InstructMesh: Selective Refinement of Generative 3D Models for Fabrication

Faraz Faruqi
ffaruqi@mit.edu
MIT CSAIL
Cambridge, MA, USA

Ahmed Katary
atkatary@google.com
Google
Sunnyvale, CA, USA

Demircan Tas
tasd@mit.edu
MIT CSAIL
Cambridge, MA, USA

Theresa Hradilak
hradilak@mit.edu
MIT CSAIL
Cambridge, MA, USA

Ning Zhang
ningrz@mit.edu
MIT CSAIL
Cambridge, MA, USA

Jiaji Li
jiaji@mit.edu
MIT CSAIL
Cambridge, MA, USA

Fabian Manhardt
fabianmanhardt@google.com
Google
Munich, Germany

Martin Nisser
nisser@uw.edu
University of Washington
Seattle, WA, USA

Vrushank Phadnis
vrushank@google.com
Google
Mountain View, CA, Germany

Federico Tombari
tombari@google.com
Google
Zurich, Switzerland

Varun Jampani
varunjampani@gmail.com
Stability AI
Boston, MA, USA

Megan Hofmann
Khoury College of Computer Sciences, Northeastern University
Boston, MA, USA

Stefanie Mueller
stefanie.mueller@mit.edu
MIT CSAIL
Cambridge, MA, USA

Figure 1: InstructMesh enables fabrication-relevant refinement of generative 3D models through selective editing. a) A user provides an initial prompt. b) The system generates a 3D model, but may include fabrication-related flaws (e.g., a sealed lid). c) The user selects the problematic region and provides a descriptive edit prompt. d) InstructMesh applies an edit operation in the latent-space to remove the lid while preserving other regions and regenerates the model. e) The final model is fabricated as a usable mug.

ABSTRACT

Recent advances in generative AI allow users to create 3D models from text or images. However, these generative models prioritize visual plausibility and may generate results with flaws that compromise their intended use post-fabrication. We present InstructMesh, a system that enables selective refinement of generative 3D models through region selection and targeted operations, such as opening or sealing voids, or adjusting local thickness. Users can invoke edit operations either through natural language prompts or sliders controls. By operating on the intermediate representation, InstructMesh allows users to apply geometric corrections without requiring expert modeling skills. To inform our design, we first analyze common fabrication-related failure modes in outputs from state-of-the-art generative tools. We then conduct two user studies with novices, demonstrating that InstructMesh improves fabrication viability of generative outputs and revealing user preference

for hybrid interfaces that combine a slider interface with natural language input.

CCS CONCEPTS

• **Human-centered computing → Human computer interaction (HCI)**.

KEYWORDS

Personal Fabrication; Digital Fabrication; 3D Printing; Generative AI.

ACM Reference Format:
Faraz Faruqi, Ahmed Katary, Demircan Tas, Theresa Hradilak, Ning Zhang, Jiaji Li, Fabian Manhardt, Martin Nisser, Vrushank Phadnis, Federico Tombari, Varun Jampani, Megan Hofmann, and Stefanie Mueller. 2026. InstructMesh: Selective Refinement of Generative 3D Models for Fabrication. In *Proceedings of (Preprint).* ACM, New York, NY, USA, 21 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Generative AI has enabled new forms of creative expression across modalities like text [9], images [39], music [30], and 3D geometry [16]. In the 3D domain, users with minimal modeling experience can now quickly generate novel 3D models using text or image prompts, without learning complex 3D modeling techniques. However, while these generative models unlock new digital workflows, their outputs are often disconnected from the geometrical constraints required for real-world fabrication.

3D models intended for fabrication must satisfy both aesthetic and functional requirements [14]. In this work, we use the term 'functionality' to refer to fabrication relevant geometric properties — such as openings, geometric thickness, and surface continuity — that affect the fabrication and real-world usability of 3D models. Novice makers often circumvent the complexity of 3D modeling by utilizing open-source designs, which are typically created manually by 3D designers and shared online [26, 28, 49]. However, they are often limited to a static library of models, and can only perform limited customization [1]. Tools like Style2Fab [14] and TactStyle [15] extend the domain of these functionality-aware transformations. However, these methods rely on the assumption that functionality already exists in the base design, and thus focus on preserving rather than inferring or repairing it.

In contrast, 3D generative tools offer a method to create custom 3D models with text or image prompts. However, these tools operate with no prior knowledge of the underlying geometrical requirements of the desired design. Thus, when users provide prompt inputs (e.g., "a minimalist phone stand" or "a knee brace in denim style"), the system must infer both visual form and functional utility — often without access to the full intent of the user or the constraints of the target physical environment. As a result, generated models frequently exhibit flaws that affect not only aesthetics, but also their physical properties that manifest post-fabrication. While expert users may identify and refine these flaws using 3D modeling tools such as Blender [11] or MeshMixer [42], doing so requires significant domain expertise and effort. On the other hand, novice users may try to regenerate results with new prompts. But prior work shows that non-experts often struggle to craft effective prompts [48, 58], leading to vague or overly rigid instructions that fail to achieve the intended change — or worse, fix one issue only to alter other desired details. This limits the usability of generative workflows for non-experts, constraining the creative process of 3D modeling into a prompt-refinement loop [29].

To bridge this gap, we introduce InstructMesh, a system that enables fabrication-relevant refinement of generative 3D models by mapping user-specified edits to targeted operations on an intermediate representation of the generative 3D model. This allows the generative model to 'fill in' the low-level details - such as color and topology - and create a valid surface mesh while satisfying the desired edit of the user. To support diverse workflows, InstructMesh provides two complementary interaction modes: a natural language interface powered by an LLM, and a direct-selection interface exposing a library of parameterized operations with sliders. InstructMesh allows users to express geometric edits by first highlighting the problematic region on the mesh, and providing either descriptive text-prompts or selection from a slider interface. InstructMesh performs the operation on the intermediate representation of the generative 3D model, which is then passed through an AI model to create the refined 3D mesh with the localized edit. Finally, since the latent representation is an inherently non-visual representation, InstructMesh provides an opportunity for the user to verify their desired edit using a preview visualization. When satisfied with this preview, the user can trigger the AI refinement, and the refined 3D model is generated. An advantage of this approach is that it brings elements of WYSIWYG[1]-style interaction into 3D generative modeling, allowing novices to directly manipulate the 3D model rather than rely solely on abstract prompts that often produce unpredictable results [48, 58]. Together, these design choices allow users to edit 3D models in-situ without requiring mesh manipulation expertise, while maintaining transparency and control.

In summary, this work contributes:

(1) A formative analysis of failure modes in generative 3D models, identifying common flaws that prevent real-world fabrication,
(2) A set of geometric operations on intermediate model representations that repair fabrication-relevant flaws without requiring mesh manipulation expertise,
(3) An interaction design for generative modeling that combines two complementary modes: a natural language interface powered by an LLM and a slider interface with parameterized presets,
(4) Two user studies showing that novices can repair flawed models for fabrication with InstructMesh, and that hybrid workflows with previews improve usability and transparency.

2 RELATED WORK

We situate our work at the intersection of human-computer interaction, generative AI, and computational fabrication. Specifically, we draw on prior research in three key areas: (1) support tools in digital fabrication workflows, (2) 3D generative AI workflows and their integration into design processes, and (3) human-in-the-loop customization techniques in 2D and 3D generative systems.

[1]What You See Is What You Get

2.1 Supporting Users in Fabrication Workflows

3D printing is emerging as makers' digital fabrication tool of choice [26]. Online repositories such as Thingiverse provide a low barrier to entry for exploring 3D printing as users can quickly download pre-made 3D designs and explore design-specific questions. However, numerous studies have explored the practices of these novice makers [3, 20, 34, 41] and tend to find that makers struggle with modifications to existing designs due to limitations of current computer-aided design (CAD) workflows. Oehlberg et al. [35] observed that, even when designs are customizable, they lack the full range of modifiable features makers seek.

Researchers have proposed several support tools that enable editing of 3D models by users with the goal to fabricate an object. Meshmixer [42] follows the approach of 'Design to Fabricate' [41] and allows users to manipulate 3D models with the intent to 3D print them afterwards. Several parametric design tools [43, 47, 51] allow interactive customization of parametric designs by a novice user, while maintaining validity and and manufacturability of the design. Tools also support specific retargeting of mechanical components [25, 40, 61], allowing users to recombine mechanical components between 3D models. Several optimization algorithms target post-fabrication physical properties of 3D models, such as balance [37], moment-of-inertia for spinnable models [2], or constrained optimization between center-of-mass and balance [36, 62].

However, these fabrication support tools typically edit existing meshes or models, limiting the ability of novice users to freely explore new designs. Recent developments in generative AI workflows offer an alternative approach, enabling users to construct bespoke 3D models directly from image and text prompts.

2.2 3D Generative AI Workflows

Reconstructing 3D objects from a single monocular image or a text description is a longstanding challenge in computer vision. Recent machine learning methods address this problem by learning priors over large collections of 3D shapes to infer the full geometry of an object from a single image. Several approaches have been proposed for this problem, including but not limited to GANs [7, 16], flow networks [24, 57], VAEs [31, 54], and diffusion models [8, 21]. The recent Large-Reconstruction Model (LRM) architecture [18] based methods [4, 50, 56], have demonstrated generalizable and high-quality 3D reconstruction. These generative methods have enabled the development of support tools for creative exploration and rapid prototyping [45]. 3DALL-E [29] integrates text-to-image AI into CAD software to provide designers with image-based inspiration. VRCopilot [59] proposed a mixed-initiative VR authoring system that integrates generative AI with multimodal interactions and intermediate wireframe representations.

While these generative methods produce visually plausible 3D models, they are typically trained using image-based supervision. As a result, they prioritize appearance over physical or functional correctness. This leads to 3D models that may look realistic but lack critical geometric features essential for real-world functionality. Consequently, models generated through these pipelines may exhibit failure modes that only become apparent when used in downstream tasks like simulation or 3D printing.

2.3 Human-in-the-Loop Customization with Generative Models

The challenges in generating functional models with generative AI stem from a broader limitation shared across generative models for all modalities - images, language, or 3D models. The inherent problem is that generative systems must predict complex outputs from underspecified inputs, such as prompts or partial observations. Under-specification introduces ambiguity, often leading models to produce outputs that are misaligned with the user's intent or task-specific requirements. Prior studies have shown that novices, in particular, struggle with this ambiguity: they often craft vague or overly rigid prompts, fail to debug faulty outputs, or cannot anticipate how a model will respond to abstract instructions [48, 58]. As a result, there has been increasing interest in HCI for designing human-in-the-loop approaches with generative systems that allow users to guide or correct the generative process.

In the image domain, researchers have developed interfaces that move beyond static prompts to support iterative refinement and creative steering. ControlNet [60] enables users to guide text-to-image generative models through intuitive visual conditionings (e.g., sketches or segmentation maps), enhancing user control and creativity in image generation. PromptPaint [10] lets users apply multiple prompts like brushstrokes to different regions of the image, allowing localized control over generation. Promptify [5] integrates large language models to suggest prompt-based edits. PromptCharm [52] visualizes internal attention maps and allows users to revise specific image regions through inpainting and prompt re-weighting, enabling targeted corrections.

Similar trends have emerged with research on interactive systems with Large Language Models. ABScribe [38] supports structured revision of LLM-generated text by letting users invoke alternative rewrites. AI Chains [53] break complex prompts into smaller, inspectable steps, allowing users to debug and tune intermediate generations. EvalLM [23] integrates LLM-based evaluators to help users refine prompts based on user-defined criteria, making the trial-and-error process of prompt engineering more directed and intelligible. Together, these systems illustrate a broader shift toward mixed-initiative generative workflows, where users and models share creative agency.

InstructMesh extends these human-in-the-loop principles to 3D generative modeling, introducing a fabrication-relevant refinement workflow that allows users to correct flaws directly on generated models.

To ground our system design on common fabrication-related issues in generative 3D models, we conducted a formative study.

3 FORMATIVE STUDY

One of the key challenges in creating 3D models with generative AI is to ensure that they contain the relevant geometric details for the desired functionality. We hypothesize that geometries created with 3D generative models fail to replicate crucial local geometric details, such as wall thickness, openings, or manifold integrity. The models constructed with these methods align with the aesthetic properties of the input; however, the missing geometric elements affect the intended functionality of the 3D models, rendering them

non-functional in the real world. To test this, we conduct a formative study with 3D models sourced from the online repository, Thingiverse and assess local geometric errors in reconstruction with a state-of-the-art 3D generative tool.

3.1 Dataset Selection and Generative 3D Reconstruction

Thingiverse is a popular online resource for novice and expert makers to share designs or things for 3D printing. We selected the most popular 100 3D models from Thingiverse. We skipped models shared as multi-part components without an assembled version, since current generative tools reconstruct meshes as a single 3D mesh and not its components. For independent models shared as a collection, we downloaded all the 3D models. After preprocessing, the dataset comprised 100 distinct Thingiverse 'things', encompassing a total of 120 unique 3D models (i.e., complete objects or individual components). In order to evaluate the reconstruction capability of state-of-the-art 3D generative models, we took one image of each model using a rendering tool. We used this image to create a reconstructed version of the 3D model. We used the current state-of-the-art 3D generative model Trellis [55], for our study because it has reported the highest reconstruction fidelity for geometry.

3.2 Inductive Taxonomy Development

We used an iterative qualitative coding method to develop our taxonomy of geometric flaws in 3D models - geometric errors in reconstruction that impact the functionality of the 3D model. For each of the 120 3D models, two expert annotators in 3D modeling compared the generated model and the original model, and described the issues in natural language. Their task was to describe each individual issue independently, and describe the change required. Next, they discussed these annotations and came to an agreement on the issues found.

3.3 Thematic Analysis of Issues

Next, the two annotators independently grouped similar descriptions of issues, then collaboratively discussed and refined these groupings. We continued this process iteratively, merging overlapping categories and refining definitions, until we reached thematic saturation—the point at which no new categories emerged from additional data. We report these flaw categories in Table 1 and show descriptive examples in Figure 2.

3.4 Deductive Classification

After developing our taxonomy of fabrication-relevant geometric flaws, we performed a deductive classification of the dataset using the previously collected annotations. The two expert annotators revisited their initial descriptions and mapped each issue to one or more of the nine categories found in the taxonomy.

Across the 120 models analyzed, we found that fabrication-relevant flaws were widespread: 94 models (78.3%) exhibited more than one distinct type of issue, highlighting the compounding nature of geometric failures in generative reconstructions. On average, each model exhibited 2.4 issues (SD = 1.10), suggesting that these flaws commonly co-occur and cannot be addressed through isolated fixes. The most frequently encountered issues were Extraneous Artifacts (65.8%), Missing Openings (48.6%), and Hollowing Errors (36.9%), while more localized structural flaws like Broken / Missing Bridges (10.8%) or Truncated Features (9.9%) appeared less frequently.

These findings highlight a critical gap in current generative modeling approaches: while state-of-the-art generative models like Trellis achieve high visual fidelity, they can overlook fine-grained geometric flaws critical for fabrication — issues that are hard to detect through image supervision alone. These low-level errors, like missing openings or thin walls, can render models non-functional. This highlights the need for tools like InstructMesh that support targeted, user-guided refinement to ensure fabrication readiness of generated 3D models.

4 SYSTEM OVERVIEW

InstructMesh enables users to generate 3D models with text or image prompts and iteratively refine them to fix fabrication-relevant issues. Recent work in 3D generative methods [19, 55] propose a two-stage pipeline: an *encoder* produces a coarse latent structure (e.g., point cloud or voxel grid) and contextual features about appearance, and a *decoder* refines this information into a high-fidelity mesh. One key idea is that this separation enables controlled customization by modifiying this intermediate representation.

InstructMesh operates directly on this intermediate (latent) representation. Figure 3 illustrates the system design, which comprises of: (1) a set of latent-space operations derived from common fabrication flaws, (2) dual interaction modes—a natural language interface powered by an LLM and a slider interface exposing parameterized presets, and (3) a preview visualization that highlights additive edits in green and subtractive edits in red before they are committed. Together, these components provide fine-grained customization with generative 3D models while maintaining user's transparency and control.

We focus on repairing static functionality in 3D models, as current generative methods do not yet support dynamic mechanisms (e.g., hinges or interlocking parts) that require precise geometric and mechanical constraints. Our goal is to address the static fabrication-related flaws identified in our formative study, with the exception of *Fused Joints*, which involves articulated or mechanical components.

4.1 User Interface and Workflow

Figure 4 shows the InstructMesh web interface. Users begin by providing a text or image prompt to generate a 3D model (Fig. 4a), which is rendered for inspection (Fig. 4b). If a flaw is identified, they activate *Highlight Region* and paint the problematic area (Fig. 4c). Users can then specify the desired edit either by entering a natural language instruction or by selecting an operation from the interface with preset parameters. (Fig. 4d)

Before the edit is applied, the system presents a *preview* overlay: additive changes are shown in green and subtractive changes in red (Fig. 5). This makes latent edits transparent and gives users the opportunity to revise or cancel. Once confirmed, the operation is executed on the corresponding subset of the latent representation and decoded into an updated mesh (Fig. 4f). The edited model appears next to the original for side-by-side comparison, and users

Table 1: Taxonomy of Fabrication-Relevant Geometric Flaws in 3D Model Reconstructions. Based on 100 most popular 3D models on 3D model website Thingiverse.

Flaw Category	Description	Models Affected
Missing Openings	Absence of essential through-holes, ventilation gaps, volumes to contain liquids or support airflow.	48.6%
Fused Joints	Components meant to rotate or articulate (e.g., hinges, pivots, ball joints, axles) are fused, preventing intended motion.	13.5%
Broken or Missing Bridges	Crucial geometric connections between parts are missing, leaving the structure fragmented or unsupported.	10.8%
Extraneous Artifacts	Unintended bumps, extensions, or disconnected mesh elements.	65.8%
Fused / Duplicated Features	Features are erroneously duplicated, stacked, or fused together, often due to generative ambiguity.	27%
Hollowing Errors	Solid elements are mistakenly hollowed or vice versa.	36.9%
Wall Thickness Issues	Non-uniform or insufficient wall thicknesses.	20.7%
Topological Errors	Gaps, tears, or ambiguous topologies in the mesh.	9.9%
Truncated Features	Extrusions or geometric extensions are prematurely cut off, resulting in incomplete or malformed shapes.	9.9%

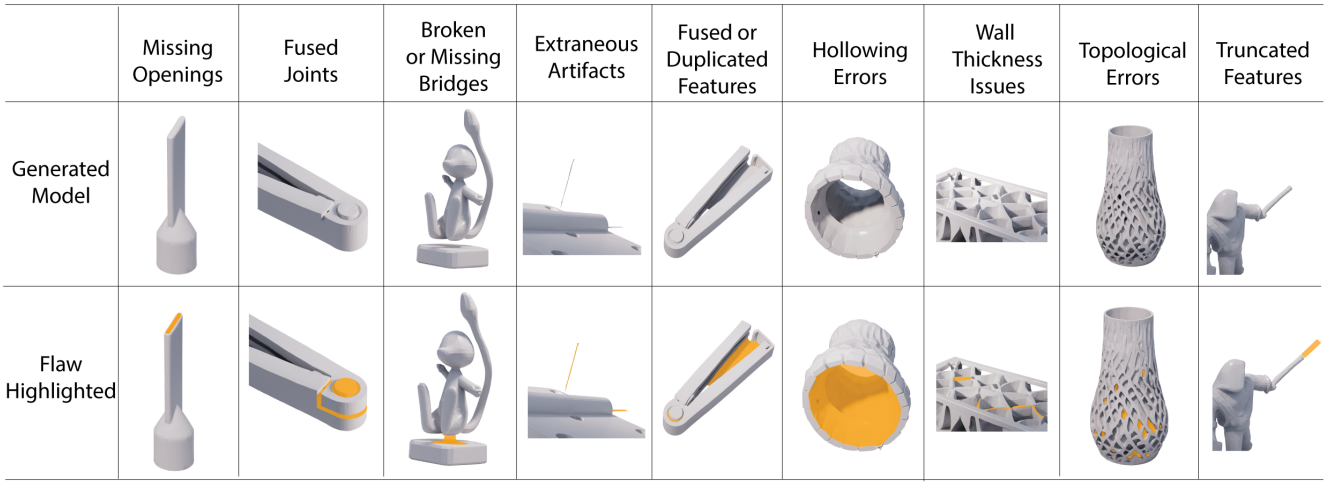

Figure 2: Descriptive examples of flaws found in our formative study. Models were sourced from Thingiverse, reconstructed with 3D generative model Trellis [55] using their image as input, and evaluated manually for geometric functionality errors by reviewers.

can continue refining, undo an operation, or download the final fabrication-ready model.

This workflow enables localized, in-situ corrections, abstracting away the complexity of latent manipulation, while providing transparent feedback loops through preview visualizations. Our goal with this design is to enable novices to iteratively refine generative 3D models with both expressiveness and control.

4.2 Canonical Operations to Modify Latent Representation

In this section, we describe our set of latent-space operations that enable localized, precise edits of a generative 3D model. Editing the latent voxel representation, rather than directly modifying the mesh, offers several advantages. Firstly, latent grids are coarser and lack explicit connectivity, making them easier to manipulate without requiring users to navigate complex mesh topology. Secondly, they are processed by an AI model (*decoder*) that combines them with the previously learned appearance features to produce a coherent mesh. By contrast, editing models in mesh manipulation tools like Blender requires an explicit set of topology-aware color-preserving edits, which often introduce artifacts, making them difficult for novices. By editing the latent representation, we leverage the decoder's ability to synthesize missing geometry, producing a coherent model without requiring the minute adjustments typical in mesh editors.

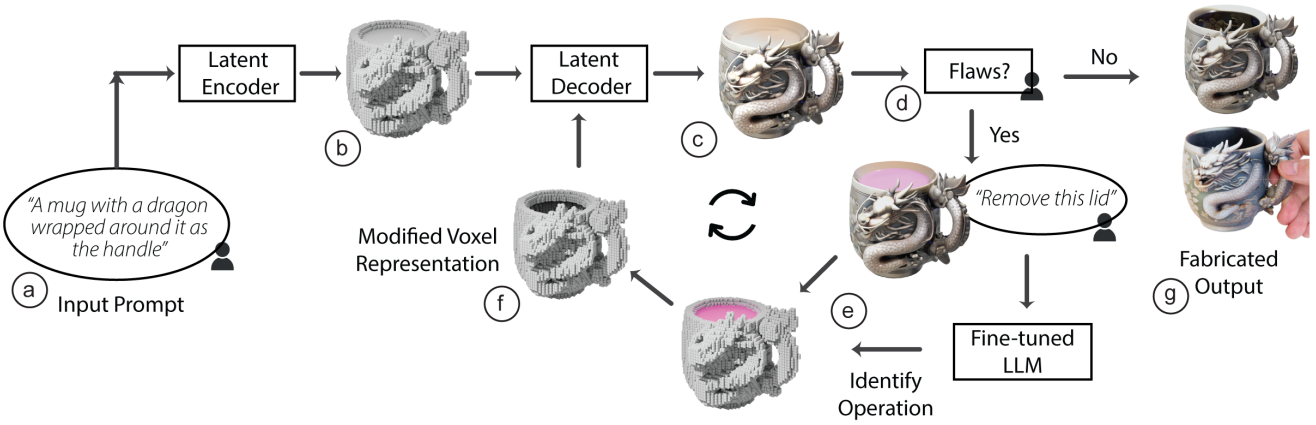

Figure 3: InstructMesh System Overview. (a) A user provides a text or image prompt to generate a 3D model (e.g., "a mug with a dragon wrapped around it as the handle"). (b) The model is encoded into a latent voxel representation and (c) decoded into a textured mesh. (d) The user inspects the mesh for fabrication-related flaws; if none are found, the model proceeds to (g) fabrication. Otherwise, the user highlights a flawed region and specifies an edit via either natural language input or the slider interface. (e) The corresponding latent operation is applied and a preview is presented to the user, (f) Once the user confirms the edit, the edited latents are decoded into a refined 3D model and presented to the user.

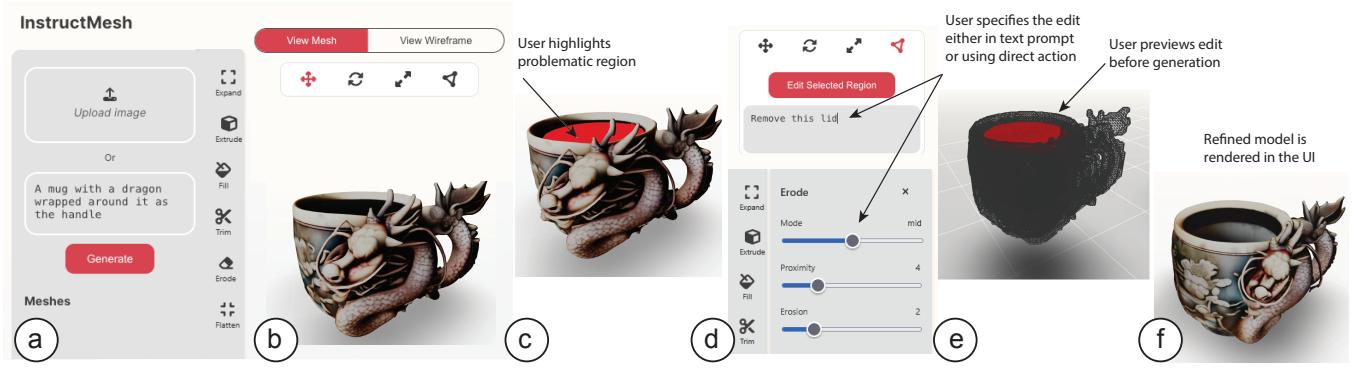

Figure 4: The InstructMesh interface for text-guided refinement of generative 3D models. (a) Users provide a text or image prompt to generate an initial 3D mesh. (b) Users inspect the mesh in rendered or wireframe view to identify flaws, such as the non-functional lid in this example. (c) Users paint the problematic region and (d) describe the desired edit either in natural language (e.g., "Remove this lid") or by direct selection on the interface (e) user inspects the preview and accept or reject the latent-space edit, (f) The system applies a localized latent-space operation to produce an updated model.

Thus, our goal was to create a small set of low-level, controllable operations that generalize across many repair scenarios identified in the formative study. Inspired by common operations in mesh-manipulation tools, we defined operations that are compatible with latent voxel grids and can be flexibly extended. We constructed these as *canonical* operations, analogous to primitive geometric operations such as extrude or subtract. By keeping the operations simple and well-defined, we get a composable vocabulary: individual edits can be combined or sequenced to address more complex repair scenarios. Thus, when users highlight a flawed region on the mesh, the system maps this region to the latent representation, and applies one of these canonical operations before decoding back into a refined mesh. Figure 6 shows examples of these operations.

Geometric Representations. We denote the decoded mesh as $\mathcal{M}$, a user-selected region as $\mathcal{S} \subseteq \mathcal{M}$, and the latent voxel grid as $\mathcal{V}$. A mapping ψ projects voxel indices into world-space coordinates, enabling the use of continuous functions such as signed distance fields (SDFs) when needed. All operations are defined as transformations of $\mathcal{V}$, guided by geometry derived from $\mathcal{M}$ or $\mathcal{S}$.

Canonical Operations. We group our operations into two classes:

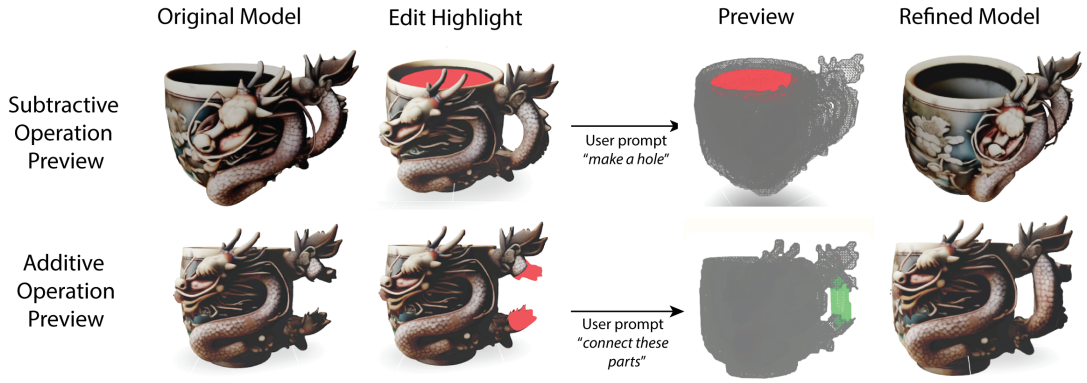

Figure 5: Preview visualizations of canonical operations in InstructMesh. Top: a subtractive operation where the user highlights a region and prompts "make a hole," producing a fabricated-ready mug with an opening. Bottom: an additive operation where the user highlights regions and prompts "connect these parts," resulting in the handle being generated. Previews show localized voxel edits before refinement, helping users anticipate the effect of their edits done in latent representation.

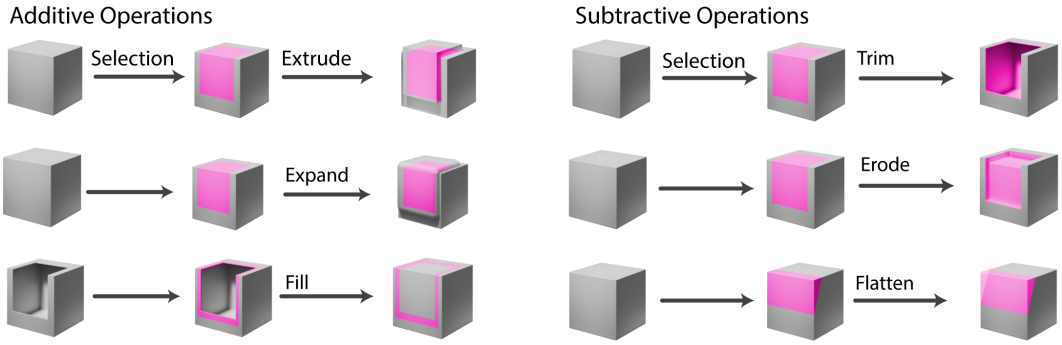

Figure 6: Canonical Operations on Voxel Representations: User highlights on the model are mapped to regions in an intermediate voxel grid and modified according to prompted operations. This modified voxel grid is then used for regeneration to create a refined model with the user modification. Additive operations (left) grow voxel occupancy through extrusion, expansion, or filling, while subtractive operations (right) reduce occupancy via trimming, erosion, or flattening.

- **Additive** (*extrude, expand, fill*): These increase voxel occupancy by extending surfaces outward, thickening thin regions, or filling enclosed volumes. They are used to reinforce fragile areas or close gaps.
- **Subtractive** (*trim, erode, flatten*): These reduce occupancy by cutting away selected regions, thinning shells, or smoothing surfaces. They are used to remove unwanted material, hollow interiors, or prepare flat contact areas.

Together, these operations form a compact, extensible vocabulary for repairing generated 3D models. Each operation can be parameterized (e.g., extrusion depth, erosion thickness), enabling a range of functional modifications while remaining grounded in the latent voxel space. Formal definitions and pseudocode for all operations are provided in Appendix A.

4.2.1 Additive Operations. Additive operations increase voxel occupancy in the latent grid, allowing users to grow, reinforce, or close regions of a model.

Extrude. Extends the selected surface region by adding voxels along its average outward normal direction: $\mathcal{V}' = \mathcal{V} \cup \Delta_{\text{extrude}}(\mathcal{S})$ where Δ_{extrude} denotes voxels sampled along surface normals. This is useful for lengthening handles, extending supports, or creating new protrusions.

Expand. Thickens the overall structure by adding a uniform shell of voxels around the selection: $\mathcal{V}' = \mathcal{V} \cup \Delta_{\text{expand}}(\mathcal{S})$ with Δ_{expand} defined by a dilation radius in voxel space. This reinforces thin walls and makes fragile structures more robust.

Fill. Closes surface gaps or hollow volumes by populating voxels inside the convex hull of the selection: $\mathcal{V}' = \mathcal{V} \cup \Delta_{\text{fill}}(\mathcal{S})$ where Δ_{fill} captures the interior volume of $\mathcal{S}$. Users commonly apply fill to solidify hollow parts or create bases and lids.

Together, these additive operations provide a compact vocabulary for reinforcing weak regions and extending geometry while abstracting away mesh-level complexity. Full pseudocode and derivations are given in Appendix A.

4.2.2 Subtractive Operations. Subtractive operations reduce voxel occupancy in the latent grid, allowing users to remove unwanted regions, thin out shells, or smooth surfaces. Each operation defines a transformation $\mathcal{V}' = \mathcal{V} \setminus \Delta(\mathcal{S})$ where $\Delta(\mathcal{S})$ captures voxels to be removed.

Trim. Removes voxels near the selected surface region, based on a distance threshold derived from a signed distance field. Trim is typically used to clean artifacts, enforce symmetry, or cut away extraneous material.

Erode. Thins a surface or creates voids by subtracting a shell of voxels from the interior of the selection. The erosion depth controls how aggressively material is removed. Erode is often used to hollow parts, lighten structures, or carve channels.

Flatten. Smooths a highlighted region toward a best-fit plane, removing deviations beyond a threshold. Flatten is used to prepare contact surfaces, repair uneven geometry, or produce flat bases for fabrication.

Together, these subtractive operations provide users with a vocabulary for cleaning, thinning, and regularizing generative outputs. Full pseudocode and implementation details are provided in Appendix A.

While these six operations form the core of our toolkit, they are implemented as parameterized variants of a shared voxel manipulation framework. This makes them modular and composable: for instance, *erode* can act as a volumetric counterpart to *trim*, and *flatten* can create bevel-like transitions, while combinations such as *fill* + *erode* can realize higher-level intents like bridges or lids. In this way, a small set of low-level operations supports a wide range of functional edits while maintaining a coherent editing model grounded in latent voxel space.

4.2.3 Operation Prediction Using LLMs with In-Context Learning. To map user-provided descriptions of desired edits to canonical operations, we employ in-context learning (ICL) with OpenAI's GPT-4 model. In-context learning refers to extending the prompt with examples of input-output pairs so that the model can learn the task purely from context—without any fine-tuning. In our case, we provide a set of labeled description-operation pairs from our formative dataset directly in the prompt. This allows the model to infer the most appropriate operation for any new user instruction from a fixed vocabulary of fabrication-relevant operations such as expand, fill, erode, etc.

4.3 Adaptive Representation for Robust 3D Operations

The canonical operations leverage three complementary geometric representations—voxel grids, surface meshes, and signed distance fields (SDFs). Our system dynamically switches between these representations, selecting the most suitable one for each operation: voxels support morphological edits, meshes capture surface detail and user selections, and SDFs enable smooth distance-based queries. This flexibility ensures stable execution of operations that might be brittle in a single mode (e.g., *trim* via SDF thresholds, *fill* via convex hulls, *erode* via distance-aware shells).

This design builds on prior work showing that no single representation is adequate for all geometric tasks [12, 22, 33, 44]. By abstracting these internal switches away from the user, InstructMesh lets users focus on high-level design intent while the system ensures robustness across representations.

5 EVALUATING AND CALIBRATING OPERATIONS

In the previous section, we introduced a set of canonical operations for making fine-grained edits to the latent voxel representation prior to mesh regeneration. Each operation has tunable parameters that must be calibrated to reliably rectify the flaws identified in our taxonomy. To this end, we conducted a technical evaluation. We first randomize our dataset, and split it using a 80%-20% split. We used 20% set as a 'development set' to calibrate each operation, and use the 80% to evaluate its accuracy in resolving an issue.

Using the development set, we manually calibrated each operation (e.g., *expand*, *extrude*, *erode*, *fill*) by highlighting affected regions and testing parameter ranges for their ability to resolve the target flaw. We found that different flaw contexts required distinct parameterizations. To cater to diverse editing scenarios while controlling complexity, we defined a limited set of calibrated presets. Too many presets would increase interface complexity for novices and affect LLM prediction accuracy, whereas too few would reduce flexibility. We therefore defined twelve calibrated presets across six operations: *Erode* (3), *Trim* (1), *Extrude* (3), *Expand* (2), *Flatten* (1), and *Fill* (2). These presets function as operational modes that can be invoked via natural language or the slider interface; detailed descriptions are provided in Appendix B.

5.1 Technical Evaluation

5.1.1 Flaw Correction Accuracy: We tested our calibrated operations on the 80% test split from our formative study. The operations were conducted by the authors with InstructMesh, and the resulting 3D models were stored. To evaluate the repairs, an independent reviewer assessed each flaw and judged whether the correction was successful or unsuccessful. This evaluation was conducted at the flaw level, meaning each flaw was independently repaired, stored, and assessed by the independent reviewer.

As shown in Figure 7, InstructMesh successfully repaired a high proportion of flaws across all categories. The most reliably repaired issues included Missing Openings (96.3%) and Wall Thickness Issues (95.65%), while the lowest success rate was observed for Fused / Duplicated Features at 83.33%. These results demonstrate that the latent operations implemented in InstructMesh are broadly effective across a diverse set of fabrication-relevant flaw types.

5.1.2 Runtime Efficiency: We also evaluated runtime performance. Using Trellis for image-to-3D generation, the average generation time was 31.7s (SD = 3.7s). Applying region-specific latent operations increased this to 43.7s (SD = 5.2s), reflecting the added computation for localized manipulation and regeneration. Despite this overhead, edits remained well under a minute, supporting iterative design workflows.

5.1.3 LLM Accuracy for Operation Selection: We also evaluated how accurately the language model could map natural language descriptions to canonical operations. As described in Section 4.2.3, we used in-context learning (ICL): 20% of annotated flaw instances were provided as example description–operation pairs, and the

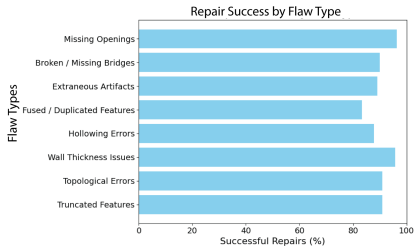

Figure 7: Success rates of canonical operations in resolving fabrication-relevant geometric flaws. This evaluation was conducted on 80% of flaw instances from our annotated dataset. For each flaw, a predefined canonical operation was applied through region selection and LLM-guided latent editing.

model was then tested on the remaining 80%. Across all categories, GPT-4 achieved 92.1% accuracy in predicting the correct operation, demonstrating that ICL is sufficient to integrate natural language input with our editing framework without additional fine-tuning.

5.1.4 Implementation Details: The generative pipeline is implemented in Python and deployed on a cloud-based server (Intel CPU, 32 GB RAM, NVIDIA L4 GPU with 24 GB memory). Canonical latent-space operations were implemented using `Pymeshlab` [32] and `Trimesh` [13].

6 USER EVALUATION I: NOVICE FLAW IDENTIFICATION AND REPAIR

Our goal with InstructMesh is to support novice 3D modelers with little to no prior experience in creating meshes for fabrication. Unlike traditional CAD or mesh-editing tools, InstructMesh abstracts away low-level mesh manipulation, enabling novices to generate and refine bespoke 3D models. Recent generative AI methods can create complex 3D geometries from simple text or image prompts in under a minute, but as our formative study showed, these outputs often contain flaws that undermine fabrication.

For InstructMesh to be effective, novices must be able to (1) identify functional flaws in generated models and (2) repair them independently and efficiently using our system.

6.1 Study Design

We recruited 12 participants (7 male, 5 female; ages 20–39, $M = 27.3$, $SD = 5.6$), all with little to no prior experience in 3D modeling. Participants worked with five flawed 3D models randomly drawn from our formative study (Fig. 8), each containing between one and four annotated fabrication flaws.

For each model, participants first inspected the mesh and identified flaws while thinking aloud. They then used InstructMesh to attempt repairs, with up to 10 minutes per model. Participants could address flaws in any order, and researchers reminded them of unaddressed flaws if overlooked. After completing their edits, participants judged for each flaw whether the repair was successful or unsuccessful. An independent researcher, post user-study sessions, evaluated the final meshes in the same way. Both participant and expert judgments were recorded as binary outcomes for analysis.

We measured task completion time, the number of flaws identified and repaired, and the success rate of repairs based on these binary judgments.

We also evaluated whether participants could successfully repair the flaws they identified. For each annotated flaw, we recorded whether the participant's repair was judged successful or unsuccessful, based on both their own binary judgment and an independent expert's assessment. This allowed us to compute repair success rates across flaw categories and to compare novice performance with expert verification. Together, these measures capture both the ability to diagnose problems in generative 3D models by novices, and their ability to resolve them using InstructMesh.

6.2 Results

We summarize our findings on a per-participant and per-model basis. We use t-tests and ANOVA tests to determine if either participants, models, or specific flaws had a significant effect on any of our evaluation metrics (i.e., flaw identification, repair time, repair success rates, and resulting model evaluations).

6.2.1 Error Identification. Participants successfully identified 90.4% of fabrication-relevant flaws in AI-generated models (Fig. 12), showing that novices could reliably recognize most issues. However, performance varied by both model and flaw type. A repeated-measures ANOVA revealed a significant effect of model ($F(4, 55) = 3.06, p < 0.05$); post-hoc tests showed that flaws in Model M1 were significantly easier to detect than those in Model M5 ($p < 0.05$). This suggests that geometric complexity can influence how easily flaws are perceived.

Flaw type also had a significant effect ($F(5, 139) = 19.24, p < 0.0001$). Topological issues were the hardest to detect, performing significantly worse than truncated features, extraneous artifacts, missing openings, and fused features. In contrast, extraneous artifacts and missing openings were among the most readily identified. Together, these results demonstrate that while novices can generally identify fabrication flaws, both the geometry of the model and the type of flaw strongly affect detectability.

6.2.2 Repair Times. Repair efficiency varied significantly across both models and flaw types (Fig. 10). A repeated-measures ANOVA revealed a significant effect of model on repair time ($F(4, 55) = 7.27, p < 0.001$); post-hoc tests showed that Model M1 was repaired significantly faster than M3 ($p < 0.01$), M4 ($p < 0.01$), and M5 ($p < 0.05$). These differences reflect variation in the number and type of flaws present across models.

Flaw type also had a significant effect on repair time ($F(5, 121) = 5.43, p < 0.001$). Truncated features (3.00 ± 0.95 min) took significantly longer to repair than topological errors (1.50 ± 0.52 min, $p < 0.01$) and fused features (1.84 ± 0.75 min, $p < 0.05$); missing openings (2.33 ± 1.24 min) also took longer than topological errors ($p < 0.05$). These results suggest that while flaws such as topological errors were harder to identify, they could be resolved quickly once recognized, whereas flaws like truncated features required more time-consuming interventions.

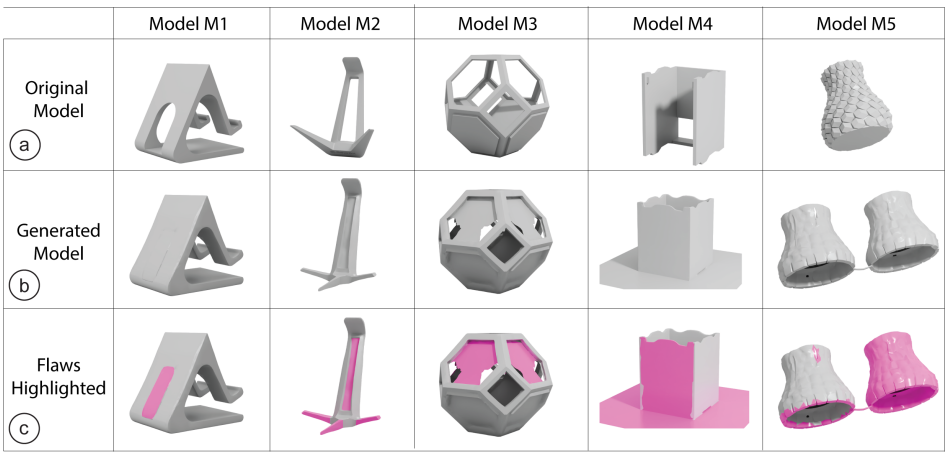

Figure 8: Overview of the five models used in our user study. Each column shows one of the five exemplar tasks (M1–M5). a) The original model demonstrating the intended geometry and function. b) A flawed version of the model generated by a state-of-the-art text-to-3D generative model. c) The same generated model with fabrication-related flaws highlighted in pink. These flaws—such as missing geometry, excessive thickness, or disconnected parts—were identified in our formative study and used as the basis for evaluating user-driven repair with InstructMesh.

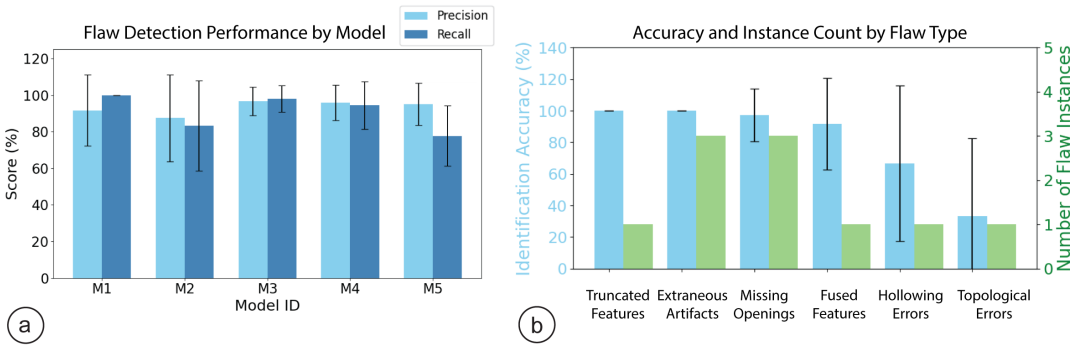

Figure 9: Flaw Identification Performance. (a) Precision and recall of flaw detection across five generated 3D models (M1–M5). (b) Average identification accuracy (left axis) and number of flaw instances (right axis) for each flaw type.

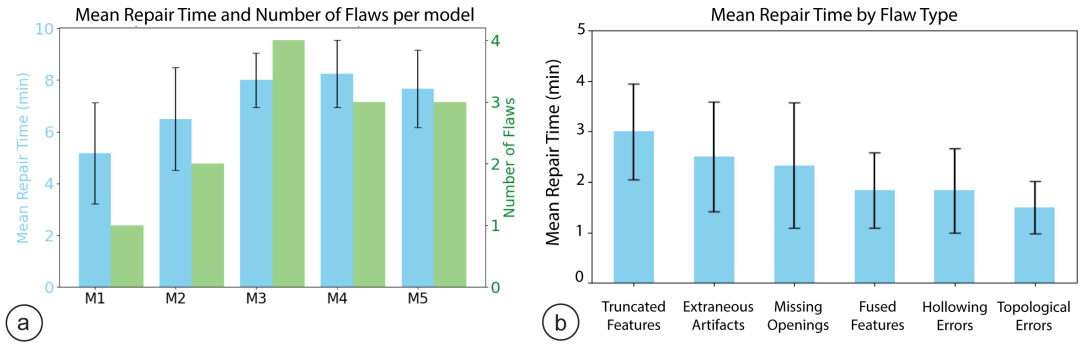

Figure 10: Repair Time and Effort. (a) Mean repair time and number of flaws per model, indicating that models with more flaws required longer repair times. (b) Mean repair time by flaw type. Simpler flaws like topological errors and fused features were repaired faster, while truncated features and missing openings took longer.

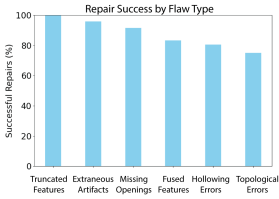

Figure 11: Repair Success by Flaw Type. Participants successfully repaired most flaw types, with success rates exceeding 80% for all categories.

6.2.3 Repair Correction Performance. An independent expert in 3D modeling and fabrication evaluated all user-corrected meshes (Fig. 11). Each flaw was judged as successfully repaired or not, based on whether functionality was restored. Overall, 89.7% of flaws were successfully repaired, with only 16 failures out of 156 annotated flaws.

A repeated-measures ANOVA showed no significant effect of model on repair correctness ($F(4, 55) = 2.25, p > 0.05$), but flaw type had a significant effect ($F(5, 121) = 2.33, p < 0.05$). Post-hoc tests, however, did not reveal significant differences between specific flaw categories. This suggests that while some flaw types may be more challenging, InstructMesh enabled novices to produce fabrication-ready repairs across diverse models.

We also compared participants' self-assessments with the expert's ratings. No significant difference was found ($F(1, 587) = 0.40, p > 0.05$), indicating close alignment between novices' perception of successful repair and expert evaluation. Only 11 cases (7.1%) showed disagreement, where participants believed a flaw was fixed but the reviewer did not.

Overall, these results suggest that novice users were able to identify and correct functional errors in AI-generated 3D models with a high degree of accuracy.

6.3 Participant Feedback

Participants responded positively to the InstructMesh system, highlighting its ease of use, speed, and intuitive natural language interface. Many appreciated the ability to describe changes in text and see results immediately. P2 remarked, "*I like that you can undo your changes, and I just have to describe my changes in text. It's very easy to get used to the tool.*" P1 noted that "*The tool is very easy to use, and makes me excited to do 3D modeling*," while P6 emphasized that "*I don't feel the need to learn CAD software.*" The iterative workflow and undo capability were particularly valued (P6, P11), as was the ability to review previous edits: "*The fact that I see all my previous versions allows me to easily refer to previous edits and compare the outputs*" (P4). Several participants also commented on the clean and simple UI (P7, P11) and its low barrier to entry for novices: "*First time 3D modeling - Didn't know it could be easy!*". Users envisioned a wide range of use cases for the tool. These included rapid prototyping and functional editing (P2, P6, P11), creative hobbyist 3D printing (P3, P7, P9), and educational contexts or toy creation (P4).

7 USER EVALUATION II: INTERACTION MODES AND PREVIEW VISUALIZATION

Our first user study demonstrated that novices can use InstructMesh to identify and repair fabrication-relevant flaws in AI-generated 3D models. However, while that study demonstrated the feasibility of our approach for conducting latent-edits, it left open the question of how different *interaction modes* shape users' experience when applying these edits. Our system supports two complementary interaction modes—natural language input and slider-based controls — that offer different affordances for expressing edits. To investigate the trade-offs between these two interaction modes, we conducted a second study comparing two interface conditions: (1) an LLM-based interface, where participants describe edits in natural language; (2) a Slider-based interface, where participants choose operations through direct selection. Finally, we investigate user's experience with *Preview Visualization*, that highlights predicted latent-space edits in green (additive) and red (subtractive), before the new model is generated. Unlike the first study, which evaluated whether novices could identify and fix flaws, this comparative study focuses on how interaction design impacts usability, clarity of control, accuracy, and user trust.

We address the following research questions:

- **RQ1 (Usability & Control):** How do the LLM and slider interfaces differ in terms of ease of use, effort, clarity of intent expression, perceived accuracy, and sense of control?
- **RQ2 (Previews):** How do latent-space previews influence participants' understanding of edits, their confidence, and their decision to accept or revise?
- **RQ3 (Trade-offs):** Do participants prefer natural language versus direct operation selection, or a combination?

We recruited 12 new participants (7 female, 5 male), none of whom had prior experience with 3D modeling or CAD tools. All were compensated with 20 USD for their time.

7.1 Tasks and Procedure

Each participant completed 12 editing tasks, drawn from the same flawed 3D models used in Study 1. Tasks were split into two blocks of six: one completed with the LLM interface and one with the slider interface. Block order was counterbalanced, to control for learning effects.

For each task, participants were shown a flawed model along with the target corrected version, and the goal was to rectify the error using the assigned interface. This approach standardized the expected outcome, ensuring that differences in performance reflected the interface and preview rather than variation in user interpretation of the edit.

In the LLM condition, participants described the desired edit in natural language, while in the slider condition they selected an operation and either chose a preset or adjusted parameters directly. In both conditions, a preview visualization was always shown before committing the edit, with green overlays indicating additive changes and red overlays indicating subtractive ones.

After each block, participants completed a post-condition questionnaire (7-point Likert scales) assessing ease of use, clarity of intent, perceived accuracy, sense of control, usefulness of the preview, and effort required. After both blocks, a post-study questionnaire

asked participants to compare the two interfaces, reflect on when they preferred natural language versus slider, and indicate whether they would favor LLMs, slider, or a hybrid workflow in a future tool.

7.2 Results

We analyzed the questionnaire data using descriptive statistics and within-subject comparisons between the LLM and slider conditions. For each Likert item, we computed mean ratings across participants, and we compared distributions across conditions.

RQ1 (Usability & Control). Overall, participants found both interfaces usable, with subtle differences in perceived effort and control. The LLM interface was rated as slightly less effortful to use (M = 2.67, SD = 1.23) compared to the slider interface (M = 3.25, SD = 1.36). Participants reported that it felt natural to "*just say what I wanted*" when using language input. At the same time, the slider interface was perceived as slightly more accurate (M = 5.17, SD = 1.19 vs. M = 5.00, SD = 1.21). Perceived sense of control was comparable across both interfaces (both M = 5.58). These results suggest a trade-off: LLMs lowered the barrier to entry by reducing effort, while slider provided a stronger sense of precision and correctness.

RQ2 (Previews). Across both interfaces, previews received consistently high ratings. Participants strongly agreed that the preview clearly showed what the system was going to do (*LLM: M = 6.67, SD = 0.65; sliders: M = 6.75, SD = 0.45*), and that the preview helped them decide whether to accept or revise the edit (*LLM: M = 7.00, SD = 0.00; sliders: M = 6.75, SD = 0.45*). Participants repeatedly described the red/green overlays as making "latent" edits tangible, with one noting that the preview "*made invisible operations visible.*" This suggests that previews are critical in building trust for edits whose outcomes are not inherently WYSIWYG, and directly shaped participants' willingness to proceed with or modify an edit.

RQ3 (Trade-offs). The post-study questionnaire revealed complementary strengths of the two interfaces. When asked to choose an overall preference, participants were evenly split (*6 preferred sliders, 6 preferred LLMs*). All participants (12/12) favored a hybrid design, where both LLMs and sliders are available, due to their complementary strengths.

Participants who favored the slider interface emphasized its precision and reliability: *"I had more control of the edits rather than waiting to see if my intent was clearly understood by the LLM."* (P1) Others noted that sliders were *"more straightforward and gave more accurate results"* (P8, P10). In contrast, participants who preferred the LLM interface highlighted its expressiveness: *"It was easier to specify the modification I wanted"* [P6], and *"I like the quickness of sliders but having LLM interface makes me feel like I can get closest to exactly what I want to be done"* [P11].

When reflecting on scenarios, participants consistently described the two modes as complementary. Natural language was considered useful for broad or creative edits, such as *"generating initial sketches and making big fundamental changes"* (P4), while sliders were preferred for precise local refinements: *"slider for quick edits, fill in a gap, make a hole"* (P2). Nearly all participants indicated that they would prefer a **hybrid workflow** in future 3D generative modeling tools, combining LLMs for learnability and rapid exploration and sliders for precise edits. They also noted that the LLM mode was useful to get started with the interface, since it would provide a low entry-barrier to generating and refining 3D models.

Taken together, these findings suggest that novices value the expressiveness of LLMs and the precision of sliders, and see the hybrid combination with previews as the most effective design direction.

8 APPLICATIONS

In this section, we demonstrate how InstructMesh supports fabrication-aware, prompt-driven editing across six application scenarios covering personal accessories, home decor, medical devices, and robotic enclosures. Each example showcases how users can guide localized latent-space edits to improve functionality and printability while preserving or enhancing a model's intended aesthetic. All examples were printed on a multi-color printer Stratasys J55.

8.1 Home Decor and Functional Objects

Personal fabrication tools are increasingly used to design expressive yet functional home objects. InstructMesh can support users in creating and refining their 3D designs with text- and image-driven prompting. We first showcase a mug with a dragon-shaped handle, where the generated model included a sealed lid that made it unusable. With InstructMesh, the lid region was removed, producing a fabrication-ready version (Figure 13a).

Next, we demonstrate a liquid dispenser inspired by an octopus. The generated model captured the desired aesthetic but was nonfunctional, as the head and legs were sealed. Using InstructMesh, we created a top opening, thickened the surrounding area, and added tunnels through each leg to enable fluid flow. The final design was successfully fabricated, as shown in Figure 13f.

8.2 Medical and Assistive Devices

Personalized fabrication is increasingly important in areas such as "Medical Making" [27] and DIY Assistive Technology [6], where combining functional and aesthetic considerations can improve adoption [46]. Yet many people with disabilities and their care providers lack the time or technical skills to customize designs [17]. To illustrate how InstructMesh can support the design of personalized health and mobility aids, we generated a model of *"a knee brace in the style of blue denim jeans"*. While the model achieved the desired aesthetic, it lacked functional openings for straps. By selecting the relevant regions and applying localized edits to create strap holes, we transformed the model into a fabrication-ready design that could be worn as a knee brace (Fig. 13c).

8.3 Personal Accessories

InstructMesh also supports the refinement of personalized accessories. For instance, we generated *"butterfly-shaped eyeglass frames"* with ornate wing patterns. The initial design looked appealing but was unusable: the lens regions were filled, the bridge was too thin, and the temples were disconnected and too short. With localized edits, we corrected these flaws and regenerated a functional design suitable for fabrication (Fig. 13d).

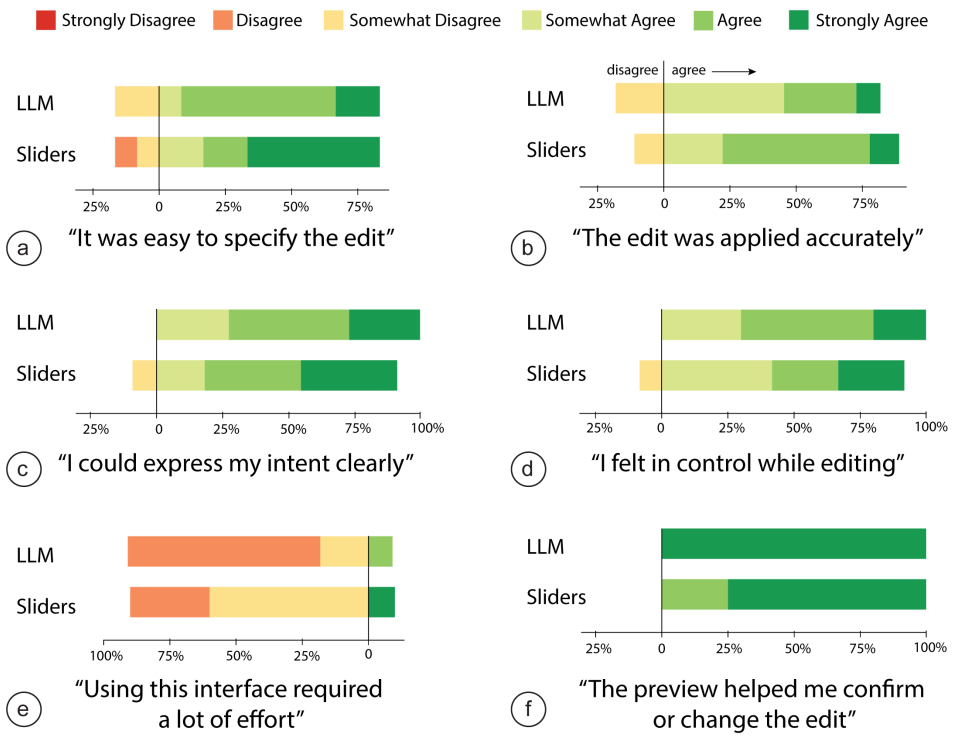

Figure 12: Distribution of Likert results from user study comparing the LLM and slider interfaces. (a–c) LLM was rated slightly easier and more natural for specifying and expressing edits. (b,d) slider scored higher on perceived accuracy and control. (e) Effort ratings were lower for LLMs, indicating reduced cognitive load. (f) Previews were rated very highly across both conditions, with participants noting that the red/green overlays made latent edits visible and guided their decision to accept or revise.

Figure 13: 3D models generated and refined by InstructMesh, demonstrating several application domains including home decor, assistive devices, personal accessories, and enclosures. a) A mug *"with a dragon shaped handle"*, where the user prompted to "*remove the lid*". b) A functional whistle "*in the shape of a seashell*". c) A knee brace generated with a "denim aesthetic", made functional by selectively adding openings for straps. d) A *"butterfly-shaped eyeglass frame"* with refinement using prompts "remove lens region" and "thicken nose bridge". e) A *shrimp-shaped bristle bot enclosure*, edited to remove legs that obstructed motion and to create a cavity for embedding electronics (left: aesthetic shell, right: electronics view). (f) A liquid dispenser inspired by an octopus, refined through region-based prompts like "*create an opening*" at the top and "*tunnel through the legs*" to ensure fluid flow.

We further demonstrate a functional whistle prompted as *"an ornate whistle in the shape of a seashell"*. While the generated model achieved the aesthetic, it lacked the openings and acoustic chambers required to produce sound. Using InstructMesh, we added the necessary tunnels and refined the resonant geometry while preserving the seashell exterior. The fabricated whistle produces a clear tone and remains visually distinctive (Fig. 13b).

8.4 Robotic Enclosures

Finally, we demonstrate InstructMesh's utility in robotics with a shrimp-shaped enclosure for a bristle bot. We generated a *"colorful shrimp with several legs"*, aiming to house a small DC motor for vibration-driven locomotion. While the model captured the desired aesthetic, its front and back legs would cancel out forward motion, and it lacked a cavity for electronics. Using InstructMesh, we shortened the obstructing legs, created an underbelly cavity, and added a tunnel for motor rotation. These localized edits produced a fabrication-ready enclosure that housed the electronics and enabled directed walking once assembled (Fig. 13e).

9 DISCUSSION AND FUTURE WORK

InstructMesh demonstrates how fabrication-aware editing can be integrated into generative 3D workflows through selective highlights and natural language interactions. In this section, we discuss the broader implications of our system design, its current limitations, and future directions.

9.1 Design Guidelines for Fabrication-Aware Generative Tools

From our studies and system implementation, we distill four design guidelines for future systems that support novice users in editing generative 3D models:

- **Offer complementary input modes.** Natural language provides flexibility and expressivity, while structured controls (e.g., sliders, presets) provide predictability. Supporting both allows users to select the mode that best fits their task or comfort level.
- **Provide previews for non-WYSIWYG edits.** Latent-space operations are inherently opaque since they are applied on intermediate representations. Providing users with visual previews that make the edits visible, allow for refinement, improve transparency, trust and decision-making.
- **Bridge intent and operation with parameterization.** Calibrated presets make latent operations more reliable while still leaving room for user adjustment. This balance reduces cognitive load for novices to learn an interface without oversimplifying functionality.
- **Prioritize near real-time feedback.** While instant responses are not always feasible with generative models, edits and previews that complete within short-timeframes are sufficient to maintain iterative workflows and keep novice users engaged.

Together, these guidelines emphasize transparency, flexibility, and responsiveness as core principles for designing usable, fabrication-aware generative AI systems.

9.2 Augmenting Generative 3D Modeling with Traditional Modeling Paradigms

While generative AI has made 3D model creation more accessible, it often abstracts away the fine-grained control traditionally afforded by interactive modeling tools. We see an opportunity to augment generative workflows with principles from established 3D modeling paradigms, where users iteratively sculpt, parametrize, and refine geometry with precision. InstructMesh brings some of this agency back into generative pipelines by enabling interpretable, fabrication-aware operations that let users guide refinement without requiring expert modeling skills.

Although our current focus is on repairing common fabrication flaws, the canonical operations introduced, such as region-based detachment, void sealing, and thickness adjustment, also serve as versatile design primitives. They allow localized edits that preserve global geometry, supporting both corrective and creative use. Because these operations are model- and representation-agnostic, they can be applied across different generative backends. Moreover, InstructMesh is implemented modularly, so advances in 3D generative modeling can be incorporated without altering its core editing framework.

9.3 Extending Functional Control in Generative Design Tools

Most generative 3D pipelines today produce static, monolithic models—useful for visualization or aesthetic inspiration, but often unsuitable for real-world deployment or fabrication. In contrast, traditional 3D design workflows routinely incorporate models with rich functional semantics, including multi-component assemblies, articulated joints, and material-aware design features. We believe that enabling similar functionality-aware generation and editing is a critical next step in the evolution of generative 3D design tools.

InstructMesh takes a step toward this goal by enabling low-level, fabrication-relevant repairs through text-guided edits. However, the tool is currently limited to static geometry. Future systems could extend this approach to support dynamic functionality, allowing users to specify edits such as *"add a hinge"*, *"make these parts interlock"*, or *"ensure it flexes under pressure"*, and generate structurally plausible models that embody these capabilities. Beyond geometry, functional control also requires reasoning about physical properties. Integrating simulation feedback—such as structural, thermal, or material analysis—into generative pipelines would enable users to evaluate not only the visual form of a model, but also its real-world performance. Such advances would shift generative design from producing visually plausible artifacts to creating functionally viable objects for domains like product design, robotics, prosthetics, and architecture.

9.4 Toward Multi-Agentic Generative Design Workflows

Future systems can extend our approach by embedding diverse forms of expertise directly into the generative environment. Rather than relying on a single model, such workflows could introduce specialized "expert agents" for tasks like mechanical simulation,

thermal analysis, range-of-motion checks, or material property evaluation.

For example, a user might highlight the handle of a generated mug and ask, *"Can this support the weight of a full cup of water?"*—invoking a structural stress test. Or a designer creating an enclosure might ask, *"Can I fit a microcontroller here?"*—prompting a spatial reasoning check. The agents would provide feedback, propose corrections, or even trigger latent-space edits, scaffolding novice users with professional-grade analysis.

This shift positions generative systems as co-creators: users articulate design goals and aesthetics, while the system ensures functional viability through embedded expertise. InstructMesh takes a first step in this direction by demonstrating how fabrication-aware edits, guided through natural language and previews, can integrate seamlessly into creative workflows.

10 CONCLUSION

In this work, we present InstructMesh, a system that enables fabrication-aware refinement of generative 3D models through selective editing enabled by text prompts and slider interface. Our approach empowers users to identify and correct structural flaws in AI-generated models by interacting with them at the level of intent rather than low-level geometry. By mapping user instructions and selected regions to a set of canonical latent-space operations, InstructMesh bridges the gap between 3D generative AI models and the functional constraints of physical fabrication. Our formative study revealed common geometric failure modes in state-of-the-art generative models and informed the design of a set of canonical editing operations for local mesh correction. We demonstrate that these canonical operations can be effectively triggered through natural language or accessed directly via the interface. A preview visualization makes this latent-editing process transparent, allowing the user to revise an edit before triggering generative refinement. Our user study shows that novice users can use InstructMesh to detect and repair flaws in under ten minutes per model—without prior 3D modeling experience, and that hybrid workflows with previews improve usability and transparency. Our applications show that InstructMesh supports practical, goal-oriented refinements across domains from decorative accessories, to assistive devices and robotic objects.

REFERENCES

[1] Celena Alcock, Nathaniel Hudson, and Parmit K. Chilana. 2016. Barriers to Using, Customizing, and Printing 3D Designs on Thingiverse. In *Proceedings of the 2016 ACM International Conference on Supporting Group Work* (Sanibel Island, Florida, USA) *(GROUP '16)*. Association for Computing Machinery, New York, NY, USA, 195–199. https://doi.org/10.1145/2957276.2957301

[2] Moritz Bächer, Emily Whiting, Bernd Bickel, and Olga Sorkine-Hornung. 2014. Spin-It: Optimizing Moment of Inertia for Spinnable Objects. *ACM Trans. Graph.* 33, 4, Article 96 (jul 2014), 10 pages. https://doi.org/10.1145/2601097.2601157

[3] Alexander Berman, Ketan Thakare, Joshua Howell, Francis Quek, and Jeeeun Kim. 2021. HowDIY: Towards Meta-Design Tools to Support Anyone to 3D Print Anywhere. In *26th International Conference on Intelligent User Interfaces* (College Station, TX, USA) *(IUI '21)*. Association for Computing Machinery, New York, NY, USA, 491–503. https://doi.org/10.1145/3397481.3450638

[4] Mark Boss, Zixuan Huang, Aaryaman Vasishta, and Varun Jampani. 2024. Sf3d: Stable fast 3d mesh reconstruction with uv-unwrapping and illumination disentanglement. *arXiv preprint arXiv:2408.00653* (2024).

[5] Stephen Brade, Bryan Wang, Mauricio Sousa, Sageev Oore, and Tovi Grossman. 2023. Promptify: Text-to-image generation through interactive prompt exploration with large language models. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–14.

[6] Erin Buehler, Stacy Branham, Abdullah Ali, Jeremy J. Chang, Megan Kelly Hofmann, Amy Hurst, and Shaun K. Kane. 2015. Sharing is Caring: Assistive Technology Designs on Thingiverse. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. Association for Computing Machinery, New York, NY, USA, 525–534. https://doi.org/10.1145/2702123.2702525

[7] Ruojin Cai, Guandao Yang, Hadar Averbuch-Elor, Zekun Hao, Serge Belongie, Noah Snavely, and Bharath Hariharan. 2020. Learning gradient fields for shape generation. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*. Springer, 364–381.

[8] Gene Chou, Yuval Bahat, and Felix Heide. 2023. Diffusion-sdf: Conditional generative modeling of signed distance functions. In *Proceedings of the IEEE/CVF international conference on computer vision*. 2262–2272.

[9] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research* 24, 240 (2023), 1–113.

[10] John Joon Young Chung and Eytan Adar. 2023. Promptpaint: Steering text-to-image generation through paint medium-like interactions. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–17.

[11] Blender Online Community. 2018. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam. http://www.blender.org

[12] Randall Davis, Howard Shrobe, and Peter Szolovits. 1993. What Is a Knowledge Representation? *AI Magazine* 14, 1 (Mar. 1993), 17. https://doi.org/10.1609/aimag.v14i1.1029

[13] Michael Dawson-Haggerty. 2019. Trimesh. https://github.com/mikedh/trimesh. Computer software.

[14] Faraz Faruqi, Ahmed Katary, Tarik Hasic, Amira Abdel-Rahman, Nayeemur Rahman, Leandra Tejedor, Mackenzie Leake, Megan Hofmann, and Stefanie Mueller. 2023. Style2Fab: Functionality-Aware Segmentation for Fabricating Personalized 3D Models with Generative AI. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–13.

[15] Faraz Faruqi, Maxine Perroni-Scharf, Jaskaran Singh Walia, Yunyi Zhu, Shuyue Feng, Donald Degraen, and Stefanie Mueller. 2025. TactStyle: Generating Tactile Textures with Generative AI for Digital Fabrication. *arXiv preprint arXiv:2503.02007* (2025).

[16] Jun Gao, Tianchang Shen, Zian Wang, Wenzheng Chen, Kangxue Yin, Daiqing Li, Or Litany, Zan Gojcic, and Sanja Fidler. 2022. GET3D: A Generative Model of High Quality 3D Textured Shapes Learned from Images. In *Advances In Neural Information Processing Systems*.

[17] Megan Hofmann, Kristin Williams, Toni Kaplan, Stephanie Valencia, Gabriella Hann, Scott E. Hudson, Jennifer Mankoff, and Patrick Carrington. 2019. "Occupational Therapy is Making": Clinical Rapid Prototyping and Digital Fabrication. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3290605.3300544

[18] Yicong Hong, Kai Zhang, Jiuxiang Gu, Sai Bi, Yang Zhou, Difan Liu, Feng Liu, Kalyan Sunkavalli, Trung Bui, and Hao Tan. 2023. Lrm: Large reconstruction model for single image to 3d. *arXiv preprint arXiv:2311.04400* (2023).

[19] Zixuan Huang, Mark Boss, Aaryaman Vasishta, James M. Rehg, and Varun Jampani. 2025. SPAR3D: Stable Point-Aware Reconstruction of 3D Objects from Single Images. arXiv:2501.04689 [cs.CV] https://arxiv.org/abs/2501.04689

[20] Nathaniel Hudson, Celena Alcock, and Parmit K. Chilana. 2016. Understanding Newcomers to 3D Printing: Motivations, Workflows, and Barriers of Casual Makers. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 384–396. https://doi.org/10.1145/2858036.2858266

[21] Heewoo Jun and Alex Nichol. 2023. Shap-e: Generating conditional 3d implicit functions. *arXiv preprint arXiv:2305.02463* (2023).

[22] Michael Kazhdan, Michael Bolitho, and Hugues Hoppe. 2006. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*. Eurographics Association, 61–70.

[23] Tae Soo Kim, Yoonjoo Lee, Jamin Shin, Young-Ho Kim, and Juho Kim. 2024. Evallm: Interactive evaluation of large language model prompts on user-defined criteria. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–21.

[24] Roman Klokov, Edmond Boyer, and Jakob Verbeek. 2020. Discrete point flow networks for efficient point cloud generation. In *European Conference on Computer Vision*. Springer, 694–710.

[25] Yuki Koyama, Shinjiro Sueda, Emma Steinhardt, Takeo Igarashi, Ariel Shamir, and Wojciech Matusik. 2015. AutoConnect: Computational Design of 3D-Printable Connectors. *ACM Trans. Graph.* 34, 6, Article 231 (nov 2015), 11 pages. https://doi.org/10.1145/2816795.2818060

[26] Stacey Kuznetsov and Eric Paulos. 2010. Rise of the Expert Amateur: DIY Projects, Communities, and Cultures. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries (NordiCHI '10)*. ACM, 295–304. https://doi.org/10.1145/1868914.1868950

[27] Udaya Lakshmi, Megan Hofmann, Stephanie Valencia, Lauren Wilcox, Jennifer Mankoff, and Rosa I. Arriaga. 2019. "Point-of-Care Manufacturing": Maker Perspectives on Digital Fabrication in Medical Practice. *Proc. ACM Hum.-Comput. Interact.* 3, CSCW, Article 91 (nov 2019), 23 pages. https://doi.org/10.1145/3359193

[28] Jiaji Li, Mingming Li, Junzhe Ji, Deying Pan, Yitao Fan, Kuangqi Zhu, Yue Yang, Zihan Yan, Lingyun Sun, Ye Tao, et al. 2023. All-in-one print: Designing and 3D printing dynamic objects using kinematic mechanism without assembly. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–15.

[29] Vivian Liu, Jo Vermeulen, George Fitzmaurice, and Justin Matejka. 2023. 3DALL-E: Integrating text-to-image AI in 3D design workflows. In *Proceedings of the 2023 ACM designing interactive systems conference*. 1955–1977.

[30] Ryan Louie, Andy Coenen, Cheng Zhi Huang, Michael Terry, and Carrie J Cai. 2020. Novice-AI music co-creation via AI-steering tools for deep generative models. In *Proceedings of the 2020 CHI conference on human factors in computing systems*. 1–13.

[31] Paritosh Mittal, Yen-Chi Cheng, Maneesh Singh, and Shubham Tulsiani. 2022. Autosdf: Shape priors for 3d completion, reconstruction and generation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 306–315.

[32] Alessandro Muntoni and Paolo Cignoni. 2021. *PyMeshLab*. https://doi.org/10.5281/zenodo.4438750

[33] Ken Museth, David E. Breen, Ross T. Whitaker, and Alan H. Barr. 2002. Level set surface editing operators. *ACM Trans. Graph.* 21, 3 (July 2002), 330–338. https://doi.org/10.1145/566654.566585

[34] Behnaz Norouzi, Marianne Kinnula, and Netta Iivari. 2021. Making Sense of 3D Modelling and 3D Printing Activities of Young People: A Nexus Analytic Inquiry. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 481, 16 pages. https://doi.org/10.1145/3411764.3445139

[35] Lora Oehlberg, Wesley Willett, and Wendy E. Mackay. 2015. Patterns of Physical Design Remixing in Online Maker Communities. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. Association for Computing Machinery, New York, NY, USA, 639–648. https://doi.org/10.1145/2702123.2702175

[36] Romain Prévost, Moritz Bächer, Wojciech Jarosz, and Olga Sorkine-Hornung. 2016. Balancing 3D Models with Movable Masses. In *Proceedings of the Conference on Vision, Modeling and Visualization* (Bayreuth, Germany) *(VMV '16)*. Eurographics Association, Goslar, DEU, 9–16.

[37] Romain Prévost, Emily Whiting, Sylvain Lefebvre, and Olga Sorkine-Hornung. 2013. Make It Stand: Balancing Shapes for 3D Fabrication. *ACM Trans. Graph.* 32, 4, Article 81 (jul 2013), 10 pages. https://doi.org/10.1145/2461912.2461957

[38] Mohi Reza, Nathan M Laundry, Ilya Musabirov, Peter Dushniku, Zhi Yuan "Michael" Yu, Kashish Mittal, Tovi Grossman, Michael Liut, Anastasia Kuzminykh, and Joseph Jay Williams. 2024. ABScribe: Rapid Exploration & Organization of Multiple Writing Variations in Human-AI Co-Writing Tasks using Large Language Models. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–18.

[39] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 10684–10695.

[40] Thijs Jan Roumen, Willi Müller, and Patrick Baudisch. 2018. Grafter: Remixing 3D-Printed Machines. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3173574.3173637

[41] Ryan Schmidt and Matt Ratto. 2013. Design-to-Fabricate: Maker Hardware Requires Maker Software. *IEEE Computer Graphics and Applications* 33, 6 (2013), 26–34. https://doi.org/10.1109/MCG.2013.90

[42] Ryan Schmidt and Karan Singh. 2010. Meshmixer: An Interface for Rapid Mesh Composition. In *ACM SIGGRAPH 2010 Talks*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/1837026.1837034

[43] Adriana Schulz, Ariel Shamir, David I. W. Levin, Pitchaya Sitthi-amorn, and Wojciech Matusik. 2014. Design and Fabrication by Example. *ACM Trans. Graph.* 33, 4, Article 62 (jul 2014), 11 pages. https://doi.org/10.1145/2601097.2601127

[44] Tianchang Shen, Jacob Munkberg, Jon Hasselgren, Kangxue Yin, Zian Wang, Wenzheng Chen, Zan Gojcic, Sanja Fidler, Nicholas Sharp, and Jun Gao. 2023. Flexible Isosurface Extraction for Gradient-Based Mesh Optimization. *ACM Trans. Graph.* 42, 4, Article 37 (July 2023), 16 pages. https://doi.org/10.1145/3592430

[45] Yulin Shen, Yifei Shen, Jiawen Cheng, Chutian Jiang, Mingming Fan, and Zeyu Wang. 2024. Neural canvas: Supporting scenic design prototyping by integrating 3d sketching and generative AI. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–18.

[46] Kristen Shinohara, Cynthia L. Bennett, Wanda Pratt, and Jacob O. Wobbrock. 2018. Tenets for Social Accessibility: Towards Humanizing Disabled People in Design. *ACM Trans. Access. Comput.* 11, 1, Article 6 (mar 2018), 31 pages. https://doi.org/10.1145/3178855

[47] Maria Shugrina, Ariel Shamir, and Wojciech Matusik. 2015. Fab Forms: Customizable Objects for Fabrication with Validity and Geometry Caching. *ACM Trans. Graph.* 34, 4, Article 100 (jul 2015), 12 pages. https://doi.org/10.1145/2766994

[48] Hariharan Subramonyam, Roy Pea, Christopher Lawrence Pondoc, Maneesh Agrawala, and Colleen Seifert. 2023. Bridging the Gulf of envisioning: Cognitive design challenges in LLM interfaces. *arXiv preprint arXiv:2309.14459* (2023).

[49] Lingyun Sun, Jiaji Li, Junzhe Ji, Deying Pan, Mingming Li, Kuangqi Zhu, Yitao Fan, Yue Yang, Ye Tao, and Guanyun Wang. 2022. X-Bridges: Designing Tunable Bridges to Enrich 3D Printed Objects' Deformation and Stiffness. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology* (Bend, OR, USA) *(UIST '22)*. Association for Computing Machinery, New York, NY, USA, Article 20, 12 pages. https://doi.org/10.1145/3526113.3545710

[50] Jiaxiang Tang, Zhaoxi Chen, Xiaokang Chen, Tengfei Wang, Gang Zeng, and Ziwei Liu. 2024. Lgm: Large multi-view gaussian model for high-resolution 3d content creation. In *European Conference on Computer Vision*. Springer, 1–18.

[51] Tom Veuskens, Florian Heller, and Raf Ramakers. 2021. CODA: A Design Assistant to Facilitate Specifying Constraints and Parametric Behavior in CAD Models. In *Graphics Interface 2021*. https://openreview.net/forum?id=1dLDPJeafRZ

[52] Zhijie Wang, Yuheng Huang, Da Song, Lei Ma, and Tianyi Zhang. 2024. Promptcharm: Text-to-image generation through multi-modal prompting and refinement. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–21.

[53] Tongshuang Wu, Michael Terry, and Carrie Jun Cai. 2022. Ai chains: Transparent and controllable human-ai interaction by chaining large language model prompts. In *Proceedings of the 2022 CHI conference on human factors in computing systems*. 1–22.

[54] Zhijie Wu, Xiang Wang, Di Lin, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. 2019. Sagnet: Structure-aware generative network for 3d-shape modeling. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–14.

[55] Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong, and Jiaolong Yang. 2024. Structured 3D Latents for Scalable and Versatile 3D Generation. arXiv:2412.01506 [cs.CV] https://arxiv.org/abs/2412.01506

[56] Jiale Xu, Weihao Cheng, Yiming Gao, Xintao Wang, Shenghua Gao, and Ying Shan. 2024. Instantmesh: Efficient 3d mesh generation from a single image with sparse-view large reconstruction models. *arXiv preprint arXiv:2404.07191* (2024).

[57] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. 2019. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF international conference on computer vision*. 4541–4550.

[58] J Diego Zamfirescu-Pereira, Richmond Y Wong, Bjoern Hartmann, and Qian Yang. 2023. Why Johnny can't prompt: how non-AI experts try (and fail) to design LLM prompts. In *Proceedings of the 2023 CHI conference on human factors in computing systems*. 1–21.

[59] Lei Zhang, Jin Pan, Jacob Gettig, Steve Oney, and Anhong Guo. 2024. Vrcopilot: Authoring 3d layouts with generative ai models in vr. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. 1–13.

[60] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. 2023. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF international conference on computer vision*. 3836–3847.

[61] Ran Zhang, Thomas Auzinger, Duygu Ceylan, Wilmot Li, and Bernd Bickel. 2017. Functionality-Aware Retargeting of Mechanisms to 3D Shapes. *ACM Trans. Graph.* 36, 4, Article 81 (jul 2017), 13 pages. https://doi.org/10.1145/3072959.3073710

[62] Haiming Zhao, Chengkuan Hong, Juncong Lin, Xiaogang Jin, and Weiwei Xu. 2016. Make it swing: Fabricating personalized roly-poly toys. *Computer Aided Geometric Design* 43 (2016), 226–236. https://doi.org/10.1016/j.cagd.2016.02.001 Geometric Modeling and Processing 2016.

A CANONICAL OPERATIONS

In this section, we detail the canonical voxel-based operations from our system. These operations allow users to apply low-level geometry modifications, by simply describing their desired edit in text or picking the operation and parameters through the slider interface. Each operation corresponds to a low-level transformation that addresses specific fabrication-related flaws in a generative 3D model, such as disconnected parts, insufficient thickness, or residual internal geometry. These operations are composable and regionally applied, and each one is implemented through simple voxel manipulations guided by local surface geometry and user instruction.

Below, we provide pseudocode and a brief explanation for each canonical operation.

A.1 ExpandVoxels: Isotropic growth of voxel grid

This operation uniformly grows the selected voxel region by a radius r, enabling users to thicken thin structures or reinforce fragile parts. It operates by taking a union of the input voxel set with a dilation neighborhood defined by a spherical structuring element.

Let $B_r \subset \mathbb{Z}^3$ denote the discrete ball of radius r in voxel units:

$$B_r = \left\{ u \in \mathbb{Z}^3 \;\middle|\; \|\delta \cdot u\|_2 \le r \right\}$$

The expanded voxel set is given by:

$$\mathcal{V}' = \bigcup_{v \in \mathcal{V}} \{ v + u \mid u \in B_r \}$$

Algorithm 1 ExpandVoxels: Isotropic growth of voxel grid

```
function ExpandVoxels(V, r, δ)
    B_r ← {u ∈ Z^3 | ||δ · u||_2 ≤ r}
    V' ← ∅
    for each v ∈ V do
        for each u ∈ B_r do
            V' ← V' ∪ {v + u}
        end for
    end for
    return V ∪ V'
end function
```

A.2 ExtrudeVoxels: Directional growth along mesh normal

This operation expands geometry outward in the direction of the average surface normal, allowing targeted directional reinforcement such as bumping out thin areas or adding mass beneath protruding components.

Given a voxel grid $\mathcal{V} \subset \mathbb{Z}^3$, a user-selected mesh region $\mathcal{S} = (V_S, F_S)$, and an extrusion amount $d \in \mathbb{R}^+$, the extrude operation grows new voxels in the direction of the average surface normal. We define:

$$\vec{n} = \frac{1}{|F_S|} \sum_{f \in F_S} \hat{n}_f$$

where $\hat{n}_f$ is the unit normal of face f. Let $\mathcal{V}_{\text{near}} \subseteq \mathcal{V}$ denote voxels whose world-space centers lie within a fixed distance ϵ of $\mathcal{S}$.

New voxels are generated by sampling along the normal direction:

$$\mathcal{V}_{\text{extrude}} = \left\{ \left\lfloor \frac{\psi(v) + t \cdot \vec{n}}{\delta} \right\rfloor \;\middle|\; v \in \mathcal{V}_{\text{near}},\ t \in [0, d] \right\}$$

The final extruded voxel set is:

$$\mathcal{V}' = \mathcal{V} \cup \mathcal{V}_{\text{extrude}}$$

Algorithm 2 ExtrudeVoxels: Directional growth along mesh normal

```
function ExtrudeVoxels(V, S, d, δ)
    n⃗ ← AverageNormal(S)
    V_sel ← VoxelsNearMesh(V, S, δ)
    V' ← ∅
    for each v ∈ V_sel do
        x ← ψ(v)
        for t ∈ [0, d] in steps of δ do
            x_t ← x + t · n⃗
            v_t ← ⌊x_t/δ⌋
            V' ← V' ∪ {v_t}
        end for
    end for
    return V ∪ V'
end function
```

A.3 FillVoxels: Solidify interior under selected mesh

Filling is used to resolve hollow shells or disconnected boundaries by computing the convex hull of the selected mesh and filling its interior with voxels. This solidifies regions that may otherwise be unprintable or structurally unsound.

Given a voxel grid $\mathcal{V} \subset \mathbb{Z}^3$ and a user-selected mesh region $\mathcal{S} = (V_S, F_S)$, the fill operation adds voxels inside the convex hull of the selected region. An optional dilation parameter $d \in \mathbb{R}_{\geq 0}$ can be used to expand the hull before voxelization.

Let $\Omega \subset \mathbb{R}^3$ denote the convex hull of the selected vertices, optionally dilated outward:

$$\Omega = \texttt{ConvexHull}(V_S)$$

The fill voxel set is:

$$\mathcal{V}_{\text{fill}} = \left\{ v \in \mathbb{Z}^3 \;\middle|\; \psi(v) \in \Omega \right\}$$

The updated voxel grid is:

$$\mathcal{V}' = \mathcal{V} \cup \mathcal{V}_{\text{fill}}$$

Algorithm 3 FillVoxels: Solidify interior under selected mesh

function FillVoxels($\mathcal{V}$, $\mathcal{S}$, d, δ)
  $\Omega \leftarrow$ DilatedConvexHull($\mathcal{S}$, d)
  $\mathcal{V}' \leftarrow \emptyset$
  for each v in voxel grid bounds **do**
    $x \leftarrow \psi(v)$
    if $x \in \Omega$ **then**
      $\mathcal{V}' \leftarrow \mathcal{V}' \cup \{v\}$
    end if
  end for
  return $\mathcal{V} \cup \mathcal{V}'$
end function

A.4 TrimVoxels: Remove surface-near voxels

Trimming removes a thin shell of voxels near the surface of a selected region. This is useful for eliminating surface noise or overgrowth that protrudes beyond a desired threshold, effectively performing a localized simplification.

Given a voxel grid $\mathcal{V} \subset \mathbb{Z}^3$, a mesh region $\mathcal{S} = (V_S, F_S)$, and a trimming threshold $\tau \in \mathbb{R}_{\geq 0}$, we first compute a signed distance field $\phi : \mathbb{R}^3 \to \mathbb{R}$ from the mesh surface.

Voxels whose world-space centers lie closer than τ to the surface are removed:

$$\mathcal{V}' = \{ v \in \mathcal{V} \mid \phi(\psi(v)) \geq \tau \}$$

This operation is commonly used to clean up generative artifacts, isolate regions, or subtract material based on proximity.

Algorithm 4 TrimVoxels: Remove surface-near voxels

function TrimVoxels($\mathcal{V}$, $\mathcal{S}$, τ)
  $\phi \leftarrow$ SignedDistanceField($\mathcal{S}$)
  $\mathcal{V}' \leftarrow \emptyset$
  for each $v \in \mathcal{V}$ **do**
    $x \leftarrow \psi(v)$
    if $\phi(x) \geq \tau$ **then**
      $\mathcal{V}' \leftarrow \mathcal{V}' \cup \{v\}$
    end if
  end for
  return $\mathcal{V}'$
end function

A.5 ErodeVoxels: Remove interior voxels after fill

Erosion reverses a prior filling operation by removing voxels that are deep within the object interior. It is particularly useful after overfills, to recover desired hollowness or to reduce weight while retaining structural support.

Since generative models produce surface-only voxels with no volumetric depth, erosion is implemented as a two-step process:

(1) **Volumetric fill.** Given a selected mesh region $\mathcal{S} = (V_S, F_S)$, we compute a signed distance field $\phi : \mathbb{R}^3 \to \mathbb{R}$ and define a filled region as:

$$\Omega_{\text{fill}} = \left\{ x \in \mathbb{R}^3 \;\middle|\; \phi(x) \leq d_{\text{fill}} \right\}$$

where $d_{\text{fill}} = d_{\text{erode}} + d_{\text{surface}}$, and d_{surface} is a constant estimated thickness of the surface shell. We voxelize Ω_{fill} to obtain $\mathcal{V}_{\text{filled}} \subset \mathbb{Z}^3$.

(2) **Erosion.** We define the eroded voxel set by removing voxels close to the surface, keeping only those deep enough inside:

$$\mathcal{V}' = \{ v \in \mathcal{V}_{\text{filled}} \mid \phi(\psi(v)) \leq -d_{\text{erode}} \}$$

Algorithm 5 ErodeVoxels: Remove interior voxels after fill

function ErodeVoxels($\mathcal{S}$, $\mathcal{V}$, d_e, d_s, δ)
  $\phi \leftarrow$ SignedDistanceField($\mathcal{S}$)
  $d_f \leftarrow d_e + d_s$
  $\mathcal{V}_{\text{filled}} \leftarrow$ FillInteriorFromMesh($\mathcal{S}$, d_f, δ)
  $\mathcal{V}' \leftarrow \emptyset$
  for each $v \in \mathcal{V}_{\text{filled}}$ **do**
    $x \leftarrow \psi(v)$
    if $\phi(x) \leq -d_e$ **then**
      $\mathcal{V}' \leftarrow \mathcal{V}' \cup \{v\}$
    end if
  end for
  return $\mathcal{V}'$
end function

A.6 FlattenVoxels: Remove deviations from best-fit plane

Flattening removes small surface deviations to enforce geometric planarity. It is useful for fabricating objects that must interface with flat surfaces (e.g., bases or mounting brackets) or when a smooth contact area is needed.

Given a voxel grid $\mathcal{V} \subset \mathbb{Z}^3$ and a mesh selection $\mathcal{S} = (V_S, F_S)$, we first compute a best-fit plane P through the selected vertices. Let $x_0 \in \mathbb{R}^3$ be a point on the plane and $\vec{n} \in \mathbb{R}^3$ its unit normal.

We remove voxels that deviate from the plane by more than a threshold $\tau \in \mathbb{R}_{\geq 0}$:

$$\mathcal{V}' = \{v \in \mathcal{V} \mid |\vec{n} \cdot (\psi(v) - x_0)| \leq \tau\}$$

Depending on the prompt or parameter settings, additional voxels aligned with the fitted plane may be added to reinforce flatness. This operation is typically used to clean surface irregularities or prepare regions for joining, fabrication, or contact.

Algorithm 6 FlattenVoxels: Remove deviations from best-fit plane

function FlattenVoxels($\mathcal{V}$, $\mathcal{S}$, τ)
  $x_0, \vec{n} \leftarrow$ FitBestPlane($\mathcal{S}$)
  $\mathcal{V}' \leftarrow \emptyset$
  for each $v \in \mathcal{V}$ **do**
    $x \leftarrow \psi(v)$
    $d \leftarrow |\vec{n} \cdot (x - x_0)|$
    if $d \leq \tau$ **then**
      $\mathcal{V}' \leftarrow \mathcal{V}' \cup \{v\}$
    end if
  end for
  return $\mathcal{V}'$
end function

B PARAMETER PRESETS FOR CALIBRATING CANONICAL OPERATION

During system development, we defined a set of parameter presets tailored to common editing scenarios. Each preset configures the operation's thresholds, growth directions, and extent parameters to match typical flaw correction tasks (e.g., sealing holes, merging parts, reinforcing thin walls).

Here, we provide two examples of calibrated operations - variations of canonical operations, calibrated for a specific task. *BridgeVoxels* is a variation of *Fill*, used to connect two surfaces together. *Tunnel* is a variation of *Fill*, but creates a negative volume that is removed.

B.1 BridgeVoxels: Connect disjoint parts via convex fill

This operation connects two disjoint regions by computing a dilated convex hull over their union and filling the resulting shape. It is commonly used to attach floating elements or fuse structurally weak joints.

Algorithm 7 BridgeVoxels: Connect disjoint parts via convex fill

function BridgeVoxels($\mathcal{V}$, $\mathcal{S}_1$, $\mathcal{S}_2$, d, δ)
  $V_{\text{all}} \leftarrow V_{\mathcal{S}_1} \cup V_{\mathcal{S}_2}$
  $\Omega \leftarrow$ DilatedConvexHull(V_{all}, d)
  $\mathcal{V}' \leftarrow \emptyset$
  for each v in voxel grid bounds **do**
    $x \leftarrow \psi(v)$
    if $x \in \Omega$ **then**
      $\mathcal{V}' \leftarrow \mathcal{V}' \cup \{v\}$
    end if
  end for
  return $\mathcal{V} \cup \mathcal{V}'$
end function

B.2 TunnelVoxels: Carve channel between regions

This operation creates a channel connecting two mesh regions by constructing a negative convex hull, and removing the voxels from that region. It can be used to create internal airflow, or channels for fluids to flow or to enable wiring paths. This is derivation of *Fill*, but a negative operation - subtracting geometry instead of adding.

Algorithm 8 TunnelVoxels: Carve channel between regions

function TunnelVoxels($\mathcal{V}$, $\mathcal{S}_1$, $\mathcal{S}_2$, r)
  $c_1 \leftarrow$ Centroid($\mathcal{S}_1$), $c_2 \leftarrow$ Centroid($\mathcal{S}_2$)
  $L \leftarrow$ LineSegment(c_1, c_2)
  $\mathcal{V}' \leftarrow \emptyset$
  for each $v \in \mathcal{V}$ **do**
    $x \leftarrow \psi(v)$
    $d \leftarrow$ DistancePointToSegment(x, L)
    if $d > r$ **then**
      $\mathcal{V}' \leftarrow \mathcal{V}' \cup \{v\}$
    end if
  end for
  return $\mathcal{V}'$
end function

In all, we created 12 calibrated operations of all our canonical operations. Here, we list all of our operations.

- **Erode**: `erode_low`, `erode_high`, `tunnel`
- **Trim**: `trim`
- **Extrude**: `extrude_low`, `extrude_med`, `extrude_high`
- **Expand**: `expand`, `expand_negative`
- **Flatten**: `flatten`
- **Fill**: `fill`, `bridge`

C FLAWS IDENTIFIED IN 3D MODELS FROM APPLICATION SECTION

To demonstrate the fabrication-relevant flaws that arise in real-world generative modeling scenarios, we tested our system with set of six application examples (see Figure 14). Each model was generated from a user-provided prompt using a state-of-the-art text-to-3D pipeline and then corrected for flaws identified. We summarize the specific geometric flaws found in the initial outputs and the corresponding refinements applied using InstructMesh.

Butterfly-shaped eyeglass frame: The generated model lacked hollow regions for the lenses and failed to connect the temple arms to the frame geometry. These issues correspond to 'Missing Openings' and 'Broken or Missing Bridges'. We resolved these flaws by hollowing the lens region and reinforcing the frame connections.

Denim-styled knee brace: The output included purely decorative straps with no openings, rendering the model non-functional for wear. This represents a 'Missing Openings' flaw, which we addressed by locally adding holes to insert physical straps and removing extraneous rear geometry for better fit.

Liquid dispenser in the shape of an octopus. : While visually accurate, the model lacked flow pathways for dispensing liquid and included sealed extremities, reflecting both Missing Openings and Hollowing Errors. We added tunnels through each leg and a top opening to allow proper fluid flow.

Shrimp-shaped bristle bot enclosure. : The generated model contained front legs in conflicting directions and lacked space for internal components — examples of Extraneous Artifacts and Missing Internal Cavities. We trimmed obstructing legs and introduced a cavity for electronics and motor clearance.

Seashell-shaped whistle. : This model failed to include airflow passages necessary for acoustic function, and contained uneven geometry across the mouthpiece. These are examples of Missing Openings and lack of channels. We created internal channels and created blowhole and airflow openings for the functionality of a whistle.

InstructMesh enables targeted correction of these issues, preserving aesthetic intent while improving fabrication viability and desired functionality.

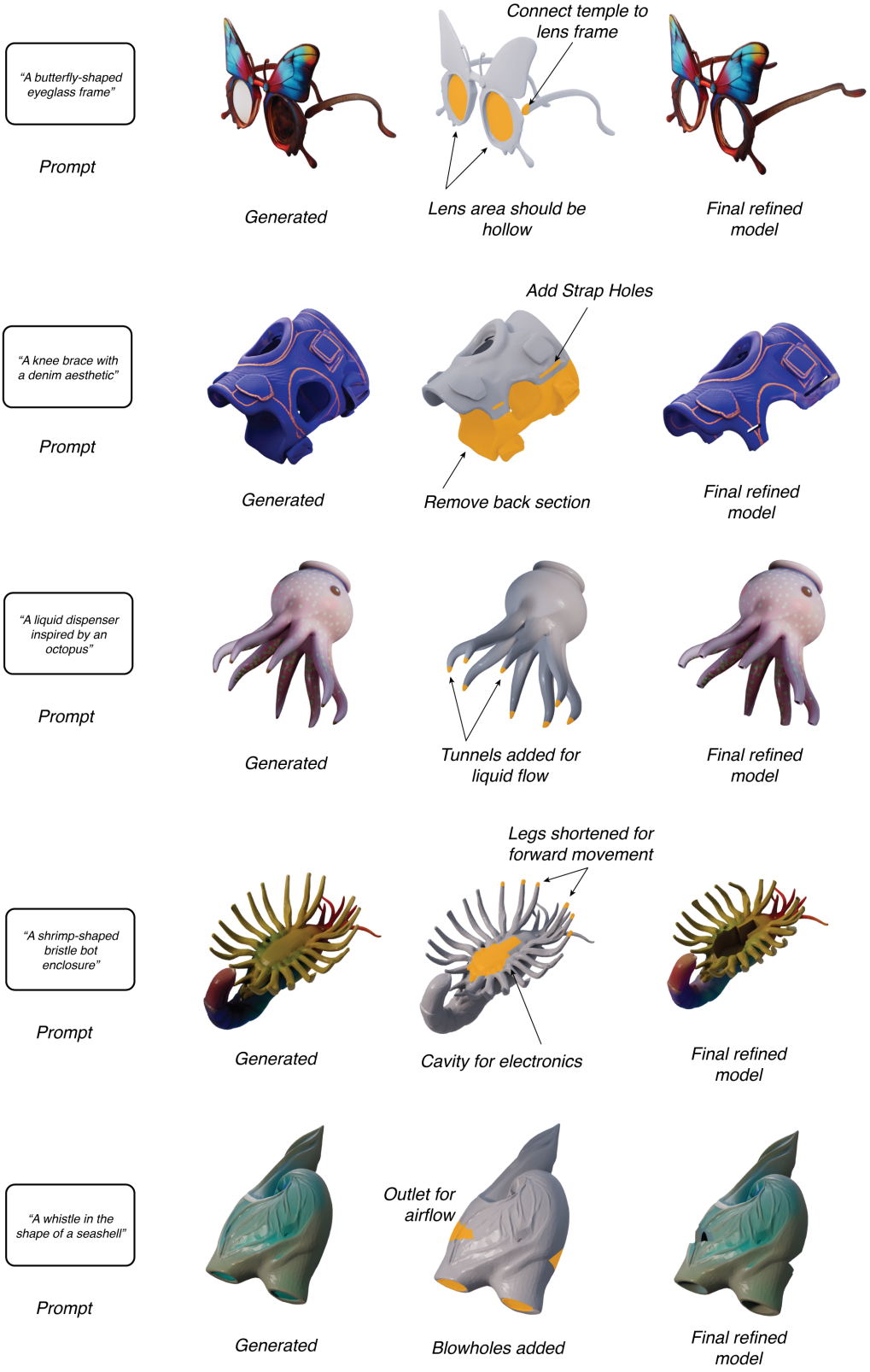

Figure 14: Detailed descriptions of application examples: For each prompt, the initial model exhibits critical flaws (e.g., missing openings, unsupported structures, lack of internal cavities) that hinder real-world use. Our system enables targeted repairs through localized canonical operations provided in natural language.