AutoPartGen: Autoregressive 3D Part Generation and Discovery

Minghao Chen[1,2] **Jianyuan Wang**[1,2] **Roman Shapovalov**[2] **Tom Monnier**[2]
Hyunyoung Jung[2] **Dilin Wang**[2] **Rakesh Ranjan**[2] **Iro Laina**[1] **Andrea Vedaldi**[1,2]
[1]Visual Geometry Group, University of Oxford [2]Meta AI
silent-chen.github.io/AutoPartGen

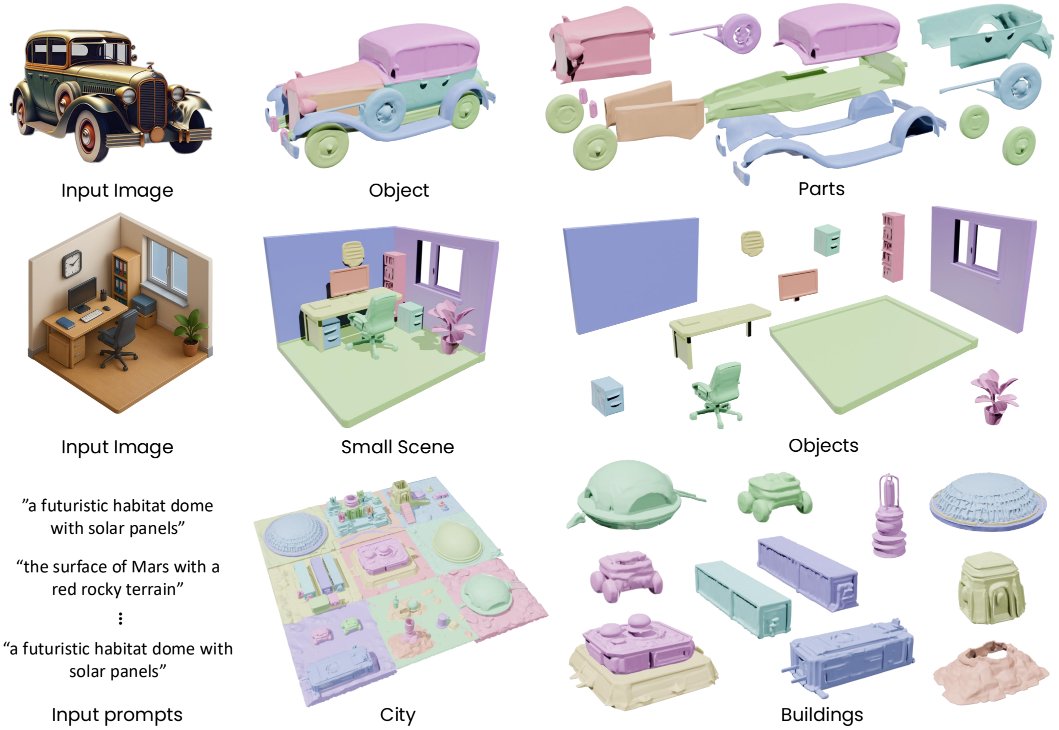
Input Image
Object
Parts
Input Image
Small Scene
Objects
”a futuristic habitat dome with solar panels”
“the surface of Mars with a red rocky terrain”
⋮
“a futuristic habitat dome with solar panels”
Input prompts
City
Buildings

Figure 1: **AutoPartGen** can be *applied*, by itself or in combination with other models, to the generation of compositional 3D objects, scenes and cities starting from 3D models, images or text.

Abstract

We introduce **AutoPartGen**, a model that generates objects composed of 3D parts in an autoregressive manner. This model can take as input an image of an object, 2D masks of the object's parts, or an existing 3D object, and generate a corresponding compositional 3D reconstruction. Our approach builds upon 3DShape2VecSet, a recent latent 3D representation with powerful geometric expressiveness. We observe that this latent space exhibits strong compositional properties, making it particularly well-suited for part-based generation tasks. Specifically, AutoPartGen generates object parts autoregressively, predicting one part at a time while conditioning on previously generated parts and additional inputs, such as 2D images, masks, or 3D objects. This process continues until the model decides that all parts have been generated, thus determining automatically the type and number of parts. The resulting parts can be seamlessly assembled into coherent objects or scenes without requiring additional optimization. We evaluate both the overall 3D generation capabilities and the part-level generation quality of AutoPartGen, demonstrating that it achieves state-of-the-art performance in 3D part generation.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

1 Introduction

Processing 3D objects, including generating them based on a textual description or an image, is an important aspect of Spatial Intelligence. Current 3D generators often treat objects or even entire scenes as monolithic shells. However, many applications require modeling their compositional structure, decomposing them into well-defined 3D parts to enable reasoning or manipulation at a finer granularity, such as applying textures and materials to each part separately. More specifically, a character in a video game should be decomposable into different parts to support animation or allow the game software to swap clothes or accessories. Windows and doors in the 3D model of a house need to be separate entities to allow user interaction, such as opening or closing them. Similarly, the design of a machine must consist of distinct parts to be functional (e.g., the cogs in a clock) or to enable 3D printing or other kinds of CNC manufacturing.

In this paper, we address the problem of generating 3D objects with a *compositional structure*. We introduce *AutoPartGen*, a new autoregressive model that can *directly* generate a 3D object part by part, building on a powerful latent 3D representation. AutoPartGen is robust, flexible, and scalable. As shown in Fig. 1, AutoPartGen can be applied, either independently or in combination with other models, to generate compositional 3D objects, scenes, or even cities, starting from 3D models, images, or text prompts. AutoPartGen solves three key problems to enable such applications: (i) object-to-parts, where it decomposes an existing 3D object into meaningful parts; (ii) image-to-parts, where the model generates 3D parts from an unstructured input image; and (iii) masks-to-parts, where users can provide 2D part masks to guide the generation. In the first two scenarios, AutoPartGen *automatically* predicts semantically meaningful 3D parts without requiring part annotations. In the third scenario, user-provided masks offer fine-grained control over the model partitioning.

Our autoregressive approach has two key benefits. First, it models the joint distribution over the object parts, ensuring that they fit together cohesively. Second, it enables the model to generate a variable number of parts, which is crucial since the number of parts is not fixed or known a priori.

We build AutoPartGen on recent advances in latent 3D representations and parameterize the 3D surface $\boldsymbol{x} \subset \mathbb{R}^3$ of the object using a latent vector $\boldsymbol{z} \in \mathbb{R}^{M \times d}$, where M is the latent length and d is the latent dimension. We use the 3DShape2VecSet representation [65, 31, 66] and observe, for the first time, that this representation is inherently compositional. Specifically, we show that the concatenation $\boldsymbol{z} = \boldsymbol{z}^{(1)} \oplus \boldsymbol{z}^{(2)} \in \mathbb{R}^{2M \times d}$ of two codes $\boldsymbol{z}^{(1)}$ and $\boldsymbol{z}^{(2)}$ decodes into the union $\boldsymbol{x} = \boldsymbol{x}^{(1)} \cup \boldsymbol{x}^{(2)}$ of the corresponding surfaces $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$.

Based on this insight, we propose generating a sequence of latent codes $\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(K)}$, each decoding into a corresponding 3D part $\boldsymbol{x}^{(k)}$. Crucially, the generation of each part is conditioned on an overall latent representation of the target 3D object $\boldsymbol{x}$ as well as on all previously generated parts $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(k-1)}$. This conditioning improves the consistency of the generated parts, meaning that they better fit each other compared to the output of the methods that extract parts independently [5, 60].

As noted in [5], object decomposition is an ambiguous problem. For example, a chair might be decomposed into few high-level parts (e.g., seat, back, legs) or a more granular set of components (e.g., individual leg segments, cushion, backrest slats). This choice typically depends on the application or the preferences of the 3D artist creating the asset. We address this ambiguity by making the 3D autoregressive model *stochastic*, using denoising diffusion to generate the next part vector $\boldsymbol{z}^{(k)}$ based on the previously generated parts $\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(k-1)}$ and the available evidence (i.e., the full 3D object, an image of the object, or 2D part masks, depending on the application). Importantly, we train a *single* diffusion model capable of handling all three cases.

We evaluate AutoPartGen against state-of-the-art part-aware 3D generators. Compared to the recent PartGen [5], AutoPartGen is easier to implement and maintain (as it does not require training several multi-view image generators) and more accurate. Compared to HoloPart [60], a method that completes a *pre-segmented* outer surface of a mesh to form 3D parts, AutoPartGen is more accurate and significantly more capable, as it can automatically discover parts and reconstruct them from either a 2D image or a "shell" 3D object, optionally guided by 2D masks, not requiring any 3D annotation.

2 Related Work

Generating a 3D object from a single image, or even just text, faces an obvious challenge: the 3D object contains significantly more information than the image or the text. This is similar to the problem of generating images or videos from text, and it is solved by learning a prior, or conditional distribution, from billions of data samples. However, data of this size is unavailable for 3D objects. Authors address this problem by involving 2D image or video generators in the 3D generation process. We distinguish two main camps: multi-view direct and single-view latent 3D generation.

Multi-view 3D Generation. In multi-view 3D generation, one asks the image generator to do most of the lifting, generating several views of the 3D objects, and thus simplifying extracting a 3D object from them. First, this was done using Score Distillation Sampling (SDS) [42], an idea explored extensively in follow-ups like GET3D [13], ProlificDreamer [54], DreamGaussian [51], Lucid Dreamer [9] which seek to achieve multi-view consistency via iterative (and slow) optimization of a radiance field (NeRF [44] or 3DGS [24]). A significant innovation, pioneered by UpFusion [23], 3DiM [55], Zero-1-to-3 [36] and MVDream [45], was to fine-tune the image generator to directly produce multiple consistent views of the object. By making the image generator more 3D aware, 3D reconstruction becomes simpler, as noted in InstantMesh [57], GRM [58], and others [56, 50].

Single-view Latent 3D Generation. The alternative approach is to start from a single image of the object and directly reconstruct the 3D object from it. Because single-view reconstruction is extremely ambiguous, this requires to learn a reconstruction function. This was the path taken, for example, by LRM [19] and others [17, 48]. However, their deterministic reconstruction model cannot cope well with this ambiguity and often produces blurry outputs. Much better results were recently obtained by switching to stochastic 3D reconstruction based on *latent* diffusion. Some of the best single-image 3D reconstructors are based on the 3DShape2VecSet [65] latent representation (also similar to Michelangelo's [67]). Building on it, CLAY [66], DreamCraft3D [47], CraftsMan [30], TripoSG [31], and others [10, 59, 63] are able to generate highly detailed and accurate 3D shapes. We build on this representation as well and show that it also supports compositionality very effectively.

Composable 3D Generation. Approaches to composable 3D generation typically start by decomposing objects into constituent parts. One common strategy represents objects as mixtures of primitives, often without semantic labels. For instance, SIF [16] models object occupancy using mixtures of 3D Gaussians. LDIF [15] represents shapes as a set of local deep implicit functions (DIFs), spatially arranged and blended using a template of Gaussian primitives. Methods such as Neural Template [21] and SPAGHETTI [1] achieve decompositions through auto-decoding. SALAD [29] utilizes SPAGHETTI for diffusion-based generation. PartNeRF [52] expands this concept by employing mixtures of NeRFs. NeuForm [32] and DifFacto [39] specifically target part-level controllability. DBW [38] uses textured superquadrics to decompose scenes. In contrast, another research direction emphasizes explicitly semantic parts. PartSLIP [35] and PartSLIP++ [71] segment objects into semantic components from point clouds using vision-language models. Part123 [34] adapts techniques from scene-level approaches like Contrastive Lift [2] to reconstruct object parts. PartSDF [49] learns latent codes for parts using an auto-decoder and then uses SALAD for part prediction. Comboverse [8] leverages single-view inpainting model and 3D generator for composable 3D generation with spatial-aware SDS optimization. Deep Prior Assembly [69] reconstructs 3D scenes from a single image in a zero-shot manner by assembling large models. MIDI [20] extend pre-trained image-to-3D generator to multi-instance generator through costly global attention. CAST [62] reconstructs physically consistent 3D scenes from a single RGB image using occlusion-aware diffusion and GPT-guided physics correction. HoloPart [60], a recent work, starts from the shell of a 3D object and a part-level segmentation for it and performs 3D amodal part completion.

The work most related to ours is PartGen [5]. This squarely sits on the 'multi-view direct' camp (see above). It uses multi-view diffusion models for segmentation and completion of compositional 3D objects from diverse modalities.

3D Segmentation. 3D parts can be obtained by segmenting a 3D object (although the resulting parts will generally be incomplete). Some approaches for semantic 3D segmentation such as [68, 28, 53, 25, 2] used neural fields to 'fuse' 2D semantic features in 3D. Contrastive Lift [2] introduced a slow-fast contrastive clustering scheme for 3D instance segmentation. Recent methods such as [26, 64, 43, 3] integrate SAM [27] and often CLIP to model multi-scale concepts, where LangSplat explicitly encodes scale information and N2F2 learns to bind concepts to scales automatically. Neural

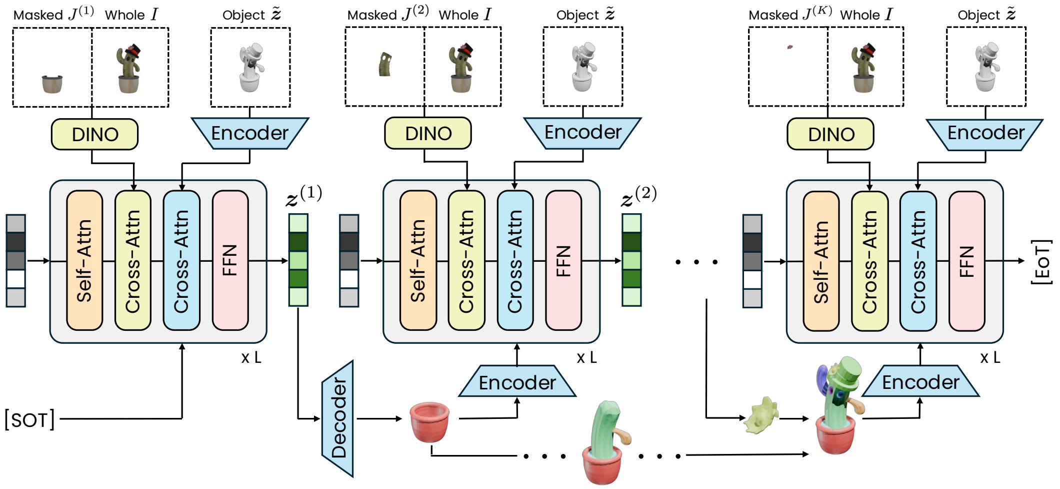

Figure 2: **AutoPartGen generates parts autoregressively.** At each step, a 3D latent diffusion model generate the next part, conditioned on the previously generated parts $\boldsymbol{z}^{(1,\dots,k)}$, the overall object $\tilde{\boldsymbol{z}}$, and, optionally, an image I of the object and an image $J^{(k)}$ of the part. The latent representation uses 3DShape2VecSet and the diffusion model is a DiT.

Part Priors [4] used learned priors for test-time decomposition. Additionally, efforts to develop 3D 'foundation' models [70, 7] are enabling zero-shot point cloud segmentation across diverse domains.

3 Method

Let $\boldsymbol{x} \subset \mathbb{R}^3$ be a *3D object* given by a surface embedded in $\mathbb{R}^3$. We assume that the object is *compositional*, meaning that it can be expressed as the union $\boldsymbol{x} = \bigcup_{k=1}^{K} \boldsymbol{x}^{(k)}$ of K disjoint *parts* $\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(K)}$, each of which is also a surface. Concretely, $\boldsymbol{x}$ is usually a 3D mesh created by an artist, and $\boldsymbol{x}^{(k)}$ are the components of the mesh that the artist has manually defined when creating the mesh. These parts are thus defined to facilitate editing of the 3D object or for functional purposes, such as animation. Generally, the same 3D object can have different and equally valid part decompositions.

Our aim is to learn to generate 3D objects $\boldsymbol{x}$ and their part decompositions $\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(K)}$. We consider three different scenarios. In the first scenario (object-to-parts), we are given the 3D object $\boldsymbol{x}$, and the goal is to sample a possible decomposition of this object into parts. Furthermore, we may potentially have an object $\hat{\boldsymbol{x}}$ which is incomplete with respect to its constituent parts — a case that may arise if $\boldsymbol{x}$ is acquired by a 3D scanner that cannot look *inside* the object or synthesized by a generator that is not aware of the internal structure of the object, as exemplified by [60]. In this case, therefore, the goal is to also *complete* the parts, thus recovering the complete object $\boldsymbol{x}$ as a byproduct.

In the second scenario (image-to-parts), the starting point is an image $I : \Omega \to \mathbb{R}^3$ of the object, where $\Omega \subset \mathbb{R}^2$ is the (finite) image domain. Inferring the object $\boldsymbol{x}$ from a single image is also known as the *image-to-3D* problem. Here, the task is to also infer its part decomposition $\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(K)}$.

In the third scenario (masks-to-parts), we are given, in addition to the image I, K 2D part *masks* $M^{(k)} : \Omega \to \{0, 1\}$ that indicate the pixels in the image I that belong to each part $\boldsymbol{x}^{(k)}$. These masks can be defined manually or, more likely, automatically, utilizing a 2D segmentation model. The problem is the same as before, but the 3D parts must match the prescribed masks.

In all these cases, recovering the parts (or the object) is *ambiguous*. These problems are thus *stochastic* and are solved by learning suitable conditional probability distributions: $p(\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(K)} \mid \boldsymbol{x})$ (object-to-parts), $p(\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(K)} \mid I)$ (image-to-parts), and $p(\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(K)} \mid I, M^{(1)}, \dots, M^{(K)})$ (masks-to-parts). We develop a single model that can handle all three cases.

3.1 Latent 3D shape space

Directly defining, modeling, and learning a distribution on 3D surfaces is difficult. We thus introduce a *latent space*, providing a finite-dimensional parametrization of the surfaces.

We utilize the *VecSet* representation developed by [65]. This representation is based on learning an encoder-decoder pair (E, D). The encoder E takes a collection of N object points $P = \{p_1, \dots, p_N\} = \text{sample}_N\, \boldsymbol{x}$ and maps them to a latent vector $\boldsymbol{z} = E(P)$. Here sample_N is a function that samples N random points from the surface of the object $\boldsymbol{x}$, so that $P \subset \boldsymbol{x}$. The decoder takes the latent vector $\boldsymbol{z}$ and a query 3D point $p \in \mathbb{R}^3$ and evaluates the *signed distance function* (SDF) at p as $\text{SDF}(p|\boldsymbol{x}) = D(p|\boldsymbol{z})$. The encoder-decoder pair is thus 'translational', in the sense that it translates one type of representation of the object (the point cloud P) into another (the signed distance function $\text{SDF}(\cdot|\boldsymbol{x})$).[1]

In more detail, the encoder E compresses the point cloud P into a sequence $\boldsymbol{z} = (z_1, \dots, z_M)$ of M tokens $z_i \in \mathbb{R}^D$. The $M \ll N$ tokens are obtained by first subsampling the point cloud P into a much smaller set of points $\tilde{P} = \{p_1, \dots, p_M\} = \text{sample}_M\, P \subset P$ and then by applying a transformer neural network to the points in $\tilde{P}$ to output $\boldsymbol{z}$. The transformer also attends to the large number of points in P efficiently via cross-attention. The network is designed to be *permutation equivariant*, meaning that the order of the points and tokens is immaterial, explaining the moniker 'VecSet'. The decoder D takes a point $p \in \mathbb{R}^3$ and the tokens $\boldsymbol{z}$ and outputs the value of the signed distance function $\text{SDF}(p|\boldsymbol{x})$, also utilizing a transformer neural network in the form of a Perceiver [22].

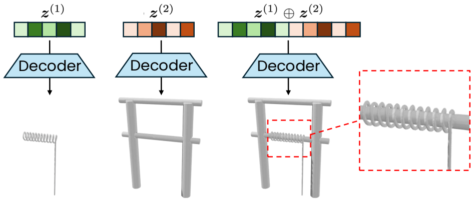

Figure 3: **Compositionality of the VecSet space.** Concatenation of two latents will result in a spatial combined mesh.

The intuition behind this representation is that each token vector z_i encodes a local region of the surface centered at the point p_i. However, the transformer allows tokens to communicate globally, which makes this interpretation somewhat loose. Empirically, we have discovered that locality, or at least compositionality, is well supported by the representation. As we show in Fig. 3, the tokens can be concatenated to form a new latent vector $\boldsymbol{z} = \boldsymbol{z}^{(1)} \oplus \boldsymbol{z}^{(2)}$ that decodes into a new surface $\boldsymbol{x} = \boldsymbol{x}^{(1)} \cup \boldsymbol{x}^{(2)}$ that is a good approximation of the union of the two parts, without any retraining.

3.2 Latent 3D diffusion

Having established the latent representation $\boldsymbol{z}$ for the shapes, the next task is to learn a model that can sample a shape given some evidence y, from a conditional probability distribution $p(\boldsymbol{z} \mid y)$ (for example, y could be the image I of the object). This utilizes (latent) diffusion. In brief, we define a sequence of progressively more noised versions of the latent vector $\boldsymbol{z}$ as $\boldsymbol{z}_t = \sqrt{\alpha_t}\boldsymbol{z} + \sqrt{1-\alpha_t}\epsilon$, where $\epsilon \sim \mathcal{N}(0, I)$ is a Gaussian noise vector and α_t, $t = 0, 1, \dots, T$ is a schedule of noise levels. Following [46, 14], we introduce the *flow velocity* $\boldsymbol{v}(t, \boldsymbol{z}_t, \epsilon) = (\boldsymbol{z}_t - \sqrt{\alpha_t}\epsilon)/\sqrt{1-\alpha_t}$. The diffusion model $\hat{\boldsymbol{v}}(t, \boldsymbol{z}_t, \epsilon)$ is trained to predict the flow velocity $\hat{\boldsymbol{v}}(t, \boldsymbol{z}_t \mid y)$ given only the latent vector $\boldsymbol{z}_t$ and the condition y, minimizing the loss $\mathcal{L}(\hat{\boldsymbol{v}}) = E_{(y,\boldsymbol{z}),t,\epsilon}\|\hat{\boldsymbol{v}}(t, \boldsymbol{z}_t \mid y) - \boldsymbol{v}(t, \boldsymbol{z}_t, \epsilon)\|^2$ averaged over a training set of evidence-latent vector pairs $(y, \boldsymbol{z})$, a random time step t and noise ϵ.

3.3 Autoregressive 3D part generation

The model described in Section 3.1 generates the entire 3D object $\boldsymbol{x}$ (or, more precisely, its latent representation $\boldsymbol{z}$) as a whole. Here, we consider the problem of generating the object parts instead. Our goal is to learn a *single model* that can handle all three part generation scenarios: object-to-part,

[1] For this to work, a few technical assumptions are required. We assume $\boldsymbol{x}$ to be the finite disjoint union of closed regular surfaces smoothly embedded in $\mathbb{R}^3$ without self-intersections. This makes the surfaces orientable; then, the surfaces split $\mathbb{R}^3$ into disjoint regions alternating between the outside and inside of the object. In this way, the signed distance function is well defined. The parts $\boldsymbol{x}^{(k)}$ are defined in the same way, and in fact, they are formed by the union of one or more of the closed surfaces that comprise $\boldsymbol{x}$, so that a signed distance function is defined for each part too.

Table 1: **3D part completion.** Reconstruction quality of the parts and whole object. *reproduced.

Method	3D Mask	Parts			Whole Object		
		IoU ↑	F-Score↑	CD↓	IoU ↑	F-Score↑	CD↓
HoloPart* [60]	✓	0.658	0.836	0.065	0.821	0.945	0.018
PartGen [5]	✗	0.614	0.812	0.121	0.779	0.921	0.033
AutoPartGen	✗	0.665	0.861	0.047	0.832	0.967	0.012

image-to-part, and masks-to-part, depending on the inputs y provided. An overview of the pipeline is shown in Figure 2.

To generate an undetermined number of parts K, we consider an autoregressive approach, where a single part $\boldsymbol{x}^{(k)}$ is generated each time, based on what was generated before. The model can thus be described as a conditional distribution $p(\boldsymbol{z}^{(k)} \mid y, \boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(k-1)})$ where $\boldsymbol{z}^{(k)}$ is the latent representation of the k-th part and the input y collects the additional evidence available to the model.

The nature of this evidence depends on the reconstruction scenario. In the object-to-part scenario, y is simply the 3D object $\boldsymbol{x}$. In the image-to-part scenario, y is the image I of the object. In the masks-to-part scenario, y is the image I as well as the masked image $J^{(k)} = M^{(k)} \odot I$, denoting which parts should be generated next.

Knowledge of the previously generated parts $\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(k-1)}$ is essential as this allows the model to ensure that the next part fits together well with the previously generated ones. Furthermore, in all cases we consider, the evidence y also provides some evidence on the overall shape of the object. As suggested in Fig. 3, we can represent the union of parts by simply concatenating their latent representations. However, for compactness, we found it useful to fuse their codes into one, which we obtain as: $\boldsymbol{z}^{(1,\ldots,k-1)} = E\left(\cup_{j=1}^{k-1} \operatorname{sample}_N D(\cdot \mid \boldsymbol{z}^{(j)})\right)$, where sample_N is a function that samples N points from the surface of the object defined by the zero level set of the SDF function $D(\cdot|\boldsymbol{z}^{(k)})$, we call this strategy re-encoding.

We found it optional but useful to pin down the overall object by adding to the evidence y a code $\tilde{\boldsymbol{z}}$ for the object as a whole, which is either given outright (object-to-part, $\tilde{\boldsymbol{z}} = E(\operatorname{sample}_N \hat{\boldsymbol{x}})$) or can be obtained by the model itself (image-to-part and masks-to-part, $\tilde{\boldsymbol{z}} \sim p(\boldsymbol{z} \mid I)$) by directly providing an unmasked image to our mode.

With all this, we learn a conditional generator model

$$\boldsymbol{z}^{(k)} \sim p(\boldsymbol{z}^{(k)} | \tilde{\boldsymbol{z}}, \boldsymbol{z}^{(1,\ldots,k-1)}, y), \tag{1}$$

where $y = \phi$ for the object-to-part scenario, $y = I$ for the image-to-part scenario, and $y = (I, M^{(k)})$ for the masks-to-part scenario. The generation process stops when all the input masks have been processed, if available, or when the model outputs a predefined special `[EoT]` token, representing empty shape. In practice, we represent the `[EoT]` token using latents whose values are all zeros.

Based on Section 3.2, learning the distribution Eq. (1) amounts to learning a velocity field $\hat{\boldsymbol{v}}(t, \boldsymbol{z}_t \mid \tilde{\boldsymbol{z}}, \boldsymbol{z}^{(1,\ldots,k-1)}, y)$. During inference, we use *classifier-free guidance* (CFG) [18] to modulate the strength of the conditioning. In the most general case, the model is conditioned by the overall (partial) object $\tilde{\boldsymbol{z}}$, the previously generated parts $\boldsymbol{z}^{(1,\ldots,k-1)}$, and a masked image pair $y = (I, J^{(k)})$. We modulate the importance of the geometric and visual conditioning as follows:

$$\begin{aligned}\hat{\boldsymbol{v}}_{\text{CFG}}(t, \boldsymbol{z}_t \mid \tilde{\boldsymbol{z}}, \boldsymbol{z}^{(1,\ldots,k-1)}, y) = w_{\text{img}} \left(\hat{\boldsymbol{v}}(t, \boldsymbol{z}_t \mid \tilde{\boldsymbol{z}}, \boldsymbol{z}^{(1,\ldots,k-1)}, I, J^{(k)}) - \hat{\boldsymbol{v}}(t, \boldsymbol{z}_t \mid \tilde{\boldsymbol{z}}, \boldsymbol{z}^{(1,\ldots,k-1)})\right) \\ + w_{\text{geom}} \left(\hat{\boldsymbol{v}}(t, \boldsymbol{z}_t \mid \tilde{\boldsymbol{z}}, \boldsymbol{z}^{(1,\ldots,k-1)}) - \hat{\boldsymbol{v}}(t, \boldsymbol{z}_t, \emptyset)\right) + \hat{\boldsymbol{v}}(t, \boldsymbol{z}_t, \emptyset),\end{aligned} \tag{2}$$

where w_{img} and w_{geom} modulate, respectively, image and geometry conditioning. The different inputs are implemented by first encoding into tokens, which are then cross-attended by a transformer neural network computing the flow velocity. Hence, to suppress an input we simply replace it with dummy tokens. In the same way, we randomly drop some input at training time to allow the model to learn to use any required subset of the inputs.

Discussion Here, we contrast our model to prior works and justify its design. The most straightforward approach to part generation is to sample each part $\boldsymbol{x}^{(k)}$ independently from a 'marginal'

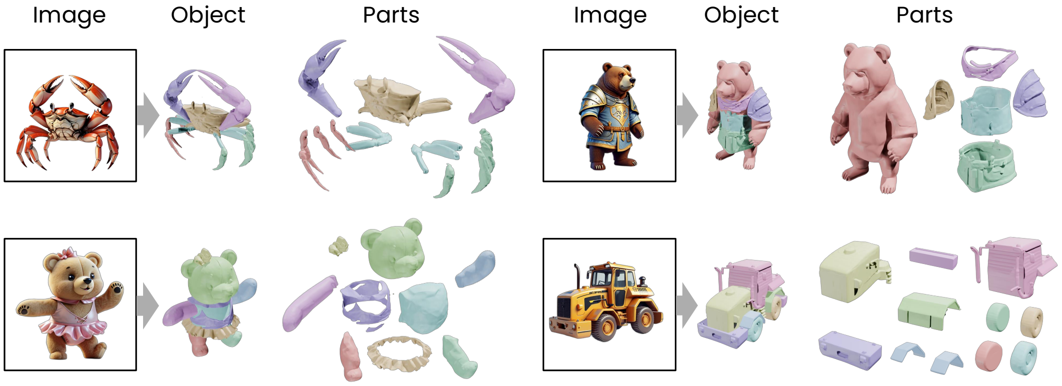

Figure 4: **Image-to-parts scenario.** Given an input image, AutoPartGen recovers a compositional 3D object made up of several meaningful and complete parts.

distribution $p(\boldsymbol{x}^{(k)})$. However, this model lacks a mechanism to tie the parts together and would result in a soup of random, uncoordinated parts. The simplest such mechanism is to provide evidence y for the overall shape of the 3D object. For instance, in the image-to-3D case, $y = I$ could be a 2D image of the object, and we may sample parts from the conditional distribution $p(\boldsymbol{x}^{(k)} \mid I)$. While I constrains the shape and position of the possible parts, these are still quite ambiguous. This explains why PartGen [5] conditions part generation on a multi-view image $y = I_{\text{MV}}$ of the 3D object $\boldsymbol{x}$, and HoloPart [60] starts from a (partial) 3D reconstruction $y = \hat{\boldsymbol{x}}$ of the object itself.

Even then, the reconstruction context y is likely insufficient because there is no indication of *which* part should be reconstructed next. We could sample the parts in a random order, but this would not be very efficient. Furthermore, because the part decomposition is not unique, we would still need to extract a coherent subset of parts from the ‘part soup’ so obtained.

Prior works address this issue by explicitly telling the 3D reconstruction model which part to extract next. PartGen does so by providing a multi-view image J_{MV} of the part, and HoloPart by providing a 3D mask M_{3D} of the part, defining distributions $p(\boldsymbol{x}^{(k)} \mid I_{\text{MV}}, J_{\text{MV}})$ and $p(\boldsymbol{x}^{(k)} \mid \hat{\boldsymbol{x}}, M_{\text{3D}})$, respectively. Hence, the problem of generating a coherent collection of parts is offloaded to a mechanism external to the 3D reconstructor. On the contrary, our 3D generator/reconstructor makes this determination by itself, operating autoregressively, one part at a time, without additional models.

4 Experiments

We first give the implementation details of AutoPartGen, including network architectures, training procedures, and datasets. We then demonstrate its performance under various conditions, highlighting its versatility for different applications. Next, we compare our approach with state-of-the-art 3D completion methods and provide ablation studies to analyze key design choices. Finally, we showcase several applications of AutoPartGen.

4.1 Implementation Details

Architecture. Our architecture builds upon the 3DShape2VecSet [65] framework, with some modifications. Specifically, we increase the input points of the VAE encoder to 32K, and utilize both point coordinates and normals as input features to better capture fine-grained geometric details. The diffusion model is implemented as a DiT [41] with a width of 2048 and 24 layers. For image-conditioned generation, we use DINOv2 [40] to encode the input image I and part-masked images $J^{(k)} = I \odot M^{(k)}$ independently. The resulting features are concatenated along the channel dimension and passed through a small MLP to match the diffusion transformer input. We provide more details in the supplementary material.

Training. We use the AdamW optimizer with a learning rate of 1e-4 and train the model for 500K iterations on 256 NVIDIA H100 GPUs. Training the full model takes approximately 4 days. More details on hyperparameters and data preprocessing are provided in the supplementary material. During

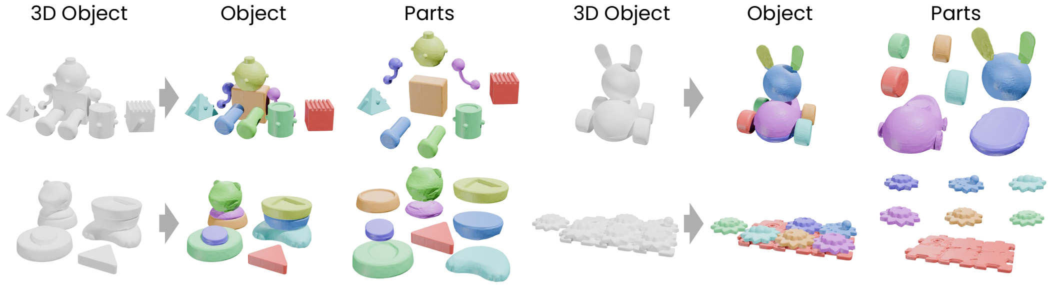

Figure 5: **Object-to-parts scenario.** Given an input 3D object, AutoPartGen regenerates it as a composition of meaningful and complete 3D parts.

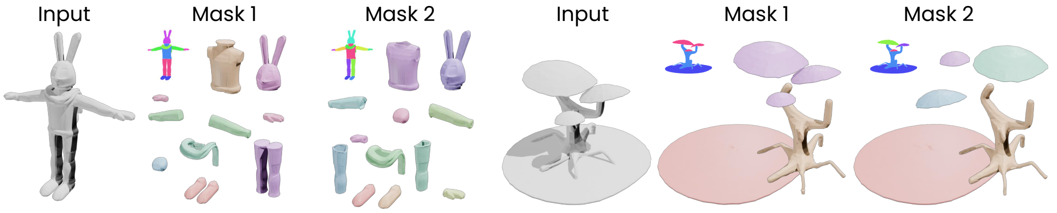

Figure 6: **Masks-to-parts scenario**. AutoPartGen reconstructs a compositional 3D object guided by user-provided 2D part masks. Varying these masks yields different decompositions, potentially at different levels of granularity.

training, we randomly drop the image condition, the geometry condition, or both with probabilities of 0.05 each. For CFG, we use $w_{\text{img}} = 7$ and $w_{\text{geom}} = 4$ as the default setting.

Training Data. Our training data pipeline draws inspiration from PartGen [5], but is substantially scaled to encompass approximately 300K assets and 2M individual parts. We start with a collection of 1.8M 3D assets, all licensed that permit AI training. Each asset is stored in glTF/GLB formats, which contains multiple meshes in it and embeds a hierarchical structure. To manage complexity, if an asset contains more than a predefined maximum of 15 meshes, we iteratively merge meshes from the bottom up, until the mesh count is within this limit. To prepare for training VAE and diffusion models, we compute a truncated signed distance for each part in a normalized space and also render different views for image-conditioned cases. More details are included in the supplementary materials.

4.2 Object, Image and Masks to 3D Parts Generation

We test AutoPartGen with different types of inputs to demonstrate its versatility. Specifically, we consider: (1) image-to-parts generation from a single input image, where the images are generated by text-to-image (2D) generators; (2) object-to-parts decomposition from a full 3D mesh, with meshes sourced from Google Scanned Objects [11]; and (3) masks-to-parts generation with user-provided 2D part masks, where the masks are taken from PartObjaverse-Tiny [61]. Figures 4 to 6 qualitatively demonstrates that AutoPartGen produces accurate and consistent 3D parts across all these input types.

4.3 Comparison to the State-of-the-Art

Evaluation Protocol. We use PartObjaverse-Tiny [61] for evaluation, filtering out very small parts following the protocol of [60]. This dataset comprises objects from diverse categories with manually annotated 3D part segmentations. We use standard metrics to assess the quality of the reconstructed geometry: Intersection-over-Union (IoU), Chamfer Distance (CD), and F-score. IoU is calculated on 64^3 voxel grids, and the F-score adopts a distance threshold of 0.02. We report the quality of the reconstruction of individual parts and of the overall object after merging them.

Results. We compare AutoPartGen to two recent methods: PartGen [5] and HoloPart [60]. We focus on mask-controlled part generation. Recall that HoloPart takes as input the overall partial 3D object and 3D part segmentations and outputs the complete parts. We adapt both PartGen and AutoPartGen

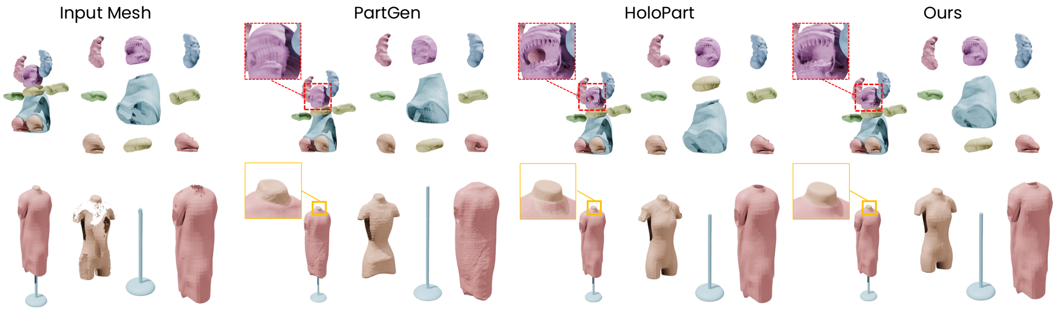

Figure 7: **Visual comparison of different completion methods.** Our approach achieves better geometric coherence by considering previously generated parts in context.

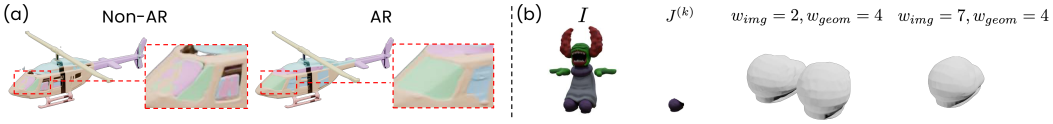

Figure 8: **Ablation Study.** (a) Without autoregressive generation, parts overlap and intersect. (b) Increasing image guidance w_{img} encourages the model to follow image mask, while a larger geometry guidance w_{geom} biases the generation towards a generic part distribution and order in the data.

to solve the same problem. For AutoPartGen, we provide the overall partial 3D mesh and 2D part masks (a variant of masks-to-parts). For PartGen, we supply four masked views of each part, which are compatible with its input requirements.

The results in Table 1 show that HoloPart outperforms PartGen, likely due to its access to more comprehensive input information (the partial 3D object and 3D part masks). Nevertheless, AutoPartGen surpasses both baselines across all key metrics: IoU, F-score, and Chamfer Distance. This advantage holds true for both part completion and overall object reconstruction, indicating that AutoPartGen generates geometrically precise parts that form a well-formed and coherent whole.

5 Ablation Study

We ablate three factors in AutoPartGen: the autoregressive design, the guidance scale, and the re-encoding strategy. We provide qualitative evidence for the first two factors and report quantitative results for all three.

Autoregressive modeling. We evaluate the contribution of the autoregressive design by removing the autoregressive condition $\boldsymbol{z}^{(1,\dots,k-1)}$ in the masks-to-part setting. This is the only setting where removal is possible, since external part guidance specifies which part to generate next and when to stop. As shown in Figure 8(a), removing this conditioning causes parts to overlap and intersect. Quantitatively, Table 2 shows that enabling autoregression improves IoU and F-score, and reduces Chamfer Distance (CD), indicating better geometric fidelity and coherence.

Effectiveness of guidance. We analyze the impact of *image guidance* w_{img} and *geometry guidance* w_{geom} in masks-to-parts generation. Because image and geometry conditions are randomly dropped during training, the model learns a data-driven prior for part partitioning and ordering when no image condition is present. As shown in Figure 8(b), increasing w_{img} aligns parts more closely with input image masks, while a higher w_{geom} biases generation toward the learned prior of plausible part structures. Quantitatively, Table 3 shows that moderate guidance values (for example, w_{geom}=4 and w_{img}=7) strike the best balance, achieving peak performance in IoU, F-score, and Chamfer Distance.

Effect of re-encoding. We further compare three ways to aggregate the VecSet tokens of different parts: (i) *Re-encoding*, which first decodes different parts into meshes, concatenates them, and re-encodes the concatenated mesh into a fixed-length latent; (ii) *Concat*, which directly feeds the concatenated latents into the cross-attention layer of the diffusion model; and (iii) *Latent fuser*, use a 6-layer Perceiver-style module that fuses the tokens into a fix length of 512 latent tokens. All models

Table 2: **Ablation study on autoregressive.** Autoregressive generation clearly show better results in terms of the part completion and overall object coherence. The models are only trained for 200 epochs. Here CD denotes Chamfer Distance.

Autoregressive	**Part Completion**			**Overall**		
	IoU ↑	F-Score ↑	CD ↓	IoU ↑	F-Score ↑	CD ↓
✗	0.574	0.795	0.067	0.783	0.917	0.031
✓	0.633	0.825	0.052	0.811	0.934	0.022

Table 3: **Effects of different guidance scales**. We report the three metrics on the part completion task, where CD denotes the Chamfer Distance.

Geometry/Image	**5**			**7**			**9**		
	IoU	F-score	CD	IoU	F-score	CD	IoU	F-score	CD
2.5	0.650	0.847	0.051	0.639	0.839	0.053	0.632	0.833	0.052
4	0.657	0.854	0.050	0.665	0.861	0.047	0.648	0.852	0.052
5	0.635	0.841	0.062	0.662	0.857	0.049	0.647	0.851	0.051

Table 4: **Effect of re-encoding**. All models are trained for 150 epochs with 512 tokens per part under the same setup.

Method	IoU ↑	F-Score ↑	Chamfer Distance ↓
Re-encoding	0.627	0.815	0.055
Concat	0.611	0.804	0.059
Latent Fuser	0.608	0.802	0.061

are trained for 150 epochs with 512 tokens per part under the same setup. As summarized in Table 4, all three methods perform similarly, with re-encoding slightly outperforms the two other strategies in terms of IoU, F-score, and CD.

5.1 Applications

3D Scene Generation. Our method's capability for decomposable object generation naturally extends to entire 3D scenes. As illustrated in Figure 1 (middle row), given an isometric view of a small scene, our approach automatically generates individual scene objects such as chairs, clocks, plants, and tables in a decomposable manner. This decomposable nature facilitates flexible manipulation and editing of individual scene components.

City Generation. Figure 1 (bottom row) further showcases our method's potential for large-scale outdoor scene generation. Drawing inspiration from SynCity [12], we employ a text-to-image generator to produce diverse tile images from text prompts. These tiles are subsequently assembled to form coherent cityscapes, enabling scalable and controllable generation of complex urban environments. Further examples and qualitative demonstrations are provided in the supplementary material.

6 Conclusion

We introduced AutoPartGen, an autoregressive model for compositional 3D part generation. By leveraging latent 3D representations, our method generates coherent object parts sequentially. The same model can handle different input types, including images, 2D masks, and 3D meshes. AutoPartGen outperforms existing methods in part completion and coherence while simplifying the overall pipeline. Our experiments demonstrate its effectiveness across various tasks and applications, highlighting its potential for scalable and controllable 3D content creation.

References

[1] Hertz Amir, Perel Or, Giryes Raja, Sorkine-Hornung Olga, and Cohen-Or Daniel. SPAGHETTI: editing implicit shapes through part aware generation. In *ACM Transactions on Graphics*, 2022.

[2] Yash Sanjay Bhalgat, Iro Laina, Joao F. Henriques, Andrea Vedaldi, and Andrew Zisserman. Contrastive Lift: 3D object instance segmentation by slow-fast contrastive fusion. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[3] Yash Sanjay Bhalgat, Iro Laina, Joao F. Henriques, Andrew Zisserman, and Andrea Vedaldi. N2F2: Hierarchical scene understanding with nested neural feature fields. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2024.

[4] Aleksei Bokhovkin and Angela Dai. Neural part priors: Learning to optimize part-based object completion in rgb-d scans. In *Proc. CVPR*, 2023.

[5] Minghao Chen, Roman Shapovalov, Iro Laina, Jianyuan Wang Tom Monnier, David Novotny, and Andrea Vedaldi. PartGen: Part-level 3d generation and reconstruction with multi-view diffusion models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

[6] Rui Chen, Jianfeng Zhang, Yixun Liang, Guan Luo, Weiyu Li, Jiarui Liu, Xiu Li, Xiaoxiao Long, Jiashi Feng, and Ping Tan. Dora: Sampling and benchmarking for 3D shape variational auto-encoders. *arXiv*, 2412.17808, 2024.

[7] Yilun Chen, Shuai Yang, Haifeng Huang, Tai Wang, Runsen Xu, Ruiyuan Lyu, Dahua Lin, and Jiangmiao Pang. Grounded 3d-llm with referent tokens. *arXiv preprint arXiv:2405.10370*, 2024.

[8] Yongwei Chen, Tengfei Wang, Tong Wu, Xingang Pan, Kui Jia, and Ziwei Liu. Comboverse: Compositional 3d assets creation using spatially-aware diffusion guidance. In *Proc. ECCV*, 2024.

[9] Jaeyoung Chung, Suyoung Lee, Hyeongjin Nam, Jaerin Lee, and Kyoung Mu Lee. LucidDreamer: Domain-free generation of 3d gaussian splatting scenes. In *arXiv*, 2023.

[10] Ken Deng, Yuanchen Guo, Jingxiang Sun, Zixin Zou, Yangguang Li, Xin Cai, Yanpei Cao, Yebin Liu, and Ding Liang. DetailGen3D: generative 3D geometry enhancement via data-dependent flow. *arXiv*, 2411.16820, 2024.

[11] Laura Downs, Anthony Francis, Nate Koenig, Brandon Kinman, Ryan Hickman, Krista Reymann, Thomas B. McHugh, and Vincent Vanhoucke. Google Scanned Objects: A high-quality dataset of 3D scanned household items. In *Proc. ICRA*, 2022.

[12] Paul Engstler, Aleksandar Shtedritski, Iro Laina, Christian Rupprecht, and Andrea Vedaldi. SynCity: training-free generation of 3d worlds. *arXiv*, 2503.16420, 2025.

[13] Jun Gao, Tianchang Shen, Zian Wang, Wenzheng Chen, Kangxue Yin, Daiqing Li, Or Litany, Zan Gojcic, and Sanja Fidler. GET3D: A generative model of high quality 3D textured shapes learned from images. *arXiv.cs*, abs/2209.11163, 2022.

[14] Ruiqi Gao, Emiel Hoogeboom, Jonathan Heek, Valentin De Bortoli, Kevin P. Murphy, and Tim Salimans. Diffusion meets flow matching: Two sides of the same coin, 2024.

[15] Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas A. Funkhouser. Local deep implicit functions for 3D shape. In *Proc. CVPR*, 2020.

[16] Kyle Genova, Forrester Cole, Daniel Vlasic, Aaron Sarna, William T. Freeman, and Thomas Funkhouser. Learning shape templates with structured implicit functions. In *Proc. CVPR*, 2019.

[17] Junlin Han, Filippos Kokkinos, and Philip Torr. Vfusion3d: Learning scalable 3d generative models from video diffusion models. In *European Conference on Computer Vision*, pages 333–350. Springer, 2024.

[18] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *Proc. NeurIPS*, 2021.

[19] Yicong Hong, Kai Zhang, Jiuxiang Gu, Sai Bi, Yang Zhou, Difan Liu, Feng Liu, Kalyan Sunkavalli, Trung Bui, and Hao Tan. LRM: Large reconstruction model for single image to 3D. In *Proc. ICLR*, 2024.

[20] Zehuan Huang, Yuan-Chen Guo, Xingqiao An, Yunhan Yang, Yangguang Li, Zi-Xin Zou, Ding Liang, Xihui Liu, Yan-Pei Cao, and Lu Sheng. Midi: Multi-instance diffusion for single image to 3d scene generation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, 2025.

[21] Ka-Hei Hui, Ruihui Li, Jingyu Hu, and Chi-Wing Fu. Neural template: Topology-aware reconstruction and disentangled generation of 3d meshes. In *Proc. CVPR*, 2022.

[22] Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, Olivier J. Hénaff, Matthew M. Botvinick, Andrew Zisserman, Oriol Vinyals, and João Carreira. Perceiver IO: A general architecture for structured inputs & outputs. In *Proc. ICLR*, 2022.

[23] Bharath Raj Nagoor Kani, Hsin-Ying Lee, Sergey Tulyakov, and Shubham Tulsiani. UpFusion: novel view diffusion from unposed sparse view observations. *arXiv*, 2024.

[24] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian Splatting for real-time radiance field rendering. *Proc. SIGGRAPH*, 42(4), 2023.

[25] Justin Kerr, Chung Min Kim, Ken Goldberg, Angjoo Kanazawa, and Matthew Tancik. LERF: language embedded radiance fields. In *Proc. ICCV*, 2023.

[26] Chung Min Kim, Mingxuan Wu, Justin Kerr, Ken Goldberg, Matthew Tancik, and Angjoo Kanazawa. Garfield: Group anything with radiance fields. *arXiv.cs*, abs/2401.09419, 2024.

[27] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything. In *Proc. CVPR*, 2023.

[28] Sosuke Kobayashi, Eiichi Matsumoto, and Vincent Sitzmann. Decomposing NeRF for editing via feature field distillation. In *Proc. NeurIPS*, 2022.

[29] Juil Koo, Seungwoo Yoo, Minh Hieu Nguyen, and Minhyuk Sung. SALAD: part-level latent diffusion for 3D shape generation and manipulation. In *Proc. ICCV*, 2023.

[30] Weiyu Li, Jiarui Liu, Rui Chen, Yixun Liang, Xuelin Chen, Ping Tan, and Xiaoxiao Long. CraftsMan: high-fidelity mesh generation with 3d native generation and interactive geometry refiner. *arXiv*, 2405.14979, 2024.

[31] Yangguang Li, Zi-Xin Zou, Zexiang Liu, Dehu Wang, Yuan Liang, Zhipeng Yu, Xingchao Liu, Yuan-Chen Guo, Ding Liang, Wanli Ouyang, and Yan-Pei Cao. TripoSG: high-fidelity 3D shape synthesis using large-scale rectified flow models. *arXiv*, 2502.06608, 2025.

[32] Connor Lin, Niloy Mitra, Gordon Wetzstein, Leonidas J. Guibas, and Paul Guerrero. NeuForm: adaptive overfitting for neural shape editing. In *Proc. NeurIPS*, 2022.

[33] Shanchuan Lin, Bingchen Liu, Jiashi Li, and Xiao Yang. Common diffusion noise schedules and sample steps are flawed. *arXiv.cs*, abs/2305.08891, 2023.

[34] Anran Liu, Cheng Lin, Yuan Liu, Xiaoxiao Long, Zhiyang Dou, Hao-Xiang Guo, Ping Luo, and Wenping Wang. Part123: Part-aware 3d reconstruction from a single-view image. *arXiv*, 2405.16888, 2024.

[35] Minghua Liu, Yinhao Zhu, Hong Cai, Shizhong Han, Zhan Ling, Fatih Porikli, and Hao Su. PartSLIP: low-shot part segmentation for 3D point clouds via pretrained image-language models. In *Proc. CVPR*, 2023.

[36] Ruoshi Liu, Rundi Wu, Basile Van Hoorick, Pavel Tokmakov, Sergey Zakharov, and Carl Vondrick. Zero-1-to-3: Zero-shot one image to 3D object. In *Proc. ICCV*, 2023.

[37] Yuan Liu, Cheng Lin, Zijiao Zeng, Xiaoxiao Long, Lingjie Liu, Taku Komura, and Wenping Wang. SyncDreamer: Generating multiview-consistent images from a single-view image. *arXiv*, 2309.03453, 2023.

[38] Tom Monnier, Jake Austin, Angjoo Kanazawa, Alexei Efros, and Mathieu Aubry. Differentiable blocks world: Qualitative 3d decomposition by rendering primitives. In *Proc. NeurIPS*, 2023.

[39] George Kiyohiro Nakayama, Mikaela Angelina Uy, Jiahui Huang, Shi-Min Hu, Ke Li, and Leonidas Guibas. DiffFacto: controllable part-based 3D point cloud generation with cross diffusion. In *Proc. ICCV*, 2023.

[40] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel HAZIZA, Francisco Massa, Alaaeldin El-Nouby, Mido Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*, 2024.

[41] 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.

[42] Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. DreamFusion: Text-to-3D using 2D diffusion. In *Proc. ICLR*, 2023.

[43] Minghan Qin, Wanhua Li, Jiawei Zhou, Haoqian Wang, and Hanspeter Pfister. LangSplat: 3D language Gaussian splatting. In *Proc. CVPR*, 2024.

[44] Viktor Rudnev, Mohamed Elgharib, William Smith, Lingjie Liu, Vladislav Golyanik, and Christian Theobalt. NeRF for outdoor scene relighting. In *Proc. ECCV*, 2021.

[45] Yichun Shi, Peng Wang, Jianglong Ye, Mai Long, Kejie Li, and Xiao Yang. MVDream: Multi-view diffusion for 3D generation. In *Proc. ICLR*, 2024.

[46] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *Proc. ICLR*, 2021.

[47] Jingxiang Sun, Bo Zhang, Ruizhi Shao, Lizhen Wang, Wen Liu, Zhenda Xie, and Yebin Liu. DreamCraft3D: Hierarchical 3D generation with bootstrapped diffusion prior. *arXiv.cs*, abs/2310.16818, 2023.

[48] Stanislaw Szymanowicz, Christian Rupprecht, and Andrea Vedaldi. Splatter Image: Ultra-fast single-view 3D reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[49] Nicolas Talabot, Olivier Clerc, Arda Cinar Demirtas, Doruk Oner, and Pascal Fua. PartSDF: part-based implicit neural representation for composite 3d shape parametrization and optimization. *arXiv*, 2502.12985, 2025.

[50] Jiaxiang Tang, Zhaoxi Chen, Xiaokang Chen, Tengfei Wang, Gang Zeng, and Ziwei Liu. LGM: Large multi-view Gaussian model for high-resolution 3D content creation. *arXiv*, 2402.05054, 2024.

[51] Jiaxiang Tang, Jiawei Ren, Hang Zhou, Ziwei Liu, and Gang Zeng. DreamGaussian: Generative gaussian splatting for efficient 3D content creation. *arXiv*, 2309.16653, 2023.

[52] Konstantinos Tertikas, Despoina Paschalidou, Boxiao Pan, Jeong Joon Park, Mikaela Angelina Uy, Ioannis Z. Emiris, Yannis Avrithis, and Leonidas J. Guibas. PartNeRF: Generating part-aware editable 3D shapes without 3D supervision. *arXiv.cs*, abs/2303.09554, 2023.

[53] Vadim Tschernezki, Iro Laina, Diane Larlus, and Andrea Vedaldi. Neural Feature Fusion Fields: 3D distillation of self-supervised 2D image representation. In *Proceedings of the International Conference on 3D Vision (3DV)*, 2022.

[54] Zhengyi Wang, Cheng Lu, Yikai Wang, Fan Bao, Chongxuan Li, Hang Su, and Jun Zhu. ProlificDreamer: High-fidelity and diverse text-to-3D generation with variational score distillation. *arXiv.cs*, abs/2305.16213, 2023.

[55] Daniel Watson, William Chan, Ricardo Martin-Brualla, Jonathan Ho, Andrea Tagliasacchi, and Mohammad Norouzi. Novel view synthesis with diffusion models. In *Proc. ICLR*, 2023.

[56] Xinyue Wei, Kai Zhang, Sai Bi, Hao Tan, Fujun Luan, Valentin Deschaintre, Kalyan Sunkavalli, Hao Su, and Zexiang Xu. MeshLRM: large reconstruction model for high-quality mesh. *arXiv*, 2404.12385, 2024.

[57] Jiale Xu, Weihao Cheng, Yiming Gao, Xintao Wang, Shenghua Gao, and Ying Shan. InstantMesh: efficient 3D mesh generation from a single image with sparse-view large reconstruction models. *arXiv*, 2404.07191, 2024.

[58] Yinghao Xu, Zifan Shi, Wang Yifan, Hansheng Chen, Ceyuan Yang, Sida Peng, Yujun Shen, and Gordon Wetzstein. GRM: Large gaussian reconstruction model for efficient 3D reconstruction and generation. *arXiv*, 2403.14621, 2024.

[59] 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*, 2502.14247, 2025.

[60] Yunhan Yang, Yuan-Chen Guo, Yukun Huang, Zi-Xin Zou, Zhipeng Yu, Yangguang Li, Yan-Pei Cao, and Xihui Liu. HoloPart: generative 3d part amodal segmentation. *arXiv*, 2504.07943, 2025.

[61] Yunhan Yang, Yukun Huang, Yuan-Chen Guo, Liangjun Lu, Xiaoyang Wu, Edmund Y Lam, Yan-Pei Cao, and Xihui Liu. Sampart3d: Segment any part in 3d objects. *arXiv preprint arXiv:2411.07184*, 2024.

[62] Kaixin Yao, Longwen Zhang, Xinhao Yan, Yan Zeng, Qixuan Zhang, Lan Xu, Wei Yang, Jiayuan Gu, and Jingyi Yu. CAST: component-aligned 3D scene reconstruction from an RGB image. *arXiv*, 2502.12894, 2025.

[63] Chongjie Ye, Yushuang Wu, Ziteng Lu, Jiahao Chang, Xiaoyang Guo, Jiaqing Zhou, Hao Zhao, and Xiaoguang Han. Hi3DGen: High-fidelity 3D geometry generation from images via normal bridging. *arXiv*, 2025.

[64] Haiyang Ying, Yixuan Yin, Jinzhi Zhang, Fan Wang, Tao Yu, Ruqi Huang, and Lu Fang. Omniseg3d: Omniversal 3d segmentation via hierarchical contrastive learning. In *Proc. CVPR*, 2024.

[65] Biao Zhang, Jiapeng Tang, Matthias Niessner, and Peter Wonka. 3DShape2VecSet: A 3D shape representation for neural fields and generative diffusion models. In *ACM Transactions on Graphics*, 2023.

[66] 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. *arXiv*, 2024.

[67] 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 *Proc. NeurIPS*, 2023.

[68] Shuaifeng Zhi, Tristan Laidlow, Stefan Leutenegger, and Andrew J. Davison. In-place scene labelling and understanding with implicit scene representation. In *Proc. ICCV*, 2021.

[69] Junsheng Zhou, Yu-Shen Liu, and Zhizhong Han. Zero-shot scene reconstruction from single images with deep prior assembly. *Advances in Neural Information Processing Systems*, 2024.

[70] Junsheng Zhou, Jinsheng Wang, Baorui Ma, Yu-Shen Liu, Tiejun Huang, and Xinlong Wang. Uni3D: Exploring unified 3D representation at scale. In *Proc. ICLR*, 2024.

[71] Yuchen Zhou, Jiayuan Gu, Xuanlin Li, Minghua Liu, Yunhao Fang, and Hao Su. PartSLIP++: enhancing low-shot 3d part segmentation via multi-view instance segmentation and maximum likelihood estimation. *arXiv*, 2312.03015, 2023.

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]

   Justification: As mentioned in the abstract and introduction, we propose a model AutoPartGen that can generate objects composed of 3D parts in an autoregressive manner. Quantitative and qualitative results indicate that AutoPartGen is efficient and accurate.

   Guidelines:

   - The answer NA 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 NA 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: We discuss the Limitations in detail in the Supplementary Materials.

   Guidelines:

   - The answer NA 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: [NA]

Justification: Our work does not include any theoretical claims or results that require formal proofs.

Guidelines:

- The answer NA 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: We provide enough information to reproduce our results. We provide more details in the Supplementary Material.

   Guidelines:

   - The answer NA 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: We will strive to release the code and model weights as open source. However, our data license does not allow us to release the data for training the model. Other authors should be able to obtain a similar dataset using resources like Objaverse.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://nips.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://nips.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, etc.) necessary to understand the results?

Answer: [Yes]

Justification: We provide such implementation details in the Experiment section and the Supplementary Material.

Guidelines:

- The answer NA 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: We are unable to randomize experiments due to the cost of retraining the models for each run.

Guidelines:

- The answer NA 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 provide these details in the Experiment section and the Supplementary Material.

   Guidelines:

   - The answer NA 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: We strictly follow the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA 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: [Yes]

    Justification: We provide a discussion of Broader impacts in the Supplementary Material.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.

- If the authors answer NA 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., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA] Our model is unlikely to present any significant direct risk of misuse.

    Guidelines:

    - The answer NA 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: We use a dataset comprising artist-created 3D meshes and commercially licensed from them.

    Guidelines:

    - The answer NA 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: [NA]

    Justification: We do not release new assets.

    Guidelines:

    - The answer NA 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: [NA]

    Justification: Our work does not involve crowdsourcing experiments or research with human subjects.

    Guidelines:

    - The answer NA 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: [NA]

    Justification: Our work does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA 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 rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer NA 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 (https://neurips.cc/Conferences/2025/LLM) for what should or should not be described.

Supplementary Material

This supplementary material provides additional details and results to complement the main paper. It includes the following sections:

- **Implementation Details:** Detailed descriptions of model architectures, training and inference procedures, and evaluation protocols.
- **Additional Comparisons with PartGen:** Extended evaluation against PartGen.
- **3D Scene Generation:** Additional qualitative examples for scene generation.
- **City Generation:** Details of the city generation pipeline and additional visual results.
- **Failure Case:** Visualization of failure cases in AutoPartGen.
- **Limitations and Broader Impact:** Discussion of limitations and potential societal implications of AutoPartGen.

A Implementation Details

A.1 Training

Our model consists of two primary components: a 3D Variational Autoencoder (VAE) and a diffusion model.

3D Variational Autoencoder. We adopt the 3D representation from 3DShape2VecSet, extending it to a larger model capacity compared to the original VAE [65]. Our VAE architecture comprises an 8-layer encoder with a dimension of 768 and a 16-layer decoder with a dimension of 1024. The model is trained on approximately 1.7M 3D assets, with data augmentation techniques including point cloud rotations, as suggested by Dora [6]. We employ a signed distance function (SDF) representation for smoother isosurface extraction. During training, we supervise the VAE using a combination of surface normal loss, Eikonal loss, and KL divergence regularization, weighted by 10, 0.1, and 0.001, respectively, following TripoSG [31]. To learn a single signed distance field, we calculate a combination of L1 and MSE loss on a total of 24,576 points per shape: 8192 each from surface points, near-surface points, and randomly in the volume. We randomly vary the number of input tokens between {512, 2048} during training. The model is optimized using AdamW with a learning rate of 1e$-$4, linearly warmed up from 1e$-$5 over the first 3 epochs. We use a batch size of 1536 and set the weight decay to 0.01. Training is conducted on 128 NVIDIA H100 GPUs for 150 epochs.

Pretraining and fine-tuning. For diffusion training, we first pretrain a general image-to-3D model on the same 1.7M assets. The diffusion backbone follows DiT [41], with 24 transformer layers and hidden dimension 2048. We train with a fixed token length of 512 for 300 epochs, learning rate 1e$-$4, and batch size 10 per GPU on 128 GPUs. We then fine-tune the model to additionally condition on masked image and geometry tokens on the part dataset in an autoregressive manner for approximately 300k steps. The image condition is encoded with DINO-v2 [40], and the geometry token is encoded with our trained 3D VAE. To reduce computation, geometry tokens are used only in the first 12 transformer layers. We apply condition dropping with probability 0.05 independently for the different combination of inputs simultaneously. Fine-tuning uses AdamW with weight decay 0.01 and batch size 6 per GPU on 128 GPUs. Subsequently, we increase the token length to 2048 and continue training for an additional 100k steps on 256 GPUs with batch size 1 per GPU. We adopt the DDIM scheduler [46] with 1000 steps, use v-prediction, and a zero signal-to-noise ratio [33].

Ordering of parts. In the *masks-to-parts* setting, the order of the input masks specifies the order in which parts are generated. Both unmasked and masked images are provided so the model can learn spatial relationships between each part and the whole object in 3D space. In practice, the model can accurately infer very small parts (less than 0.1% of the object volume) from very small masks, and at inference time, users can interactively adjust the image guidance scale to control how strongly masks and images influence generation. In the *automatic setting*, a predefined order is used during training and followed at inference. Training assets are defined in a canonical space, and parts are sorted lexicographically by their axis-aligned bounding boxes: bottom to top (Z), then left to right (X), then front to back (Y), following Blender's ZXY axis convention. Concretely, the minimum Z values are compared first; if they are similar, the minimum X values are compared, and if still similar, the minimum Y values are used..

A.2 Inference

In all scenarios, we use 50 denoising steps during inference. For the object-to-parts setting, geometry guidance is set to 10, while image guidance is disabled (set to 0). For both the image-to-parts and masks-to-parts settings, we first perform image-to-3D reconstruction to obtain the overall object shape. When user-provided masks are available, we apply the default guidance setting, with image guidance set to 7 and geometry guidance set to 4.

A.3 Evaluation

The most relevant baseline to AutoPartGen is PartGen. We compare the two methods under the object-to-parts setting, without incorporating any user inputs. For this comparison, we use objects from the Google Scanned Objects (GSO) dataset [11].

When users provide masks to guide part partitioning, we compare our method with recent approaches, including HoloPart and PartGen. For evaluation, we use the PartObjaverse-Tiny dataset [61]. To exclude negligible parts, we filter out objects containing segments that occupy only a small fraction of the total object volume as in [60].

B Additional Comparison with PartGen

To further highlight the improvements over PartGen, we provide a qualitative comparison in Figure 9. As shown in the figure, AutoPartGen produces sharper and more detailed meshes, as highlighted by the red circle. Additionally, the autoregressive generation in AutoPartGen avoids over-generation issues seen in PartGen, which arise from its lack of explicit modeling of the joint distribution of different parts. This is a key capability addressed by AutoPartGen.

C 3D Scene Generation

Small scene generation is a natural extension of our method. Specifically, we begin by providing prompts such as “an isometric view of an office” or “an isometric view of a small bedroom” to an off-the-shelf 2D text-to-image generator to produce corresponding images. We first generate the overall shape of the small scene. Subsequently, we apply AutoPartGen again to decompose the scene into distinct components such as “chair”, “table” and so on. More qualitative examples are provided in Figure 10.

D City Generation

As shown in Figure 1 of the main paper, we demonstrate the ability of AutoPartGen to generate 3D cities by integrating it into the SynCity [37] pipeline, replacing its original 3D generator. Specifically, the pipeline begins with text prompts generated by a large language model, which are then used to guide a 2D image generator in creating isometric views of individual tiles, taking into account the context of neighboring tiles. These generated images are then passed to AutoPartGen to produce compositional 3D tiles.

We present additional examples in Figure 11. As illustrated, AutoPartGen can be seamlessly integrated into the city generation pipeline, generating different cities, such as a ’medieval town‘ or a “solarpunk city”. For further details on prompt generation and 2D image synthesis, please refer to the SynCity paper.

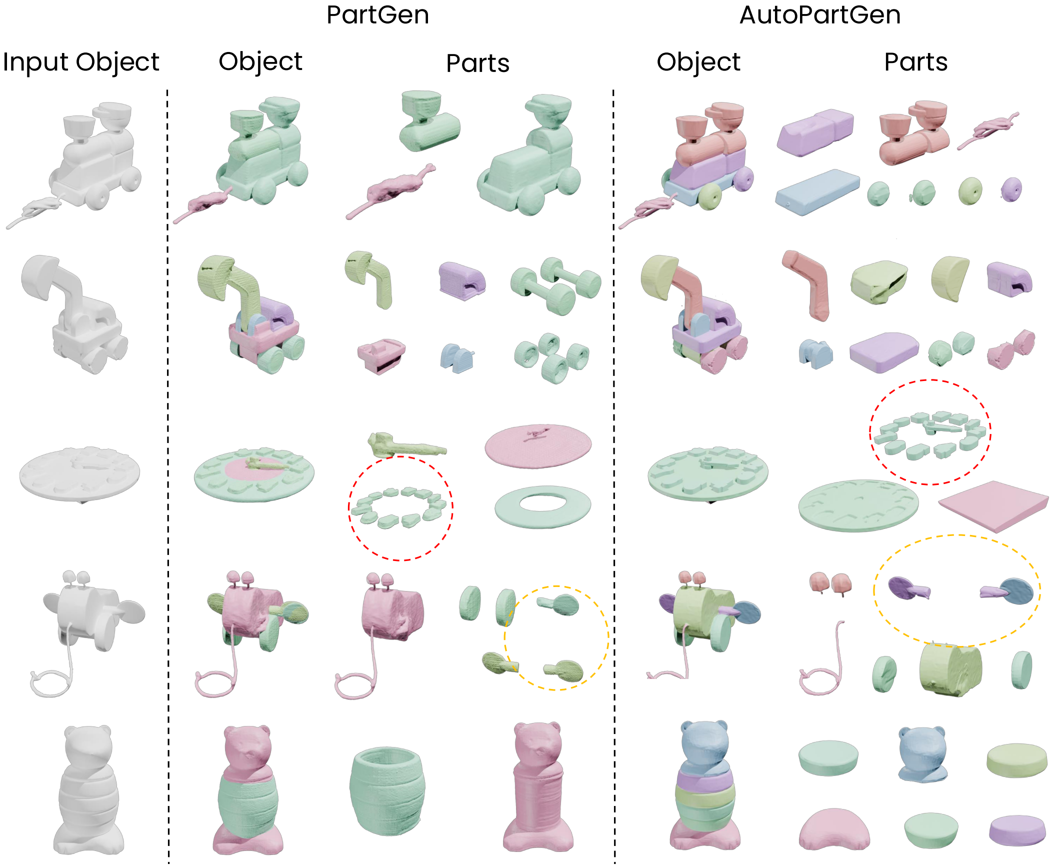

Figure 9: **Comparison between AutoPartGen and PartGen.** AutoPartGen produces more accurate geometry, as highlighted in the red circle. Additionally, its autoregressive generation prevents the overlapping parts observed in PartGen, as shown in the yellow circle.

E Failure Cases

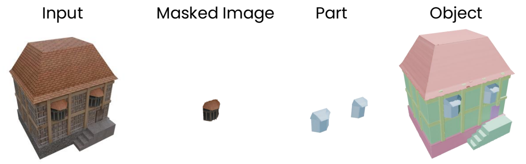

Figure 12: **Failure Case**. When there are identical parts, the model sometimes will try to predict the parts together even if masks are given.

We present a failure case in Figure 12. As shown in the figure, when the object contains identical parts, the model attempts to predict multiple parts together, even when only one is masked. We conjecture that this behavior comes from the bias in the training data, where some artists may have grouped identical parts into a single mesh. Additionally, as discussed in the ablation study, users may adjust the guidance scale to enforce stronger adherence to the image, which can amplify this effect. Due to the model's autoregressive nature, each prediction depends on previously generated parts. In the example shown in Figure 12, where both window instances are generated together, when the mask for the second window is given as input, the model's prediction will be (nearly) empty, since the model has already generated that content. Hence, despite this potential failure mode, the model produces coherent results, maintaining consistency in the overall shape.

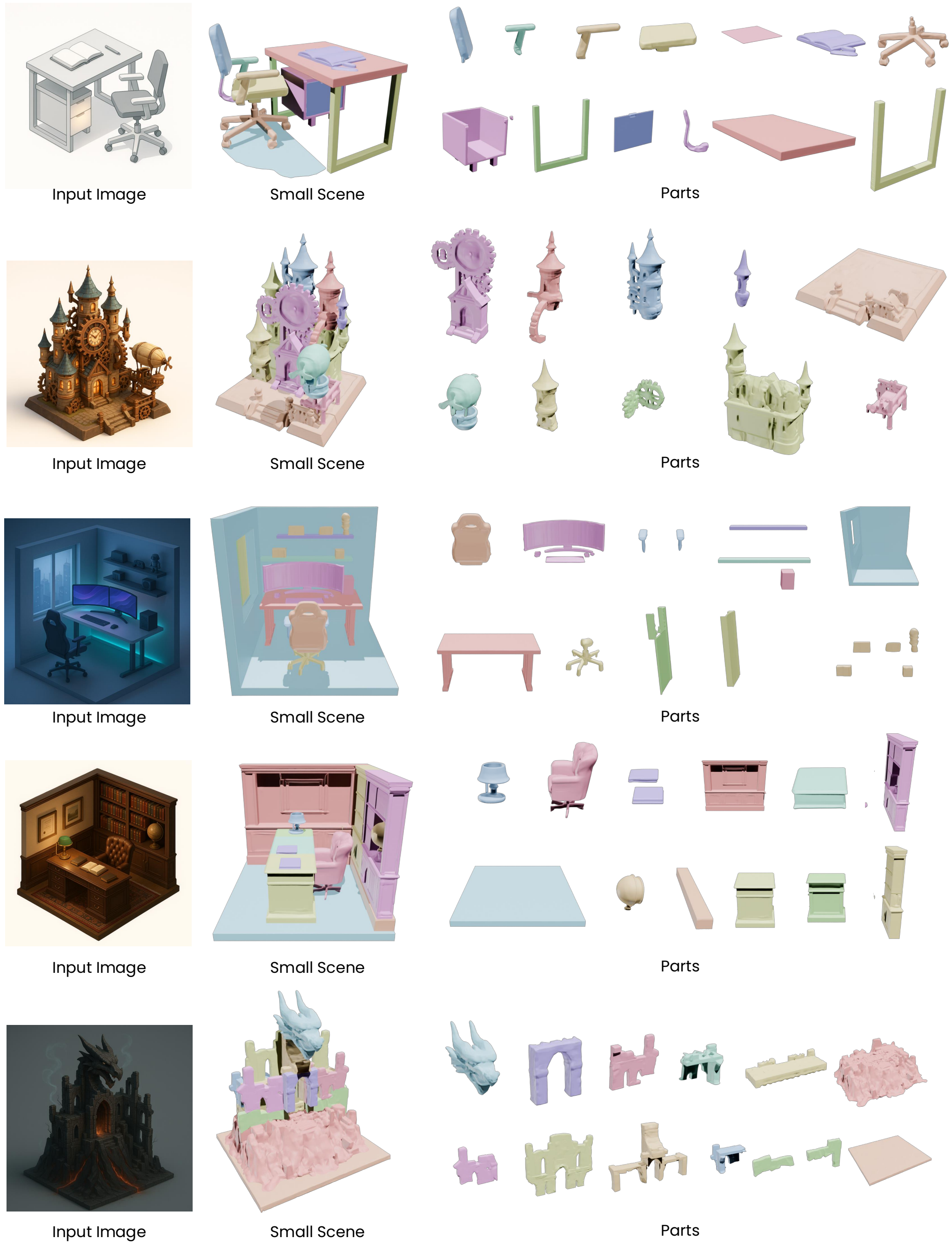

Figure 10: **3D scene generation.** AutoPartGen generates 3D scenes while decomposing them into their constituent elements. The input images are generated by a 2D text-to-image generator.

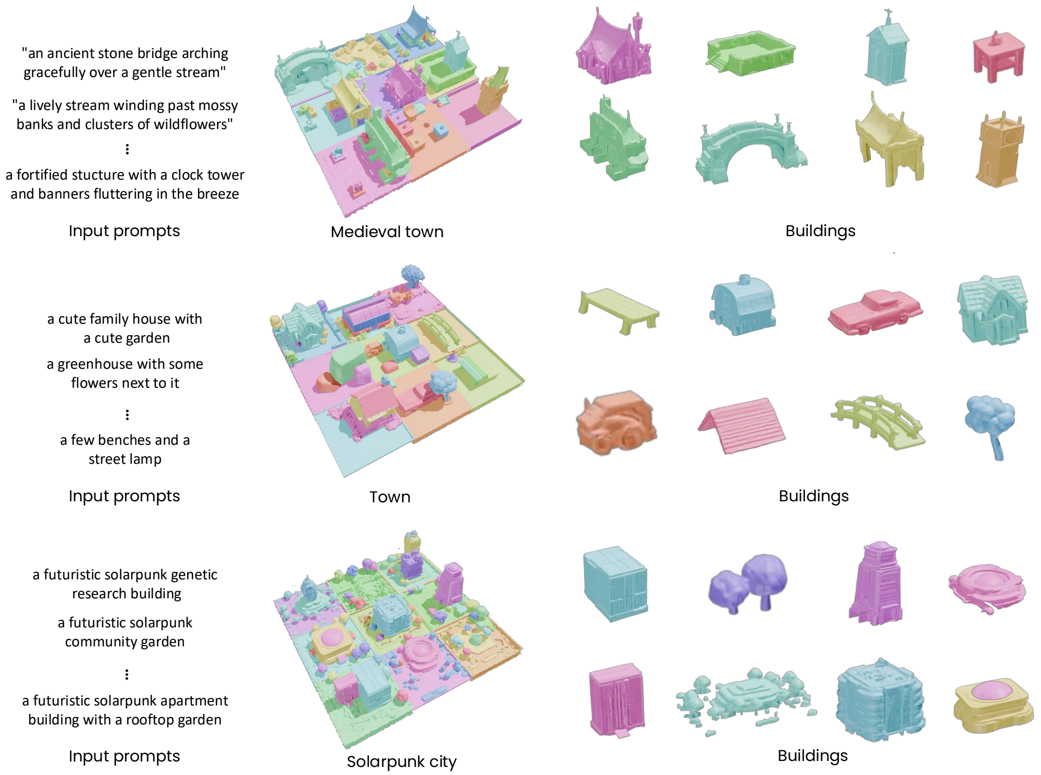
"an ancient stone bridge arching gracefully over a gentle stream"
"a lively stream winding past mossy banks and clusters of wildflowers"
⋮
a fortified stucture with a clock tower and banners fluttering in the breeze
Input prompts
Medieval town
Buildings
a cute family house with a cute garden
a greenhouse with some flowers next to it
⋮
a few benches and a street lamp
Input prompts
Town
Buildings
a futuristic solarpunk genetic research building
a futuristic solarpunk community garden
⋮
a futuristic solarpunk apartment building with a rooftop garden
Input prompts
Solarpunk city
Buildings

Figure 11: **City Generation.** We showcase AutoPartGen on larger scenes by integrating it within Syncity [12]. From top to bottom, the images depict a medieval town, a cozy town, and a solarpunk city.

F Limitations and Boarder Impact

Limitations. While AutoPartGen demonstrates strong performance across all three scenarios, it also has some limitations that point to potential directions for future improvement. First, the current model can only generate bounded scenes, as it inherits the spatial constraints from the underlying VAE latent space. Extending the framework to support unbounded world generation, where scenes can grow or evolve without a predefined spatial limit, would be both an interesting challenge and a promising research direction. Second, the method currently lacks explicit control over the granularity of part partitioning, except in the masks-to-parts setting, where masks can be provided as a way of control. In the image-to-parts and object-to-parts settings, the decomposition of parts can vary. In future iterations, one may incorporate high-level controls for granularity levels, such as ‘simple‘, ‘medium‘, and ‘complicated‘, to make the system more flexible and interactive. Finally, the model learns part distributions directly from the training data, which introduces the risk of bias being inherited from the dataset.

Broader Impact. Although our model is trained on a large amount of data, it may still exhibit biases that reflect imbalances in the underlying distribution. These biases can influence downstream tasks and should be carefully evaluated before practical deployment. To mitigate potential misuse or harmful applications, we recommend implementing safeguards and conducting thorough audits to ensure responsible usage. Additionally, the training process requires substantial computational resources, particularly in terms of GPU usage. This raises concerns about energy consumption and the associated environmental impact, which should be taken into account when scaling or deploying the model in real-world settings.