MESHCARVE: ARTISAN MESH GENERATION WITH FLOW MATCHING IN COMPACT LATENT SPACES

Anonymous authors
Paper under double-blind review

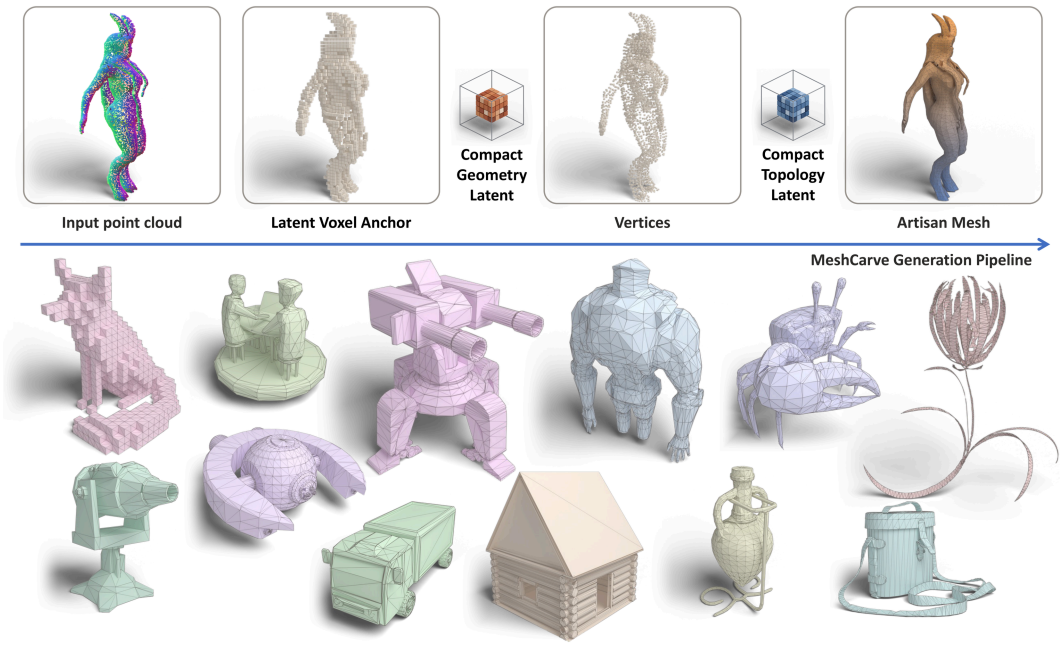

Figure 1: MeshCarve turns a dense point cloud into an artisan mesh with two flows on a generated anchor. The latent voxel anchor is generated with a variational autoencoder. The vertex flow then places the vertices, and the edge flow decides the connectivity between vertices. Both flows run in a spatially compressed 64^3 latent, drawn as the two compact cubes. Bottom: meshes generated by MeshCarve.

ABSTRACT

Prior artisan mesh generation works largely predict face tokens autoregressively, which makes inference slow. Recent methods instead flow match continuous latents built by Variational AutoEncoders (VAEs), but reconstruction quality drops significantly when geometry and topology are jointly encoded, and further when the latent space is compressed. We present **MeshCarve**, a flow matching method that generates entirely in compact latent spaces, generating vertex positions and edge connections separately and sidestepping the difficulty of a joint compact latent. To shorten the token sequence, we propose a hierarchical sparse transformer backbone, instantiated as VertexVAE and EdgeVAE. Instead of encoding fields over the surface voxels, both VAEs anchor on discrete vertices in their latent spaces, which drastically reduces the token sequence length, and our spatial-aware compression shortens it further without costing reconstruction. VertexVAE directly encodes vertex occupancy. For connectivity, we propose vertex-link encoding, which turns arbitrary connectivity between vertices into fixed-length continuous per-vertex embeddings and recovers complex artistic topology faithfully. MeshCarve combines these VAEs with an anchor generator and flow matches on the shortened token sequences. It shows advantages over state-of-the-art autoregressive and flow matching methods on Objaverse and generalizes to Toys4K. To the best of our knowledge, it is among the first artisan mesh generation methods

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

whose every generative stage runs in a spatially compressed latent, with a token sequence only a fraction of the most compressed previous autoregressive and flow matching works.

1 INTRODUCTION

3D mesh models are widely used as digital representations of 3D shapes. They are used in applications such as video games, visual effects, animation, and virtual reality. In practice, artisan meshes handcrafted by human artists with well-structured topology are particularly valuable: their compact geometry supports efficient rendering, while their carefully designed topology facilitates downstream tasks such as editing, deformation, and texture mapping. Hence, generating meshes with artistic geometry and topology is a central problem for 3D content creation (Siddiqui et al., 2024; Chen et al., 2025b; Weng et al., 2025; Zhao et al., 2025; 2026; Li et al., 2026b).

Many works model mesh generation as autoregressive generation of face tokens with transformers. Faces are tokenized and predicted one by one, starting from vertex-coordinate tokens (Nash et al., 2020; Siddiqui et al., 2024) and progressing through increasingly compact tokenizations that share vertices across adjacent faces (Chen et al., 2025c), walk edges (Tang et al., 2025), or index blocks and patches (Weng et al., 2025; Zhao et al., 2025). These efforts have greatly reduced the token budget for a given mesh, and the most recent methods reach one token per face (Wang et al., 2026a). Diffusion and flow matching methods (Alliegro et al., 2023; He et al., 2025; Shen et al., 2024) show greater potential, as they sidestep serialized generation, which is slow by nature. Among them, two recent works anchor the latent to the sparse structure of the mesh, one token per surface voxel (Zhao et al., 2026) or per vertex (Li et al., 2026b), and reach quality competitive with autoregressive generation. Both carry geometry and topology in one latent, and neither compresses it spatially. The surface-anchored latent keeps every surface voxel, and the vertex-anchored one reports a large reconstruction loss when its tokens are spatially down-sampled. The flow transformer therefore attends over a sequence that grows with the mesh, and its cost grows quadratically with it. *We ask whether flow matching can instead run entirely in a spatially compressed latent.* The token budget would then grow only sublinearly as the meshes size up.

The surface-anchored and the vertex-anchored latent each have further limitations that motivate our design. First, both jointly encode geometry and topology in a shared latent representation. Echoing the known limitations of shared latent spaces in multimodal VAEs (Daunhawer et al., 2022; Javaloy et al., 2022), our preliminary experiments show that such joint encoding substantially degrades the reconstruction quality of prior work (Zhao et al., 2026). Second, neither representation in its existing form is well suited for further compression. A surface-anchored latent assigns tokens to every surface voxel, even though surface voxels are several times more numerous than vertex-containing voxels. In existing vertex-anchored designs, topology is represented using a fixed-length ordered list of neighbors for each vertex, making the representation sensitive to neighbor ordering and imposing an upper bound on vertex degree. Consequently, the former contains substantial spatial redundancy, whereas the latter cannot merge vertex tokens without risking the loss of essential connectivity information. Importantly, this limitation arises from how topology is represented at the vertices, rather than from vertex anchoring itself.

MeshCarve addresses each of these limitations. Given an input point cloud, it first generates the coarse vertex occupancy with a VAE in a single forward pass and then generates vertex positions and edge connections sequentially through flow matching, each in its own spatially compressed latent space. These latent spaces are learned by VertexVAE and EdgeVAE, respectively, which use the same hierarchical sparse transformer VAE architecture built around a redesigned vertex-anchored representation. The architecture compresses the latents by merging the eight children of each voxel through a learned projection, allowing the token count to adapt to the local vertex density. For connectivity, EdgeVAE uses vertex-link encoding to embed the links from each vertex to its neighbors into a fixed-width, order-invariant representation, from which its decoder recovers edges through classification. Recent and concurrent works (Wang et al., 2026b; Long et al., 2026) also separate geometry from topology. MeshCarve further compresses both latents, preserving the reconstruction benefit of separation while shortening both generative sequences. Evaluated against artist references, MeshCarve shows advantages over prior state-of-the-art methods on Objaverse and generalizes to Toys4K.

Table 1: Tokens per triangle face, lower is a shorter sequence for the generative transformer. Target: G = geometry, T = topology, joint = one latent for both. Two-stage methods list both stages, and the two stages of MeshCarve share the same anchor and hence one value.

Autoregressive		Diffusion / Flow Matching			
Method	Tokens / face	Method	Token	Target	Tokens / face
MeshAnything (Chen et al., 2025b)	9	LATO (Zhao et al., 2026)	coarse surface voxel	joint	6.5
MeshGPT (Siddiqui et al., 2024)	6	LATO.2 (Long et al., 2026)	coarse surface voxel / vertex	G+T	0.88 / 0.32
EdgeRunner (Tang et al., 2025)	4.2	MeshFlow (Li et al., 2026b)	vertex	joint	0.52
MeshAnything V2 (Chen et al., 2025c)	4.1	Nexus (Wang et al., 2026b)	vertex	G+T	0.79 / 0.52
DeepMesh (Zhao et al., 2025)	2.5	MeshCraft (He et al., 2025)	face	joint	1.0
BPT (Weng et al., 2025)	2.3	MeshCarve (Ours)	coarse vertex voxel	G+T	**0.13**
TreeMeshGPT (Lionar et al., 2025)	2.0				
FACE (Wang et al., 2026a)	1.0				

In sum, our contributions are threefold:

- We propose MeshCarve, which samples the coarse vertex occupancy with a VAE in one forward pass and then flow matches vertex positioning and edge connection in turn, each in its own spatially compressed latent.
- We propose a hierarchical sparse transformer VAE backbone with **spatial-aware compression**, instantiated as the VertexVAE and the EdgeVAE, and the **vertex-link encoding**, which turns arbitrary connectivity into a fixed-length, order-free per-vertex embedding.
- The latent holds 0.13 tokens per face on average, the shortest among the compared methods. That budget affords a larger flow transformer, and MeshCarve shows advantages over previous state-of-the-art methods on Objaverse and generalizes to Toys4K.

2 Related Work

Autoregressive artisan mesh generation. PolyGen (Nash et al., 2020) and MeshGPT (Siddiqui et al., 2024) established the face-token paradigm: faces are tokenized and a transformer predicts them autoregressively. Subsequently, MeshAnything (Chen et al., 2025b) scales up based on them and tokenizes each face with 9 tokens, i.e., the 3D coordinates of its three vertices. Due to this inefficient tokenization, it is limited to generating meshes with a few hundred faces. Subsequent works (Chen et al., 2025c; Tang et al., 2025; Weng et al., 2025; Zhao et al., 2025; Lionar et al., 2025) improve the token efficiency progressively. Recent efforts (Wang et al., 2026a) reach one token per face with an MLP tokenizer. Nevertheless, a few challenges remain. First, artisan meshes frequently contain thousands or tens of thousands of faces. At one token per face, the token sequence is still unaffordably long and the generation can take very long. Second, the serialized generation is prone to accumulated errors and often fails to emit a mesh with complete geometry because the stopping token is emitted too early. MeshMosaic (Xu et al., 2026) thus proposes to generate block by block to decompose the generation process, but these generic limitations still apply.

Latent generative models for artisan meshes. Flow matching and diffusion models sidestep the serialization of autoregressive generation, since vertices, edges and faces are produced concurrently. The pioneering works PolyDiff and SpaceMesh (Alliegro et al., 2023; Shen et al., 2024) apply diffusion to triangle soups and to raw vertex coordinates, respectively, and MeshCraft (He et al., 2025) flow matches a latent of independent faces. Later works mix the paradigms. FastMesh (Kim et al., 2026) generates the vertex positions autoregressively and obtains the connections from a predictor in one shot. TriFlow (Li et al., 2026a) generates a topology field over a given surface and takes its vertices from that surface. Nexus (Wang et al., 2026b) generates the vertices by diffusion in raw voxel space, as subdivision and pruning, and then diffuses in latent space to generate the connections. Notably, LATO (Zhao et al., 2026) and MeshFlow (Li et al., 2026b) flow match in a continuous latent built by a Variational AutoEncoder (VAE) that encodes vertex positions and connections jointly. LATO anchors the latent to the surface voxels and MeshFlow to the vertices, and both reach quality competitive with autoregressive generation. Both also keep one token per element, so the latent grows with the mesh and flow matching in it is costly. Their joint encoding of vertex positions and connections further degrades reconstruction. In MeshCarve, we anchor flow matching to vertices and construct compact latent spaces that reduce the token sequence length, allowing a larger flow matching backbone to run comfortably.

3 METHOD

MeshCarve generates an artisan mesh in two stages (Figure 2). We propose an anchor generator that settles the latent voxel positions for flow matching. The vertex flow then generates a compact latent on the anchor, which decodes into the fine vertex occupancy. The edge flow generates a second compact latent on the same anchor, and this latent decodes into the connectivity. Both latents are encoded using the same VAE architecture, instantiated as the VertexVAE and the EdgeVAE. We describe the backbone and the two VAEs first and the pipeline second.

Throughout, we write a mesh as a pair $(\mathbf{V}, \mathbf{A})$, with $\mathbf{V} \in \mathbb{R}^{N\times 3}$ the N vertex positions in the unit cube and $\mathbf{A} \in \{0,1\}^{N\times N}$ the symmetric adjacency, $A_{ij} = 1$ when vertices i and j share an edge. Faces are triangular, not stored, and recovered as the 3-cycles of $\mathbf{A}$. Discretizing the unit cube into an R^3 grid records which voxels contain vertices as a binary occupancy $\mathbf{S}_R$, and an occupied voxel is a vertex voxel. Both stages of MeshCarve follow the latent-diffusion recipe (Rombach et al., 2022), a VAE (Kingma & Welling, 2014) mapping the mesh to a compact latent whose posterior is kept near a standard Gaussian and a flow model trained in that space. The reconstruction term of each VAE is given below and the flow objective in Appendix A.

3.1 OVERALL VAE DESIGN

Flow matching needs the mesh as a continuous latent, and a mesh carries two signals of different kinds: discrete vertex positions and combinatorial connectivity. Prior works such as LATO (Zhao et al., 2026) and MeshFlow (Li et al., 2026b) choose to encode both into one latent, costing reconstruction quality. Our preliminary studies also show similar failure modes. Notably, this is a known issue in the field of multi-modal VAEs (Daunhawer et al., 2022) and modality-specific latents recover the quality (Palumbo et al., 2023). We therefore encode vertex position and edge connection with separate VAEs that share one backbone.

To design the VAE, we point out that vertices are the fewest of a mesh's primitives (around 12-16% of surface voxels) and naturally fit into quantized voxel grids, so we anchor our VAEs to vertex voxels. We also observe that the key to running flow matching at a larger scale is to compress the latent space and shorten the token sequence, but current generation works (Li et al., 2026b) encounter a noticeable reconstruction barrier when compressing the latent space. To this end, we propose to compress spatially through a learned projection rather than by fixed-interval pooling (Li et al., 2026b). Notably, we found that convolutional networks operate inefficiently on vertices, which are highly sparse. Thus, our VAE is fully based on sparse transformers that assign 3D positional embeddings to sparse voxels.

Spatial-Aware Compression In recent years, with the rapid improvement of hardware, e.g., GPUs, applied artisan meshes are also growing larger. With vertex-anchored tokens, we can shorten the token sequence considerably, but it remains difficult to afford a larger flow matching network because the vertex-anchored token sequence still grows linearly as the demanded mesh sizes up. Worse, the cost of running mainstream DiT networks grows quadratically with the token sequence length. MeshFlow (Li et al., 2026b) proposed TokenMerge, which merges tokens that are consecutive in the vertex index and projects them down. Nevertheless, this compression trades reconstruction significantly. To this end, we point out that compression should attend to the spatial relationship between voxels and propose spatial-aware compression. Specifically, inspired by Chen et al. (2025a); Xiang et al. (2026), eight children of a parent voxel are stacked channel-wise and then a linear mapping is learned to reach the dimension of a single voxel, reaching a coarser level. The spatial-aware compression and its mirrored decompression are defined as

$$\mathbf{h}_c = \sum_{k=1}^{8} \mathbf{W}_k^{\downarrow}\, \mathbf{h}_{c,k}, \qquad \mathbf{f}_{c,k} = \mathbf{W}_k^{\uparrow}\, \mathbf{f}_c, \quad k = 1, \ldots, 8, \tag{1}$$

where c is a parent voxel, c, k its child in octant k, $\mathbf{h}$ and $\mathbf{f}$ are encoder and decoder features, and $\mathbf{W}_k^{\downarrow}, \mathbf{W}_k^{\uparrow} \in \mathbb{R}^{D\times D}$ are learned per octant. Empty children drop out of the sum on the way down, and only occupied children are instantiated on the way up. The compression is therefore spatial-aware, and the token count follows the local vertex density rather than the grid resolution. In both VertexVAE and EdgeVAE the spatial-aware compression is applied after transformer blocks. It

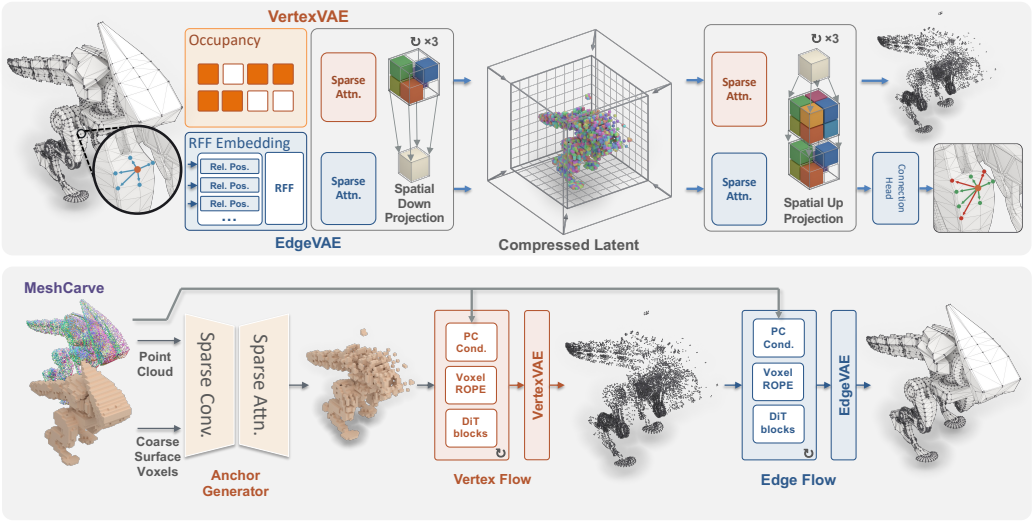

Figure 2: Overview of MeshCarve. Top: VertexVAE and EdgeVAE share one sparse transformer backbone, reaching the compressed 64^3 latent through the spatial down projection of Eq. 1 and decoding the vertex voxels and their connections through the mirrored up projection. Bottom: end-to-end generation from a dense point cloud through the anchor generator and two flows.

reduces the input resolution of 512^3 to 256^3, then 128^3, and finally 64^3, which is the resolution of our sparse compact latent. Following standard VAE practice, a linear layer maps each 768-dimensional latent token to the mean and log-variance of a 16-channel Gaussian for sampling.

3.2 VertexVAE: Compressing Vertex Geometry

The geometry side only has to decide which voxels hold vertices, so its decoder classifies octants rather than regressing coordinates. VertexVAE encodes vertex occupancy at 512^3 into the 64^3 latent, and its decoder recovers the vertex voxels by subdivision and pruning, following Xiang et al. (2026); Zhao et al. (2026). At each of the three levels, every latent voxel is expanded into its eight child features by the upsampling of Eq. 1. A prune head then scores each child with the probability that it contains a vertex, $p_{c,k} = \mathrm{sigmoid}(\mathrm{MLP}(\mathbf{f}_{c,k}))$, where MLP is a small multilayer perceptron. Children with $p < 0.5$ are discarded and the survivors subdivided again until the 512^3 vertex voxels are reached, whose centers are the vertex positions. The prune heads are trained with the asymmetric loss (ASL) (Ridnik et al., 2021) following Zhao et al. (2026). The network is trained with teacher-forcing. The inference is cascaded subdivision and pruning.

3.3 EdgeVAE: Compressing Vertex Connectivity

Topology is the set of connections between vertices. It is conventionally a byproduct of generating faces (Nash et al., 2020; Siddiqui et al., 2024; Chen et al., 2025b; Weng et al., 2025). Recent works (Shen et al., 2024; Long et al., 2026; Wang et al., 2026b) instead generate the adjacency matrix for a given set of vertices directly. Connectivity carries long-range information and is harder to reconstruct faithfully, let alone from a spatially compact latent. EdgeVAE therefore keeps the backbone above (Eq. 1).

Vertex-Link Encoding. To encode connectivity with a single fixed-width token per vertex, the variable-size set of its neighbors must be mapped to a fixed-dimensional representation. A straightforward fixed-width construction assigns neighbor coordinates to ordered slots and pads the list to a prescribed maximum degree (Li et al., 2026b). However, this construction is sensitive to permutations of the neighbor set, imposes a hard cap on vertex degree, and reserves dimensions for uninformative padding. We therefore propose vertex-link encoding, which carries the connections of a vertex as the relative positions of the vertices it is connected to, encoded with random Fourier features (RFF) (Rahimi & Recht, 2007) and summed over the neighbors. The encoding is straight-

forward, with no parameters to learn and no hand-crafted rule beyond one fixed random matrix. It has a fixed length without padding. It is order-free, so two exports of the same mesh give the same token. It accepts any valence, so no capacity is spent on empty slots and no cap is placed on the degree. Tancik et al. (2020) compare frequency mappings for low-dimensional coordinates and find isotropic Gaussian frequencies preferable to on-axis positional encodings, which are biased toward axis-aligned structure. The offsets from a vertex to its linked vertices point in arbitrary directions, so we draw $\mathbf{B} \in \mathbb{R}^{m/2\times 3}$ once from $\mathcal{N}(0, \sigma^2\mathbf{I})$ and keep it fixed. Each relative position $\mathbf{d}$ is encoded as $\gamma(\mathbf{d}) = [\cos(\mathbf{Bd}); \sin(\mathbf{Bd})] \in \mathbb{R}^m$, and the input token of vertex i sums the encodings of all its connections,

$$\mathbf{r}_i = \sum_{j:\, A_{ij}=1} \gamma(\mathbf{v}_j - \mathbf{v}_i), \qquad \langle \gamma(\mathbf{a}), \gamma(\mathbf{b}) \rangle \approx \tfrac{m}{2}\, e^{-\sigma^2 \|\mathbf{a}-\mathbf{b}\|^2/2}, \tag{2}$$

with $m = 128$ and $\sigma = 3$ in voxel units. The sum has a fixed width whatever the number of connections, and since the inner product of two encodings approximates a Gaussian kernel of their difference, distinct relative positions remain separable inside the sum. During decoding, the vertex voxels are given, so the decoder expands the 64^3 latent along them with the upsampling of Eq. 1 without pruning and produces a feature for every vertex. Following LATO (Zhao et al., 2026), a connection head classifies each vertex pair and is trained with binary cross-entropy.

3.4 Generating Artisan Mesh

3.4.1 Generating the Latent Voxel Anchor

To begin flow matching, position of the latent voxel should be given. This is an essential step because otherwise the flow matching needs to operate on a dense tensor with hundreds of thousands of tokens. Our preliminary study generated it with a flow matching model following the setup of TRELLIS (Xiang et al., 2025). However, it costs significant compute and the result remains unsatisfactory (Table 4). Our insight is that the coarse geometry of an artisan mesh is mostly determined by the surface itself. Vertices gather where the curvature is. At the coarse resolution, such as 64^3, the different tessellations an artist might choose occupy the same voxels, leaving less ambiguity and making flow matching or diffusion redundant. We therefore generate the anchor in one shot with an anchor generator. Notably, this is faster than flow matching and better for this task. The generator is a sparse VAE composed of convolutional and attention blocks. Its input is the dense point cloud and the coarse surface occupancy. We use a PointNet to pool the position and normals of point cloud samples into the voxels, giving coarse voxels richer details. The vertex budget is also given as a condition since it is up to user preference. The latent space is a dense tensor with a spatial dimension of 16^3. The decoder subdivides and prunes redundant voxels to generate the placement of the coarse vertex positions, i.e., the anchor that flow matching networks run on. We use the asymmetric loss (ASL) (Ridnik et al., 2021) since vertices are highly sparse. Additional backbone details can be found in the supplementary.

3.4.2 Vertex and Edge Latent Space Flow Matching

Both latents are generated with rectified flow (Lipman et al., 2023) by the sparse transformer of Xiang et al. (2025; 2026), with one token per latent voxel and the dense point cloud as the condition through a frozen shape encoder (Zhao et al., 2023). The vertex flow runs on the generated anchor and is additionally conditioned on the vertex budget N. Its sample decodes into the vertex voxels at 512^3, whose centers are the generated vertices. The edge flow runs on the same anchor and is additionally conditioned on the vertex latent of the same voxel. Its sample decodes over the generated vertex voxels into the adjacency and the faces through the connection head. Both flows train on the ground-truth anchor, vertex voxels and vertex latents, and run on generated ones at inference. The objective, the sampler and the conditioning are detailed in Appendix A.

4 Experiments

Implementation VertexVAE and EdgeVAE share the sparse transformer backbone, with a hidden width of 768 and eight blocks at each resolution level, at 429M and 483M parameters. The anchor

Table 2: Shape-conditioned artisan mesh generation on Objaverse and Toys4K. The top block gives the edge stage the reference vertices. Best in bold within each block.

	Objaverse				Toys4K			
Method	$CD_{\ell_2}\downarrow$	$CD_{\ell_1}\downarrow$	HD↓	NC↑	$CD_{\ell_2}\downarrow$	$CD_{\ell_1}\downarrow$	HD↓	NC↑
w/ GT vertices								
LATO.2 T-Flow (Long et al., 2026)	.00595	.00824	.0709	.8775	.00483	.00689	.0490	.9245
MeshCarve edge flow (ours)	**.00462**	**.00644**	**.0495**	**.9218**	**.00374**	**.00538**	**.0461**	**.9488**
End-to-end generation								
MeshAnything (Chen et al., 2025b)	.02838	.03655	.1661	.7537	.03525	.04798	.1765	.7754
MeshAnythingV2 (Chen et al., 2025c)	.01907	.02611	.1513	.7867	.03549	.04858	.1683	.7667
TreeMeshGPT (Lionar et al., 2025)	.02611	.03554	.1659	.7584	.01459	.02061	.0841	.8607
BPT (Weng et al., 2025)	.01561	.02155	.0991	.8406	.00866	.01256	.0617	.8962
DeepMesh (Zhao et al., 2025)	.01991	.02738	.1382	.7833	.01055	.01515	.0705	.8871
FastMesh (Kim et al., 2026)	.00998	.01415	.0922	.8010	.00780	.01128	.0618	.8702
MeshFlow (Li et al., 2026b)	.00854	.01212	**.0547**	.8119	.00676	.00978	**.0355**	.8793
LATO.2 (Long et al., 2026)	.00693	.00993	.0754	.8238	.00562	.00820	.0476	.8855
MeshCarve (ours)	**.00649**	**.00933**	.0789	**.8529**	**.00560**	**.00814**	.0666	**.9069**

generator has 613M parameters. The vertex and edge flows follow the implementation of TREL-LIS (Xiang et al., 2025; 2026), with 30 blocks at width 1,536 and 32 blocks at width 2,048, 1.27B and 2.37B parameters. MeshCarve is trained on 558K meshes from Objaverse (Deitke et al., 2023). Generation is evaluated on 185 held-out Objaverse meshes and on 500 meshes drawn at random from Toys4K (Stojanov et al., 2021), and the VAE ablations on 5,000 Objaverse validation meshes. Training schedules and inference settings are given in the supplementary.

Baselines and metrics We compare against MeshAnything, MeshAnything V2, TreeMeshGPT, BPT, DeepMesh, FastMesh, MeshFlow and LATO.2 (Chen et al., 2025b;c; Lionar et al., 2025; Weng et al., 2025; Zhao et al., 2025; Kim et al., 2026; Li et al., 2026b; Long et al., 2026), each run from its released weights under its stock inference setting. Autoregressive methods that publish a face cap are scored on the evaluation meshes inside it. The topology flow of LATO.2 is scored on the meshes within its vertex bound. In each case MeshCarve is scored on the same meshes, and on the full sets otherwise. MeshCarve and LATO.2 take the reference vertex count as their user-set budget. The supplementary gives the settings of every baseline, the count conditioning, and the comparison outside the caps. LATO (Zhao et al., 2026) released only its VAE. All methods are scored in the same way, following Li et al. (2026b); Wang et al. (2026b). Prediction and reference are normalized to the $[-0.5, 0.5]$ box, and 50K points are sampled from each surface. We report Chamfer distance (CD_{ℓ_2}, CD_{ℓ_1}), Hausdorff distance (HD) and normal consistency (NC).

4.1 Main Results

As shown in Table 2, MeshCarve achieves the highest normal consistency on both benchmarks and the lowest Chamfer distance on Objaverse among all compared methods. On Toys4K it is level with LATO.2 in Chamfer distance. As a maximum-based metric, Hausdorff distance can be dominated by a single hole or uncovered region. MeshFlow (Li et al., 2026b) achieves the lowest value with coarse, low-polygon outputs that capture the global shape without large local gaps. This robust surface coverage is favored by Hausdorff distance, which does not directly assess tessellation density or edge layout. Additional analysis of vertex count adherence is included in the supplementary, where we find that MeshCarve follows the given vertex count better than MeshFlow and LATO.2. The autoregressive baselines are scored inside their published face caps, the range they were built for, and the gaps widen on the full sets (Appendix F).

We also compare the edge flows alone. Both are given the same reference vertices, on the meshes that the topology flow of LATO.2 admits. Our edge flow leads on every metric (Table 2, top block) by significant margins. On Objaverse the lead is 22% to 30% of the mean Chamfer and Hausdorff distance. It also recovers more of the artist's wiring, as measured by the edge F1 score (Table 3). We believe the lead comes mostly from the much higher compression rate of our EdgeVAE, which allows flow matching to run at an affordable computational cost. Our VertexVAE and the vertex flow also show stronger performance than the V-VAE of LATO.2 (Table 3). In sum, we demonstrate

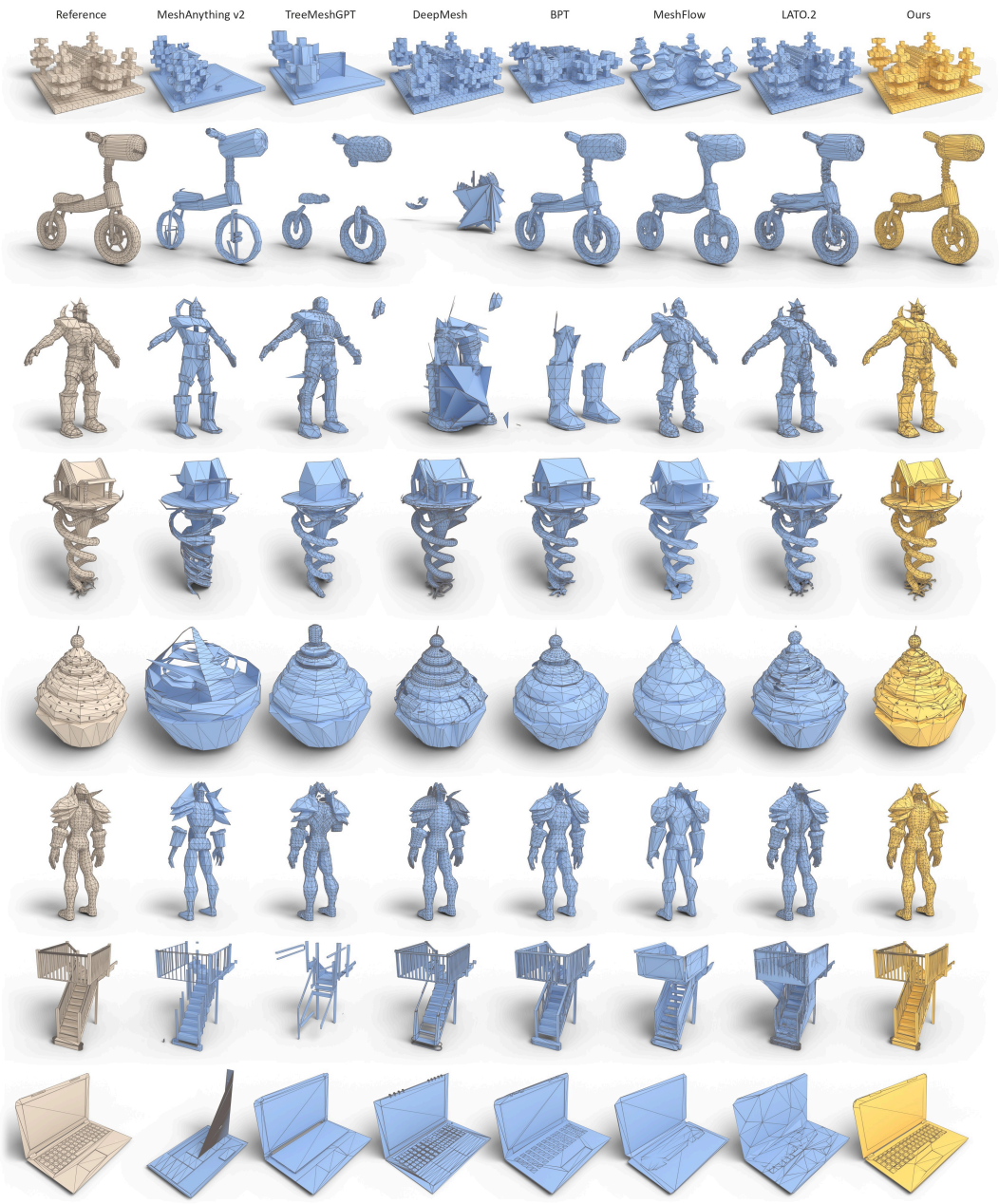

Figure 3: Qualitative comparison on Objaverse and Toys4K. References are in clay, baselines in blue and MeshCarve in amber; all outputs are shown as generated with wireframes. Zoom in for better view.

that both vertex position and edge connectivity can be flow matched in much more compact latent spaces, allowing easy scale-up of the flow matching model. Notably, the V-VAE of LATO.2 does not generalize to our test dataset despite our best effort to reproduce it. Surface metrics do not see the connectivity, so Appendix D reports statistics of the generated graphs, where our edge flow given the reference vertices produces far fewer non-manifold edges than the topology flow of LATO.2.

4.2 Empirical Study

Qualitative results. In Figure 3, MeshCarve follows the reference surface closely and places vertices where the artist did, while the autoregressive baselines leave parts of the shape uncovered.

Table 3: **VAE reconstruction and flow generation.** (a) F1 of the reconstructed vertex voxels and edges. (b) The vertex flow scored against, and the edge flow given, the reference vertices. Best in bold.

(a) VAE reconstruction F1↑

	Vertex VAE		Edge VAE	
Method	Toys4K	Objaverse	Toys4K	Objaverse
LATO	.762	.656	.866	.667
MeshFlow	.017	.020	.973	.920
LATO.2	.375	.343	**.999**	**.997**
MeshCarve	**.998**	**.996**	.973	.962

(b) Flow generation

	Vertex flow				Edge flow	
	CD_{ℓ_1}↓		HD↓		F1↑	
Method	Toys4K	Obj.	Toys4K	Obj.	Toys4K	Obj.
MeshFlow	.0260	.0298	.144	.172	–	–
LATO.2	.0149	.0145	.117	.144	.729	.673
MeshCarve	**.0091**	**.0079**	**.106**	**.111**	**.795**	**.820**

Table 4: **Ablations.** (a) EdgeVAE variants scored on 5,000 validation samples with the all-pair edge metric, our model first and one component changed per row. Precision and recall are computed over all vertex pairs of a mesh and averaged over meshes. (b) Anchor generation, the anchor generated by a flow against our anchor generator, the two sharing the same downstream stages on Objaverse.

(a) EdgeVAE variants

Variant	precision	recall	F1
MeshCarve	**.95180**	**.99019**	**.96965**
mean pooling and broadcast	.81978	.96361	.88111
w/o vertex-link encoding	.61701	.85324	.70793
slotted vertex neighbor encoding	.93806	.98959	.96167
w/o RoPE	.36967	.50487	.40872
joint latent for vertices and edges	.90258	.95743	.92743

(b) Anchor generation

Anchor	CD_{ℓ_2}↓	CD_{ℓ_1}↓	HD↓	NC↑
generated by a flow	.01007	.01422	.0971	.8012
anchor generator (MeshCarve)	**.00649**	**.00933**	**.0789**	**.8529**

MeshFlow, in comparison, generates meshes with an overly low face count that cannot describe the object shape precisely, which explains its low HD yet higher CD_{ℓ_2} and CD_{ℓ_1}. LATO.2 also has difficulty capturing fine details and does not handle thin structures well. More examples are in Appendix G and the limitations are discussed in Appendix H.

Ablations. Table 4 ablates the critical components of our VAE backbone. We replace the spatial-aware compression with naive mean pooling and broadcast, remove vertex-link encoding, swap vertex-link encoding with slotted vertex neighbor encoding, and remove RoPE in the sparse transformer blocks. All four variants degrade the performance, justifying our VAE design. Removing RoPE causes the largest drop, since the connections between vertex voxels depend on their relative positions. The slotted vertex neighbor encoding comes closest to MeshCarve. It is trained from scratch for the full schedule, so this small gap is not an artifact of a shorter schedule. In every variant the precision drops more than the recall, so the changed components mainly prevent connections that the artist did not draw. We also show the results when the same VAE backbone is used to encode vertex position and edge connection simultaneously. The degradation is noticeable and is amplified by inaccurate vertex subdivision and pruning, because the error accumulates fast during inference. We also ablate the use of the anchor generator versus a flow matching network. The anchor generator shows significant advantages over a flow matching network in obtaining the latent voxel positions. Both anchors feed the same vertex and edge flows, so the gap in Table 4(b) comes from the anchor alone. Notably, the anchor generator is based on a VAE backbone and therefore also generates with diversity, as shown in Figure 6 of the supplementary.

Limitations and future work. While MeshCarve demonstrates its effectiveness in our evaluation, we notice some limitations imposed by the network. Specifically, for complex meshes over 10K vertices, the 512^3 resolution might not be sufficient and fine details might not be generated faithfully. We also find that the flow matching networks trained with teacher forcing encounter a domain gap when cascaded, and we believe further adaptation will unlock more of the edge flow's potential.

5 Conclusion

We presented MeshCarve, which generates artisan meshes by flow matching entirely in spatially compressed latents. The pipeline samples the coarse vertex occupancy with a VAE in one forward pass and then generates vertex positions and edge connections in turn. The latents are compact because of the VAE design. One hierarchical sparse transformer backbone is anchored to the vertex voxels and compressed spatially through a learned projection. It is instantiated separately for vertex positions and for edge connections, and the connections reach it as the vertex-link encoding. At 0.13 tokens per face, MeshCarve shows advantages over previous state of the art on Objaverse and generalizes to Toys4K.

AI USE STATEMENT

We used generative AI coding assistants. They helped with writing and language polishing of the manuscript. They helped with the code of the evaluation, rendering and figure-generation scripts. We did not use generative AI to generate research ideas and to design the method. All AI-assisted text was revised by the authors, all AI-assisted code was verified and tested by the authors, and every number reported was computed by our own evaluation pipeline. We take responsibility for the final content of this work.

REFERENCES

Antonio Alliegro, Yawar Siddiqui, Tatiana Tommasi, and Matthias Nießner. PolyDiff: Generating 3d polygonal meshes with diffusion models. *arXiv preprint arXiv:2312.11417*, 2023.

Junyu Chen, Han Cai, Junsong Chen, Enze Xie, Shang Yang, Haotian Tang, Muyang Li, Yao Lu, and Song Han. Deep compression autoencoder for efficient high-resolution diffusion models. In *ICLR*, 2025a. arXiv:2410.10733.

Yiwen Chen, Tong He, Di Huang, Weicai Ye, Sijin Chen, Jiaxiang Tang, Xin Chen, Zhongang Cai, Lei Yang, Gang Yu, Guosheng Lin, and Chi Zhang. MeshAnything: Artist-created mesh generation with autoregressive transformers. In *ICLR*, 2025b.

Yiwen Chen, Yikai Wang, Yihao Luo, Zhengyi Wang, Zilong Chen, Jun Zhu, Chi Zhang, and Guosheng Lin. MeshAnything V2: Artist-created mesh generation with adjacent mesh tokenization. In *ICCV*, 2025c.

Imant Daunhawer, Thomas M. Sutter, Kieran Chin-Cheong, Emanuele Palumbo, and Julia E. Vogt. On the limitations of multimodal VAEs. In *ICLR*, 2022.

Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3D objects. In *CVPR*, 2023.

Xianglong He, Junyi Chen, Di Huang, Zexiang Liu, Xiaoshui Huang, Wanli Ouyang, Chun Yuan, and Yangguang Li. MeshCraft: Exploring efficient and controllable mesh generation with flow-based DiTs. *arXiv preprint arXiv:2503.23022*, 2025.

Adrián Javaloy, Maryam Meghdadi, and Isabel Valera. Mitigating modality collapse in multimodal VAEs via impartial optimization. In *ICML*, 2022.

Jeonghwan Kim, Yushi Lan, Armando Fortes, Yongwei Chen, and Xingang Pan. FastMesh: Efficient artistic mesh generation via component decoupling. In *International Conference on 3D Vision (3DV)*, 2026. arXiv:2508.19188.

Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. In *ICLR*, 2014. arXiv:1312.6114.

Haoxuan Li, Ziya Erkoç, Daniele Sirigatti, Vladislav Rosov, Lei Li, Angela Dai, and Matthias Nießner. TriFlow: Generating artist-like 3d mesh topology via nearest-vertex vector fields. In *ECCV*, 2026a.

Weiyu Li, Antoine Toisoul, Tom Monnier, Roman Shapovalov, Rakesh Ranjan, Ping Tan, and Andrea Vedaldi. MeshFlow: Efficient artistic mesh generation via MeshVAE and flow-based diffusion transformer. In *CVPR*, 2026b.

Stefan Lionar, Jiabin Liang, and Gim Hee Lee. TreeMeshGPT: Artistic mesh generation with autoregressive tree sequencing. In *CVPR*, 2025.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *ICLR*, 2023.

Hang Long, Tianhao Zhao, Junkai Lin, Youjia Zhang, Huipeng Guo, Rendong Liang, Jiale Xu, Jozef Hladký, Matthias Nießner, Yuanming Hu, and Wei Yang. LATO.2: Factorized 3d mesh generation with vertex and topology flow. *arXiv preprint arXiv:2607.10623*, 2026.

Charlie Nash, Yaroslav Ganin, S. M. Ali Eslami, and Peter W. Battaglia. PolyGen: An autoregressive generative model of 3D meshes. In *ICML*, 2020. arXiv:2002.10880.

Emanuele Palumbo, Imant Daunhawer, and Julia E. Vogt. MMVAE+: Enhancing the generative quality of multimodal VAEs without compromises. In *ICLR*, 2023.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *NeurIPS*, 2007.

Tal Ridnik, Emanuel Ben-Baruch, Nadav Zamir, Asaf Noy, Itamar Friedman, Matan Protter, and Lihi Zelnik-Manor. Asymmetric loss for multi-label classification. In *ICCV*, 2021.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022. arXiv:2112.10752.

Tianchang Shen, Zhaoshuo Li, Marc Law, Matan Atzmon, Sanja Fidler, James Lucas, Jun Gao, and Nicholas Sharp. SpaceMesh: A continuous representation for learning manifold surface meshes. In *SIGGRAPH Asia*, 2024.

Yawar Siddiqui, Antonio Alliegro, Alexey Artemov, Tatiana Tommasi, Daniele Sirigatti, Vladislav Levy, Angela Dai, and Matthias Nießner. MeshGPT: Generating triangle meshes with decoder-only transformers. In *CVPR*, 2024. arXiv:2311.15475.

Stefan Stojanov, Anh Thai, and James M. Rehg. Using shape to categorize: Low-shot learning with an explicit shape bias. In *CVPR*, 2021.

Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. In *NeurIPS*, 2020.

Jiaxiang Tang, Zhaoshuo Li, Zekun Hao, Xian Liu, Gang Zeng, Ming-Yu Liu, and Qinsheng Zhang. EdgeRunner: Auto-regressive auto-encoder for artistic mesh generation. In *ICLR*, 2025.

Hanxiao Wang, Yuan-Chen Guo, Ying-Tian Liu, Zi-Xin Zou, Biao Zhang, Weize Quan, Ding Liang, Yan-Pei Cao, and Dong-Ming Yan. FACE: A face-based autoregressive representation for high-fidelity and efficient mesh generation. In *CVPR*, 2026a.

Hanxiao Wang, Ying-Tian Liu, Yuan-Chen Guo, Qi-Yuan Feng, Zi-Xin Zou, Ding Liang, Biao Zhang, and Yan-Pei Cao. Nexus: Native mesh generation with diffusion. *ACM Transactions on Graphics*, 45(4), 2026b.

Haohan Weng, Zibo Zhao, Biwen Lei, Xianghui Yang, Jian Liu, Zeqiang Lai, Zhuo Chen, Yuhong Liu, Jie Jiang, Chunchao Guo, Tong Zhang, Shenghua Gao, and C. L. Philip Chen. Scaling mesh generation via compressive tokenization. In *CVPR*, 2025.

Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong, and Jiaolong Yang. Structured 3D latents for scalable and versatile 3D generation. In *CVPR*, 2025.

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 *CVPR*, 2026.

Rui Xu, Tianyang Xue, Qiujie Dong, Le Wan, Zhe Zhu, Peng Li, Zhiyang Dou, Cheng Lin, Shiqing Xin, Yuan Liu, Wenping Wang, and Taku Komura. MeshMosaic: Scaling artist mesh generation via local-to-global assembly. In *CVPR*, 2026.

Ruowen Zhao, Junliang Ye, Zhengyi Wang, Guangce Liu, Yiwen Chen, Yikai Wang, and Jun Zhu. DeepMesh: Auto-regressive artist-mesh creation with reinforcement learning. In *ICCV*, 2025.

Tianhao Zhao, Youjia Zhang, Hang Long, Jinshen Zhang, Wenbing Li, Yang Yang, Gongbo Zhang, Jozef Hladký, Matthias Nießner, and Wei Yang. LATO: 3d mesh flow matching with structured topology preserving latents. In *ICML*, 2026.

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.