are not released with this submission; however, we have plans to release code upon acceptance. All experiments are conducted on the publicly available Objaverse-XL TRELLIS subset. We will release the dataset processing scripts upon acceptance.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 3.2.1, 4.1, 4.2 and Appendix A.2, A.3 specify the training details and data splits.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Error bars and statistical significance tests are not reported due to the prohibitive compute cost of retraining the models. Quantitative results are instead reported as single-run metrics aggregated over a fixed test set.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)

- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We specify the hardware used along with training iteration counts and batch size (256). We also report the inference time and memory usage.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research complies with the NeurIPS Code of Ethics. It uses publicly available synthetic 3D assets with appropriate license attribution, involves no human subjects or personally identifiable data, and poses minimal risk of misuse beyond general concerns common to 3D generative modeling.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This work is foundational research on latent 3D representations and reconstruction/generation of 3D assets; it does not involve human subjects, personal data, or deployed decision-making systems, and does not present a direct path to negative societal applications beyond the general concerns already inherent to 3D generative modeling.

    Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release artifacts posing high-risk misuse concerns that would require dedicated safeguards.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All existing assets are credited via citations in the paper. Each asset is explicitly noted as licensed under Creative Commons.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not release any new assets alongside the submission.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human subjects research, so IRB approval is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs do not have impact on the core methodology.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.