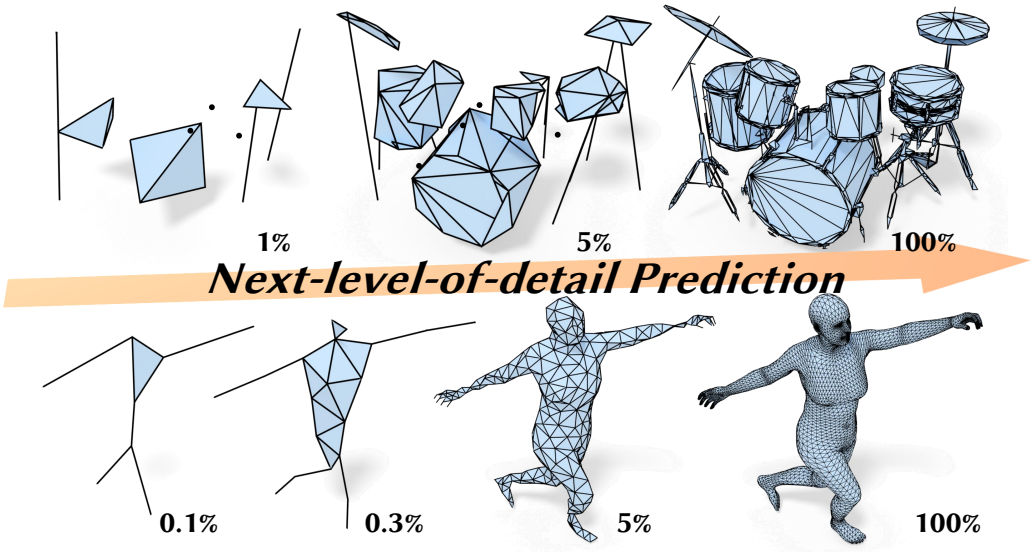
ARMesh: Autoregressive Mesh Generation via Next-Level-of-Detail Prediction

Jiabao Lei[1*], **Kewei Shi**[2*], **Zhihao Liang**[3], **Kui Jia**[1†]
[1] The Chinese University of Hong Kong, Shenzhen
[2] The University of Hong Kong
[3] Tencent Hunyuan
https://jblei.site/proj/armesh



Abstract

Directly generating 3D meshes, the default representation for 3D shapes in the graphics industry, using auto-regressive (AR) models has become popular these days, thanks to their sharpness, compactness in the generated results, and ability to represent various types of surfaces. However, AR mesh generative models typically construct meshes face by face in lexicographic order, which does not effectively capture the underlying geometry in a manner consistent with human perception. Inspired by 2D models that progressively refine images, such as the prevailing next-scale prediction AR models, we propose generating meshes auto-regressively in a progressive coarse-to-fine manner. Specifically, we view mesh simplification algorithms, which gradually merge mesh faces to build simpler meshes, as a natural fine-to-coarse process. Therefore, we generalize meshes to simplicial complexes and develop a transformer-based AR model to approximate the reverse process of simplification in the order of level of detail, constructing meshes initially from a single point and gradually adding geometric details through local remeshing, where the topology is not predefined and is alterable. Our experiments show that this novel progressive mesh generation approach not only provides intuitive control over generation quality and time consumption by early stopping the auto-regressive process but also enables applications such as mesh refinement and editing.

[*] Equal Contribution; [†] Corresponding Author: *kuijia@cuhk.edu.cn*

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

https://doi.org/10.52202/085713-5397

1 Introduction

3D triangular surface mesh is the predominant representation in graphics. Due to its explicit control nature, compactness, and strong compatibility with modern graphics pipelines, it is important and has been widely used in various applications, such as the gaming industry and film production. Since 3D meshes are discrete and often non-watertight, autoregressive (AR) models are increasingly popular for directly generating meshes instead of relying on intermediate representations like implicit fields, which may struggle with non-watertight, non-manifold, or thin-sheet surfaces. While AR models prove highly effective in modeling discrete ordered sequences in many other domains, such as natural languages, their application to meshes is less straightforward since mesh vertices and faces are inherently unordered. Prior works typically assign lexicographic orders [29] (for instance, from left to right and bottom to top in a canonical coordinate system) or face traversal orders [4, 48, 60] to transform a mesh into a 1D sequence, and later employ an autoregressive model to learn the sequence directly for mesh generation. However, these artificially created orders are often less meaningful in geometry, failing to capture the overall shape during generation, and do not align well with how humans perceive objects in a hierarchical, coarse-to-fine manner.

Such an issue also exists for AR models in 2D, where a raster scanning order for images is easy to implement but not optimal. Recent approaches, such as VAR [49], instead define the AR sequence as predicting the next scales, *i.e.*, generating a higher-resolution image from a lower-resolution one. We are therefore prompted to explore *what the "next-scale" prediction for 3D meshes could be.*

Unlike images or voxel grids that are structured regularly and allow for easy up- or down-sampling to obtain different resolutions, the "scale," more commonly known as level-of-detail (LOD) in graphics, is rarely explored for 3D meshes with deep learning due to the irregular nature and complexity of explicitly managing a mesh's connectivity and topology. Nevertheless, one can still define the LOD for a mesh in terms of the number of faces or vertices. The widely accepted mesh simplification algorithms, such as QSlim [16], help by progressively reducing the face count of a mesh through the repeated collapsing of edges, where an edge connecting two vertices is merged into a single vertex, thus creating different levels of detail for the mesh. By leveraging a mesh simplification algorithm, we are motivated to train an AR model to learn from this reversed simplification sequence and generate meshes by progressively adding geometric details through local remeshing.

Our paper primarily draws upon two works [17, 41] in the traditional area of mesh processing from several decades ago. Specifically, we propose an adapted version of [17], termed GSlim in this paper, which is designed to operate on and simplify simplicial complexes (introduced later in Section 3.1), a generalization of triangular meshes, within a unified framework. The algorithm is enhanced with the capability to allow for topological changes during simplification. With our algorithm, a mesh, regardless of how complex its initial topology may be, will ultimately be reduced to a single point, benefiting from the use of simplicial complexes[2], which is unsupported by QSlim. The algorithm can provide information as a byproduct on how we simplify a mesh in the LOD order. By reversing the simplification process, we can obtain a 1D refinement sequence that progressively refines a mesh through local remeshing, an idea that has been roughly discussed by Popović and Hoppe [41]. The above algorithms essentially convert a mesh into a 1D sequence. We then design a tokenization scheme to discretize the 1D sequence into compact tokens and subsequently train an autoregressive (AR) transformer using the tokenized sequences. This process can also be referred to as next-level-of-detail prediction, as the model essentially predicts refinement operations that add details to construct the next LOD. By employing our method, we can easily obtain a mesh with a user-specified LOD/complexity by early stopping the AR generation, which was unachievable in earlier work. Our experiments also show that such a trained transformer model is a useful and effective shape encoder that can faithfully capture shapes with high fidelity in the task of unconditional mesh generation.

In summary, to achieve direct mesh generation in the order of LODs, this paper proposes reversing the mesh simplification algorithm, thereby converting a mesh into a coarse-to-fine mesh refinement sequence. We train an AR model on these sequences to predict the next LODs. During generation, the mesh initially starts from a single point and is iteratively refined according to the predictions made by the AR model, progressively adding geometric and topological details through local remeshing. Experiments show its streaming generation with flexible controllability over quality and time usage.

[2]If we do not use simplicial complexes, an isolated single point cannot be described along with a triangular mesh. Additionally, simplicial complexes accommodate various types of meshes, including those with different topologies, non-watertight or non-manifold cases, mixed dimensions, and so on, as indicated by [17].

https://doi.org/10.52202/085713-5397

2 Related Work

Indirect Mesh Generation. A mesh can be indirectly obtained by converting it from other representations. Researchers have shifted their focus to studying generation with implicit fields after recognizing the ease of using these fields to handle different topologies [6, 34, 39]. Numerous studies [7, 8, 13, 15, 25, 26, 27, 28, 39, 43] have emerged along this line. Some aim to describe shapes purely geometrically, typically using signed distance fields [6, 34, 39], while others further explore how to encode both geometry and appearance within the same representation, typically employing neural radiance fields [20, 24, 35, 36, 55]. Their representations can be either continuous [34, 35, 36, 39] or discrete [28, 30, 43]. They often rely on a post-processing step of iso-surface extraction [31] to convert their representation into a surface mesh. The drawbacks of indirectly generating meshes prompt us to study direct mesh generation methods, as detailed below.

Direct Mesh Generation. Initial attempts in this area involve deforming a shape template (*e.g.*, a sphere) to a target [54, 58], which is limited to a fixed topology. Some approaches attempt to explicitly modify the topology [38] or leverage local surface patches to adapt the topology [18, 19], but still fail to produce a mesh with concise tessellation. Some methods capable of obtaining a concise tessellation are based on intermediate representations and need to derive the mesh indirectly from them [8, 13]. PolyGen [37] is the first to directly generate a mesh by explicitly generating its vertices and faces. MeshGPT [45] learns codebooks to facilitate the encoding of triangular faces; some methods [3, 4, 21, 29, 60] strive to develop efficient tokenization strategies that convert a mesh into a one-dimensional sequence; some researchers [21, 53] study the architecture for efficient training of a mesh generation network. However, although there have been numerous works in this area, the paradigm they use mainly originates from [37], where the mesh is constructed face by face. In contrast, we attempt to generate a mesh progressively in the order of levels of detail.

Levels-of-Detail. Representing signals with multiple resolutions can generally be effective. Traditional methods [10, 16, 22, 42] focus on designing simplification algorithms that produce multiple levels from a given mesh input. Some works in deep learning [5, 36, 40, 47] suggest producing implicit fields hierarchically or using hierarchical data structures to facilitate fine-grained representation; however, they do not natively operate on the mesh itself. This is mainly due to the challenges associated with handling a mesh's connectivity. A recent work [48] in direct mesh generation attempted to incorporate an additional token for face count control; however, it does not intrinsically model different levels of detail, still essentially treating various levels as distinct meshes. A concurrent work named VertexRegen [61] is built upon progressive meshes [22], which require the initialization to be homeomorphic to the target mesh. However, our method does not have these restrictions.

3 Preliminaries

3.1 Generalizing Mesh into Simplical Complex

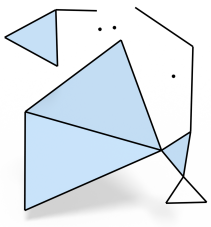


Figure 2: An simplicial complex example.

A mesh can be considered exclusively composed of triangles. However, triangles alone are insufficient in this paper. Our method relies on a new representation called simplicial complex (SC), which is a generalization of meshes. A simplicial complex further generalizes a mesh to accommodate (isolated) points and line segments that are not necessarily part of any triangles. Figure 2 illustrates this idea. We sometimes refer to an SC simply as a mesh in this paper for simplicity. Another benefit of using SC is its conciseness and abstractness in describing geometry. For example, line segments are better suited for describing skeletons, while an SC composed exclusively of points is essentially a point cloud.

3.2 QSlim: The Standard Quadric-Based Simplification Algorithm

To make our paper self-contained, we briefly introduce the widely accepted mesh simplification algorithm QSlim [16] here before formally introducing the GSlim algorithm that we actually used. For a triangular face σ defined by the plane $\mathbf{n}^\top(\mathbf{x}-\mathbf{p})=0$ with $\|\mathbf{n}\|=1$ and $\mathbf{p}$ being the barycenter, we can easily see that the squared distance from a point $\mathbf{x}\in\mathbb{R}^3$ to the plane $\mathbf{n}^\top(\mathbf{x}-\mathbf{p})=0$ can be calculated as:

$$Q_\sigma(\mathbf{x}) = \left(\mathbf{n}^\top(\mathbf{x}-\mathbf{p})\right)^2 = \mathbf{x}^\top\mathbf{A}\mathbf{x} + 2\mathbf{b}^\top\mathbf{x} + c$$
$$\text{with} \quad \mathbf{A}=\mathbf{n}\mathbf{n}^\top, \quad \mathbf{b}=-\mathbf{A}\mathbf{p}, \quad c=\mathbf{p}^\top\mathbf{A}\mathbf{p}$$

https://doi.org/10.52202/085713-5397

The above distance, also called the cost, is uniquely defined by the triplet $Q_\sigma = (\mathbf{A}, \mathbf{b}, c)$, which is referred to as the fundamental quadric of σ. We note that since geometric error can be measured in total as the sum of costs, the quadric is essentially additive. The quadric of a vertex v is therefore the sum of the quadrics of all faces incident to it, that is, $Q_v = \sum_{\sigma \in \mathcal{N}_v} Q_\sigma$, where $\mathcal{N}_v$ is the set of faces incident to v. The quadric of an edge e is defined as the sum of the quadrics of its two endpoints, that is, $Q_e = Q_{v_1} + Q_{v_2}$, where v_1 and v_2 are the two endpoints of e. If the edge e will collapse, it will merge v_1 and v_2 into a single vertex v^* which is the minimizer of Q_e, and the quadric of v^* is updated with $Q_{v^*} = Q_{v_1} + Q_{v_2}$. For simplicity, v^* is practically determined by

$$v^* = \underset{u \in \{v_1, v_2, \frac{v_1+v_2}{2}\}}{\arg\min} Q_e(u)$$

with the collapse cost being $Q_e(v^*)$. This means the collapse position is taken from either the endpoints or the midpoint of the edge. We can calculate the collapsing costs for every edge in the mesh and employ a greedy strategy to iteratively collapse the edge with the minimum cost found. A min-heap data structure will help find the minimum. The algorithm terminates when certain criteria are met, such as when the face count falls below the desired number. At the end of the algorithm, we will obtain a coarse mesh that is homeomorphic (has the same topology) to the original mesh.

To preserve boundary edges, the algorithm needs to generate a perpendicular plane that runs through the edge. These constraint planes are later converted into quadrics, weighted with a large penalty, and added to the initial quadrics for the endpoints of the edge. Apart from the inconvenience of preserving boundaries, there are also several limitations, including unchangeable topology and the inability to handle isolated points and edges. These limitations make the QSlim algorithm unusable for our task, since we envision that any mesh will eventually be simplified to a single vertex, which will require changes to both the topology and simplex dimension during simplification.

4 Method

4.1 GSlim: Generalized Mesh Simplification Algorithm

The GSlim algorithm we used addressed the aforementioned limitations (Section 3.2). It is adapted from [17] to enable topological changes. It is compatible with SC and supports reducing a mesh to the coarsest shape possible, which is a single point that also serves as the starting point for generation.

Unlike QSlim, which only defines fundamental quadrics for triangles, GSlim defines them for every simplex[3]. The quadric coefficient $\mathbf{A}$ for a simplex σ of intrinsic dimension d is instead calculated as

$$\mathbf{A} = \mathbf{I} - \sum_{i=1}^{d} \mathbf{e}_i \mathbf{e}_i^\top$$

where $\{\mathbf{e}_i\}$ is its tangent orthonormal basis of σ, and $\mathbf{I}$ is an identity matrix.
A larger quadric value in the above formulation will indicate that updates involving σ are difficult, thereby postponing its edge collapse. Similarly, we need to aggregate nearby quadrics to compute the edge collapse quadric Q_e, the calculation process of which has been detailed in Section 3.2. During the aggregation, we can optionally weight the fundamental quadrics with a large penalty factor to adjust the strength of preserving that simplex. For example, if we assign a large penalty to the boundary edges, it will attempt to preserve the boundary, which is much easier than QSlim. We have included an ablation study in Section 5.1 to investigate which set of penalty factors is the best.

It is interesting to note that for a mesh with n vertices, the algorithm requires $n - 1$ steps to simplify it until only a single vertex remains, as each edge collapse reduces the vertex count by exactly one.

4.2 PSC: Reversing Simplification into Generation

Since the simplification process is a many-to-one mapping, it is not inherently reversible unless additional information is recorded to specify how to reverse this process. The reverse operation of edge collapse is called "vertex split," and an example is provided in Figure 4. By reversing simplification with vertex splits, one can reformulate a simplicial complex into a new representation called a Progressive Simplicial Complex (PSC) [41], which consists of a starting point followed by a sequence of vertex splits for coarse-to-fine refinement. Since in PSC only new simplices are added to the existing mesh with new indices, without altering the indices of pre-existing simplices, to convert an edge collapse sequence into a vertex split sequence, one must also reorder the simplices' indices so that they are in ascending order. The following information must be recorded for each vertex split:

[3]A simplex may refer to a point ($d = 0$), an edge ($d = 1$), or a triangle ($d = 2$).

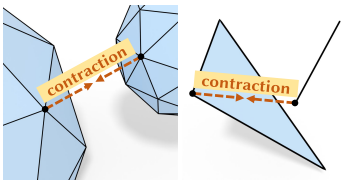


Figure 3: Two examples of generalized edge collapse. Two points will collapse into one point after contraction, leading to topological changes.

Before the algorithm begins, we need to specify a set of candidate vertex pairs for collapse. Unlike the convention [16] that uses candidate pairs from triangles, we further include "virtual edges" that bridge up disconnected components to facilitate topological changes. Figure 3 presents two examples of virtual edges for collapse. Following [41], these virtual edges are derived from the Delaunay tetrahedralization [1, 57] of the input vertices. After the simplification is applied, the algorithm will produce a sequence of increasingly coarsened meshes as side products, visualized from right to left in Teaser. In the next section, we will discuss what information should be recorded during progressive simplification to facilitate our future use.

- **which vertex to split**: an integer index is sufficient for selecting the vertex for splitting
- **how the split is performed**: a boolean indicating whether the current position remains unchanged or becomes the midpoint of the split vertex pair
- **the relative positional offset**: an offset vector for translating the new vertex position relative to the current vertex position
- **how the topology changes in adjacency**: a list of topological labels for adjacent simplices

A vertex split operation can be represented as a JSON configuration, which may look like:

```
{ "vsid":0, "midpoint":true, "offset":[1.2,0.7,2.5], "topo":[1,2,5,3,3,4,6,7] }
```

The first three are quite straightforward, while the fourth is less intuitive, and we will elaborate on what it actually means and how we derive it by explaining the example below.

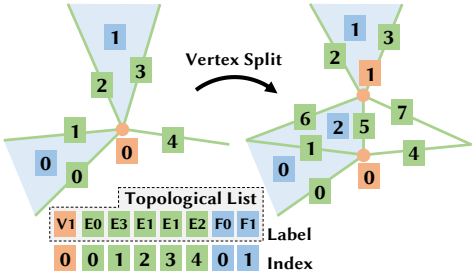


Figure 4: An example of vertex split.

Table 1: Enumeration of topological cases.

Input	Output Cases After Vertex Split			
	V0	V1		
	E0	E1	E2	E3
	F0	F1	F2	

In Figure 4, a vertex `0` is split into two, resulting in the creation of a new vertex `1`. Since the vertex split only influences its adjacent primitives (vertex `0` itself, five edges `0-4`, and two faces `0,1`), the topological changes must occur solely in its adjacency. We can use a list called "topological list" to carry the topological change information for each adjacent simplices (always including vertex `0` itself), and its length is exactly the number of adjacent primitives, as shown at the bottom of Figure 4. The list is first sorted by the dimension of the primitives, and then by ascending indices. Each entry in the list represents a classification label ranging from $0 \sim 8$, corresponding to the nine possible cases (`V0`, `V1`, `E0`, `E1`, `E2`, `E3`, `F0`, `F1` and `F2`) summarized in Table 1.

Let's interpret the topological cases in Table 1. For a single vertex marked as `s` (source), it can either be split to form a new vertex `t` (target) or create a new edge `st` connecting `s` and `t`. They are respectively denoted as case `V0` and `V1`. Similarly, there are four cases for an edge: `E0` merely adds a `t`; `E1` switches the edge to connect with `t`; `E2` creates a new edge to connect with `t`; and finally, `E3` creates a new triangle. The explanation for triangle cases is analogous and omitted for brevity.

With Table 1, we can interpret the case presented in Figure 4. For instance, for edge `1` in Figure 4, after splitting vertex `0` to create vertex `1`, a new face `2` is created, corresponding to case `E3` in Table 1. We note that creating face `2` also implies that all its sub-simplices exist or are created as well, including edges `6`, `5`, and `1`. This also suggests a constraint on the topological labels, *i.e.*, case `V0` is inherently incompatible with `E3`, because the former suggests that edge `st` (specifically, the edge `5`) exists, while the latter does not. There are many more hard constraints like this that the configuration must adhere to. The original paper [41] provides only incomplete rules and cannot be used directly for our purposes. We analyze and summarize all topological constraints into the following four rules:

https://doi.org/10.52202/085713-5397

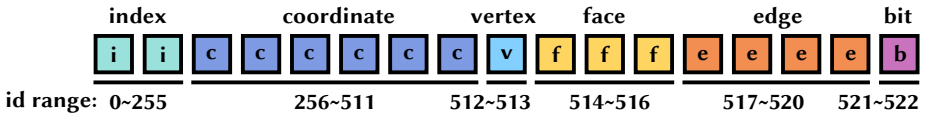


Figure 5: An example of a tokenization layout for a refinement operation.

1. If `V0`, then cannot have `E3`.
2. If $v \subset e_1, e_2$ and $e_1, e_2 \subset$ `F0`, then $e_1, e_2 \neq$ `E1`.
3. If $v \subset e_1, e_2$ and $e_1, e_2 \subset$ `F1`, then $e_1, e_2 \neq$ `E0`.
4. If $v \subset e_1, e_2$ and $e_1, e_2 \subset$ `F2`, then $e_1, e_2 \neq$ `E0` and $\neq$ `E1`.

A detailed explanation of rules (2-4) is provided in the footnote[4]. These four rules are complete and can be rigorously derived by thoroughly enumerating all possible cases in a table and summarizing the rules that encompass each entry. They also indicate that we can modify only the predictions of the edges to ensure that all the topological label predictions are consistent. We will leverage this property to design our tokenization layout, as shown in Figure 5, where the edge predictions follow the vertex and face predictions to facilitate the constrained decoding process introduced in Section 4.3.

By formulating the vertex split operations (refinement sequence) as a series of JSON configurations, we are able to progressively reconstruct a simplicial complex from a single point with the sequence. In the next section, we will detail how to turn this into a learnable process with AR models.

4.3 Auto-Regressive Learning from Refinement Sequences

After obtaining the refinement sequences, we can utilize transformers to learn from these sequences[5]. We need to tokenize the sequence before feeding it into the transformer network. Unlike [56], which directly converts the structured input to textual format, we organize it in a compact layout. We show the operation layout after tokenization in Figure 5. In this example, there are three faces and four edges incident to the selected vertex, resulting in three squares colored in yellow and four squares colored in coral. We now explain the layout in detail: (1) The first two bytes encode the vertex index using `int16`, and each slot ranges from 0 to 255; (2) The subsequent six bytes encode the relative offset using $3 \times$ `fp16`, with the first three bytes storing their most significant bytes, and each slot ranges from 256 to 511; (3) We then store the topological labels for the selected vertex, as well as incident faces and edges, respectively. There are only two cases for the selected vertex, so it ranges from 512 to 513. There are only three cases for the incident faces, so it ranges from 514 to 516. There are four cases for the incident edges, so it ranges from 517 to 520; (4) The last byte denotes a boolean indicating whether the current vertex is the midpoint after the vertex split. After all vertex split operations are tokenized, we should be able to obtain a long sequence by concatenating all tokens for autoregressive learning. Figure 6 below illustrates the network design for learning.

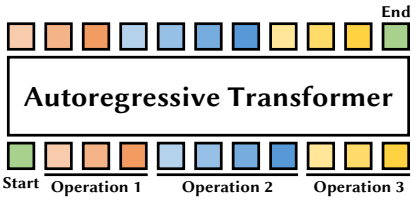


Figure 6: An example of autoregressive learning of three refinement operations.

We simply adopt a transformer network [52] to fit the sequences. Since we primarily focus on unconditional generation in this paper, the network receives no input conditions. The network is responsible for predicting the next token until it reaches the end token and stops generation. However, predictions generated by the transformer may not always correspond to a valid operation, as analyzed in Section 4.2. We may need a solution to determine whether the prediction is valid, possibly through constrained decoding, as shown below, to address this issue and ensure validity.

Apart from the four rules summarized in Section 4.2, there are additional constraints, such as those related to the vertex index and the relationships among different refinement configurations. To facilitate implementation, we can manually hardcode a function, denoted as $\phi(x_i \mid x_1, \ldots, x_{i-1})$, that takes the newly generated token x_i as input, conditioned on all the previous tokens $\{x_1, \ldots, x_{i-1}\}$, and outputs a boolean value indicating whether the new token x_i is compatible with the previous

[4]The second rule states that if there are two edges e_1 and e_2 incident to the split vertex and are subset to a face with a topological label of `F0`, then the topological labels of e_1 and e_2 cannot be `E1`. The explanation for the other rules is quite similar.

[5]We would like to note that our major contribution (Sections 4.1 and 4.2) is theoretically independent of how the learning scheme is designed; we simply present the simplest learning solution here.

https://doi.org/10.52202/085713-5397

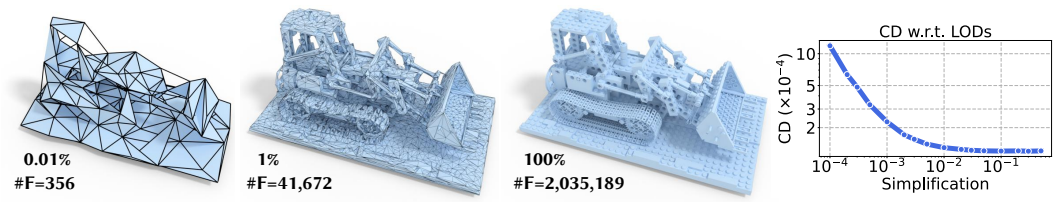


Figure 7: Mesh simplification results at different levels of detail. Our method incorporates isolated points and edges, which also help to approximate the original fine shapes. When simplifying to 1%, the geometric accuracy remains nearly the same.

tokens $\{x_1, \ldots, x_{i-1}\}$. We hardcode the function ϕ in Python with caching. The basic idea to ensure validity is to use a depth-first search-based tree traversal combined with random sampling to introduce diversity. The function ϕ is evaluated at each node, and the most probable node is selected for deeper traversal. If it cannot proceed further, it backtracks to the parent nodes to find alternative solutions. Using this constrained decoding strategy provides an implementation guarantee that the operations predicted by the transformer are always consistent with the previously generated ones.

5 Experiments

Dataset. Our 3D mesh data mainly come from Stanford 3D Scans [50], Thingi10K [63], NeRF [35], AMASS [33], and ShapeNet [2]. The ShapeNet data has been preprocessed by Siddiqui et al. [45], which contains $< 1,700$ vertices and < 800 faces for each mesh, and we follow their settings.

Metrics. Following previous works [32, 45, 51, 62], we adopt the following metrics for evaluation: Minimum Matching Distance (MMD), Coverage (COV), and 1-Nearest-Neighbor Accuracy (1-NNA). For MMD, lower values are better; for COV, higher values are better; and for 1-NNA, 50% is optimal. These metrics are calculated based on Chamfer Distance (CD). To measure visual similarity, we also use FID and KID scores to assess visual quality, with lower values being better. We also report mesh compactness as the average number of vertices and faces per mesh.

Implementation Details. For simplification, we set the penalty factors for vertices, boundary edges, and faces to 0, 1, and 1, respectively, as their default values in our experiments. We follow the standard techniques in LLM, employing Byte Pair Encoding (BPE) to compress tokens. The entire vocabulary size is 16,384, resulting in a length reduction of $2 \sim 3$ times. We basically follow the setup of MeshGPT [45] for our comparative experiments. We adopt a 12-layer transformer with a width of 768 for training, utilizing variable-length `bf16` flash attention [11, 12] for efficient training. The average batch size is ≈ 60 per GPU. The learning rate is 10^{-4} for pre-training across all categories and 10^{-5} for supervised fine-tuning on specific categories (chair, table, bench, and lamp). The model is pretrained on 4 H20 GPUs for ≈ 4 days and fine-tuned on 2 H20 GPUs for ≈ 2 days.

5.1 Ablation Studies

Simplification with Different Levels-of-detail. In this part, we study how the geometric reconstruction accuracy varies w.r.t. different LODs using our simplification algorithm. We test a very complex LEGO shape from [35] that contains up to 2 million faces. Visualization results, along with the numerical accuracy, are presented in Figure 7. We find that our simplification algorithm indeed produces meshes with different LODs that consistently mimic the original shape.

Simplification on Non-manifold Meshes. We study the necessity of using our proposed generalized simplification (GSlim) algorithm over the widely used QSlim [16] and present the results in Figure 8. Since QSlim cannot handle non-manifold cases, it either crashes during processing or produces erroneous results. Our algorithm is often useful since many datasets contain non-manifold cases; using GSlim will enable direct processing of a non-manifold mesh without preprocessing.

Simplification with Different Penalties. To investigate how the penalty factors affect the simplification results, we conduct experiments and present the visualizations in Figure 9. We find that without the boundary edge penalty applied, the simplified mesh cannot maintain a hole in the middle, and the leg length is clearly shortened. With the vertex penalty applied, the resulting triangles become close

https://doi.org/10.52202/085713-5397

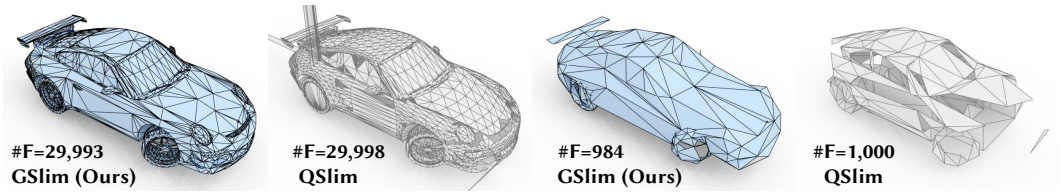


Figure 8: Different mesh simplification algorithms. The popular QSlim algorithm may result in broken and cracked surfaces, while our algorithm (GSlim) still captures the overall shape perfectly, even when the mesh is non-manifold. The QSlim we used is implemented by MeshLab [9].

to equilateral; however, it cannot adaptively distribute its triangles to fit the geometric curvature. The best configuration is therefore $\texttt{VEF} = (0, 1, 1)$, which is the default for all our other experiments.

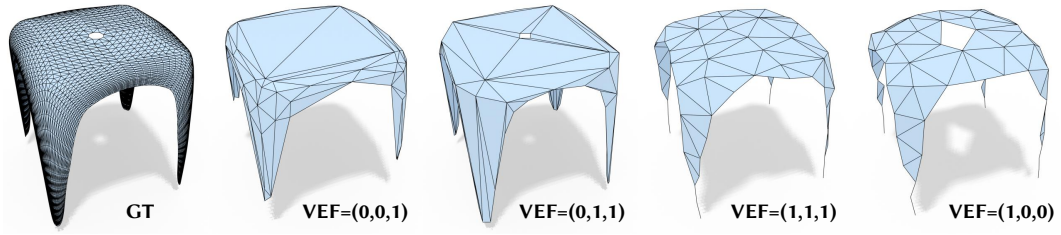


Figure 9: Different penalty factors used in simplification. The four right subfigures show the results with only 2% refinement steps applied. The notation "VEF" at the bottom denotes the penalty values for vertices, boundary edges, and faces, respectively. It is clear that $\texttt{VEF} = (0, 1, 1)$ is the best.

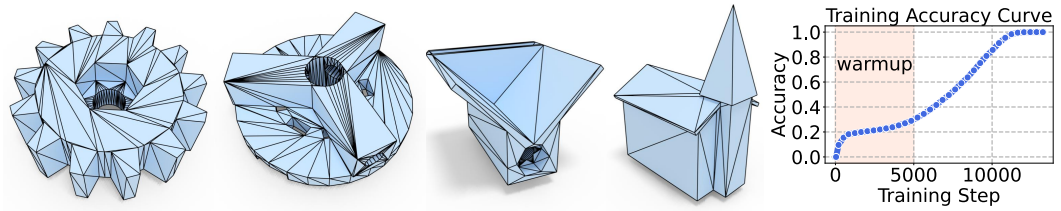


Figure 10: Mesh collection learning with 100 objects taken from the Thingi10K [63] dataset.

5.2 Direct Mesh Encoding

Direct Single Mesh Encoding at Different Levels of Detail. In this part, we study and compare methods capable of obtaining meshes with different resolutions in the context of single mesh fitting. From Figure 11, we find that our method can faithfully reconstruct the shape, while other methods, such as DMesh [46], require significantly more training steps to achieve better results. Additionally, EdgeRunner [48], a pretrained model that receives a point cloud as input, produces only three LODs and struggles to accurately reconstruct shapes at higher LODs. These three LODs not only may correspond to poor quality and deviate significantly from the claimed number of generated faces but are also insufficient to meet practical requirements due to the limited choices of LODs. In contrast, our method can produce any LODs as needed, and the number of generated vertices is precise.

Direct Mesh Collection Encoding with Transformer Networks. Following [44], we investigate whether the refinement sequences can be effectively learned by the transformer using 100 objects. Although simple, the learning process could fail if the sequences are meaningless (nearly random) or if local minima exist, which makes convergence unstable and slow. We present some fitted meshes from the Thingi10K [63] dataset, along with the accuracy curve shown in Figure 10. Fortunately, we did not observe any of these obstacles during training. After the learning rate warms up (increasing linearly to 10^{-4}), we find that our method converges quickly to a very high accuracy, indicating its effectiveness in learning the refinement sequences with transformer networks.

5.3 Unconditional Mesh Generation

We conducted unconditional generation experiments on four categories of the ShapeNet [2] dataset. Visualization results are presented in Figure 12. PolyGen [37] tends to produce simple and incomplete

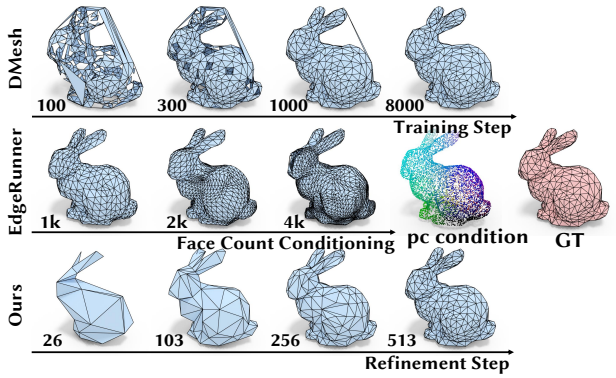


Figure 11: Different LOD methods in mesh reconstruction. EdgeRunner [48] is conditioned on point clouds.

Table 2: Quantitative results of unconditional mesh generation for two categories of the ShapeNet [2] dataset with different LOD ratios.

Class	Ratio	COV↑	MMD↓	1-NNA	FID↓	KID↓	$\lvert V\rvert$ / $\lvert F\rvert$
Lamp	0.1	57.37	2.76	72.47	49.91	0.0246	23 / 36
	0.2	65.45	2.05	63.21	27.91	0.0096	44 / 80
	0.5	**71.79**	1.54	**51.74**	10.44	0.0013	109 / 209
	1.0	70.68	**1.54**	41.20	**6.60**	**0.0005**	216 / 418
Bench	0.1	41.07	1.67	86.53	41.80	0.0187	22 / 30
	0.2	47.40	1.15	83.83	21.94	0.0106	42 / 74
	0.5	54.84	0.87	66.88	10.23	0.0022	105 / 196
	1.0	**56.29**	**0.75**	**39.81**	**2.63**	**0.0001**	208 / 384

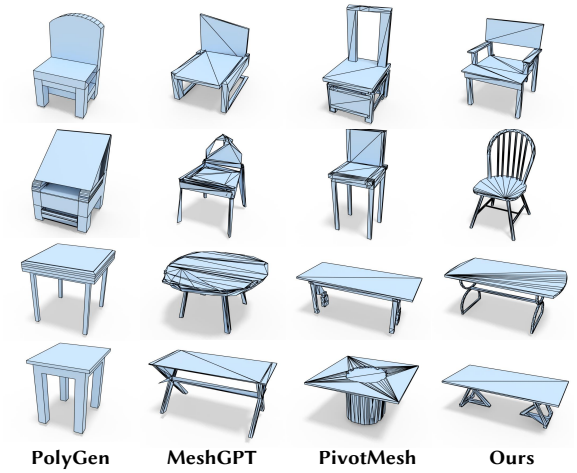


Figure 12: Visualization of unconditional direct mesh generation on the ShapeNet [2] dataset.

Table 3: Quantitative results on unconditional generation across four categories of the ShapeNet [2] dataset. Numerics are copied from [45].

Class	Method	COV↑	MMD↓	1-NNA	FID↓	KID↓	$\lvert V\rvert$ / $\lvert F\rvert$
Chair	AtlasNet [19]	9.03	4.05	95.13	170.71	0.169	2500 / 4050
	BSPNet [8]	16.48	3.62	91.75	46.73	0.030	673 / 1165
	Polygen [37]	31.22	4.41	93.56	61.10	0.043	248 / 603
	GET3D [15]	40.85	3.56	83.04	81.45	0.054	13725 / 27457
	GET3D*	38.75	3.57	84.07	78.29	0.065	199 / 399
	MeshGPT	**43.28**	3.29	75.51	18.46	0.010	125 / 228
	Ours	36.67	**2.44**	**67.40**	**1.54**	**0.0001**	198 / 377
Table	AtlasNet [19]	7.16	3.85	96.30	161.38	0.150	2500 / 4050
	BSPNet [8]	16.83	3.14	93.58	30.78	0.017	420 / 699
	Polygen [37]	32.99	3.00	88.65	38.53	0.029	147 / 454
	GET3D [15]	41.70	2.78	85.54	93.93	0.076	13767 / 27537
	GET3D*	37.95	2.85	81.93	50.46	0.037	199 / 399
	MeshGPT	**45.68**	2.36	72.88	6.24	0.002	99 / 187
	Ours	32.94	**1.94**	**69.46**	**2.05**	**0.0002**	156 / 305
Bench	AtlasNet [19]	20.53	2.47	90.58	189.39	0.163	2500 / 4050
	BSPNet [8]	28.74	2.05	88.44	59.11	0.030	457 / 756
	Polygen [37]	51.92	1.97	76.98	49.34	0.031	172 / 430
	MeshGPT	55.23	1.44	68.24	8.72	0.001	159 / 291
	Ours	**56.29**	**0.75**	**39.81**	**2.63**	**0.0001**	208 / 384
Lamp	AtlasNet [19]	19.97	4.68	91.85	177.91	0.139	2500 / 4050
	BSPNet [8]	18.38	5.32	93.13	112.65	0.077	587 / 1011
	Polygen [37]	47.86	4.18	81.42	52.48	0.025	185 / 558
	MeshGPT	53.88	3.94	65.73	19.91	0.004	150 / 288
	Ours	**70.68**	**1.54**	**41.20**	**6.60**	**0.0005**	216 / 418

meshes because it requires separate training of the vertex and face models. MeshGPT [45] may generate meshes with messy tessellation due to its limited capabilities and learning challenges. PivotMesh [59] can produce plausible results, but some artifacts may be observed in local details. Our method successfully generates shapes with complex topology and clean tessellation. The numerical results in Table 3 further validate the superiority of our approach in terms of generation diversity and fidelity, with most metrics surpassing those of other mesh generation methods. Table 2 presents metrics for different simplification ratios. We find that a 50% simplification still yields decent results. The average generation time (full) is ≈ 1 minute, suggesting a potential additional speedup of 50%.

We also compare our tokenization scheme with other tokenization methods, such as EdgeRunner [48] and BPT [60], in Table 4 under an unconditional setting. We observe that our method achieves comparable performance to the other methods when all (100%) auto-regressive steps are applied. Notably, we find that it is considerably superior when only a few steps (10%, 20%, 50%) are applied compared to the other methods. This is because the intermediate results produced by our scheme are consistently good approximations of the finest level of detail, whereas other approaches produce meshes in a partial-to-complete manner, causing their intermediate results to deviate significantly from the targets if the generation process is not fully completed.

In the last two columns of Table 4, we also present the average number of tokens required by each tokenization scheme, either per shape or per geometric element. We notice that our method, on average, needs fewer tokens to represent a mesh than other methods, even though our method generally requires more tokens to represent a vertex than other methods do to represent a triangle. This demonstrates the superiority of our tokenization scheme regarding compactness.

https://doi.org/10.52202/085713-5397

Table 4: Quantitative results of various tokenization approaches [48, 60] with different percentages of auto-regressive steps applied are measured on the bench category of the ShapeNet [2] dataset. EdgeRunner [48] and BPT [60] are based on triangles, while the elements in our method are vertices.

Methods	COV ↑				MMD ↓				#Token ↓	
	10%	20%	50%	100%	10%	20%	50%	100%	per shape	per elem.
EdgeRunner [48]	9.43	30.99	40.33	54.90	10.32	4.87	2.59	0.81	5840	4.5
BPT [60]	15.15	26.65	34.11	**56.35**	3.02	1.97	1.26	0.76	3235	**2.73**
Ours	**41.07**	**47.40**	**54.84**	56.29	**1.67**	**1.15**	**0.87**	**0.75**	**2556**	9.1

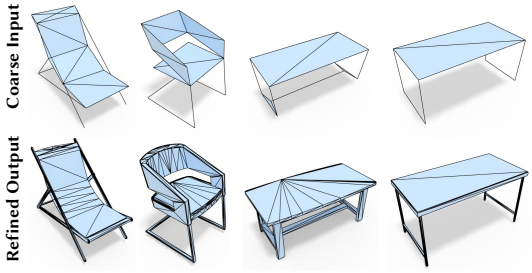


Figure 13: Our representation supports refining a coarse mesh input (top row) into a fine-grained output mesh (bottom row).

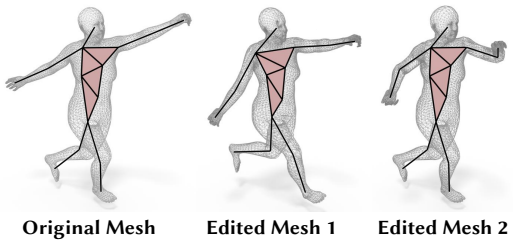


Figure 14: A coarse skeleton can be easily derived using our approach for fine-grained mesh editing using surface deformation.

5.4 Possible Applications with Our Approach

Shape Refinement from Sketchy Inputs. Our method can also be used to refine a 3D sketchy shape, as shown in Figure 13. This is practically useful for 3D modelers, as they only need to specify a coarse outline without expending effort on fine-grained geometric details, significantly saving time.

Shape Editing by Manipulating a Coarse Skeleton. A skeleton can be easily derived using our approach to facilitate the manipulation of a fine shape, as shown in Figure 14. By modifying the coarse skeleton, marked in red, one can easily manipulate the human shape through surface deformation.

6 Limitation, Future Work and Conclusion

Like prior works [45], the main drawback of our method is its relatively limited generalization ability compared to those built upon continuous space (*e.g.*, Diffusion). Using an order-of-magnitude larger amount of training data may alleviate this problem. To suppress excessive topological flexibility, we may use PSC to create an initial result with the desired topology and then use the normal progressive meshes representation [22] for the remaining steps of further refinement without changing the topology. During generation, we may reduce the linear complexity to logarithmic complexity by using parallel vertex splits [14, 23] in future work. Lastly, our method lacks gradients to update the PSC; therefore, it would be interesting to derive a differentiable PSC that can adaptively fit a mesh in a coarse-to-fine manner through local remeshing with gradient guidance.

This paper suggests formulating direct mesh generation in the order of levels of detail (LODs), consistent with how humans perceive objects, and generating in a preferable coarse-to-fine manner. The basic idea is to generalize meshes to simplicial complexes, reverse the mesh simplification process, and train a model to learn mesh generation, initially starting with a single point and iteratively refining the previous coarse results. Experiments verify the promise and effectiveness of our approach.

Acknowledgment

We sincerely thank Prof. Zhen Liu for his valuable discussions regarding our work. This work was supported by the Key-Area Research and Development Program of Guangdong Province, China, under Grant 2024B0101040004 for the project titled “Research and Application of Common Key Technologies for Embodied Intelligence of Robots Based on AI Large Models.”

References

[1] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, December 1996. ISSN 0098-3500. doi: 10.1145/235815.235821. URL https://doi.org/10.1145/235815.235821.

[2] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[3] 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, 2024.

[4] 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, 2024. URL https://arxiv.org/abs/2408.02555.

[5] Zhang Chen, Yinda Zhang, Kyle Genova, Sean Fanello, Sofien Bouaziz, Christian Häne, Ruofei Du, Cem Keskin, Thomas Funkhouser, and Danhang Tang. Multiresolution deep implicit functions for 3d shape representation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13087–13096, 2021.

[6] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5939–5948, 2019.

[7] Zhiqin Chen and Hao Zhang. Neural marching cubes. *ACM Transactions on Graphics (TOG)*, 40(6):1–15, 2021.

[8] Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. Bsp-net: Generating compact meshes via binary space partitioning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 45–54, 2020.

[9] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool. In Vittorio Scarano, Rosario De Chiara, and Ugo Erra, editors, *Eurographics Italian Chapter Conference*. The Eurographics Association, 2008. ISBN 978-3-905673-68-5. doi: 10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136.

[10] Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, Frederick Brooks, and William Wright. Simplification envelopes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, page 119–128, New York, NY, USA, 1996. Association for Computing Machinery. ISBN 0897917464. doi: 10.1145/237170.237220. URL https://doi.org/10.1145/237170.237220.

[11] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024.

[12] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[13] Boyang Deng, Kyle Genova, Soroosh Yazdani, Sofien Bouaziz, Geoffrey Hinton, and Andrea Tagliasacchi. Cvxnet: Learnable convex decomposition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 31–44, 2020.

[14] E. Derzapf and M. Guthe. Dependency-free parallel progressive meshes. *Computer Graphics Forum*, 31(8):2288–2302, 2012. doi: https://doi.org/10.1111/j.1467-8659.2012.03154.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2012.03154.x.

[15] 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. In *Advances In Neural Information Processing Systems*, 2022.

[16] Michael Garland and Paul S Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, 1997.

[17] Michael Garland and Yuan Zhou. Quadric-based simplification in any dimension. *ACM Transactions on Graphics (TOG)*, 24(2):209–239, 2005.

[18] Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas Funkhouser. Local deep implicit functions for 3d shape. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4857–4866, 2020.

[19] Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. A papier-mâché approach to learning 3d surface generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 216–224, 2018.

[20] Antoine Guédon and Vincent Lepetit. Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5354–5363, 2024.

[21] Zekun Hao, David W. Romero, Tsung-Yi Lin, and Ming-Yu Liu. Meshtron: High-fidelity, artist-like 3d mesh generation at scale. *arXiv preprint arXiv:2412.09548*, 2024.

[22] Hugues Hoppe. Progressive meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, page 99–108, New York, NY, USA, 1996. Association for Computing Machinery. ISBN 0897917464. doi: 10.1145/237170.237216. URL `https://doi.org/10.1145/237170.237216`.

[23] Liang Hu, Pedro V Sander, and Hugues Hoppe. Parallel view-dependent refinement of progressive meshes. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 169–176, 2009.

[24] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2d gaussian splatting for geometrically accurate radiance fields. In *SIGGRAPH 2024 Conference Papers*. Association for Computing Machinery, 2024. doi: 10.1145/3641519.3657428.

[25] Jiabao Lei and Kui Jia. Analytic marching: An analytic meshing solution from deep implicit surface networks. In *International Conference on Machine Learning 2020 ICML-20*, 7 2020.

[26] Jiabao Lei, Kui Jia, and Yi Ma. Learning and meshing from deep implicit surface networks using an efficient implementation of analytic marching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021. doi: 10.1109/TPAMI.2021.3135007.

[27] Jiahao Li, Hao Tan, Kai Zhang, Zexiang Xu, Fujun Luan, Yinghao Xu, Yicong Hong, Kalyan Sunkavalli, Greg Shakhnarovich, and Sai Bi. Instant3d: Fast text-to-3d with sparse-view generation and large reconstruction model. *arXiv preprint arXiv:2311.06214*, 2023.

[28] Yiyi Liao, Simon Donne, and Andreas Geiger. Deep marching cubes: Learning explicit surface representations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2916–2925, 2018.

[29] Stefan Lionar, Jiabin Liang, and Gim Hee Lee. Treemeshgpt: Artistic mesh generation with autoregressive tree sequencing. *arXiv preprint arXiv:2503.11629*, 2025.

[30] Zhen Liu, Yao Feng, Michael J Black, Derek Nowrouzezahrai, Liam Paull, and Weiyang Liu. Meshdiffusion: Score-based generative 3d mesh modeling. *arXiv preprint arXiv:2303.08133*, 2023.

[31] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. 21(4):163–169, 1987.

[32] Shitong Luo and Wei Hu. Diffusion probabilistic models for 3d point cloud generation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2837–2845, 2021.

[33] Naureen Mahmood, Nima Ghorbani, Nikolaus F Troje, Gerard Pons-Moll, and Michael J Black. Amass: Archive of motion capture as surface shapes. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5442–5451, 2019.

[34] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4460–4470, 2019.

[35] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[36] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM transactions on graphics (TOG)*, 41(4): 1–15, 2022.

[37] Charlie Nash, Yaroslav Ganin, S. M. Ali Eslami, and Peter W. Battaglia. Polygen: An autoregressive generative model of 3d meshes. *ICML*, 2020.

[38] Junyi Pan, Xiaoguang Han, Weikai Chen, Jiapeng Tang, and Kui Jia. Deep mesh reconstruction from single rgb images via topology modification networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9964–9973, 2019.

[39] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 165–174, 2019.

[40] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pages 523–540. Springer, 2020.

[41] Jovan Popović and Hugues Hoppe. Progressive simplicial complexes. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 217–224, 1997.

[42] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. *SIGGRAPH Comput. Graph.*, 26(2):65–70, July 1992. ISSN 0097-8930. doi: 10.1145/142920.134010. URL `https://doi.org/10.1145/142920.134010`.

[43] Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. *Advances in Neural Information Processing Systems*, 34:6087–6101, 2021.

[44] 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. *arXiv preprint arXiv:2409.20562*, 2025.

[45] Yawar Siddiqui, Antonio Alliegro, Alexey Artemov, Tatiana Tommasi, Daniele Sirigatti, Vladislav Rosov, Angela Dai, and Matthias Nießner. Meshgpt: Generating triangle meshes with decoder-only transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[46] Sanghyun Son, Matheus Gadelha, Yang Zhou, Zexiang Xu, Ming C Lin, and Yi Zhou. Dmesh: A differentiable mesh representation. *arXiv preprint arXiv:2404.13445*, 2024.

[47] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3d shapes. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11358–11367, 2021.

[48] 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. *arXiv preprint arXiv:2409.18114*, 2024.

[49] Keyu Tian, Yi Jiang, Zehuan Yuan, Bingyue Peng, and Liwei Wang. Visual autoregressive modeling: Scalable image generation via next-scale prediction. *Advances in neural information processing systems*, 37:84839–84865, 2024.

[50] Stanford University. Stanford 3d scanning repository. `http://graphics.stanford.edu/data/3Dscanrep/`, 1999.

[51] Arash Vahdat, Francis Williams, Zan Gojcic, Or Litany, Sanja Fidler, Karsten Kreis, et al. Lion: Latent point diffusion models for 3d shape generation. *Advances in Neural Information Processing Systems*, 35:10021–10039, 2022.

[52] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[53] Hanxiao Wang, Biao Zhang, Weize Quan, Dong-Ming Yan, and Peter Wonka. iFlame: Interleaving full and linear attention for efficient mesh generation. 2025. URL `https://arxiv.org/abs/2503.16653`.

 https://doi.org/10.52202/085713-5397

[54] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *Proceedings of the European conference on computer vision (ECCV)*, pages 52–67, 2018.

[55] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *arXiv preprint arXiv:2106.10689*, 2021.

[56] Zhengyi Wang, Jonathan Lorraine, Yikai Wang, Hang Su, Jun Zhu, Sanja Fidler, and Xiaohui Zeng. Llama-mesh: Unifying 3d mesh generation with language models. *arXiv preprint arXiv:2411.09595*, 2024.

[57] D. F. Watson. Computing the n-dimensional delaunay tessellation with application to voronoi polytopes*. *The Computer Journal*, 24(2):167–172, 01 1981. ISSN 0010-4620. doi: 10.1093/comjnl/24.2.167. URL `https://doi.org/10.1093/comjnl/24.2.167`.

[58] Chao Wen, Yinda Zhang, Zhuwen Li, and Yanwei Fu. Pixel2mesh++: Multi-view 3d mesh generation via deformation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1042–1051, 2019.

[59] Haohan Weng, Yikai Wang, Tong Zhang, C. L. Philip Chen, and Jun Zhu. Pivotmesh: Generic 3d mesh generation via pivot vertices guidance, 2024.

[60] 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. *arXiv preprint arXiv:2411.07025*, 2024.

[61] Xiang Zhang, Yawar Siddiqui, Armen Avetisyan, Chris Xie, Jakob Engel, and Henry Howard-Jenkins. Vertexregen: Mesh generation with continuous level of detail. *arXiv preprint arXiv:2508.09062*, 2025.

[62] Linqi Zhou, Yilun Du, and Jiajun Wu. 3d shape generation and completion through point-voxel diffusion. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5826–5835, 2021.

[63] Qingnan Zhou and Alec Jacobson. Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797*, 2016.

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: We have clearly pointed out our contributions in the abstract and introduction sections, particularly in the last paragraph of the introduction section.

   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: The limitations are discussed in the final section of this paper.

   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]

https://doi.org/10.52202/085713-5397

Justification: There are no theoretical results presented in this paper.

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: Key information is provided in this paper to enhance reproducibility. The method section and the experimental section include sufficient implementation details.

   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: [Yes]

Justification: We provide demo code in the supplementary materials to help examine our method.

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 have provided sufficient training and testing details in both the experimental section and the supplementary materials.

   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 lacked computational resources, so they are not included in this paper.

   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).

https://doi.org/10.52202/085713-5397

- 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 have provided this information in the experimental section.

   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 comply with the 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: [NA]

    Justification: We do not think that our method can have any positive or negative societal impacts.

    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]

    Justification: Our paper poses no such risks.

    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: All the assets, including code, data, and models, are properly credited and respected.

    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.

https://doi.org/10.52202/085713-5397

- 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: [Yes]

    Justification: We include the code for reproducibility in the supplementary materials.

    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 paper 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.
    - 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: No human participants were involved; thus, IRB approval and risk disclosure are not required.

    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.

https://doi.org/10.52202/085713-5397