PrintAnything: Learning Geometric Plan Map for 3D Printing G-code Generation from Unoriented Point Clouds

Sangmin Hong[1], Daniel Sungho Jung[1], Heewon Kim[†3,4], and Kyoung Mu Lee[†1,2]

[1] IPAI, Seoul National University, Seoul, Korea
[2] Dept. of ECE & ASRI, Seoul National University, Seoul, Korea
[3] Soongsil University, Seoul, Korea
[4] Kairoba Inc.
{mchiash2,dqj5182,kyoungmu}@snu.ac.kr, hwkim@ssu.ac.kr

Abstract. Point clouds are one of the most fundamental and widely used 3D representations, serving as the most basic geometric representation of 3D shapes. Nevertheless, most existing 3D printing pipelines require a watertight mesh as input, preventing the direct use of point clouds for fabrication. A common workaround is to reconstruct meshes from point clouds; however, the resulting meshes often contain geometric artifacts, such as incorrect faces or topological inconsistencies, that are difficult to repair and may lead to printing failures. To overcome these limitations, we propose PrintAnything, a novel framework that learns to produce executable 3D printing G-code directly from 3D point clouds without requiring mesh reconstruction. To enable point clouds to serve as direct input for slice-wise toolpath generation, we introduce a slice-wise point projection strategy that transforms unstructured 3D point clouds into slice-aligned 2D representations consistent with layer-by-layer nature of fused deposition modeling in 3D printing. To eliminate mesh dependency and provide a unified representation that bridges point clouds and G-code, we propose Geometric plan (G-plan) map, a compact 2D representation composed of occupancy, region, and flow maps that encode the geometric and extrusion properties required for toolpath synthesis in 3D printing. As a result, our proposed method accurately generates printable G-code directly from point clouds, enabling a practical and fully mesh-free pipeline for 3D printing. The code is publicly available at https://github.com/Sangminhong/PrintAnything.

Keywords: 3D Printing · Manufacturing · G-code · Point cloud

1 Introduction

Point clouds serve as a fundamental representation of 3D geometry and frequently appear as the native output of modern sensors and cameras. They are

† co-corresponding author.

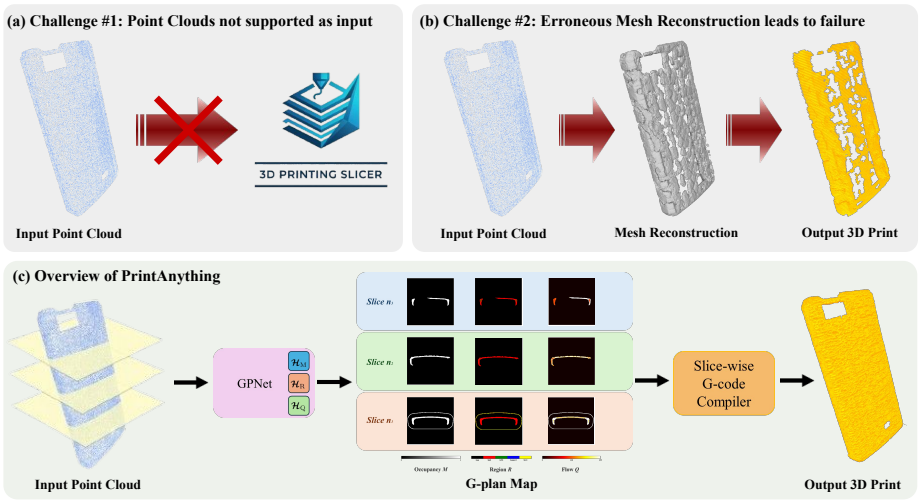


Fig. 1: Motivated by Two Challenges: (a) No Point Cloud support for 3D Printing Slicers (b) Point Cloud to Mesh reconstruction is limited and error-prone. We present **(c) PrintAnything** converts point clouds into a G-plan Map and compiles slice-wise G-code for printing.

naturally produced by LiDAR sensors [10], RGB-D cameras [9], and increasingly by 3D generative models [6,12,15,20,30,37,39,42,55]. At the same time, 3D printing has emerged as a widely adopted technology for fabricating 3D geometry due to its accessibility and effectiveness for rapid prototyping and customized manufacturing. However, the integration of point cloud and 3D printing remains limited in practice. Consequently, enabling the direct fabrication of point clouds through 3D printing presents an important yet underexplored problem.

There are two major issues that prevent point clouds from being directly used for 3D printing. First, point clouds cannot be directly taken as input by existing 3D printing pipelines. In typical fabrication workflows, a digital 3D model is converted into slice-wise toolpaths using slicing software [36], and these toolpaths are executed through G-code instructions that specify nozzle motion and material extrusion. These pipelines assume a watertight mesh as the input representation, making point clouds incompatible with standard slicing tools even though they are often the most readily available 3D signal in many acquisition pipelines. Second, converting point clouds into meshes can introduce errors that lead to printing failures. A common practice is to reconstruct meshes from point clouds [3, 8, 14, 24, 34] before slicing, but reconstructing surfaces from sparse or noisy point clouds is inherently ill-posed and frequently produces artifacts such as incorrect faces, holes, or topological inconsistencies. These defects are difficult to repair and can propagate into the slicing stage, resulting in invalid toolpaths or unstable extrusion during fabrication. The problem is particularly severe for unoriented point clouds that consist only of raw point coordinates without nor-

mals as shown in Figure 1. Consequently, the dependence on mesh both prevents unoriented point clouds from serving as direct inputs and introduces potential failure propagation in the 3D printing pipeline for point clouds.

To overcome these challenges, we propose PrintAnything, a framework that learns to generate executable 3D printing G-code directly from point clouds without requiring mesh reconstruction. To enable point clouds to be used as input for 3D printing pipelines, we introduce a slice-wise point projection strategy that converts an unstructured point cloud into slice-aligned 2D inputs encoding local geometric context around each slice height. These slice-aligned representations allow the model to reason about the geometry of each printing layer while preserving information from the original 3D point cloud. Furthermore, to eliminate failures caused by erroneous mesh reconstruction, we introduce the Geometric plan (G-plan) map, a compact slice-wise representation composed of an occupancy map, a region map, and a flow map. The occupancy map specifies which spatial locations correspond to printed material, the region map encodes structural categories required for toolpath synthesis (*e.g.,* wall or infill), and the flow map represents extrusion-related properties for material deposition as illustrated in Figure 1. Based on this representation, PrintAnything predicts G-plan maps for each slice from the input point cloud, generates appropriate infill patterns within the predicted infill regions, and finally converts the resulting maps into ordered toolpaths to produce executable G-code. Through this pipeline, PrintAnything enables a fully mesh-free approach that directly transforms point clouds into printable instructions for 3D fabrication.

In summary, our key contributions are as follows:

- We introduce PrintAnything, a framework that generates executable 3D printing G-code directly from point clouds, enabling a practical and fully mesh-free fabrication pipeline.
- To align point-cloud geometry with slice-wise printing, we propose a slice-wise point projection strategy that converts unstructured 3D points into slice-aligned 2D representations suitable for toolpath-related learning.
- To bridge point clouds and G-code with a unified representation, we propose the Geometric plan (G-plan) map, a compact slice-wise representation composed of occupancy, region, and flow maps that encode both geometric and extrusion properties required for toolpath synthesis.
- As a result, our method accurately generates printable G-code directly from point clouds without mesh reconstruction, demonstrating a robust mesh-free pipeline for 3D printing.

2 Related works

3D Printing. Recent advances in 3D printing have improved speed, accuracy, structural coherence, and robustness across diverse geometries [7, 17, 21, 31, 45, 56]. Prior work has explored printable connectors [17], large-scale model datasets [56], collision-aware wireframe printing [45], stiffness optimization for cable-driven systems [7], mobile 3D printing with sub-mm accuracy [21], and

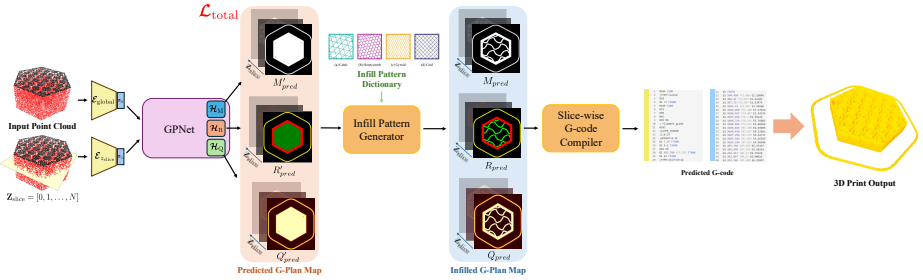


Fig. 2: The framework of PrintAnything. From input point cloud and slice indices, we encode them and pass to GPNet that predicts M, R, Q. The M, R, Q along with infill pattern dictionary are given to infill pattern generator to generate infilled M, R, Q. Lastly, we feed the infilled M, R, Q to layer-wise G-code generator to get predicted G-code which is directly converted to 3D print output.

extrusion-based fabrication for sensor manufacturing [31]. More recent studies have investigated multi-axis and learning-based slicing, including curved slice generation [53], representation-agnostic neural slicing [23], paired CAD–G-code datasets [13], reinforcement learning-based toolpath planning [11, 29], and image-to-G-code generation [44]. We use Slice-100K [13] as our main experimental dataset. Unlike existing methods, PrintAnything generates printable G-code directly from point clouds, enabling users with 3D-scanned data to obtain fabrication-ready toolpaths.

AI for Manufacturing. AI has been widely adopted in manufacturing to automate repetitive processes and exploit task-specific data. Representative directions include shape design [5, 16, 49, 50], assembly [1, 2, 26, 32, 40, 43], human monitoring and skill transfer [19, 27], visual inspection [22, 48, 51], and factory simulation [52]. Recent studies further address deployment tradeoffs in manufacturing models [47], lithography simulation and mask optimization [54], vision-based automated sewing [18], reinforcement learning for robotic task sequencing [27], representation learning for assembly state recognition [40], simulation-compatible assembly asset generation [43], industrial anomaly generation [41], and LLM adaptation for manufacturing inspection [22].

3 Method

Figure 2 illustrates the overall pipeline of PrintAnything, which consists of three stages: geometric plan map prediction, infill pattern generation, and slice-wise G-code generation. Prior to training, we construct ground-truth (GT) G-plan maps from the GT G-code and 3D model, which are then used to supervise the training of PrintAnything.

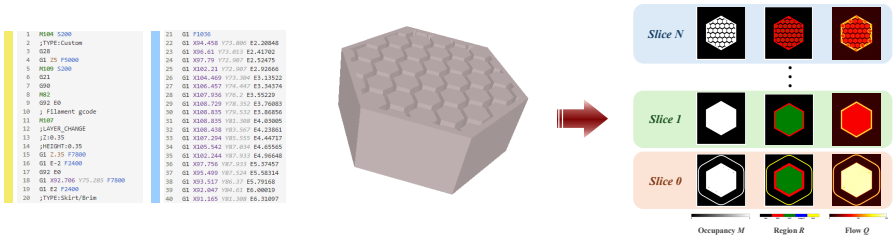


Fig. 3: GT G-plan map generation. We generate geometric plan (G-plan) map from a pair of G-code and 3D model.

3.1 GT Geometric plan (G-plan) map generation

Given a pair consisting of a 3D mesh $\mathcal{M}$ and its corresponding sliced G-code $\mathbf{G}$, we construct three geometric plan (G-plan) maps: an occupancy map $\mathbf{M}$, a region map $\mathbf{R}$, and a flow map $\mathbf{Q}$. The sliced G-code $\mathbf{G}$ encodes the toolpath coordinates of the 3D printer together with printing attributes, including structural categories (*e.g.,* `gap`, `wall`, `infill`, `support`, `skirt`) and the extrusion length along each path segment for a given slice. For each slice, we first build the region map $\mathbf{R}$ by rasterizing the structural categories specified in $\mathbf{G}$. Based on $\mathbf{R}$, we then construct the occupancy map $\mathbf{M}$, which represents the union of all printed regions. Finally, we compute the flow map $\mathbf{Q}$ by assigning normalized extrusion-length values to the corresponding coordinates. All maps are resized to a fixed resolution of 256×256 for compatibility with image-based encoder-decoder frameworks (*e.g.,* U-Net [38]).

3.2 Geometric plan map prediction

Given an input point cloud $\mathbf{P} \in \mathbb{R}^{N\times 3}$, our model predicts G-plan maps defined in Section 3.1. For each slice indexed by $Z_{\text{slice}} \in \{0, 1, \ldots, N\}$, we estimate an occupancy map M'_{pred}, region map R'_{pred}, and flow map Q'_{pred}. We denote the normalized slice height as $\tilde{Z}_{\text{slice}} = \frac{Z_{\text{slice}}}{N} \in [0, 1]$. We first encode the input point cloud using a point-cloud encoder $\mathcal{E}_{\text{global}}$ (*i.e.,* Point Transformer V3 [46]), producing a global feature: $\mathbf{z}_{\text{g}} = \mathcal{E}_{\text{global}}(\mathbf{P}) \in \mathbb{R}^{D}$. For each slice, we construct a slice-aligned 2D input by projecting the point cloud onto the printer XY grid at height $\tilde{Z}_{\text{slice}}$. Points satisfying $|z - \tilde{Z}_{\text{slice}}| \leq \Delta z/2$ are selected, together with adjacent bins above and below to provide local context which we call Multi-Slice Conditioning (MSC). We rasterize these points into an $H \times W$ grid and compute, for each bin, an occupancy mask, a density map, and a height-offset map. Concatenating these channels yields a tensor: $\mathbf{H}_{Z_{\text{slice}}} \in \mathbb{R}^{C\times H\times W}$. The normalized slice height is embedded using an MLP: $\mathbf{z}_{z_{\text{s}}} = \mathcal{E}_{z_{\text{slice}}}(\tilde{Z}_{\text{slice}})$. Based on global feature $\mathbf{z}_{\text{g}}$ and embedded slice height $\mathbf{z}_{z_{\text{s}}}$, we build slice conditioning vector as $\mathbf{h}_{Z_{\text{slice}}} = [\,\mathbf{z}_{\text{g}}; \mathbf{z}_{z_{\text{s}}}\,]$. Then, we use a U-Net [38] architecture that consists of an encoder and decoder (GPNet). The encoder processes the slice input as:

$\mathbf{X}_{z_{\text{slice}}} = \mathcal{E}_{\text{GPNet}}(\mathbf{H}_{z_{\text{slice}}})$. The conditioning vector $\mathbf{h}_{Z_{\text{slice}}}$ is injected into the bottleneck using FiLM [35]:

$$\mathbf{X}'_{z_{\text{slice}}} = \mathbf{X}_{z_{\text{slice}}} \odot (1 + \gamma(\mathbf{h}_{z_{\text{slice}}})) + \beta(\mathbf{h}_{z_{\text{slice}}}). \tag{1}$$

In the end, the decoder produces feature maps: $\mathbf{Y}_{z_{\text{slice}}} = \mathcal{D}_{\text{GPNet}}(\mathbf{X}'_{z_{\text{slice}}})$. Lastly, the final G-plan map are predicted using 1×1 convolution heads:

$$M'_{\text{pred}},\ R'_{\text{pred}},\ Q'_{\text{pred}} = \mathcal{H}_{\text{M}}(\mathbf{Y}_{z_{\text{slice}}}),\ \mathcal{H}_{\text{R}}(\mathbf{Y}_{z_{\text{slice}}}),\ \mathcal{H}_{\text{Q}}(\mathbf{Y}_{z_{\text{slice}}}). \tag{2}$$

All maps are predicted at resolution 256×256.

3.3 Infill pattern generation

Our goal is to choose an infill pattern p and scale s that balance part strength and printing cost. Rather than learning infill geometry from scratch, we leverage a small library of hand-designed templates (e.g., grid, cubic, honeycomb, gyroid) and train a lightweight recommender using automatically generated supervision. We construct a bootstrap dataset by repeatedly sampling a training shape and randomly selecting a template pattern p and a continuous scale parameter s. For each sampled pair (p, s), we warp the corresponding template to the target layer resolution using periodic wrapping so the pattern tiles seamlessly, and insert it only inside the infill-designated regions of the predicted region map $\mathbf{R}$, while keeping structural regions (perimeter/support/skirt) unchanged. This produces a synthesized G-plan map M_{pred}, R_{pred}, and Q_{pred} for the chosen (p, s). We then compute two inexpensive proxy objectives from the synthesized maps: a strength proxy S and a cost proxy C. The strength proxy increases with (i) the infill ratio within occupied regions, (ii) infill connectivity measured by the largest connected-component fraction, and (iii) directional balance; the cost proxy increases with material usage and infill boundary transitions, which serve as a simple proxy for path complexity and printing overhead. Repeating this procedure yields many labeled examples of the form

$$(\phi(M'_{\text{pred}}, R'_{\text{pred}}, Q'_{\text{pred}}),\ p,\ s) \ \longrightarrow\ (S,\ C), \tag{3}$$

where $\phi(M'_{\text{pred}}, R'_{\text{pred}}, Q'_{\text{pred}})$ denotes simple features computed from the predicted G-plan maps.

Using this dataset, we fit a small surrogate recommender with two regressors that predict $\hat{S}(p, s)$ and $\hat{C}(p, s)$ from the candidate choice. At inference time, we enumerate a discrete candidate set $\{(p_i, s_i)\}$, predict $(\hat{S}_i, \hat{C}_i)$ for each candidate, and retain non-dominated solutions that form the Pareto frontier (higher predicted strength with lower predicted cost). We select the final policy from this frontier and generate the final infill by warping and inserting the corresponding template into the predicted G-plan maps.

3.4 Slice-wise G-code generation

Given the infilled G-plan maps M_{pred}, R_{pred}, and Q_{pred}, we generate G-code by converting each slice into ordered toolpaths on the printer XY plane. Each slice provides the pixel-to-millimeter scale `px_mm`, the slice height, and the raster origin in physical coordinates. Before conversion, we center the slice by placing the raster origin at the center of the print bed. For a pixel location $\mathbf{u} = (u_x, u_y)$, the physical coordinate at the pixel center is:

$$(x, y) = \big((u_x + 0.5)\,\texttt{px\_mm} + x_0,\ (u_y + 0.5)\,\texttt{px\_mm} + y_0\big), \tag{4}$$

where (x_0, y_0) is the centered origin offset in millimeters. We generate perimeter and infill toolpaths from the occupancy mask M_{pred}, where the region map R_{pred} determines the structural type of each area (*e.g.,* wall, infill), and compute extrusion for each segment from its physical length, printing parameters, and a flow value sampled from Q_{pred}. For example, with `px_mm` $= 0.5$ mm, slice height **0.20** mm, line width 0.45 mm, centered origin $(100, 100)$ mm, filament diameter 1.75 mm, and $Q_{pred} = 1.0$, three boundary pixels $(20, 40) \to (60, 40) \to (60, 80)$ (labeled as **wall** in R_{pred}) map to $(\mathbf{110.25}, \mathbf{120.25})$, $(130.25, 120.25)$, and $(130.25, 140.25)$ mm, computed by $(20{+}0.5)\times 0.5 + 100 = 110.25$ mm and $(40{+}0.5)\times 0.5 + 100 = 120.25$ mm for $(20, 40)$. Each segment has length 20 mm, yielding a deposited volume of $0.45 \times 0.20 \times 20 = 1.8$ mm^3, which corresponds to an extrusion length of $1.8 \div \left(\pi(1.75/2)^2\right) \approx \mathbf{0.75}$ mm. Accumulating extrusion along consecutive segments produces G-code such as:

Example G-code

```
; Perimeter (wall)
G1 X110.25 Y120.25 Z0.20 E0.75
G1 X130.25 Y120.25 Z0.20 E1.50
G1 X130.25 Y140.25 Z0.20 E1.05
```

which move the nozzle along connected segments while depositing material. Processing all slices in increasing height order produces the final slice-wise G-code, where M_{pred} determines printed regions, R_{pred} specifies structural roles, and Q_{pred} controls spatially varying extrusion.

3.5 Final outputs and loss functions

During training, the model predicts G-plan maps M'_{pred}, R'_{pred}, and Q'_{pred} for each slice indexed by Z_{slice}, which are supervised using the ground-truth maps M, R, and Q described in Section 3.1. We optimize a multi-task objective that combines losses for occupancy, region classification, and flow estimation. The occupancy loss encourages accurate prediction of printed versus empty regions. We use binary cross-entropy (BCE) together with a Dice [28] loss:

$$\mathcal{L}_M = \text{BCE}(M'_{pred}, M) + \text{DICE}(M'_{pred}, M). \tag{5}$$

The region loss supervises the structural labels using weighted cross-entropy:

$$\mathcal{L}_R = \mathrm{WCE}(R'_{pred}, R), \tag{6}$$

where WCE denotes weighted cross-entropy. The flow loss measures the error of the predicted extrusion flow only in printed regions. We use a masked Huber [4] loss:

$$\mathcal{L}_Q = \mathrm{Huber}\big(Q'_{pred}, Q; M\big)\,, \tag{7}$$

where the mask M restricts the loss to printed regions. The final training objective is the weighted sum of the three losses, where λ_M, λ_R, and λ_Q are scalar weights that balance the contribution of each term:

$$\mathcal{L} = \lambda_M \mathcal{L}_M + \lambda_R \mathcal{L}_R + \lambda_Q \mathcal{L}_Q. \tag{8}$$

4 Implementation details

We implement PrintAnything in PyTorch [33] and train with AdamW [25] using mixed precision. We sample N=30,000 surface points per shape and normalize them to $[-1, 1]$. All G-plan maps are rasterized and predicted at a fixed resolution of 256×256 per slice. Following Section 3, we encode the input point cloud with a global point-cloud encoder E_{global} (Point Transformer V3 [46]), producing a global feature $\mathbf{z}_g$. For each slice, we build a slice-aligned 2D input by projecting points near the target height including adjacent bins above and below for local context onto the printer XY grid. We embed the normalized slice height $\tilde{z} \in [0, 1]$ with an MLP and concatenate it with $\mathbf{z}_g$ to form the slice conditioning vector. We use a U-Net [38] backbone for GPNet to process the slice input, inject the conditioning at the bottleneck via FiLM [35], and predict occupancy, region, and flow maps using separate 1×1 convolution heads. We train for 50 epochs with learning rate 2×10^{-4} and weight decay 10^{-4}. All experiments are conducted on a single NVIDIA Quadro RTX 8000 GPU. For all generated G-code, we use PrusaSlicer [36] to simulate the printing process and visualize the 3D prints.

5 Experiments

5.1 Datasets

We conduct our experiments on Slice-100K [13], a large-scale dataset for learning printing-oriented 3D representations. Each sample contains a 3D shape together with manufacturing supervision, including its CAD model (STL) and the corresponding ground-truth G-code produced by a standard slicing pipeline. We randomly partition the shapes into training and test sets with a 9:1 ratio and report mean performance on the held-out test set.

As input, we represent each shape as a surface point cloud: we uniformly sample 30k points and normalize coordinates to the range $[-1, 1]$. For supervision and model outputs, we construct layer-wise G-plan maps by rasterizing the ground-truth G-code into occupancy, region, and flow maps at the target slice resolution, which are used to train and evaluate our model.

5.2 Evaluation Metrics

In all experiments, we report the average performance over the test set of Slice-100K [13] dataset. For evaluating 3D geometry, our main evaluation metric is Chamfer Distance (CD) and F1-score ($F1_{3D}$). Let $\mathcal{P}$ denote points sampled from the predicted shape and $\mathcal{G}$ denote points sampled from the ground-truth (GT) mesh. We evaluate with Chamfer Distance (CD) defined as :

$$\mathrm{CD}(\mathcal{P},\mathcal{G}) = \frac{1}{|\mathcal{P}|}\sum_{p\in\mathcal{P}}\min_{g\in\mathcal{G}}\|p-g\|_2 + \frac{1}{|\mathcal{G}|}\sum_{g\in\mathcal{G}}\min_{p\in\mathcal{P}}\|g-p\|_2. \tag{9}$$

We also report F1 score for 3D points after setting a distance threshold $\tau = 1\,\mathrm{mm}$:

$$F1_{3D} = \frac{2PR}{P+R}, \tag{10}$$

where P and R denote precision and recall computed between $\mathcal{P}$ and $\mathcal{G}$. To evaluate slice-wise structural accuracy, we rasterize slices from the GT G-code and compute a binary occupancy F1 score per each rasterized slices:

$$F1_{2D} = \frac{2\,\mathrm{P}_{2D}\,\mathrm{R}_{2D}}{\mathrm{P}_{2D}+\mathrm{R}_{2D}}, \tag{11}$$

where P_{2D} and R_{2D} denote precision and recall of predicted slice occupancy. The final score is obtained by averaging across slices and samples. For the ablation studies on G-plan map design, we further evaluate extrusion-flow properties derived from the generated G-code. We consider only extruding moves ($\Delta E_k > 0$) with non-zero XY motion. For each move, we define the planar travel length $\Delta s_k = \|(x_k, y_k) - (x_{k-1}, y_{k-1})\|_2$ and the extrusion density $\rho_k = \frac{\Delta E_k}{\Delta s_k + \epsilon}$. We measure extrusion smoothness using:

$$\Delta\rho\text{-smooth} = \frac{1}{N-1}\sum_{k=2}^{N}\left|\rho_k - \rho_{k-1}\right|, \tag{12}$$

which captures local variation of extrusion density along toolpaths. We also evaluate per-slice extrusion consistency by computing the coefficient of variation of extrusion density within each slice:

$$\mathrm{CV}_z = \frac{\mathrm{std}(\rho^{(z)})}{|\mathrm{mean}(\rho^{(z)})| + \epsilon}, \qquad \rho\text{-CV} = \frac{1}{Z'}\sum_{z=1}^{Z'}\mathrm{CV}_z, \tag{13}$$

where $\rho^{(z)}$ denotes the extrusion densities of moves within slices z, and Z' is the number of slices containing at least one extruding move. Lower values indicate smoother and more consistent extrusion behavior.

5.3 Comparison with state-of-the-art methods

As noted in section 1, a natural and widely adopted baseline for printing from point observations is to first convert the input into a mesh and then rely on a

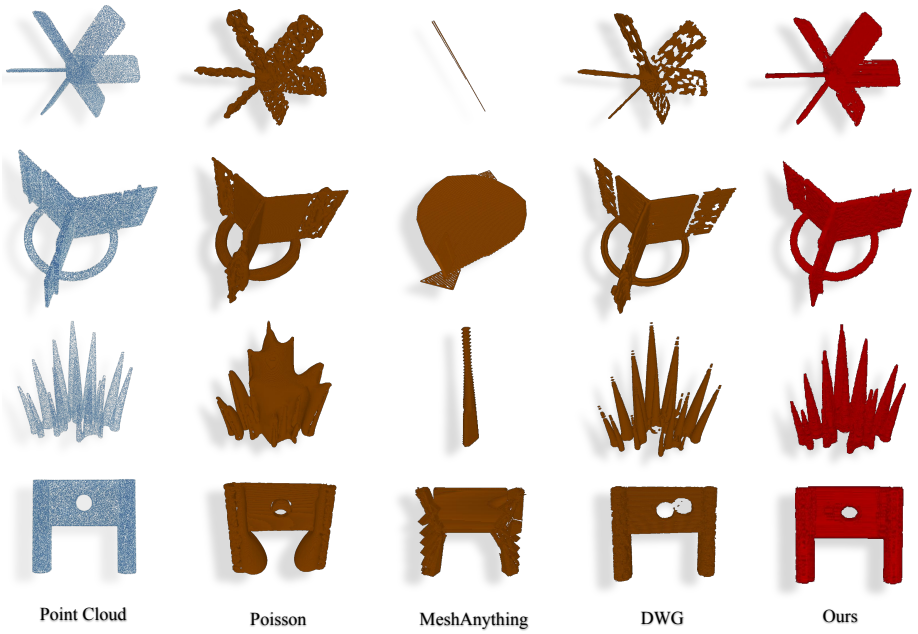


Fig. 4: Qualitative Results on Slice-100K [13] dataset.

mature slicer to produce G-code. Following this principle, we compare against strong mesh-conversion pipelines built on classical Poisson surface reconstruction [14] and recent learning-based methods, DWG [24] and MeshAnything [3]. For each baseline, we reconstruct a mesh from the input point cloud and feed the resulting mesh into PrusaSlicer [36] to generate the final toolpath, yielding a standard "point cloud → mesh → slicer → G-code" workflow.

Table 1 summarizes the results on Slice-100K [13]. Our method achieves the best performance across all metrics, reaching 0.047 CD, 0.7414 $F1_{3D}$, and 0.677 $F1_{2D}$. In comparison, mesh-based pipelines tend to accumulate errors across stages: small geometric artifacts introduced during reconstruction can be amplified after slicing, which is reflected most clearly in the lower slice-level scores.

Table 1: Quantitative comparison of various point-to-G-code prediction techniques on Slice-100K [13] dataset.

Representation	Method	CD ↓	$F1_{3D}$↑	$F1_{2D}$↑
Mesh	Poisson [14]	0.088	0.682	0.587
	DWG [24]	0.062	0.712	0.496
	MeshAnything [3]	0.157	0.480	0.356
G-plan map	PrintAnything (Ours)	**0.047**	**0.741**	**0.677**

We further visualize output 3D prints in Figure 4. Poisson reconstruction often smooths away thin structures, while learning-based mesh methods may introduce broken surfaces or local clutter, which can translate into missing or fragmented toolpaths after slicing. In contrast, our approach produces more coherent and complete print structures, particularly on shapes with slender parts and sharp features, leading to toolpaths that better match the target slices. Overall, these quantitative and qualitative results indicate that representing the input with our G-plan map yields more reliable, slice-faithful toolpaths than mesh-based conversion pipelines followed by a conventional slicer.

Table 2: Ablation of multi-slice conditioning (MSC) on Slice-100K [13].

Multi-slice conditioning	CD ↓	$F1_{3D}$ ↑	$F1_{2D}$ ↑
✗	0.059	0.702	0.652
✓	**0.047**	**0.741**	**0.677**

5.4 Multi-slice conditioning

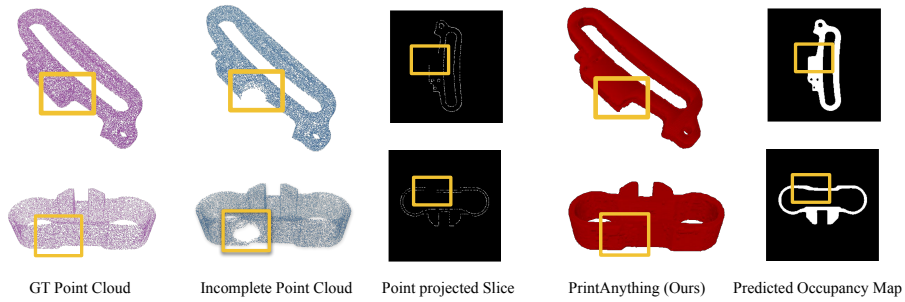
Table 2 shows the impact of multi-slice conditioning (MSC) on Slice-100K [13]. Enabling MSC notably improves geometric accuracy: CD drops from 0.059 to 0.047, a 20.3% reduction. It also improves 3D shape fidelity, with $F1_{3D}$ increasing from 0.702 to 0.741, which is a 5.6% improvement. These gains suggest that conditioning on multiple neighboring slices helps the model resolve slice-wise ambiguities. By enforcing inter-slice continuity and structural support cues, our PrintAnything leads to more consistent toolpath decisions than relying only on a single-slice context.

Table 3: Comparison of our recommender against random-scale/fixed-scale baselines on Slice-100K [13].

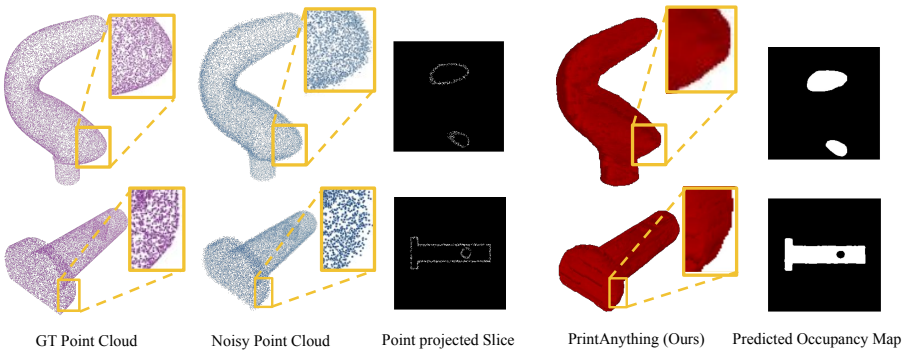
Method	Strength ↑	Cost ↓	Combined↑
Random Pattern + Random Scale	0.294	38,808	0.757
Cubic + Random Scale	0.298	36,994	0.806
Grid + Random Scale	**0.310**	41,535	0.748
Gyroid + Random Scale	0.271	42,247	0.643
Honeycomb + Random Scale	0.294	<u>35,445</u>	0.830
Random Pattern + Fixed Scale (s=1.0)	0.300	35,866	<u>0.837</u>
Ours (Recommender)	<u>0.308</u>	**32,974**	**0.937**

5.5 Infill policy recommendation

Table 3 compares our infill policy recommender with random-scale and fixed-scale baselines across a diverse set of infill patterns. We evaluate each method



(a) Qualitative Results for Incomplete Point cloud



(b) Qualitative Results for Noisy Point cloud

Fig. 5: Qualitative Results on incomplete and noisy point cloud input on Slice-100K [13]. Yellow boxes highlight regions with missing observations for incomplete input. For the noisy case, enlarged insets are provided for clearer comparison.

using three criteria: a proxy for structural strength, a proxy for fabrication cost, and a combined score that balances the two. Concretely, the strength proxy measures (i) the infill material ratio within occupied regions, (ii) the connectivity of the infill (largest connected-component ratio), and (iii) a direction-balance term that favors well-distributed infill boundaries, while the cost proxy increases with total occupied area and the number of infill boundary transitions. Detailed computation of these measures is included in the Supplementary Details. We compute the combined score as $\text{Combined} = 10^5 \cdot \text{Strength}/\text{Cost}$.

Among the baselines, random-scale strategies can occasionally achieve competitive strength. For example, Grid + Random Scale attains the highest strength value of 0.310, but it incurs a high cost of 41,535 and therefore yields a low combined score of 0.748. In contrast, our recommender achieves a strong strength value of 0.308 while producing the lowest fabrication cost of 32,974. As a result, it obtains the best combined score of 0.937. Overall, these results indicate that our recommender learns to select infill strategies that better balance the strength–cost trade-off, leading to more efficient yet reliable fabrication outcomes.

5.6 Robustness to scan artifacts

In practical settings, point clouds acquired from commodity depth sensors or 3D scanners are often affected by occlusions and measurement noise. To encourage robustness under such scan artifacts, we additionally apply two augmentations during training: structured hole dropout, which removes points inside a contiguous 3D spherical region to mimic occlusion, and additive Gaussian noise with standard deviation 0.01 in normalized coordinates.

Figure 5 shows qualitative results for incomplete point clouds with a missing circular portion of the geometry, a common failure mode caused by sensor-object occlusion. Despite the missing observations, our method recovers a complete printable shape, producing stable occupancy predictions and coherent toolpaths. Figure 5 presents a noisy input case. Even when the point cloud is corrupted by noise, the projected slices remain stable, and both the predicted occupancy map and the generated G-code preserve the intended structure without introducing spurious artifacts. Overall, these results indicate that our pipeline is robust to realistic scan imperfections and can reliably generate printable toolpaths from imperfect point observations.

Table 4: Ablation of representations on Slice-100K [13].

M	R	Q	CD ↓	$F1_{3D}$ ↑	$F1_{2D}$ ↑	$\Delta\rho$-smooth ↓	ρ-CV ↓
✓	✗	✗	0.052	0.709	0.635	0.082	<u>1.055</u>
✓	✓	✗	**0.046**	<u>0.728</u>	<u>0.675</u>	<u>0.081</u>	1.059
✓	✓	✓	<u>0.047</u>	**0.741**	**0.677**	**0.035**	**0.909**

5.7 Ablations on G-plan maps design

We analyze the design of the proposed geometric plan (G-plan) map by incrementally adding its components: **M**, **R**, **Q**. Specifically, we compare three configurations: using only the occupancy map **M** (first row), using occupancy map **M** and region map **R** (second row), and using the full representation with the flow map **Q** (third row). Table 4 reports the results on Slice-100K [13]. Using only **M** provides a coarse printable geometry, but it cannot distinguish between structurally different regions that require different deposition behaviors. Region map **R** improves geometric fidelity, reducing CD by 11.5% and increasing $F1_{3D}$ by 2.7%. This improvement suggests that region-level cues help disambiguate toolpath assignment around boundaries and interior supports. Adding the flow map **Q** further improves flow-related metrics of the predicted toolpaths. In particular, the flow smoothness measure $\Delta\rho$-smooth decreases from 0.081 to 0.035, corresponding to a 56.8% reduction, and the extrusion variability ρ-CV decreases from 1.059 to 0.909, which is a 14.2% reduction. Overall, the results highlight that all components of our G-plan map are critical not only for geometric reconstruction but also for achieving extrusion consistency per path length during G-code generation.

5.8 Results on real-world 3D printer machines

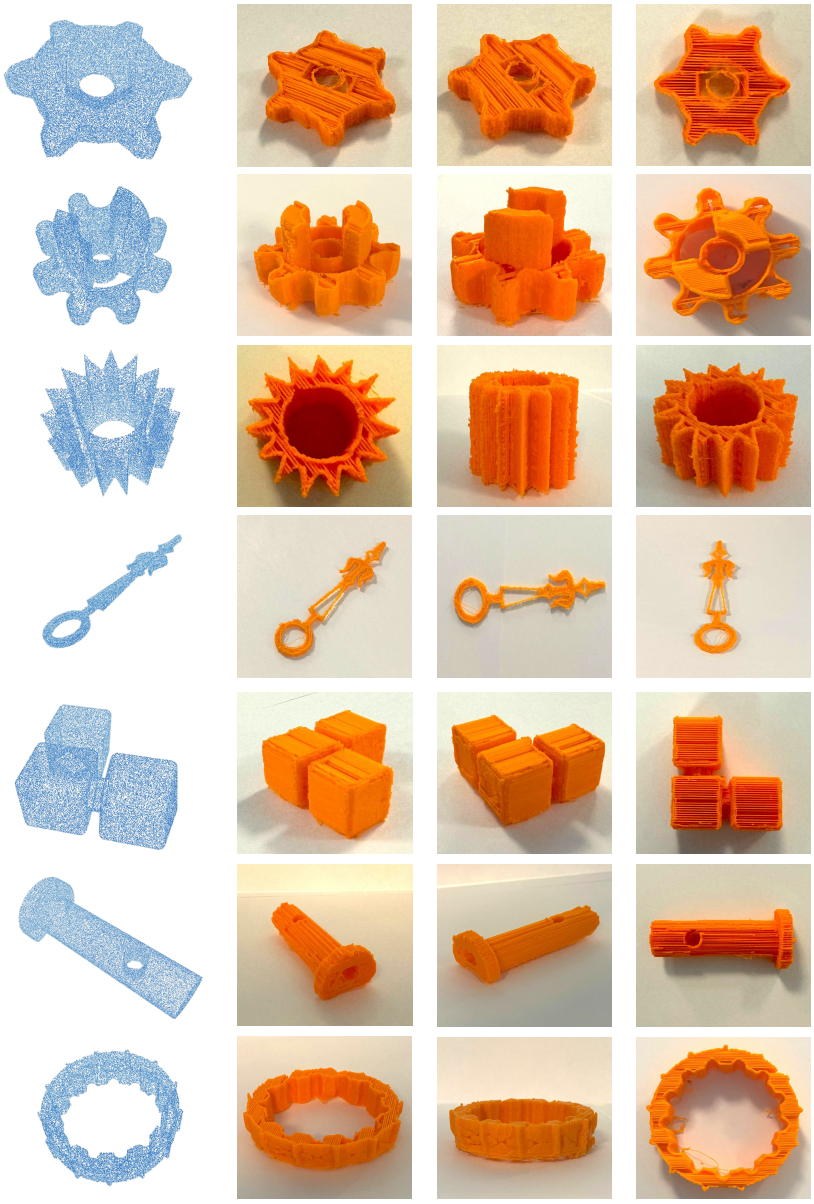
Figure 6 presents real-world 3D printing results produced directly from the G-code generated by PrintAnything. All objects were fabricated using a Bambu Lab X1-Carbon printer without any additional post-processing of the generated G-code, allowing us to evaluate the raw printing quality of our method. The results demonstrate that the predicted G-code is robust across a variety of shapes and geometric complexities. In particular, the first, second, third, and seventh rows show mechanical parts with cogwheel structures commonly used in manufacturing, where the printed outputs exhibit consistent and well-formed teeth across the entire gear. The fourth row illustrates a complex object with thin and delicate structures, for which PrintAnything still produces detailed and stable prints. We further evaluated the structural integrity of the printed objects through manual pressure tests, where the printed parts remained intact under reasonable external force. These results indicate that PrintAnything can generate reliable G-code that translates effectively into high-quality physical prints on real-world 3D printing hardware.

6 Conclusion

We present PrintAnything, a framework that generates executable 3D printing G-code directly from point clouds without requiring mesh reconstruction. To bridge point clouds and slice-wise fabrication, we introduce a slice-wise point projection strategy that converts point clouds into slice-aligned 2D representations. We further propose the Geometric plan (G-plan) map, composed of occupancy, region, and flow maps that encode the geometric and extrusion properties required for toolpath synthesis. In the end, PrintAnything predicts G-plan maps from the input point cloud and converts them into executable toolpaths to produce printable G-code, enabling a practical mesh-free fabrication pipeline. As future work, we plan to extend the framework to support more diverse printer settings and materials with physical constraints.

Acknowledgement

This work was supported in part by the IITP grants [No. RS-2021-II211343, Artificial Intelligence Graduate School Program (Seoul National University), No. RS-2024-00426853, No. RS-2025-02303870, No.2022-0-00156] funded by the Korea government (MSIT). We thank Juhyoung Lee for assistance with the physical 3D printing and sample fabrication.



Input Point Cloud **3D Printed Output by Real Machine**

Fig. 6: Visualization of 3D printed output by Real Machine.

References

1. Ben-Shabat, Y., Yu, X., Saleh, F., Campbell, D., Rodriguez-Opazo, C., Li, H., Gould, S.: The IKEA ASM Dataset: Understanding people assembling furniture through actions, objects and pose. In: WACV. pp. 847–859 (2021) 4
2. Bhatt, P.M., Kulkarni, A., Malhan, R.K., Gupta, S.K.: Optimizing part placement for improving accuracy of robot-based additive manufacturing. In: ICRA (2021) 4
3. Chen, Y., He, T., Huang, D., Ye, W., Chen, S., Tang, J., Cai, Z., Yang, L., Yu, G., Lin, G., et al.: MeshAnything: Artist-created mesh generation with autoregressive Transformers. In: ICLR 2, 10
4. Collins, J.R.: Robust estimation of a location parameter in the presence of asymmetry. The Annals of Statistics (1976) 8
5. Du, T., Inala, J.P., Pu, Y., Spielberg, A., Schulz, A., Rus, D., Solar-Lezama, A., Matusik, W.: InverseCSG: Automatic conversion of 3D models to CSG trees. ACM TOG (2018) 4
6. Du, Y., Zhao, Z., Su, S., Golluri, S., Zheng, H., Yao, R., Wang, C.: SuperPC: A single diffusion model for point cloud completion, upsampling, denoising, and colorization. In: CVPR (2025) 2
7. Gueners, D., Chanal, H., Bouzgarrou, B.: Stiffness optimization of a cable driven parallel robot for additive manufacturing. In: ICRA (2020) 3
8. Hanocka, R., Metzer, G., Giryes, R., Cohen-Or, D.: Point2Mesh: A self-prior for deformable meshes. ACM TOG (2020) 2
9. Hong, S., Lee, S., Lee, K.M.: StarryGazer: Leveraging monocular depth estimation models for domain-agnostic single depth image completion. arXiv preprint arXiv:2512.13147 (2025) 2
10. Hong, S., Yavartanoo, M., Neshatavar, R., Lee, K.M.: ACL-SPC: Adaptive closed-loop system for self-supervised point cloud completion. In: CVPR (2023) 2
11. Huang, Y., Guo, Y., Su, R., Han, X., Ding, J., Zhang, T., Liu, T., Wang, W., Fang, G., Song, X., et al.: Learning based toolpath planner on diverse graphs for 3D printing. ACM TOG (2024) 4
12. Huang, Z., Johnson, J., Debnath, S., Rehg, J.M., Wu, C.Y.: PointInfinity: Resolution-invariant point diffusion models. In: CVPR (2024) 2
13. Jignasu, A.N., Marshall, K., Mishra, A.K., Nerone Rillo, L., Ganapathysubramanian, B., Balu, A., Hegde, C., Krishnamurthy, A.: Slice-100K: A multimodal dataset for extrusion-based 3D printing. NeurIPS (2024) 4, 8, 9, 10, 11, 12, 13
14. Kazhdan, M., Bolitho, M., Hoppe, H.: Poisson surface reconstruction. In: SGP (2006) 2, 10
15. Kim, J., Yoo, J., Lee, J., Hong, S.: SetVAE: Learning hierarchical composition for generative modeling of set-structured data. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 15059–15068 (2021) 2
16. Koch, S., Matveev, A., Jiang, Z., Williams, F., Artemov, A., Burnaev, E., Alexa, M., Zorin, D., Panozzo, D.: ABC: A big CAD model dataset for geometric deep learning. In: CVPR (2019) 4
17. Koyama, Y., Sueda, S., Steinhardt, E., Igarashi, T., Shamir, A., Matusik, W.: AutoConnect: Computational design of 3D-printable connectors. ACM TOG (2015) 3
18. Ku, S., Choi, H., Kim, H.Y., Park, Y.L.: Automated sewing system enabled by machine vision for smart garment manufacturing. RA-L (2023) 4

19. Kubota, A., Iqbal, T., Shah, J.A., Riek, L.D.: Activity recognition in manufacturing: The roles of motion capture and sEMG+ inertial wearables in detecting fine vs. gross motion. In: ICRA (2019) 4
20. Lee, J.J., Benes, B.: RGB2Point: 3D point cloud generation from single RGB images. In: WACV (2025) 2
21. Li, J., Aubin-Fournier, P.L., Skonieczny, K.: SLAAM: Simultaneous localization and additive manufacturing. T-RO (2020) 3
22. Li, Y., Yuan, S., Wang, H., Li, Q., Liu, M., Xu, C., Shi, G., Zuo, W.: Triad: Empowering LMM-based anomaly detection with expert-guided region-of-interest tokenizer and manufacturing process. In: ICCV (2025) 4
23. Liu, T., Zhang, T., Chen, Y., Huang, Y., Wang, C.C.: Neural slicer for multi-axis 3D printing. ACM TOG (2024) 4
24. Liu, W., Li, J., Chen, X., Hou, F., Xin, S., Wang, X., Wu, Z., Qian, C., He, Y.: Diffusing winding gradients (DWG): A parallel and scalable method for 3D reconstruction from unoriented point clouds. ACM TOG (2025) 2, 10
25. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. In: ICLR (2019) 8
26. Malhan, R.K., Kabir, A.M., Shah, B., Gupta, S.K.: Identifying feasible workpiece placement with respect to redundant manipulator for complex manufacturing tasks. In: ICRA (2019) 4
27. Manyar, O.M., McNulty, Z., Nikolaidis, S., Gupta, S.K.: Inverse reinforcement learning framework for transferring task sequencing policies from humans to robots in manufacturing applications. In: ICRA (2023) 4
28. Milletari, F., Navab, N., Ahmadi, S.A.: V-Net: Fully convolutional neural networks for volumetric medical image segmentation. In: 3DV (2016) 7
29. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing Atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013) 4
30. Mo, S., Xie, E., Wu, Y., Chen, J., Nießner, M., Li, Z.: Fast training of diffusion transformer with extreme masking for 3D point clouds generation. In: ECCV (2024) 2
31. Munasinghe, N., Masangkay, J., Paul, G.: Temperature compensated 3D printed strain sensor for advanced manufacturing applications. In: ICRA (2021) 3, 4
32. Owan, P., Garbini, J., Devasia, S.: Faster confined space manufacturing teleoperation through dynamic autonomy with task dynamics imitation learning. RA-L (2020) 4
33. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: An imperative style, high-performance deep learning library. In: NeurIPS (2019) 8
34. Peng, S., Jiang, C., Liao, Y., Niemeyer, M., Pollefeys, M., Geiger, A.: Shape as points: A differentiable poisson solver. NeurIPS (2021) 2
35. Perez, E., Strub, F., De Vries, H., Dumoulin, V., Courville, A.: FiLM: Visual reasoning with a general conditioning layer. In: AAAI (2018) 6, 8
36. Prusa Research a.s.: PrusaSlicer. https://www.prusa3d.com/page/prusaslicer_424/ (2026) 2, 8, 10
37. Ren, Z., Kim, M., Liu, F., Liu, X.: TIGER: Time-varying denoising model for 3D point cloud generation with diffusion process. In: CVPR (2024) 2
38. Ronneberger, O., Fischer, P., Brox, T.: U-Net: Convolutional networks for biomedical image segmentation. In: MICCAI (2015) 5, 8

39. Sanghi, A., Chu, H., Lambourne, J.G., Wang, Y., Cheng, C.Y., Fumero, M., Malekshan, K.R.: CLIP-Forge: Towards zero-shot text-to-shape generation. In: CVPR (2022) 2
40. Schoonbeek, T.J., Balachandran, G., Onvlee, H., Houben, T., Hung, S.H., Kustra, J., de With, P.H., van der Sommen, F.: Supervised representation learning towards generalizable assembly state recognition. RA-L (2024) 4
41. Tong, X., Chang, Y., Zhao, Q., Yu, J., Wang, B., Lin, J., Lin, Y., Mai, X., Wang, H., Tao, Z., et al.: Component-aware unsupervised logical anomaly generation for industrial anomaly detection. In: ICRA (2025) 4
42. Vahdat, A., Williams, F., Gojcic, Z., Litany, O., Fidler, S., Kreis, K., et al.: LION: Latent point diffusion models for 3D shape generation. NeurIPS (2022) 2
43. Wang, Y., Tang, B., Gan, C., Fox, D., Mo, K., Narang, Y., Akinola, I.: MatchMaker: Automated asset generation for robotic assembly. In: ICRA (2025) 4
44. Wang, Z., Jadhav, Y., Pak, P., Farimani, A.B.: Image2Gcode: Image-to-G-code generation for additive manufacturing using diffusion-transformer model. arXiv preprint arXiv:2511.20636 (2025) 4
45. Wu, R., Peng, H., Guimbretière, F., Marschner, S.: Printing arbitrary meshes with a 5DOF wireframe printer. ACM TOG (2016) 3
46. Wu, X., Jiang, L., Wang, P.S., Liu, Z., Liu, X., Qiao, Y., Ouyang, W., He, T., Zhao, H.: Point Transformer V3: Simpler faster stronger. In: CVPR (2024) 5, 8
47. Yang, C., Wu, Z., Chee, J., De Sa, C., Udell, M.: How low can we go: Trading memory for error in low-precision training. In: ICLR (2022) 4
48. Yang, H., Zhu, H., Li, J., Chen, J., Yin, Z.: Multi-category decomposition editing network for the accurate visual inspection of texture defects. T-ASE (2023) 4
49. Yavartanoo, M., Hong, S., Neshatavar, R., Lee, K.M.: CNC-Net: Self-supervised learning for CNC machining operations. In: CVPR (2024) 4
50. Yavartanoo, M., Hong, S., Neshatavar, R., Lee, K.M.: Text2CAD: Text to 3D CAD generation via technical drawings. arXiv preprint arXiv:2411.06206 (2024) 4
51. Zhang, G., Cui, K., Hung, T.Y., Lu, S.: Defect-GAN: High-fidelity defect synthesis for automated defect inspection. In: WACV (2021) 4
52. Zhang, M., Wang, X., Decardi-Nelson, B., Song, B., Zhang, A., Liu, J., Tao, S., Cheng, J., Liu, X., Yu, D., et al.: SMPL: Simulated industrial manufacturing and process control learning environments. NeurIPS (2022) 4
53. Zhang, T., Fang, G., Huang, Y., Dutta, N., Lefebvre, S., Kilic, Z.M., Wang, C.C.: S^3-Slicer: A general slicing framework for multi-axis 3D printing. ACM TOG (2022) 4
54. Zheng, S., Yang, H., Zhu, B., Yu, B., Wong, M.: LithoBench: Benchmarking AI computational lithography for semiconductor manufacturing. NeurIPS (2023) 4
55. Zhou, C., Zhong, F., Hanji, P., Guo, Z., Fogarty, K., Sztrajman, A., Gao, H., Oztireli, C.: Frepolad: Frequency-rectified point latent diffusion for point cloud generation. In: ECCV (2024) 2
56. Zhou, Q., Jacobson, A.: Thingi10K: A dataset of 10,000 3D-printing models. arXiv preprint arXiv:1605.04797 (2016) 3