CoMesh: Constrained Progressive Mesh Generation for Engineering Design

Anonymous authors
Paper under double-blind review

Abstract

Engineering shape design calls for models that explicitly generate meshes, which preserve broad geometric freedom and interface naturally with downstream physical simulations. Yet existing mesh generation approaches struggle to reconcile with spatial engineering constraints, achieve simulation-level quality, and often require costly autoregressive modeling over long refinement histories. We introduce *CoMesh*, a framework that starts with a prescribed seed mesh that hard-encodes engineering constraints and progressively introduces new geometric degrees of freedom through edge splits. Its key component is a novel serialization algorithm that constructs refinement trajectories for training in the same forward direction as generation, using a shared scheduler to select the edge refined at each step. We establish sufficient conditions for self-intersection free refinement, and translate them into a dihedral-based procedure for placing new vertices. Compared to baseline serializations, our approach guarantees that hard constraints can be embedded and reduces each refinement action to a 2D local displacement. This turns autoregressive generation into a local, state-conditioned problem, enabling an efficient, compact model independent of refinement history. Across three engineering families, mounting brackets, transonic wings, and ship hulls, CoMesh consistently generates diverse meshes with substantially improved constraint validity and element quality, while enabling markedly more efficient training and inference. These results demonstrate that using a deterministic refinement schedule with learned local geometry provides an efficient formulation for mesh-level generative design.

1 Introduction

Engineering shape design requires a representation to describe feasible geometries, and then improving it with functional signals from physical solvers through iterations. A compact parameterization is traditionally used to reduce a high-dimensional geometric problem to a small number of design variables. However, they impose a strong prior that only geometries expressible by the chosen parameterization can be explored. An appealing alternative is therefore to generate the discrete mesh directly, and place engineering priors only where they are actually required, such as smoothness and spatial constraints (Wei et al., 2025b). This is also favorable for downstream solvers, either numerical (Hughes et al., 2005) or data-driven (Pfaff et al., 2021; Zhou et al., 2026), that are typically designed to work with a mesh for functional evaluation.

Recent advances in neural mesh generation make this direction increasingly plausible. Flow-based generative models (Sun et al., 2026) and Autoregressive models (Siddiqui et al., 2024) can now generate explicit triangle meshes with high geometric fidelity and scale. However, most existing work is motivated by graphics to reconstruct an asset already specified through input, whereas engineering design requires the model to act differently as a generator to create *new* and *diverse* designs while satisfying hard spatial relations. A structural bracket, for example, may be free to vary almost everywhere while being required to connect exactly to prescribed mounting panels. The downstream physical evaluation solvers also pose higher mesh-quality requirements (Knupp, 2001).

Among the approaches, progressive mesh generation provides a natural starting point. It starts from a *base mesh* that acts as a skeleton, and introduces new vertices through a sequence of refinement operations (Zhang et al., 2025a). It is attractive that the base mesh can directly encode spatial con-

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.

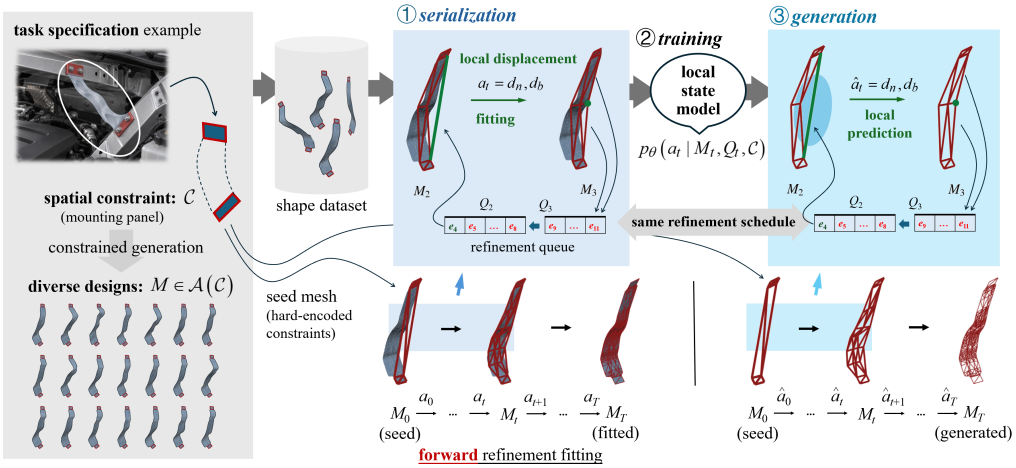


Figure 1: Example of constrained mesh generation for mounting brackets.

straints, such as mounting interfaces, and refinement can adaptively allocate resolution only where complex geometry requires it. The central difficulty then becomes how to serialize the refinement process. Existing methods typically construct *backward*: a fine mesh is repeatedly simplified, and the inverse operations define the generation trajectory. This creates two problems for engineering mesh applications. First, the resulting coarse mesh is determined by the target geometry rather than by a prescribed constraint-encoding seed, leading to a mismatch between training and inference. Second, the reversal causes serialization and generation to operate on fundamentally different information. Simplification selects operations from the full fine mesh, whereas generation must infer the corresponding inverse operations from a coarse mesh. As refinement proceeds, these decisions can depend increasingly on global geometry or long autoregressive histories. We hypothesize that this backward–forward asymmetry contributes to poor mesh quality and inefficient learning.

We address these limitations with *CoMesh*, whose core is the *Forward Refinement Fitting* (FRF) serialization algorithm. FRF starts from the prescribed seed mesh and constructs the training trajectory in the same forward direction as generation. The same deterministic scheduler is shared between serialization and inference, selecting the edge to split at each refinement step. During serialization, the ground-truth position of the new vertex is determined on the target surface via a fitting procedure supported by theoretical conditions for valid refinement. Each refinement action is then represented as a two-dimensional displacement in an edge-local coordinate frame, which the model can learn from a bounded local neighborhood. This formulation allows *CoMesh* to use a compact local model without conditioning on the full mesh or a long refinement history. We evaluate *CoMesh* on three engineering shape families: mounting brackets, transonic wings, and ship hulls. Under matched settings, comparisons with baseline mesh generation show that this shared refinement procedure yields substantially higher-quality meshes, while at the same time being more resource efficient.

Contributions. Our contributions are threefold. **(1)** We formulate constrained engineering shape generation as a progressive mesh refinement problem. It starts with a seed explicitly encoding hard spatial constraints, while the refinement process progressively introduces geometric degrees of freedom to generate diverse admissible shapes. **(2)** We introduce FRF, a serialization method that constructs training trajectories directly from the prescribed seed mesh. It aligns serialization with generation and reduces each refinement action to a local displacement, enabling a compact local-state model instead of a large causal model over the full refinement history. **(3)** We establish a benchmark spanning three representative engineering shape families, designed to stress different engineering requirements, and use it to systematically compare mesh generation methods.

2 Related Work

Generative design and shape representation. Shape representation remains a foundation for engineering design optimization. Traditionally, engineers relied on low-dimensional parameteriza-

tions that expose a compact set of design variables, such as the spline-based representations that support most modern CAD libraries, free-form deformation, and problem-specific bases like class–shape transformation (Zhu & Qin, 2014). They are attractive because they make design optimization tractable by reducing the design space, and consequently become a common choice for generative models to work on latent spaces (Chen et al., 2019; Xie et al., 2024) and CAD representations(Wu et al., 2021; Xu et al., 2022; Zhang et al., 2025b). However, the same reduction also imposes a strong prior that is often suboptimal. Implicit representations (Park et al., 2019; Mescheder et al., 2019; He et al., 2025) are straightforward, but still limited to the base grid resolution. Our work takes a complementary route to generate the surface mesh directly, following (Jabón et al., 2024; Wei et al., 2025b). This provides the broadest freedom while working naturally with downstream solvers without a problematic meshing procedure (Owen et al., 2026), but also poses challenges to model efficiency and mesh quality that this paper aims to address.

Autoregressive mesh generation. Explicit mesh generators must resolve both geometry and connectivity. PolyGen (Nash et al., 2020) separates vertex generation from face connectivity, whereas MeshGPT (Siddiqui et al., 2024) and subsequent methods serialize local mesh elements into token sequences. Adjacency-aware tokenizations such as MeshAnything (Chen et al., 2024; 2025), EdgeRunner (Tang et al., 2025), TreeMeshGPT (Lionar et al., 2025), OctGPT (Wei et al., 2025a) and MeshRipple (Lin et al., 2026) reduce sequence length and improve local coherence. Most recent works (Xu et al., 2026a; Wang et al., 2026; Xu et al., 2026b) also focus on token compression and aim for higher mesh resolution. These methods are primarily designed for reconstruction from a representation of the target geometry, whereas our engineering design problems require free generation subject to spatial constraints.

Progressive meshing and coarse-to-fine generation. Progressive meshes start with a fine mesh and repeatedly collapse the edges with representative metrics until reaching a base mesh. Then the fine mesh can be represented with the base plus a sequence of vertex splits, each obtained by reversing an edge collapse (Hoppe, 1997). Recent generative methods like VertexRegen (Zhang et al., 2025a) adopt it to generate the autoregressive sequence, and ARMesh (Lei et al., 2025) expands it with progressive simplicial complexes (Popović & Hoppe, 1997). Similar ideas also exist in neural mesh refinement. Neural Subdivision (Liu et al., 2020) and Neural Mesh Refinement (Zhu et al., 2025) construct a bijective coarse–fine parameterization during the collapse, then use it to construct the finer level mesh based on local split for every edge. SubdivAR (Guo et al., 2026) further introduces an autoregressive model for this task. However, engineering generation requires a prescribed seed mesh to encode constraints. We observe in our experiments that progressive meshing fails to preserve a specific seed complex (Section 6.1), and the bijective mapping is fragile at the seed-mesh level (Appendix C.3). More importantly, reversing these collapse sequences keeps the vertex selection and topological update within representation, which significantly increases the learning load.

3 Constrained Engineering Mesh Generation

3.1 Problem formulation

We consider engineering design problems defined by a set of spatial constraints $\mathcal{C}$ that may prescribe interfaces to existing components, packaging or keep-out regions, and other geometric requirements. These constraints define a family of admissible surfaces $\mathcal{A}(\mathcal{C})$. We directly use a triangle mesh $M = (K, x)$, to describe the surface, where K is a 2-complex and $x : V(K) \to \mathbb{R}^3$ embeds its vertices. Our goal is to learn a conditional distribution

$$M \sim p_\theta(M \mid \mathcal{C}) \tag{1}$$

that places high probability on diverse meshes in $\mathcal{A}(\mathcal{C})$.

This formulation differs from conventional engineering parameterization, which specifies a map $g_{\mathcal{C}} : \mathbb{R}^d \to \mathcal{S}(\mathcal{C})$ and optimizes a fixed-dimensional design vector $\alpha \in \mathbb{R}^d$. Its tractability comes from restricting the reachable geometry to the image of $g_{\mathcal{C}}$. Direct mesh generation instead exposes geometric degrees of freedom at the discretization level. It also differs from target-conditioned mesh reconstruction because $\mathcal{C}$ does not specify which member of the admissible family should be produced. Ground-truth surfaces are available during training only to provide geometric supervision.

3.2 SEED-CONSTRAINED PROGRESSIVE GENERATION

We hard encode spatial constraints $\mathcal{C}$ in a *seed mesh* $M_0(\mathcal{C}) = (K_0, x_0)$, whose vertices are fixed during generation. The generated meshes consequently must pass these vertices, so the prescribed information, such as interface locations, are fulfilled.

Starting from M_0, we generate a design through progressive local refinement $a_0, a_1, \cdots, a_T$, where each action a_t either introduces an edge split or stops further refinement of the current edge. Edge splitting preserves the existing vertices, thereby preserves spatial constraints embedded in seed. What remains stochastic is the geometry introduced by refinement and the amount of resolution allocated to different regions.

This representation naturally defines a sequential generative process over refinement events. In its general autoregressive form, $p_\theta(a_{0:T-1} \mid \mathcal{C}) = \prod_{t=0}^{T-1} p_\theta(a_t \mid a_{<t}, \mathcal{C})$, where $a_{<t}$ denotes the preceding refinement decisions. Our method instantiates this formulation in two stages. In Section 4, *forward refinement fitting*(FRF) constructs a refinement trajectory directly from a prescribed seed toward a ground-truth surface. We will show that FRF removes the need to explicitly model the full action history by a deterministic *refinement schedule*. In Section 5, the resulting process is implemented with a compact local-state model that predicts only the remaining geometric decision and is sampled repeatedly at inference to obtain diverse designs.

4 FORWARD REFINEMENT FITTING SERIALIZATION

4.1 DETERMINISTIC REFINEMENT SCHEDULE

FRF uses the same deterministic refinement schedule during serialization and generation, so the learned action contains only the remaining local geometric choice. This is achieved by maintaining a queue Q_t at mesh state M_t that stores eligible edges. Let $\mathcal{R}_t = (M_t, Q_t)$ represent a *refinement state* at step t. For each run, Q_0 is initialized with the active edges in the seed mesh. A deterministic priority function ρ then repeatedly selects the edge $e_t = \pi(\mathcal{R}_t) = \arg\max_{e \in Q_t} \rho(e; M_t)$. We use longest-edge priority in our experiments, although FRF only requires the selection rule to be deterministic and replayable.

Working on the selected edge, an algorithm described in Section 4.2 then determines the action a_t given the ground-truth surface S_{GT} during the serialization phase, which is replaced by the model prediction during the generation phase. If the action is STOP, the edge is marked inactive, and M_t is unchanged. Otherwise, FRF performs a standard edge split where one vertex is inserted on e_t. The edge is replaced by two sub-edges, and the new vertex is connected to the opposite vertex of each incident triangle. These contribute to four new edges that are pushed into Q_t. The updated mesh and queue define $\mathcal{R}_{t+1}$, from which the next active edge is selected again. Hence, the serialization does not need to store an edge identifier or a connectivity token. Another beneficial property is that the split action preserves the mesh topology, so the hard spatial constraints embedded in the seed mesh can be naturally maintained.

4.2 REFINEMENT TARGET SELECTION

Let the active edge have endpoints v_0, v_1, midpoint $m_e = \frac{v_0+v_1}{2}$, length $L = \|v_1 - v_0\|$, and unit direction $\hat{e} = (v_1 - v_0)/L$. From the adjacent face normals, we construct a unit vector $\hat{n} \perp \hat{e}$ and define $\hat{b} = \hat{e} \times \hat{n}$. Then $(\hat{e}, \hat{n}, \hat{b})$ becomes the edge local frame.

We restrict the new vertex to the normal plane $\Pi_e = \{p \in \mathbb{R}^3 : (p - m_e) \cdot \hat{e} = 0\}$, so the vertex should be on the one-dimensional intersection curve $\gamma_e = S_{\text{GT}} \cap \Pi_e$, and a target point $w \in \gamma_e$ is represented by two scale-normalized coordinates, $w = m_e + L(d_n \hat{n} + d_b \hat{b})$ where $d_n = (w - m_e) \cdot \hat{n}/L$, $d_b = (w - m_e) \cdot \hat{b}/L$. The learned action is thus only

$$a_t \in \{\text{STOP}\} \cup \{(d_n, d_b)\} \tag{2}$$

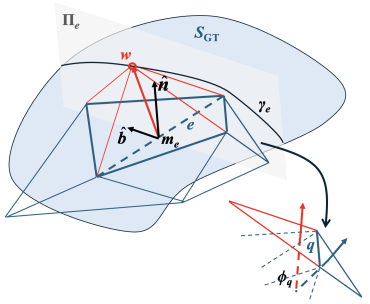


Figure 2: The illustration of one fitting step, with a demonstration of the signed dihedral ϕ_q.

Then we need a strategy for selecting w from γ_e to ensure the resulting mesh is high-quality; most importantly, to ensure that the new faces do not intersect with existing ones. This is non-trivial because of the historical effect: one step could be intersection-free but may squeeze the valid γ_e in the future, ultimately causing failure. To tackle this, we tend to a theoretical guarantee for the chosen interior split that preserves the embedding, and use it to guide the selection strategy below. Full thresholds and the shipped per-family configuration are in Appendix C.

Initial guess. For an interior edge, the target is drawn from the feasible section γ_e^* of γ_e, which is defined as the section between the projection point along the edge normal ($d_b = 0$) and the nearest point on γ_e of the midpoint m_e. It is also clipped by keep-out radii around the boundary. An initial guess of w^0 is then randomly sampled along γ_e^*.

Folding guard and rescue. We check the initial guess w^0 with a signed dihedral to increase the success rate for generating intersection-free samples. For an affected interior edge q, let $\hat{\mathbf{t}}_q$ be its oriented unit tangent and let $\mathbf{n}_q^-(w)$ and $\mathbf{n}_q^+(w)$ be the consistently oriented normals of its two adjacent faces. We define

$$\phi_q(w) = \operatorname{atan2}\left(\hat{\mathbf{t}}_q \cdot \left(\mathbf{n}_q^-(w) \times \mathbf{n}_q^+(w)\right), \mathbf{n}_q^-(w) \cdot \mathbf{n}_q^+(w)\right) \in (-\pi, \pi). \tag{3}$$

and we can obtain the worst dihedral angle affected by adding point w around edge e via

$$\Gamma_e(w) = \min_{q \in \mathcal{E}_{\text{aff}}(w)} \sigma_q^{\text{GT}} \phi_q(w), \tag{4}$$

where $\sigma_q^{\text{GT}} \in -1, +1$ is the bending direction on ground-truth surface corresponding to edge q.

Intuitively, a nonnegative $\Gamma_e(w)$ means that every affected hinge bends in the target-consistent direction, which helps avoid face intersection when splitting them in the future. We formalize this intuition under a set of stronger sufficient assumptions:

Definition 1 (Target-curvature compatibility)**.** Let S^{GT} be a compact embedded orientable C^2 surface admitting a tubular neighborhood with unique nearest-point projection P_S. A triangulated oriented surface M is *target-curvature compatible* with S^{GT} if (1) $P_S|_M : M \to S^{\text{GT}}$ is an orientation-preserving homeomorphism; and (2) for every interior edge q where the target bending sign is defined, $\sigma_q^{\text{GT}} \phi_q \geq 0$.

Theorem 1 (Safe progressive refinement)**.** *Suppose the initial mesh M_0 is target-curvature compatible. Assume that, at every interior split, (1) the refined two-triangle edge star remains inside the tubular neighborhood of S_{GT}; (2) for each new face, the target normal rotates from a neighboring face normal in the target-curvature direction by less than $\pi/2$; and (3) for every affected interior edge that σ_q^{GT} is defined, $0 \leq \sigma_q^{\text{GT}} \phi_q < \pi/2$. Then every intermediate mesh M_t remains target-curvature compatible with S_{GT} and is therefore embedded and self-intersection free.*

The formal proof is given in Appendix A. The theorem only covers interior splits. Boundary edges are handled separately, as described next, and are outside this theoretical guarantee. We also note that the theorem is deliberately conservative. Our implementation only uses the sign of Γ_e as a folding guard, and adopts a simpler surrogate $\sigma_q^{\text{GT}} \equiv +1$ because estimating a reliable target bending sign for every newly created edge is difficult. Specifically, we first evaluate the sampled target to be $\Gamma_e(w^0) > 0$. However, this is not usually possible for every action because the simpler surrogate and the seed mesh cannot fully meet the theoretical requirement. We consequently introduced a rescue procedure, that if $\Gamma_e(w^0) \leq 0$, FRF probes additional points along the feasible section and selects $w^{\text{res}} = \arg\max_{w \in \gamma_e^*} \Gamma_e(w)$.

Guideline correspondences. Beyond projection to γ_e, we also support prescribing seed mesh edges to follow designated ground-truth curves (refer to *guidelines*), such as boundary edges. To avoid conflict, we construct a correspondence from each associate mesh edge to a section on the guideline by first mapping the vertices. During refinement, whenever such an edge is split, we place the new vertex w at the intersection of Π_e with its associated section and update the correspondence accordingly. This propagates the prescribed line constraint throughout refinement.

Stopping, validity check, and multi-draw. A STOP signal is recorded when the current edge already approximates the target within a prescribed offset tolerance $||d_n \hat{n} + d_b \hat{b}||_2 < \delta$ or when

further refinement would fall below the minimum length scale. Otherwise, new faces are checked for intersections with the existing mesh, which would terminate and reject the entire serialization. Since the target selection is stochastic, we can retain several distinct sequences per sample to improve the success rate and enlarge the training dataset.

5 Local State Encoder

From autoregressive history to refinement state FRF removes the reliance of a common progressive formulation on the full action history that is needed only to determine where and how the next topological operation should occur. Given $\mathcal{R}_t$, both the active edge $e_t = \pi(\mathcal{R}_t)$ and the connectivity update are deterministic, and the refinement state evolves as $\mathcal{R}_{t+1} = F(\mathcal{R}_t, a_t)$. The progressive process can therefore be written as $p_\theta(a_{0:T-1} \mid \mathcal{C}) = \prod_{t=0}^{T-1} p_\theta(a_t \mid \mathcal{R}_t, \mathcal{C})$. Since the refinement is local, $\mathcal{R}_t$ can further reduce this state to a bounded neighborhood of the active edge. Previous decisions still affect future predictions through the geometry and processed-edge status already stored in $\mathcal{R}_t$. This brings two benefits: (1) The model cost per refinement query is independent of global mesh resolution; and (2) Every intermediate state could be a legitimate place to be edited.

Model. The input is the k-hop vertex neighborhood of the split edge, each vertex written as $(p - v_0)/L$ in the edge basis $(\hat{e}, \hat{n}, \hat{b})$ and tagged with its hop distance. This local-frame representation naturally commutes with rigid motion and uniform scaling. A bidirectional transformer over that set produces a query q from the two encoded endpoint tokens and a boundary flag. A hierarchical head first predicts d_n from q, then d_b from q and d_n, both over 2^8 uniform bins with corpus-fixed scales. The stop decision is one extra class on the d_n head. Training is per-step and independent across steps, which allows a large effective batch. Appendix D.1 provides further information.

Generation. Decoding replays the same refinement queue in FRF. An edge is popped from Q_t, and the model sample d_n, d_b from local mesh input. The vertex is then reconstructed in the frame. We only apply a self-intersection guard here, and all other code is either embedded in the queue or learned by the model.

6 Experiments

We evaluate three families of engineering parts across different design tasks. We display samples of the meshes in Fig. 3.

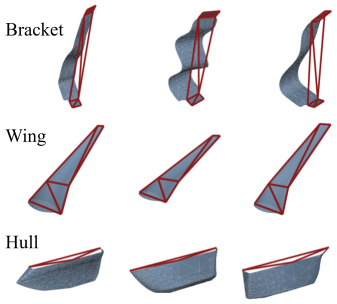


Figure 3: Seed meshes (red) for the engineering families.

Bracket. Brackets are used to mount parts together in a mechanical assembly. We focus on a simple sheet-metal bracket used to bridge *two* randomly located mounting panels. We generated 10,000 brackets with OpenCascade Paviot (2022) and assigned random bends, curves, and twists to the bridge bracket to enhance diversity. We provide a detailed description of the dataset in Appendix B.3. The design task for the bracket is straightforward: design the bracket surface given the two mounting panels. The seed meshes consist of three rectangular shapes: two panels and a bridge connecting them. The vertices and edges of the mounting panels are fixed, while the bridging edges serve as the boundary.

Wing. Wing design is a classic and yet challenging problem for generative models, as wing aerodynamics are extremely sensitive to subtle shape modification. This typically requires generation at high resolution. We focus on transonic wings for transportation aircraft and use the geometries from the *SuperWing* dataset (Yang et al., 2026), which contains 4,239 cranked wing shapes. The wing design task starts with a planform shape, i.e., the top-view projection that provides a skeleton as shown in Fig. 3, which serves as the seed mesh, and all seed edges are treated as guidelines. The most design freedom comes from the sectional airfoil shapes that support the wing surface, which is what our generation model aims for.

Hull. Hulls are a central object in hydrodynamics. We use a 10,000-hull starboard half-hulls subset of *Ship-D* (Bagazinski & Ahmed, 2023), whose designs span the parallel-midbody cross-

section, bow and stern taper, and the geometry of the bulbous bow and stern bulb. They are known to improve wave-drag and displacement performance, but they introduce high local curvature that makes the surface harder to mesh and to generate faithfully. We take the same design task and seed mesh for hull and wing, which utilize the planform shape.

Goals for mesh generation. Unlike standard mesh-generation tasks, engineering design often lacks a single ground-truth target. We therefore evaluate four aspects (more details in Appendix E): **(1) Fidelity**, that the generated mesh should meet the spatial constraints, measured by the mean displacement (disp.) of prescribed seed mesh vertices to the generated surface; **(2) Validity**, measured by face-level self-intersection ratio (self-int.) and sample-level non-clear rate that counts samples with degenerated and intersected faces; **(3) Quality**, which is critical to downstream physical solvers Knupp (2001), primarily reported with surface smoothness measured by bending energy. Additional metrics, such as minimum angle and aspect ratio, are reported in Appendix F.2; **(4) Diversity**, measured by the mean pairwise Chamfer distance among samples per spatial constraint.

6.1 SERIALIZATION

Compatibility for hard-embedding constraints. FRF hard-embeds spatial constraints by refining a prescribed seed mesh without moving its vertices. In contrast, non-progressive baselines can only enforce these constraints through soft point-cloud conditioning. Hard embedding is also nontrivial for progressive models, whose collapse process reduces the mesh to a single triangle. We therefore modify this process to keep the seed vertices fixed during collapse, but this alone does not guarantee preservation of the seed connectivity, as illustrated in Fig. 4. For VertexRegen, we therefore construct a prefix containing both the seed vertex locations and their mesh index soup, explicitly specifying the prescribed connection. ARMesh, however, loses the connectivity information needed to identify the prescribed connection. Accordingly, we use hard seed-mesh conditioning for VertexRegen and soft point-cloud conditioning for all other baselines, including ARMesh.

Table 1: Per-step prediction corpus on the bracket dataset. *site, geom., ctrl* stands for the entropy of naming where the next operation acts (incl. topology), the geometric position of the next vertex, and control signals (e.g. STOP).

model (serialization)	bits per step: site	geom.	ctrl	total	steps / face	bits / face	hard-embedding constraints
MeshGPT	0.0	56.5	0.0	56.5	1.0	**56.5**	✗
MeshAnything v2	0.0	16.2	0.6	16.8	1.7	**29.1**	✗
TreeMeshGPT	0.0	9.4	1.0	10.4	2.0	**20.9**	✗
ARMesh (PSC)	41.0	12.4	1.0	54.4	0.6	**29.8**	✗
VertexRegen (PM)	25.0	20.9	0.0	45.9	0.6	**25.1**	○
CoMesh (FRF)	0.0	4.9	1.0	5.9	1.0	**6.1**	✓

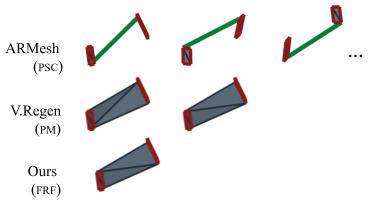


Figure 4: Possible base meshes produced by backward simplification with fixed vertices. PM produces two options of the bridging panel's diagonal. PSC stops at a single edge connecting the panels at two arbitrary vertices.

Reduced prediction complexity. We quantify the prediction complexity induced by each serialization itself. Using each method's native tokenizer, we measure the empirical context-free entropy of the fields that must be generated. As shown in Table 1, our serialization requires substantially fewer bits than all baselines, which is mostly due to the *site* field, for which progressive baselines must predict the choice of operation site from the logits. Meanwhile, FRF encodes geometry in a local frame, yielding far fewer geometric bits than methods that use global coordinates.

Beyond these benefits, FRF performs exceptionally well on our three engineering shape datasets, albeit under a simplified guarantee. It achieves a 100% success rate for transonic wings, and around 97% for bracket and hull. Their full results are reported in Appendix C.

6.2 GENERATION QUALITY AND EFFICIENCY

We compare all baselines under two protocols. VertexRegen, ARMesh, and MeshFlow are retrained with our S-size Transformer capacity (128-dim, 5 layers, and 8 heads) for 10k steps with warm-up, while MeshAnything V2 and TreeMeshGPT are fine-tuned from released weights for 2k steps. All

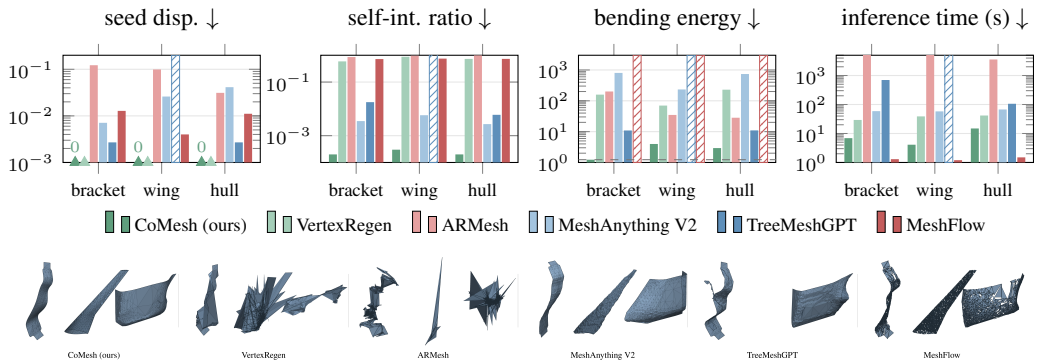


Figure 5: Headline comparison across the three families and full tables are in Appendix F.2.

baseline experiments use an effective batch size of 8 on a single A5000 GPU with a 120-hour limit and are evaluated on 32 conditions $\times$, 32 samples. The corresponding results are shown in Fig. 5.

Our method is the only one that consistently combines exact seed, low self-intersection, and smooth elements across all three families. Seed error is exactly zero by construction, whereas baselines, except for modified VertexRegen, must learn the constraint and remain less accurate. Our model also produces the best mesh quality, as self-intersection is near zero and the bending energy approaches the ground-truth mesh values shown by the dashed line. Within the baselines, the fine-tuned results are noticeably better on metrics and visualizations, justifying the pretraining effort, but remain behind CoMesh in every aspect. Model capacity is a central parameter that we kept aligned for a fair comparison. However, even when increasing the capacity of VertexRegen in Appendix F.1, it still produces noticeably worse results. MeshFlow is faster because it generates faces in parallel, but at the cost of validity and scalability, while our method is faster than all autoregressive baselines.

Serialization and model class. How the mesh is serialized is crossed with whether the model is state-conditioned or globally causal in Table 2 as an ablation (details in Appendix D.3). The serialization dominates the difference, as FRF reduces self-intersection by orders of magnitude across both models by simplifying the representation of every action a_t. FRF hard-embedded the connectivity modification led by a_t, further removing the possibility that a broken topology is introduced by the error in vertex index prediction. The state model surprisingly leads in both serializations, even though it has much less exposure to samples. The reason could be that the mesh generation cases we studied are highly Markovian, so the history of how the existing mesh is reached provides no additional information, but adds burden to a model that has to reconstruct the mesh state from a prefix.

Table 2: Ablation of serialization $\times$ model class.

	self-int ↓	bend. ↓	epoch
forward refinement fitting serialization (ours)			
state (local, **CoMesh**)	**0.0002**	**1.228**	19.1
causal	0.0062	4.705	252.9
progressive meshing serialization			
state (global)	0.2501	126.2	10.6
causal (**V.Regen**)	0.5903	158.5	1121

Mesh resolution scale. We evaluate scaling on the wing family by training models on progressively finer meshes and longer sequences, obtained by increasing the maximum face count during fitting. Fig. 6 shows the model's efficiency and performance (with marker area; smaller is better). We observed that our CoMesh model maintains stable mesh quality as the resolution increases, while generation time scales as $N_{\text{faces}}^{1.31}$ in our wing experiments, and peak model memory remains constant due to the fixed-size local mesh-patch input. In contrast, all baselines exhibit worse mesh quality or longer generation times, and fail to scale beyond 1,920 faces for method-specific reasons detailed in Appendix F.2.

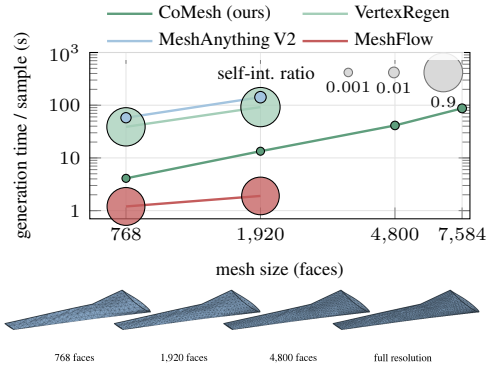


Figure 6: Scaling with mesh resolution on wings. Full report in Appendix F.2

6.3 DIVERSITY

Generation novelty. On the bracket dataset, the median Chamfer distance from each generated mesh to its nearest example in the training dataset is 0.0647, compared with 0.0426 for held-out test meshes. Since these meshes are simultaneously valid, the distance reflects genuinely new geometry rather than memorizing the data manifold.

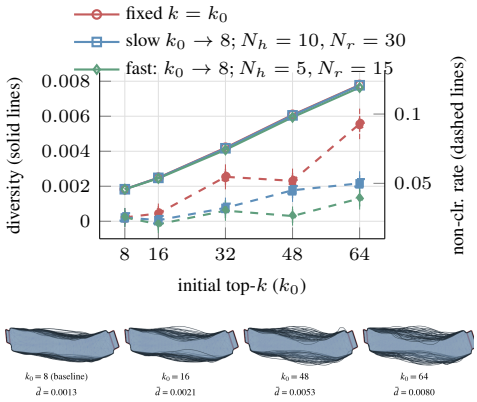


Figure 7: Effect of initial sampling width and cooling schedule. Full results in Appendix F.3.

Cooling schedule and diversity. We then ask how much variety the model produces within a single mounting-panel condition. We observe that across generations, early splits largely determine the global geometry, whereas later splits primarily refine the surface. This motivates a dynamic sampling schedule that explores broadly early and becomes increasingly conservative afterward to avoid unreasonable actions. Starting from an initial k_0, we hold top-k for N_h steps and then linearly cool it to k=8 over N_r steps. We compared two cooling schedules with fixed k, and Figure 7 shows that increasing k_0 substantially increases generational diversity. Cooling consistently reduces failure rate while being harmless to diversity. We observe the same trend in surface quality in full reports.

7 LIMITATIONS

CoMesh inherits several restrictions from its refinement formulation, i.e., exploring geometry only within a prescribed topological class. FRF further assumes that the target is reachable through a valid forward-refinement trajectory, while the theoretical guarantee is sufficient and stronger than the conditions enforced in our implementation. Extending the method to arbitrary collections of objects will therefore require seed construction and geometric guards that more closely satisfy these conditions. In addition, the spatial constraints considered here are primarily encoded in the seed mesh and do not yet cover more general engineering constraints such as packaging or clearance. Nevertheless, because CoMesh conditions on the current mesh state rather than an immutable action prefix, projections or corrective edits pose interesting directions for future work. From an application perspective, our evaluation does not yet cover volumetric meshing or end-to-end CFD/FEA simulation, although the reported geometric metrics indicate potential gains from the substantially improved mesh quality.

8 CONCLUSION

We introduced *CoMesh*, a progressive framework for constrained engineering mesh generation built around a forward serialization that matches the direction of the generative task. Instead of recovering refinement actions from a reversed simplification history, FRF starts from a prescribed seed, uses a shared deterministic scheduler to select each refinement site, and leaves the model to predict only a local two-dimensional displacement or a stop decision. This construction preserves seed-encoded constraints by design, substantially simplifies the prediction burden per step, and enables a compact state-conditioned model with bounded local context. Across mounting brackets, transonic wings, and ship hulls, CoMesh consistently produces higher-quality meshes with near-zero self-intersection and exact seed fidelity, while training and decoding more efficiently than autoregressive alternatives and scaling to substantially finer meshes. A top-k cooling schedule further enables broad sampling diversity without sacrificing validity. More broadly, our results suggest that, with an appropriate refinement strategy and progressive model, high-quality surface meshes can be generated *directly* and *efficiently* for engineering design without relying on a prescribed geometric parameterization that may constrain the design space. Coupled with functional evaluators and optimization algorithms, this offers a promising foundation for more flexible AI-assisted engineering design workflows.

AI USE STATEMENT

In this work, we used generative AI tools to implement methods, design or provide feedback on research methodology or experiments, formulate and provide critical ingredients for proving mathematical claims, and assist in writing proofs. We have not used generative AI tools to generate synthetic data sets, help develop theoretical models or conceptual frameworks, clean and reformat datasets, or support qualitative and thematic data analysis. Additionally, we used generative AI tools to create or edit software code and implement code to display experiment results. We have reviewed all AI-assisted work. LLM-generated code was verified and tested for correctness, and we reorganized the entire codebase several times to reduce duplication and improve readability. The mathematical claim and proofs are obtained from an LLM with the authors' guidance and verified for correctness. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The current paper presents a new dataset of mounting brackets, described in detail in Appendix B.3. We will release the code, raw parameters, and the mesh. We also describe the detailed experimental settings for serialization in Appendix C, and models in Appendix D to enable reproduction. We will release the full codebase that contains the implementations of our models. The formal proof of the mathematical claim in the paper is provided in Appendix A.

REFERENCES

Noah J. Bagazinski and Faez Ahmed. Ship-D: Ship Hull Dataset for Design Optimization Using Machine Learning. In *Volume 3A: 49th Design Automation Conference (DAC)*, pp. V03AT03A028, Boston, Massachusetts, USA, August 2023. American Society of Mechanical Engineers. ISBN 978-0-7918-8730-1. doi: 10.1115/DETC2023-117003. URL https://asmedigitalcollection.asme.org/IDETC-CIE/proceedings/IDETC-CIE2023/87301/V03AT03A028/1170443.

Wei Chen, Kevin Chiu, and Mark Fuge. Aerodynamic Design Optimization and Shape Exploration using Generative Adversarial Networks. In *AIAA Scitech 2019 Forum*, San Diego, California, January 2019. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-578-4. doi: 10.2514/6.2019-2351. URL https://arc.aiaa.org/doi/10.2514/6.2019-2351.

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, October 2024. URL http://arxiv.org/abs/2406.10163.

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 *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 13922–13931. IEEE, 2025.

Huipeng Guo, Zikai Song, Hang Long, Jielei Zhang, Wenbing Li, Junkai Lin, Tianhao Zhao, Jinshen Zhang, Tianle Guo, and Wei Yang. SubdivAR: Autoregressive Next-Scale Prediction for Neural Mesh Subdivision, July 2026. URL http://arxiv.org/abs/2606.27088. arXiv:2606.27088 [cs.CV].

Xianglong He, Zi-Xin Zou, Chia-Hao Chen, Yuan-Chen Guo, Ding Liang, Chun Yuan, Wanli Ouyang, Yan-Pei Cao, and Yangguang Li. Sparseflex: High-resolution and arbitrary-topology 3d shape modeling. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 14822–14833. IEEE, 2025.

Hugues Hoppe. View-dependent refinement of progressive meshes. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pp. 189–198, USA, 1997. ACM Press/Addison-Wesley Publishing Co. ISBN 978-0-89791-896-1. doi: 10.1145/258734.258843. URL https://dl.acm.org/doi/10.1145/258734.258843.

Thomas JR Hughes, John A Cottrell, and Yuri Bazilevs. Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement. *Computer methods in applied mechanics and engineering*, 194(39-41):4135–4195, 2005.

Jorge Jabón, Sergio Corbera, Roberto Álvarez, and Rafael Barea. Aerodynamic shape optimization using graph variational autoencoders and genetic algorithms. *Structural and Multidisciplinary Optimization*, 67(3):32, March 2024. ISSN 1615-147X, 1615-1488. doi: 10.1007/s00158-024-03771-5. URL https://link.springer.com/10.1007/s00158-024-03771-5.

Patrick M. Knupp. Algebraic Mesh Quality Metrics. *SIAM Journal on Scientific Computing*, 23(1): 193–218, January 2001. ISSN 1064-8275, 1095-7197. doi: 10.1137/S1064827500371499. URL https://epubs.siam.org/doi/10.1137/S1064827500371499.

Jiabao Lei, Kewei Shi, Zhihao Liang, and Kui Jia. ARMesh: Autoregressive Mesh Generation via Next-Level-of-Detail Prediction. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=xlQ4QUB9VC.

Junkai Lin, Hang Long, Huipeng Guo, Jielei Zhang, Jiayi Yang, Tianle Guo, Yang Yang, Jianwen Li, Wenxiao Zhang, Matthias Nießner, and Wei Yang. MeshRipple: Structured Autoregressive Generation of Artist-Meshes. In *Conference on Computer Vision and Pattern Recognition 2026*, 2026. URL https://openreview.net/forum?id=thic44DEkI.

Stefan Lionar, Jiabin Liang, and Gim Hee Lee. Treemeshgpt: Artistic mesh generation with autoregressive tree sequencing. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 26608–26617. IEEE, 2025.

Hsueh-Ti Derek Liu, Vladimir G. Kim, Siddhartha Chaudhuri, Noam Aigerman, and Alec Jacobson. Neural subdivision. *ACM Trans. Graph.*, 39(4), August 2020. ISSN 0730-0301. doi: 10.1145/3386569.3392418. URL https://doi.org/10.1145/3386569.3392418.

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4455–4465. IEEE, 2019.

Charlie Nash, Yaroslav Ganin, S. M. Ali Eslami, and Peter Battaglia. PolyGen: An Autoregressive Generative Model of 3D Meshes. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 7220–7229. PMLR, November 2020. URL https://proceedings.mlr.press/v119/nash20a.html.

Steven Owen, Nathan Brown, Nikos Chrisochoides, Rao Garimella, Xianfeng Gu, Franck Ledoux, Na Lei, Roshan Quadros, Navamita Ray, Nick Winovich, and others. A survey of AI methods for geometry preparation and mesh generation in engineering simulation. *Engineering with Computers*, 42(5):171, 2026.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *2019 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 165–174. IEEE, 2019.

Thomas Paviot. pythonocc, December 2022. URL https://doi.org/10.5281/zenodo.7471333.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. Learning Mesh-Based Simulation with Graph Networks. In *ICLR*, 2021. URL https://openreview.net/forum?id=roNqYL0_XP.

Jovan Popović and Hugues Hoppe. Progressive simplicial complexes. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pp. 217–224, USA, 1997. ACM Press/Addison-Wesley Publishing Co. ISBN 978-0-89791-896-1. doi: 10.1145/258734.258852. URL https://dl.acm.org/doi/10.1145/258734.258852.

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 *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 19615–19625, Seattle, WA, USA, June 2024. IEEE. ISBN 979-8-3503-5300-6. doi: 10.1109/CVPR52733.2024.01855. URL https://ieeexplore.ieee.org/document/10656131/.

Qi Sun, Kiyohiro Nakayama, Jing Nathan Yan, Qixing Huang, Alexander Rush, Leonidas Guibas, Gordon Wetzstein, Jing Liao, and Guandao Yang. MeshFlow: Mesh Generation with Equivariant Flow Matching. In *SIGGRAPH 2026*. arXiv, August 2026. doi: 10.48550/arXiv.2606.23489. URL http://arxiv.org/abs/2606.23489. arXiv:2606.23489 [cs.GR].

Jiaxiang Tang, Max Li, Zekun Hao, Xian Liu, Gang Zeng, Ming-Yu Liu, and Qinsheng Zhang. Edgerunner: Auto-regressive auto-encoder for artistic mesh generation. In *International Conference on Learning Representations*, volume 2025, pp. 35913–35934, 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12719–12729, June 2026.

Si-Tong Wei, Rui-Huan Wang, Chuan-Zhi Zhou, Baoquan Chen, and Peng-Shuai Wang. Octgpt: Octree-based multiscale autoregressive models for 3d shape generation. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers*, pp. 1–11, 2025a.

Zhen Wei, Aobo Yang, Jichao Li, Michaël Bauerheim, Rhea P. Liem, and Pascal Fua. Automated Parameterization for Aerodynamic Shape Optimization via Deep Geometric Learning. *AIAA Journal*, 63(12):5430–5447, December 2025b. ISSN 0001-1452, 1533-385X. doi: 10.2514/1.J065203. URL https://arc.aiaa.org/doi/10.2514/1.J065203.

Rundi Wu, Chang Xiao, and Changxi Zheng. Deepcad: A deep generative network for computer-aided design models. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6752–6762. IEEE, 2021.

Hairun Xie, Jing Wang, and Miao Zhang. Parametric generative schemes with geometric constraints for encoding and synthesizing airfoils. *Engineering Applications of Artificial Intelligence*, 128: 107505, February 2024. ISSN 09521976. doi: 10.1016/j.engappai.2023.107505. URL https://linkinghub.elsevier.com/retrieve/pii/S0952197623016895.

Jiale Xu, Wang Zhao, and Ying Shan. Meshweaver: Sparse-voxel-guided surface weaving for autoregressive mesh generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5912–5922, June 2026a.

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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 20003–20013, June 2026b.

Xiang Xu, Karl D. D. Willis, Joseph G. Lambourne, Chin-Yi Cheng, Pradeep Kumar Jayaraman, and Yasutaka Furukawa. SkexGen: Autoregressive Generation of CAD Construction Sequences with Disentangled Codebooks. In *ICML*, pp. 24698–24724, 2022. URL https://proceedings.mlr.press/v162/xu22k.html.

Yunjia Yang, Weishao Tang, Mengxin Liu, Nils Thuerey, Yufei Zhang, and Haixin Chen. SuperWing: a comprehensive transonic wing dataset for data-driven aerodynamic design. *Scientific Data*, 13(1):1113, July 2026. ISSN 2052-4463. doi: 10.1038/s41597-026-07769-0. URL https://www.nature.com/articles/s41597-026-07769-0.

Xiang Zhang, Yawar Siddiqui, Armen Avetisyan, Chris Xie, Jakob Engel, and Henry Howard-Jenkins. Vertexregen: Mesh generation with continuous level of detail. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 12570–12580. IEEE, 2025a.

Zhanwei Zhang, Shizhao Sun, Wenxiao Wang, Deng Cai, and Jiang Bian. FlexCAD: Unified and Versatile Controllable CAD Generation with Fine-tuned Large Language Models. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=Z0eiiV3Yyh.

Hang Zhou, Haixu Wu, Haonan Shangguan, Yuezhou Ma, Huikun Weng, Jianmin Wang, and Mingsheng Long. Transolver-3: Scaling Up Transformer Solvers to Industrial-Scale Geometries, February 2026. URL http://arxiv.org/abs/2602.04940. arXiv:2602.04940 [cs].

Feng Zhu and Ning Qin. Intuitive Class/Shape Function Parameterization for Airfoils. *AIAA Journal*, 52(1):17–25, January 2014. ISSN 0001-1452, 1533-385X. doi: 10.2514/1.J052610. URL https://arc.aiaa.org/doi/10.2514/1.J052610.

Zhiwei Zhu, Xiang Gao, Lu Yu, and Yiyi Liao. Neural Mesh Refinement. *Frontiers of Information Technology & Electronic Engineering*, 26(5):695–712, May 2025. ISSN 2095-9230. doi: 10.1631/FITEE.2400344. URL https://ieeexplore.ieee.org/document/11293036/.

APPENDIX

TABLE OF CONTENTS

A Target-Curvature Compatibility and Embedding Preservation **14**
A.1 Definitions . . . 14
A.2 From signed dihedral to face co-orientation . . . 15
A.3 Nearest-point projection of a co-oriented triangle . . . 16
A.4 Projection preservation under an interior edge split . . . 16
A.5 One-step preservation from the signed-dihedral condition . . . 16
A.6 Induction over progressive refinement . . . 17
A.7 Relation to the practical signed-dihedral guard . . . 17

B Design families and shape datasets **17**
B.1 Overview . . . 17
B.2 Dataset split . . . 18
B.3 Bracket dataset . . . 18
B.3.1 Construction . . . 18
B.3.2 Sampling and export . . . 20

C Serialization Configuration and Analysis **21**
C.1 Detailed configurations . . . 21
C.2 Success rate. . . . 23
C.3 Mapping-based target selection becomes fragile at large refinement ratios . . . 23

D Model, Training and Generation Configurations **23**
D.1 Our model . . . 23
D.2 Baselines . . . 24
D.3 Ablations . . . 25
D.4 Generation protocol . . . 26

E Metric Definitions **26**
E.1 Seed fidelity . . . 26
E.2 Mesh validity . . . 26
E.3 Mesh quality . . . 26
E.4 Mesh Diversity . . . 27

F Detailed results **27**
F.1 Model scale . . . 27
F.2 Detailed results of generation fidelity, validity, and quality. . . . 27
F.3 Sampling Schedule . . . 28

A TARGET-CURVATURE COMPATIBILITY AND EMBEDDING PRESERVATION

This appendix formalizes a conservative sufficient regime in Section 4.2, under which the signed-dihedral condition used by FRF preserves an embedding throughout progressive refinement. The argument has three steps. First, under a local acute-angle assumption, a target-consistent signed dihedral implies that each new face is co-oriented with the target surface. Second, this co-orientation makes the nearest-point projection locally orientation preserving and, for an interior edge split, preserves a homeomorphic projection of the refined edge star. Finally, the one-step result is applied inductively over a sequence of interior splits.

The assumptions below are deliberately stronger than those enforced by the practical refinement procedure. The practical interpretation of this simplification is discussed after the proof.

A.1 DEFINITIONS

Let $S \subset \mathbb{R}^3$ be a compact, embedded, orientable C^2 surface with chosen unit normal field $\mathbf{N}$. Let $U_r(S)$ be a tubular neighborhood with $r < \mathrm{reach}(S)$. Then every $x \in U_r(S)$ has a unique nearest point $P_S(x) \in S$, and the nearest-point projection $P_S : U_r(S) \to S$ is smooth.

Signed mesh dihedral. Let q be an oriented interior edge of an oriented triangular mesh. Let $\hat{\mathbf{t}}_q$ be its unit tangent, and let $\mathbf{n}_q^-$ and $\mathbf{n}_q^+$ be the oriented unit normals of its two incident faces, ordered consistently with $\hat{\mathbf{t}}_q$. Define

$$\phi_q = \operatorname{atan2}\left(\hat{\mathbf{t}}_q \cdot \left(\mathbf{n}_q^- \times \mathbf{n}_q^+\right), \mathbf{n}_q^- \cdot \mathbf{n}_q^+\right) \in (-\pi, \pi). \tag{5}$$

Reversing the orientation of q also swaps its two incident faces and leaves ϕ_q unchanged.

Target bending sign. Assume that the midpoint m_q of q lies in $U_r(S)$, and let $\bar{m}_q = P_S(m_q)$. Let $\boldsymbol{\tau}_q$ be the normalized projection of $\hat{\mathbf{t}}_q$ onto $T_{\bar{m}_q}S$, $\boldsymbol{\tau}_q = \operatorname{Norm}\left[\left(I - \mathbf{N}(\bar{m}_q)\mathbf{N}(\bar{m}_q)^\top\right)\hat{\mathbf{t}}_q\right]$, whenever τ_q is nonzero. The transverse tangent is then defined by $\boldsymbol{\eta}_q = \mathbf{N}(\bar{m}_q) \times \boldsymbol{\tau}_q$. Let $\kappa_q^S = \mathrm{II}_{\bar{m}_q}(\boldsymbol{\eta}_q, \boldsymbol{\eta}_q)$ be the signed normal curvature of S in this transverse direction, define $\sigma_q^S = \operatorname{sign}(\kappa_q^S) \in \{-1, +1\}$ when $\kappa_q^S \neq 0$.

Local target-normal sector. Consider two consistently oriented faces f^- and f^+ sharing an interior edge q, with normals $\mathbf{n}_q^-$ and $\mathbf{n}_q^+$ as in equation 5. For $x \in f^+$, project the target normal at $P_S(x)$ onto the plane orthogonal to q: $\widetilde{\mathbf{N}}_q(x) = \operatorname{Norm}\left[\left(I - \hat{\mathbf{t}}_q\hat{\mathbf{t}}_q^\top\right)\mathbf{N}(P_S(x))\right]$ whenever the denominator is nonzero. Define the signed target-normal rotation from $\mathbf{n}_q^-$ by

$$\theta_q^S(x) = \operatorname{atan2}\left(\hat{\mathbf{t}}_q \cdot \left(\mathbf{n}_q^- \times \widetilde{\mathbf{N}}_q(x)\right), \mathbf{n}_q^- \cdot \widetilde{\mathbf{N}}_q(x)\right) \in (-\pi, \pi). \tag{6}$$

We say that (q, f^+) satisfies the *acute target-sector condition* if σ_q^S and $\theta_q^S(x)$ are defined and

$$0 \leq \sigma_q^S \theta_q^S(x) < \frac{\pi}{2} \qquad \text{for every } x \in f^+. \tag{7}$$

A.2 From signed dihedral to face co-orientation

We first connect the signed-dihedral condition used by FRF to the face co-orientation condition required by the projection argument.

Lemma 2 (Signed dihedral implies co-orientation in the acute regime)**.** *Let f^- and f^+ be two consistently oriented nondegenerate triangles sharing an interior edge q. Assume that $f^+ \subset U_r(S)$, that (q, f^+) satisfies the acute target-sector condition equation 7, and that*

$$0 \leq \sigma_q^S \phi_q < \frac{\pi}{2}. \tag{8}$$

Then $\mathbf{n}_q^+ \cdot \mathbf{N}(P_S(x)) > 0$ for every $x \in f^+$.

Proof. The unit vectors $\mathbf{n}_q^-$, $\mathbf{n}_q^+$, and $\widetilde{\mathbf{N}}_q(x)$ all lie in the plane orthogonal to $\hat{\mathbf{t}}_q$. By construction, ϕ_q is the signed angle from $\mathbf{n}_q^-$ to $\mathbf{n}_q^+$ in this plane, whereas $\theta_q^S(x)$ is the signed angle from $\mathbf{n}_q^-$ to $\widetilde{\mathbf{N}}_q(x)$.

From equation 7 and equation 8, both angles lie in the same open half-sector and

$$\left|\phi_q - \theta_q^S(x)\right| < \frac{\pi}{2}.$$

Therefore

$$\mathbf{n}_q^+ \cdot \widetilde{\mathbf{N}}_q(x) = \cos\left(\phi_q - \theta_q^S(x)\right) > 0.$$

Because $\mathbf{n}_q^+ \perp \hat{\mathbf{t}}_q$, only the component of $\mathbf{N}(P_S(x))$ orthogonal to $\hat{\mathbf{t}}_q$ contributes to the dot product. Thus

$$\begin{aligned}\mathbf{n}_q^+ \cdot \mathbf{N}(P_S(x)) &= \mathbf{n}_q^+ \cdot \left(I - \hat{\mathbf{t}}_q\hat{\mathbf{t}}_q^\top\right)\mathbf{N}(P_S(x)) \\ &= \left\|\left(I - \hat{\mathbf{t}}_q\hat{\mathbf{t}}_q^\top\right)\mathbf{N}(P_S(x))\right\| \mathbf{n}_q^+ \cdot \widetilde{\mathbf{N}}_q(x) > 0.\end{aligned}$$

Hence f^+ is co-oriented with the target. □

The sign in equation 8 is essential since the magnitude bound alone prevents an arbitrarily large hinge rotation but does not distinguish whether the new face rotates toward or away from the target-oriented sector.

A.3 NEAREST-POINT PROJECTION OF A CO-ORIENTED TRIANGLE

Lemma 3 (Positive projection Jacobian)**.** *Let f be a nondegenerate oriented planar triangle contained in $U_r(S)$. Assume $\mathbf{n}_f \cdot \mathbf{N}(P_S(x)) > 0$ for every $x \in f$, where $\mathbf{n}_f$ is the oriented unit normal of f. Then P_S is an orientation-preserving local diffeomorphism on the relative interior* $\mathrm{relint}(f)$.

Proof. Write

$$x = p + s\,\mathbf{N}(p), \qquad p = P_S(x),$$

with $|s| < r$, and let $\mathcal{S}_p$ denote the shape operator of S at p. Differentiating the normal-coordinate map gives

$$\dot{P}_S(x) = \big(I - s\mathcal{S}_p\big)^{-1} P_{T_pS}$$

on tangent vectors. Since no focal point is encountered in $U_r(S)$. Thus, the eigenvalues $1 - s\kappa_i(p)$ do not vanish; by continuity from $s = 0$, they remain positive throughout the tubular neighborhood. Consequently, $\big(I - s\mathcal{S}_p\big)^{-1}$ is orientation preserving on T_pS.

Let (u, v) be a positively oriented basis of the plane of f. The sign of the oriented area Jacobian of P_{T_pS} restricted to that plane is the sign of $\mathbf{N}(p) \cdot (u \times v)$, which is positive exactly when $\mathbf{n}_f \cdot \mathbf{N}(p) > 0$. Hence, the restriction of $\dot{P}_S(x)$ to the face plane is nonsingular with positive determinant at every interior point of f. The inverse function theorem gives an orientation-preserving local diffeomorphism on $\mathrm{relint}(f)$. □

A.4 PROJECTION PRESERVATION UNDER AN INTERIOR EDGE SPLIT

Let M_t be target-curvature compatible with S, and let e be an interior edge selected for splitting. Let A_t be the closed two-triangle star of e. Because $P_S|_{M_t}$ is a homeomorphism, $B_t = P_S(A_t) \subset S$ is a closed topological disk, and $P_S|_{\partial A_t} : \partial A_t \to \partial B_t$ is an orientation-preserving homeomorphism.

An interior FRF edge split modifies only the triangulation inside A_t and leaves its outer boundary unchanged. Let A_{t+1} denote the refined star.

Lemma 4 (Homeomorphic projection of the refined star)**.** *Assume that (1)$A_{t+1} \subset U_r(S)$; and (2) every new face $f \subset A_{t+1}$ satisfies $\mathbf{n}_f \cdot \mathbf{N}(P_S(x)) > 0$ for every $x \in f$. Then $P_S|_{A_{t+1}} : A_{t+1} \to B_t$ is an orientation-preserving homeomorphism.*

Proof. Set $F = P_S|_{A_{t+1}}$. Because the split edge is interior, the outer boundary of its two-triangle star is unchanged. Hence $F|_{\partial A_{t+1}} = P_S|_{\partial A_t}$, which is an orientation-preserving homeomorphism onto ∂B_t.

By Lemma 3, F has a positive Jacobian on the relative interior of every new face. Consider a point $y \in \mathrm{int}(B_t)$ away from the projected one-skeleton. Every preimage of y lies in a face interior and contributes local degree $+1$. Since the boundary map has degree one around B_t, $\deg(F, A_{t+1}, y) = 1$. Therefore, y has exactly one preimage. Likewise, a point outside B_t has boundary degree zero and hence has no preimage.

The positive face Jacobians and the consistent orientation of the triangulation imply that F is locally open across interior edges and vertices of the refined star. Consequently, if two distinct points of A_{t+1} had the same image, every sufficiently small neighborhood of that image would contain a point away from the projected one-skeleton with at least two preimages, contradicting the degree-one result above. An interior point of A_{t+1} also cannot map to ∂B_t, because local openness would then produce images outside B_t.

Thus F is a continuous bijection from the compact disk A_{t+1} to the Hausdorff disk B_t. It is therefore a homeomorphism. Its positive local degree makes it orientation-preserving. □

A.5 ONE-STEP PRESERVATION FROM THE SIGNED-DIHEDRAL CONDITION

For every new face f created by an interior split, choose an affected interior edge $q_f \subset \partial f$ for which $\sigma^S_{q_f}$ is defined, and orient q_f so that f is the + incident face in equation 5. We call q_f a *certifying edge* for f.

Proposition 5 (Signed-dihedral preservation under an interior split)**.** *Let M_t be target-curvature compatible with S, and obtain M_{t+1} by one interior FRF edge split. Assume that (1) the refined edge star satisfies $A_{t+1} \subset U_r(S)$, (2) every new face f admits a certifying edge q_f for which the acute target-sector condition holds,*

$$0 \le \sigma^S_{q_f} \theta^S_{q_f}(x) < \frac{\pi}{2} \qquad \textit{for every } x \in f;$$

and (3) every affected interior edge q for which σ^S_q is defined satisfies

$$0 \le \sigma^S_q \phi_q < \frac{\pi}{2}.$$

Then M_{t+1} is target-curvature compatible with S.

Proof. For each new face f, Conditions 2–3 and Lemma 2, applied to its certifying edge q_f, give $\mathbf{n}_f \cdot \mathbf{N}(P_S(x)) > 0$ for every $x \in f$. Thus, all newly created faces are co-oriented with the target.

Condition 1 and Lemma 4 then imply that $P_S|_{A_{t+1}} : A_{t+1} \to B_t$ is an orientation-preserving homeomorphism onto exactly the target patch covered by the old star. Outside A_{t+1} the mesh is unchanged, and the old and new projection maps agree on the unchanged boundary. Therefore, they glue to an orientation-preserving homeomorphism

$$P_S|_{M_{t+1}} : M_{t+1} \to S.$$

It remains to verify the signed-bending condition in Definition 1. Every unaffected interior edge is unchanged and therefore retains its previous relation $\sigma^S_q \phi_q \ge 0$. Every affected interior edge for which σ^S_q is defined satisfies the same inequality by Condition 3. Hence, both requirements of target-curvature compatibility hold for M_{t+1}. □

A.6 Induction over progressive refinement

Corollary 6 (History-safe progressive refinement)**.** *Suppose M_0 is target-curvature compatible with S. Consider any sequence of interior FRF edge splits. If every split satisfies the hypotheses of Proposition 5, then every intermediate mesh M_t is target-curvature compatible with S. In particular, every M_t is embedded and self-intersection free.*

Proof. The result follows by induction on t. The base case holds by assumption for M_0. Suppose M_t is target-curvature compatible. If the next interior split satisfies the hypotheses of Proposition 5, then the proposition gives that M_{t+1} is again target-curvature compatible. Hence, the property holds for every t, and every intermediate mesh is embedded and self-intersection free. □

A.7 Relation to the practical signed-dihedral guard

For a split of edge e, the fitting score $\Gamma_e(w) \ge 0$ is exactly the sign component of Condition 3. The proposition shows that, in the additional conservative regime, this sign condition implies the face co-orientation needed for an orientation-preserving nearest-point projection. The projection argument then provides the embedding guarantee, and the one-step result closes inductively over arbitrarily many interior refinement steps. In particular, FRF can encounter affected dihedrals with $|\phi_q| \ge \pi/2$. The result should therefore be interpreted as a conservative sufficient regime that explains how the signed-dihedral condition connects to embedding preservation, rather than as a claim that every accepted fitting step satisfies the theorem hypotheses.

B Design families and shape datasets

B.1 Overview

We provide an overview of three families in Fig. 8.



Figure 8: Samples of the three engineering families.

B.2 Dataset split

For each family's dataset, we split it by condition. We use 90% of the conditions for training. From the remaining 10%, we reserve 128 conditions for generation evaluation (in the final report, we used the fixed first 32 of them to save time) and use the remaining conditions as the validation set to monitor training and to trigger early stopping if needed.

B.3 Bracket dataset

The bracket corpus is the only family we generate ourselves, so we describe its construction in full. We will release raw shape parameters and the code to construct bracket shapes from scratch. The main entry is under `generate_chain_dataset.py`, where it hands the root shape parameters to the procedural sheet-metal kernel in `sheet_metal/`, a thin modeling layer over Open CASCADE (through `pythonOCC`) (Paviot, 2022).

The dataset models a sheet-metal part as its *mid-surface* only, and a ground-truth sample is therefore a zero-thickness surface.

B.3.1 Construction

Part topology This dataset focuses on a simple type of bracket that connects two mounting panels together. It has a topology of a *chain* of five panels joined by four cylindrical bend bands (Fig. 9). Among the five panels, M1 and M2 are two rectangular flat mounting panels (serve as the seed mesh in generation tasks) that each connect to a flat rectangle connector (P1 and P2). A smooth lofted ribbon bridges (MID) the two bends, and is where the family gets its shape diversity: the loft is skinned through three movable cross-sections that may be advanced, offset, rolled, and yawed. In construction, we fix P1 and build the chain forward, placing each child at its parent's bend, with no constraint-solving or fitting involved.

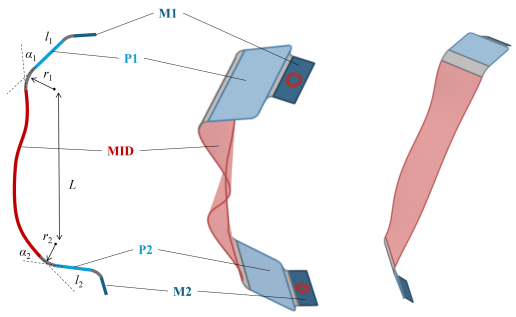


Figure 9: Bracket topology, drawn from an actual sample of the corpus.

Panel frames. A flat panel is an origin o with two orthonormal in-plane axes $\hat{T}$ (length direction) and $\hat{W}$ (width direction); its normal is $\hat{N} = \hat{W} \times \hat{T}$. Local coordinates map as $x(u, v) = o + u\hat{W} + v\hat{T}$ with $u \in [0, w]$, $v \in [0, \ell]$. The root panel is placed at $o = 0$ with $\hat{T} = \hat{e}_x$ and $\hat{W} = \hat{e}_y$, hence $\hat{N} = -\hat{e}_z$.

Bend bands. The panels are joined with bend bands. We require that the two adjacent edges on both sides of the bend are parallel in the width direction W. The bands are then cylindrical patches of opening angle α about an axis parallel to the edge that have

$$\hat{T}_{\text{out}} = \sin\alpha\, \hat{n}_b + \cos\alpha\, \hat{T}_{\text{in}}, \qquad \hat{N}_{\text{out}} = \cos\alpha\, \hat{n}_b - \sin\alpha\, \hat{T}_{\text{in}}. \tag{9}$$

and its analytical function then is:

$$X(s, a) = \Big(1 - \tfrac{s}{L}\Big) p_0 + \tfrac{s}{L}\, p_1 + \rho\big[(1 - \cos a)\, \hat{n}_b + (\sin a)\, \hat{T}_{\text{in}}\big], \qquad a \in [0, \alpha], \tag{10}$$

For mounting panels, we add a smooth taper to bridge the different widths. The band widens at its base and tapers back to the child width at $a = \alpha$: the s-range in Eq. (10) is

$$s \in \big[-b_0\, e(a/\alpha),\ L + b_1\, e(a/\alpha)\big], \qquad e(\tau) = 1 - \sqrt{\tau(2 - \tau)}, \tag{11}$$

a normalized quarter-circle taper with $e(0) = 1$ and $e(1) = 0$.

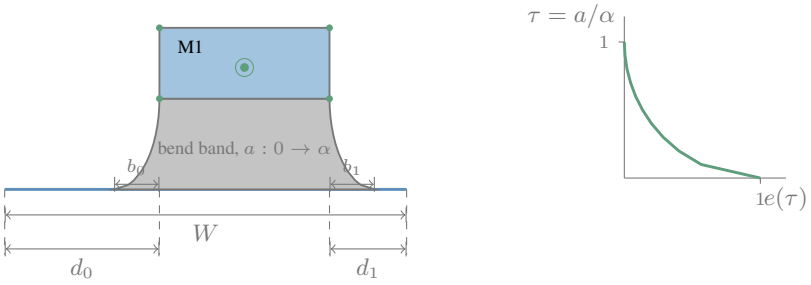


Figure 10: Mounting tab and partial bend.

Corner radius. Free convex corners of a flat panel are filleted with radius r_c, which trims $r_c \tan(\varphi/2)$ off each adjacent edge at a corner of exterior deflection φ. If the mounting panel's base would run into the corner trim c_j at a distance d_j from that corner ($d_0 = f_0 L_e$ and $d_1 = (1 - f_1) L_e$ along the parent edge of length L_e), both are cut back.

The lofted middle panel The bridge carries all of the family's interesting geometry. Let $(o; \hat{T}, \hat{N}, \hat{W})$ be the exit frame of the bending connected to P_1, with $o = \frac{1}{2}(p_0^{\text{out}} + p_1^{\text{out}})$ the midpoint of the band's exit edge, $\hat{T} = \hat{T}_{\text{out}}$, $\hat{N} = \hat{N}_{\text{out}}$, and $\hat{W}$ the edge direction; the root profile C_0 is the straight segment $[p_0^{\text{out}}, p_1^{\text{out}}]$, of length W.

Three further profiles are placed in this frame. Their sampled parameters are *increments*, which the builder accumulates along the chain. Profile k is then the straight segment of length W given by

$$C_k = \big[\, c_k + (\omega_k - \tfrac{W}{2})\hat{W}_k,\ \ c_k + (\omega_k + \tfrac{W}{2})\hat{W}_k \,\big]. \tag{12}$$

where $\hat{W}_k = R_{\hat{N}}(\Psi_k)\, R_{\hat{T}}(\Phi_k)\, \hat{W}$ and $c_k = o + \tau_k \hat{T} + \nu_k \hat{N}$. $(\tau_k, \nu_k, \omega_k)$ are the shifts of C_k in the exit frame, and (Φ_k, Ψ_k) are the rotation angles, where both rotation axes are the *fixed* root axes. Every profile inherits the root width W and is centered at c_k.

The surface is finally built with a smooth skin (`BRepOffsetAPI_ThruSections` with `ruled=False`) through the six sections, including the four profiles plus two auxiliary copies that steer the end tangents,

$$\big[\, C_0,\ \ C_0 + \delta\, \hat{T}_{\text{out}},\ \ C_1,\ \ C_2,\ \ C_3 - \delta\, \hat{t}_{\text{end}},\ \ C_3 \,\big], \qquad \delta = \max\big(1,\ 0.025\, \|m_3 - m_0\|\big). \tag{13}$$

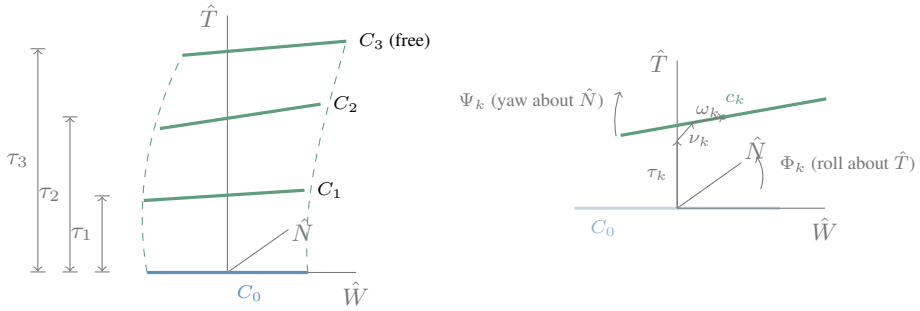


Figure 11: Parameterization of the lofted bridge.

B.3.2 Sampling and export

A corpus of 10k samples is carried out by i.i.d. uniform sampling of the parameters in Table 3. The mounting panel parameters are fixed and shown in Table 4. Each of the nine B-Rep shapes in a sample (five panels, four bend bands) is tessellated independently using `BRepMesh_IncrementalMesh` with an absolute linear deflection of 0.1 mm and an angular deflection of 0.2 rad. Each bend band is a ruled loft through 33 straight sections.

Table 3: Sampled design parameters.

symbol	code name	range	role
ℓ_1	`p1_l`	$[20, 100]$ mm	length of the root panel P1 along $\hat{T}$
W	`p1_w`	$[80, 140]$ mm	global width: P1, every loft profile, P2
r_1	`bend_r1`	$[20, 60]$ mm	nominal radius of bend B_1
r_2	`bend_r2`	$[20, 60]$ mm	nominal radius of bend B_2
θ_1	`mid_ang`	$[-80, 80]^\circ$	fold of B_1; only $\lvert\theta_1\rvert$ is used
θ_2	`p3_ang`	$[-80, 80]^\circ$	fold of B_2; magnitude *and* sign are used
ℓ_2	`p3_l`	$[20, 100]$ mm	length of the end panel P2
β_1	`sp1_ang`	$[-40, 40]^\circ$	fold of the P1 tab; only $\lvert\beta_1\rvert$ is used
β_3	`sp3_ang`	$[-40, 40]^\circ$	fold of the P2 tab; only $\lvert\beta_3\rvert$ is used
per loft profile $k = 1, 2, 3$ *increments*			
t_k	`shift_t`	$[60, 100]$ mm	advance along $\hat{T}$
n_k	`shift_n`	$[-50, 50]$ mm	offset along $\hat{N}$
w_k	`shift_w`	$[-20, 20]$ mm	slide along the profile direction
φ_k	`twist`	$[-20, 20]^\circ$	roll about $\hat{T}$
ψ_k	`twist2`	$[-20, 20]^\circ$	yaw about $\hat{N}$

Table 4: Constants held fixed across the corpus.

group	constant	value
mounting panels	spans `sp1_f0, f1` / `sp3_f0, f1`	$(0.30, 0.85)$ / $(0.15, 0.85)$
	length `sp1_len` / `sp3_len`	28 mm
	bend radius (default of `extend`)	5 mm
	blend radius `sp{1,3}_blend`	8 mm

C SERIALIZATION CONFIGURATION AND ANALYSIS

C.1 DETAILED CONFIGURATIONS

Algorithm 1 is the procedure that turns a ground-truth surface into the token stream our models are trained on. It is a single loop over an edge queue, and each iteration either retires an edge (STOP) or splits it at a point found on the ground truth. The only quantities that reach the model are the per-edge (d_n, d_b) on line 34 and the binary STOP flag. The edge is reproduced by the same queue schedule, and every guard is a function of the current mesh alone, so none of them costs a token. A draw that cannot produce a valid split is discarded in its entirety and redrawn with a fresh seed. The outer driver keeps `n_keep` traces per shape out of at most `max_retry` draws, which is where the stochasticity of `proj_random_mix` pays off.

Algorithm 1 Forward projection fit (one draw). S^{GT}: target surface; ∂S^{GT}: boundary chains; M: current mesh.

Require: target S^{GT}, guides Γ, seed mesh $M \leftarrow M_0$, frozen edges E_{fix}, random stream

1: $Q \leftarrow \textsc{EdgeQueue}\big(\text{edges}(M) \setminus E_{\text{fix}}\big)$ ▷ `longest`: pop $\arg\max L(e)$
2: $T \leftarrow ()$ ▷ Refinement action sequence
3: **while** $Q \neq \emptyset$ **and** $|T| <$ `max_expand_steps` **do**
4: $e = (v_l, v_r) \leftarrow Q.\textsc{pop}()$;
5: $F \leftarrow \mathrm{Adj}(e)$ ▷ faces that adjacent to the edge e
6: $L \leftarrow \|v_r - v_l\|$, $\hat{e} \leftarrow (v_r - v_l)/L$, $\hat{n} \leftarrow \mathrm{norm}\left(\sum_F n_f\right)$, $\hat{b} \leftarrow \hat{e} \times \hat{n}$ ▷ edge frame
7: **if** $\min_F \angle_{\text{opp}}(f, e) <$ `min_opposite_angle` **or** $L <$ `len_stop_threshold` **then**
8: **continue** ▷ retired without an event
9: **end if**
– stop probe –
10: $\delta \leftarrow \max_{r \in (0,1)} \mathrm{dist}\big(v_l + r(v_r - v_l),\ S^{\mathrm{GT}}\big)$ ▷ `n_offset_probe_points` samples
11: **if** $\delta \leq$ `offset_stop_threshold` **and** $L \leq$ `len_continue_threshold` **then**
12: $T.\textsc{append}\big(\textsc{stop}(e)\big)$; **continue**
13: **end if**
– target search –
14: $m_e \leftarrow v_l + 0.5\,(v_r - v_l)$; $\Pi \leftarrow \{x : (x - m_e) \cdot \hat{e} = 0\}$
15: **if** e is boundary edge **then** ▷ guided: stay on the ground-truth boundary curve
16: $w \leftarrow \Pi_e \cap \partial S^{\mathrm{GT}}$ restricted to the arc between the images of v_l, v_r
17: **else** ▷ free: project onto the plane–surface curve
18: $\gamma_e^* \leftarrow$ component of $\Pi_e \cap S^{\mathrm{GT}}$ with clearance of ∂S^{GT} ▷ boundary clearance
19: $w_{\text{ray}} \leftarrow \textsc{RayHit}(p_0, \hat{n}, S^{\mathrm{GT}})$, $w_{\text{near}} \leftarrow \arg\min_{x \in C} \|x - p_0\|$
20: $w^0 \leftarrow \textsc{ArcLerp}_{\gamma_e^*}\big(w^0_{\text{ray}}, w^0_{\text{near}}, t\big)$, $t \sim \mathcal{U}(0,$ `proj_random_mix`$)$ ▷ initial guess
21: $\Gamma_e(w^0)$, self-int., degen. $\leftarrow \textsc{Evaluate}(M, e, w^0)$ ▷ oriented dihedral
22: **if** self-int., degen. is True **then** stop and abort.
23: **end if**
24: **if** $\Gamma_e \leq$ `min_valley_dot` **then**
25: $w^{\text{res}} \leftarrow \arg\max_{w \in \gamma_e^*} \Gamma_e(w)$ over `arc_search_samples` points within
26: $\pm$`arc_search_window` $\cdot\, L$
27: **if** w^{res} improves **then** $w \leftarrow w^{\text{res}}$
28: **end if**
29: **else**
30: $w \leftarrow w^0$
31: **end if**
32: **end if**
33: $d_n \leftarrow (w - m_e) \cdot \hat{n}/L$; $d_b \leftarrow (w - m_e) \cdot \hat{b}/L$
34: $T.\textsc{append}\big(\textsc{split}(e, r, d_n, d_b)\big)$
35: $M \leftarrow \big(V + w, E - e +$ new edges induced from $w\big)$
36: $Q.\textsc{push}\big(\text{new edges}\big)$
37: **end while**
38: **return** T

Table 5 lists the configuration used for every result in this paper. The three families use a shared code, and their values can differ slightly.

Table 5: Shipped serialization configuration across the three families.

	parameter	Description	bracket	wing	hull
seed	`normalize_extent`	Axial scaling (x, y, z) before fitting to avoid shape bending.	$[1, 1, 1]$	$[1, 3, 1]$	$[1, 1, 0.5]$
budget	`max_expand_steps`	Cap on refinement events per draw.	1024	8192	1024
	`len_stop_threshold`	Absolute edge length below which an edge is dropped from the queue with no event.		0.005	
	`len_continue_threshold`	Absolute edge length above which an edge is refused to emit STOP	∞	0.08	∞
	`offset_stop_threshold`	An edge STOPs when its maximum distance to GT surface, sampled at the probe stations along the edge, falls below the threshold.		5×10^{-4}	
	`n_offset_probe_points`			11	
clearance	`boundary_clearance_frac`	Keep-out radii around registered boundary/guideline polylines, as fractions of the current edge length L.	0.08	0.02	0.0
	`clearance_local_coef`	Caps each clearance at this multiple of the shortest edge adjacent to the split, so a long edge beside small triangles cannot push its target outside the local feature size.		0.5	
guards	`min_opposite_angle`	Refuse a split whose opposite angles would fall below this floor (in degrees).	20	5	5
	`min_valley_dot`	Fold gate on the worst new oriented dihedral (Γ_e).		0.0	
proj.	`proj_random_mix`	Draw the initial guess uniformly between the projection hit (w^0_{ray}) and this portion of the curve until the nearest point on the feasible arc.	1.0	1.0	0.8
	`arc_search_samples`	On guard failure, search this many samples within $\pm$ window$\times L$ of the guess along the feasible arc.		11	
	`arc_search_window`			0.05	
draws	`n_keep` / `max_retry`	Distinct traces retained per sample, and the draw budget used to find them.	4 / 64	2 / 32	4 / 64

C.2 SUCCESS RATE.

With the in-serialization guide, every accepted refinement trace must produce a final mesh with zero self-intersection ratio and no degenerated triangles. We therefore report the success rate of our forward serialization in Table 6. A sample succeeds if at least one draw completes without aborting.

Table 6: Serialization success rate.

	bracket	wing	hull
samples in corpus	10,000	4,239	10,000
samples with ≥ 1 trace	9,846 (**98.5%**)	4,239 (**100%**)	9,666 (**96.7%**)
traces produced	39,105	8,478	38,492

C.3 MAPPING-BASED TARGET SELECTION BECOMES FRAGILE AT LARGE REFINEMENT RATIOS

A possible alternative to our dihedral-guided target selection for forward-fitting serialization is to use a bijective mapping between the coarse and fine meshes. This is also known as the successive self-parameterization method and has been used in neural mesh refinement methods (Liu et al., 2020; Guo et al., 2026). It starts with the full mesh and builds the mapping during a common simplification (e.g., progressive meshing) by conformally flattening the joint one-ring at each collapse and composing the per-collapse maps. In its forward phase, new vertex positions are then read off through that map rather than computed geometrically.

The composition is exact only while every collapse keeps it bijective. We reimplemented the algorithm and examined the composed map as a function of base size. We measured the failure rate for each base size as the fraction of vertices whose round-trip through the map exceeds 1% of the bounding-box diagonal, and the fraction of rebuilt triangles that collapse to below 10^{-4} of their original area. We show the results in Fig. 12.

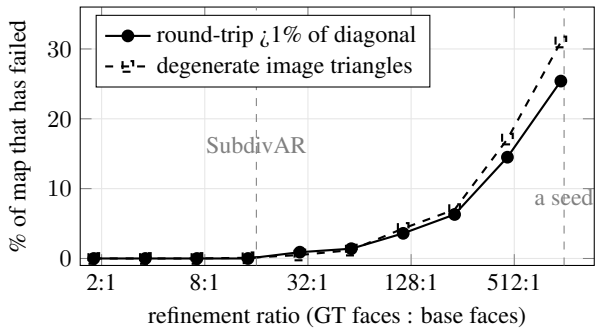


Figure 12: Degradation of the self-parameterization mapping with refinement ratio on a wing sample.

The results clearly show that around the mesh refinement task ratio (e.g., SubdivAR simplifies $\sim$12k faces to 750–850, about 16:1), the correspondence is exact. However, generation from a semantic seed mesh needs $\sim 10^3$:1 refinement ratio, where a quarter of the vertices fail the round trip, and nearly a third of image triangles have collapsed. Consequently, we abandoned this mapping mechanism in our fitting algorithm.

D MODEL, TRAINING AND GENERATION CONFIGURATIONS

D.1 OUR MODEL

Input. Model's input is a patch of the hop $\leq \eta$ (we use $\eta = 8$ for our three cases) neighborhood of the candidate edge, with coordinates expressed in that edge's own frame. The extraction canonicalizes the split edge into slots 0 and 1, so the two endpoints are always addressable without an index.

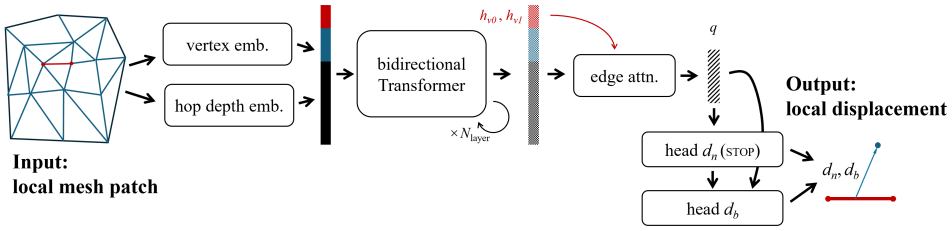


Figure 13: The local state encoder: One forward pass answers one local edge query.

Each vertex carries its hop distance, and an (depth+2)-way embedding layer embeds the hop level and serves as the source of positional information. Vertex coordinates are entered through a $\mathbb{R}^3 \to d$ linear map and are summed with the hop embedding. This input design supports large query-level batches and allows the model inherently *invariant* to global translation, rotation, and scaling of the mesh. It also makes the input at step $t+1$ is recomputed from the realized mesh rather than carried in a hidden state.

Backbone. A bidirectional stack of 5 pre-norm blocks, 128 hidden dimensions, 8 heads, GLU feed-forward, attention, and feed-forward dropout 0.1 (vendored from `x-transformers`). The causal mask is off, so every vertex in the patch sees every other. Padding is carried as an explicit key mask.

Readout. The readout layer concatenates three d-vectors before fusion: the two contextualized endpoint tokens h_{v_0}, h_{v_1}, and one masked attention pool over the whole patch whose *query* is a linear map of $[h_{v_0} \| h_{v_1}]$, with its own key and value projections and $1/\sqrt{d}$ scaling. A two-layer MLP fuses $3d \to d$, and a two-way boundary embedding is added to the result.

Heads and factorization. Two chained classifiers over 2^8 uniform bins. The d_n head reads q and predicts 257 classes, including 256 offset bins plus one STOP class. The d_b head reads $[q \| \mathrm{emb}(d_n)]$ and predicts 256, conditioning the tangential offset on the realized normal offset. The model is therefore locally scale-free.

Quantization. The 2^8 quantization bins are evenly distributed, and their scales are set to cover the full range of action values in the training corpus. We calculate them before training and store the scale as shift $\pm$ scale in the run configuration.

Optimization. The sum of two cross-entropies of d_n and d_b, with the d_b term masked wherever the d_n target is STOP. We use batch 512 *queries* with no gradient accumulation, AdamW at 2×10^{-4}, weight decay 0.01, gradient-norm clip 1.0, 20,000 warmup steps, then a warmup–stable–decay schedule with 800,000 stable and 180,000 decay steps to $0.1\times$ peak. Validation every 10,000 steps, early-stopping patience 20 validations, 16 dataloader workers, one GPU per run.

The architecture is fixed at width 128, depth 5, 8 heads, dropout 0.1, 8-bit quantization for every family, and input padding size 96, resulting in 2.583M parameters.

D.2 BASELINES

Each baseline is run from the original repository. We fine-tune the original models with released weights for TreeMeshGPT and MeshAnything V2. We train the other models from scratch at a capacity matched to ours. E

VertexRegen (Zhang et al., 2025a). We modified the original CGAL tokenizer to gain true vertex pinning. The released code exposes CGAL's `edge_is_constrained_map`, but it rejects a collapse when either endpoint touches a constrained edge, so constraining the corners' incident edges freezes their whole one-rings. We developed a `pinned_vertices` list to reject a collapse only when both endpoints are pinned via CGAL's `filter` slot.

On the model side, we condition the training on a hard seed-mesh token prefix to address the seed-mesh connectivity issue proposed in Section 6.1. It follows

`[BOS]⟨seed verts⟩[SEP]⟨base soup⟩ [SEP]|⟨vsplits⟩[EOS].`

With the base mesh prefilled, the model emits only vertex splits. Its backbone is resized to our S/L/XL rungs via OPT config overrides; the 125M rung instead fine-tunes the released `opt-125m` transformer, with token and position embeddings reinitialized for our vocabulary.

ARMesh (Lei et al., 2025). ARMesh is trained from scratch at a capacity matched to ours (128/5/8). As a soft-conditioned baseline, generation starts from a single point, as in the release, and the condition reaches the model only via a Fourier encoder with a feature dimension of 64 and 8 cross-attention tokens, which read the vertex positions of every seed mesh.

MeshAnything V2 (Chen et al., 2025). We fine-tuned the model with the released weight. We use their own conditioning path designed for point clouds, but we sample points only from our seed mesh rather than the ground-truth mesh.

TreeMeshGPT (Lionar et al., 2025). We fine-tuned the model at width 1024, depth 24, 7-bit coordinates, with their released weights. We use their own conditioning path designed for point clouds, but we sample points only from our seed mesh rather than the ground-truth mesh. Note that their 7-bit/9-bit quantizations are in the global frame, and were not enough to describe the wing dataset we studied, because there are very thin meshes at the tail edge of the wing, where two vertices can be extremely close to each other.

MeshFlow (Sun et al., 2026). This is the non-autoregressive comparison that output a triangle soup generated by equivariant flow matching in parallel. We trained the model matching our model size from scratch. We still use their own conditioning path designed for point clouds, but we sample points only from our seed mesh rather than the ground-truth mesh.

D.3 Ablations

This subsection documents the two ablations in full. Within each model, we modify either the serialization or the model type, choosing between a causal Transformer that reads history and a bidirectional Transformer that reads the mesh state. We trained both models for the bracket task with the same S-size Transformer backbone (128-dim, 5 layers, 8 heads) and the same training settings as in Appendix D.1.

PM × state This ablation retains the model class, using a bidirectional encoder to read the current mesh, and swaps our serialization for VertexRegen's operator, where a vertex split is fully determined by the split site (VSID), its two wing vertices (v_lID, v_rID), and the location of the new vertex (x, y, z). Because VertexRegen's decoder requires global context to predict the split site and wing vertices, we design the model to use the same backbone as CoMesh, but with an input of the full list of vertices at the current mesh state. The head in CoMesh is also replaced to emit the split parameters hierarchically: first VSID, then the wing vertices and the global location from the hidden state, VSID, and additional vertex information on the 1-ring around VSID. The global coordinates are still quantized based on the training dataset to cover the full range.

FRF × causal This arm holds a causal transformer over a flat token stream, but with the serialization fixed to ours FRF. The token stream primarily encodes $[d_n, d_b]$ per accepted split and the [STOP] signal, if it occurs, using the same vocabulary as our CoMesh. However, since the decoder must infer the active edge from history, we add tokens per step to encode the endpoint indices of the two split edges (v_aID, v_bID). The model is trained with the token sequences of $[v_a, v_b, d_n, d_b]$, whereas at generation, the decoder *forces* two identity tokens from its own queue rather than sampling them, so they carry no information at inference and are a training-time cost only. We use a causal transformer with the same backbone as our s-size CoMesh and plain next-token cross-entropy, with padding and identity tokens masked.

D.4 GENERATION PROTOCOL

Every arm is scored on 32 distinct unseen seed meshes $\times$ 32 samples. Face budgets are set to match the training samples. For the sampling settings, our arms use temperature 1.0 and top-k 8, except in the cooling schedule study. The baseline models follow their own sampling settings.

E METRIC DEFINITIONS

We report four groups of metrics that answer four different questions: did the mesh meet the spatial condition it was given (*seed fidelity*); is it a mesh at all (*validity*); could a solver run on it (*quality*); and does the model produce more than one answer (*diversity*). We give each metric's definition once here and refer to this section from the tables.

E.1 SEED FIDELITY

The design task prescribes a seed mesh (the two mounting panels for brackets and the planform shape for wings and hulls). The generated surface must contain the prescribed seed mesh vertices S. The metric is therefore a point-to-surface distance:

$$\text{seed fid.} \;=\; \frac{1}{|S|}\sum_{p\in S}\min_{x\in\mathcal{M}}\|p-x\| \;\Big/\; \operatorname{diag}(S), \tag{14}$$

where S is the set of prescribed seed vertices, $\mathcal{M}$ the generated triangle surface, and $\operatorname{diag}(S)$ the seed's bounding-box diagonal.

E.2 MESH VALIDITY

Validity asks whether the output is well-formed on the surface.

Non-clear rate. (smaller better) The fraction of requested samples that produced a mesh that fails during generation, or have degenerated, or have self-intersection faces. The following metrics can still be obtained for meshes reported as non-clear due to self-intersections.

Self-intersection ratio. (smaller better) The fraction of faces involved in at least one intersection with a non-adjacent face.

Euler characteristic. (should be 1) Obtained by $\chi = V - E + F$, and for all cases we studied, the mesh should be a disk and thereby have $\chi = 1$.

E.3 MESH QUALITY

This is where engineering mesh generation differs most from artist mesh generation, because we usually have a downstream solver that runs on an engineering mesh. For example, a volume cell extruded from a sliver surface triangle makes the stiffness matrix ill-conditioned and inflates the interpolation error in a finite-element or finite-volume solver. A surface that zig-zags between adjacent faces produces spurious curvature and therefore spurious stresses. Knupp (2001).

Aspect ratio. (closer to 1) $(a^2+b^2+c^2)/(4\sqrt{3}\,A)$ for a triangle with edge lengths a, b, c and area A. It is 1 for an equilateral triangle and grows as the triangle degenerates. We report the 95th percentile for a mesh.

Sliver ratio. (smaller better) The fraction of triangles whose minimum angle is below 10°.

Bending energy. (smaller better) $\sum_e \ell_e\theta_e^2/A$ over interior edges, with ℓ_e the edge length, θ_e the exterior dihedral angle and A the total area. It is our primary smoothness number because it penalizes exactly the local folding that produces spurious curvature.

Normal variation. (smaller better) $1-\hat{n}_i\cdot\hat{n}_j$ averaged over interior edges, a first-order companion to bending energy that is insensitive to edge length.

E.4 Mesh Diversity

Diversity asks whether the model explores the design space left open by a spatial condition.

Pairwise Chamfer distance. (larger better) The mean symmetric Chamfer distance between every pair of samples that share a spatial condition, with surfaces sampled uniformly by area. Note that it measures spread within the generated set, so a model emitting degenerate meshes scores well because its outputs differ from one another rather than because they explore valid designs. We therefore only use this metric for our CoMesh model.

Novelty against the training corpus. (larger better) Nearest-neighbor symmetric Chamfer distance from each generated mesh to the training set, which indicates whether the generated meshes differ from what was memorized during training.

F Detailed results

F.1 Model scale

The headline comparison shows our model produces excellent meshes with a relatively small backbone size at $\approx$ 2M. Yet it remains interesting to see how the baselines behave when scaled up, as our results show that the fine-tuned large-size models MeshAnything V2 and TreeMeshGPT perform better. We therefore train the VertexRegen model with different parameter counts from 1.4M to 125M on the bracket task. We report the two representative metrics, the self-intersection ratio and bending energy, in Fig. 14, and the detailed results are provided in Appendix F.2.

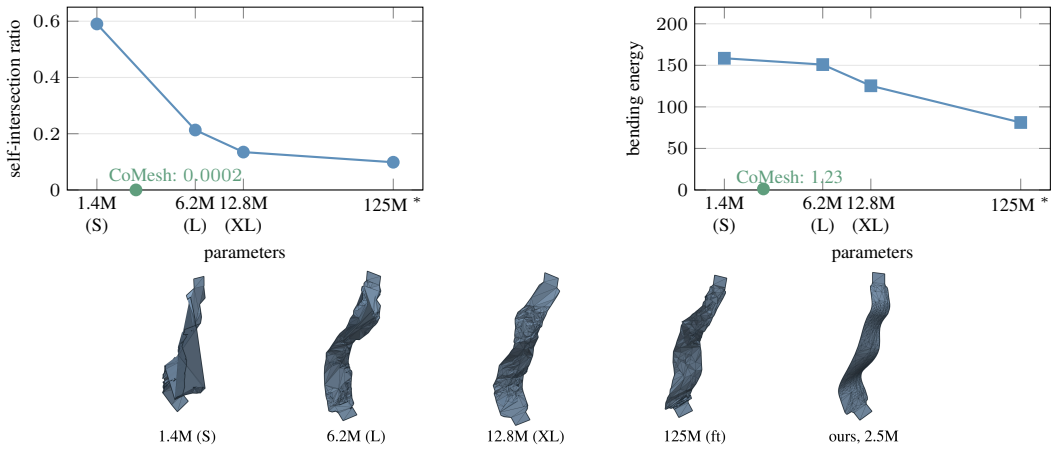


Figure 14: Scaling the VertexRegen.(* : The 125M point is fine-tuned from `OPT-125M` weights.)

As the model size increases, both metrics improve, indicating that larger models can produce better meshes. This aligns with our analysis in Section 6.1, which shows that CoMesh achieves better performance due to a simplified action corpus. Nevertheless, every evaluated model size, including 125M, still can not achieve CoMesh's results.

F.2 Detailed results of generation fidelity, validity, and quality.

Table 7 reports the full generation metrics table of the models involved in the experiments. We evaluated every model on every family with 32 spatial conditions $\times$ 32 samples. The training and validation times are measured on a single NVIDIA A5000, and numbers with * indicate that the run was stopped at our 5-day budget. The generation peak memory is measured for generating one sample, also with A5000. Note that MeshFlow doesn't emit connectivity, so its Euler characteristic, bending energy, and normal variation are not reported.

We further provide a gallery of meshes generated with different models on the three families in Figs. 15 to 17.

Table 7: Full report for the metrics of the models involved in experiments

	seed fid. ↓	self-int. ↓	non-clr. ↓	Euler χ	sliver ↓	aspect p_{95} ↓	bending ↓	norm. var.↓	train (d)	infer (s)	peak (MiB)
bracket (768 faces)											
CoMesh (ours)	0	0.0002	0.02	1.00	0.025	3.00	1.23	0.0065	1.68	6.9	20
abl. FRF×causal	0	0.0062	0.16	1.00	0.053	3.72	4.71	0.0178	5.00*	15.0	103
abl. PM×state	0	0.2501	1.00	-2.76	0.100	5.63	126.23	0.4640	5.00*	46.2	36
VertexRegen S	0	0.5903	1.00	-7.07	0.211	9.17	158.51	0.8976	5.00*	29.3	27
abl. VertexRegen L	0	0.2132	1.00	-4.94	0.151	7.42	150.94	0.4755	5.00*	34.4	73
abl. VertexRegen XL	0	0.1347	1.00	-2.81	0.129	6.55	125.37	0.3512	5.00*	33.8	125
abl. VertexRegen 125M-ft	0	0.0986	1.00	-4.44	0.114	6.50	81.17	0.2399	5.00*	58.8	715
ARMesh	0.1222	0.8576	1.00	-64.75	0.414	22.75	200.48	0.9757	5.00*	9399.9	161
MeshAnything V2	0.0071	0.0035	0.35	0.16	0.020	2.90	802.26	1.5106	5.00*	59.0	3995
TreeMeshGPT	0.0027	0.0178	0.25	-0.78	0.011	2.56	10.94	0.0502	5.00*	699.3	6718
MeshFlow	0.0128	0.7204	1.00	—	0.020	2.74	—	—	5.00*	1.3	1025
wing											
CoMesh (ours)											
768 face	0	0.0003	0.03	1.00	0.009	2.03	3.98	0.0441	0.98	4.1	20
1920 face	0	0.0003	0.03	1.00	0.005	1.88	3.40	0.0276	1.69	13.4	20
4800 face	0	0.0010	0.07	1.00	0.004	1.69	2.27	0.0122	1.69	41.3	20
full resolution	0	0.0017	0.16	1.00	0.004	1.89	1.78	0.0094	1.69	87.2	20
VertexRegen S											
768 face	0	0.8772	1.00	0.88	0.238	10.26	70.01	1.1064	3.11	38.9	35
1920 face	0	0.9325	1.00	0.85	0.279	13.67	95.90	1.1253	5.00*	92.1	58
ARMesh											
768 face	0.0992	0.9631	1.00	-11.78	0.839	2.8×10^8	34.83	1.0953	5.00*	5232.8	152
higher resolution	✗ *cannot decode mesh because cost grows super-linearly and the DFS deadlocks.*										
MeshAnything V2											
768 face	0.0260	0.0058	0.50	-6.13	0.013	2.28	233.31	1.4613	5.00*	57.9	4006
1920 face	0.0289	0.0219	1.00	-12.35	0.004	2.20	393.40	1.4531	5.00*	142.0	4432
higher resolution	✗ *mesh size exceeds fixed pretrained context.*										
TreeMeshGPT											
	✗ *Unable to qualify mesh vertices with 7-bit coordinates*										
MeshFlow											
768 face	0	0.7496	1.00	—	0.012	2.05	—	—	3.25	1.2	1025
1920 face	0	0.8158	1.00	—	0.010	1.87	—	—	5.00*	1.9	1028
higher resolution	✗ *cannot train on it because OT coupling scale with* $\mathcal{O}(n^3)$.										
hull (1152 faces)											
CoMesh (ours)	0	0.0002	0.03	1.00	0.032	2.95	2.93	0.0142	2.12	14.9	21
VertexRegen S	0.0017	0.7304	1.00	-4.45	0.189	9.38	229.27	0.9901	5.00*	41.7	39
ARMesh	0.0312	0.9855	1.00	-272.06	0.317	16.76	28.25	0.9609	5.00*	3576.4	299
MeshAnything V2	0.0413	0.0027	0.59	-0.77	0.008	2.11	738.91	1.4833	5.00*	67.4	7690
TreeMeshGPT	0.0026	0.0060	0.52	-0.19	0.013	2.34	10.97	0.0571	5.00*	105.9	4532
MeshFlow	0.0111	0.7324	1.00	—	0.015	2.28	—	—	5.00*	1.5	1025

F.3 SAMPLING SCHEDULE

Table 8 reports the full generation metrics table of the CoMesh model for the bracket dataset, with emphasis on generation diversity.

Table 8: Full report for the metrics of CoMesh on bracket generation with different initial top-k and cooling schedule.

			diversity		validity		element quality			
schedule	k_0	n	pair. CD	×base	clean	self-int.	sliver	aspect p_{95}	norm. var.	bending
baseline	8	1023	0.00183	1.00	97.5%	0.00023	0.0252	3.00	0.0065	1.24
fixed	16	1024	0.00249	1.36	97.2%	0.00040	0.0240	2.95	0.0080	1.56
	32	1024	0.00420	2.30	94.5%	0.00105	0.0256	2.98	0.0106	2.21
	48	1024	0.00608	3.33	94.8%	0.00110	0.0264	2.98	0.0124	2.54
	64	1023	0.00779	4.27	90.7%	0.00255	0.0279	3.01	0.0152	3.31
hold 10, ramp 30	16	1024	0.00248	1.36	97.7%	0.00022	0.0234	2.93	0.0071	1.37
	32	1022	0.00417	2.29	96.8%	0.00059	0.0252	2.97	0.0085	1.72
	48	1022	0.00605	3.31	95.5%	0.00124	0.0259	2.99	0.0101	2.18
	64	1021	0.00777	4.25	95.0%	0.00127	0.0272	3.02	0.0111	2.33
hold 5, ramp 15	16	1023	0.00245	1.34	97.9%	0.00023	0.0235	2.94	0.0068	1.33
	32	1023	0.00408	2.23	97.0%	0.00044	0.0243	2.96	0.0079	1.56
	48	1022	0.00594	3.26	97.4%	0.00035	0.0252	2.96	0.0084	1.63
	64	1023	0.00764	4.18	96.1%	0.00133	0.0267	3.02	0.0100	2.08

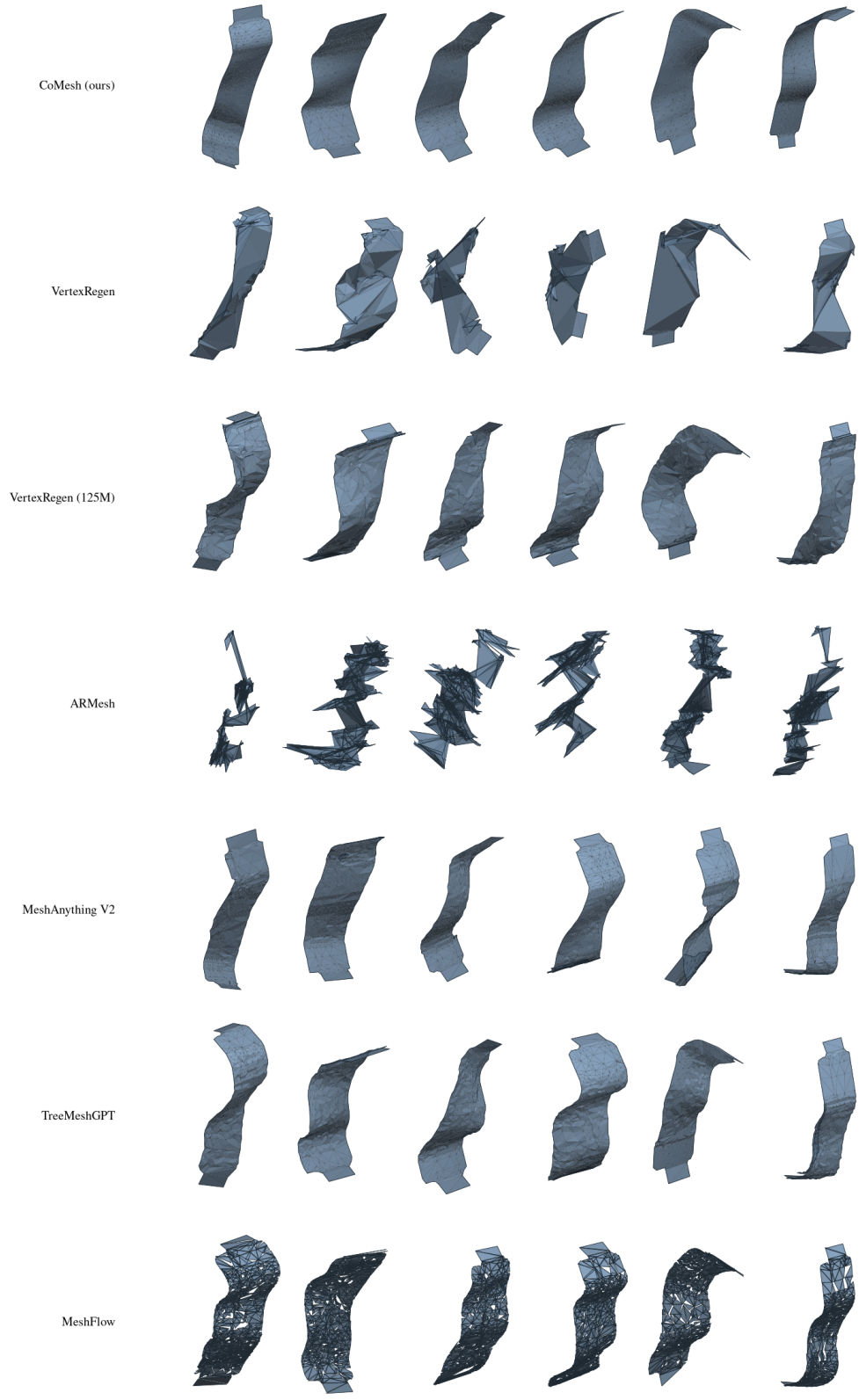


Figure 15: Bracket meshes generated with our and baseline models.

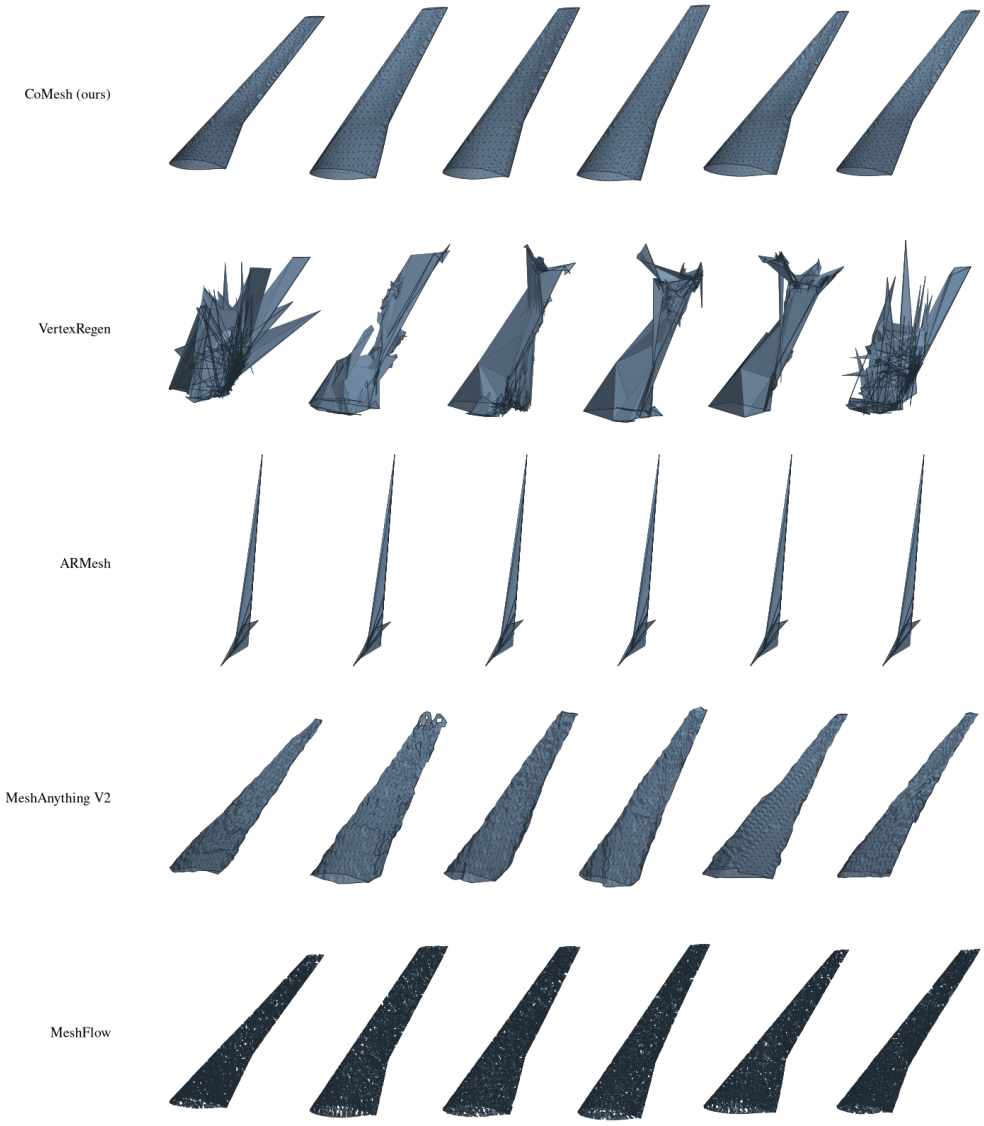


Figure 16: Wing meshes generated with our and baseline models.

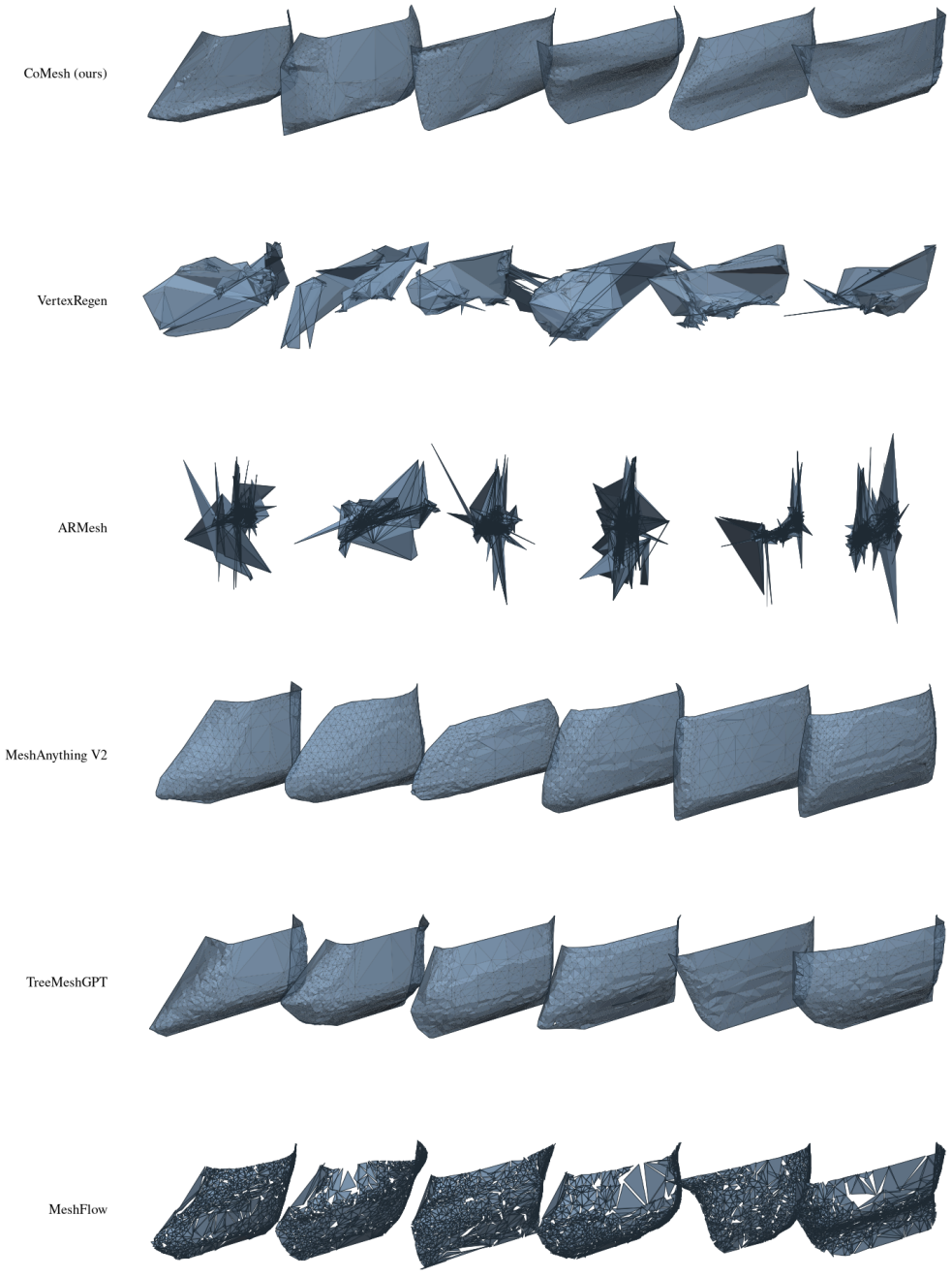


Figure 17: Hull meshes generated with our and baseline models.