PARTCODER: RECONSTRUCTING MULTI-FILE CAD CODE ASSEMBLIES FROM POINT CLOUDS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent LLM-based CAD reverse-engineering methods recover editable CAD code from point clouds but represent entire objects as monolithic programs. This creates generation and representation bottlenecks because the model must reconstruct the complete geometry within a single script, while the output lacks the explicit part structure and part-level controllability of industrial CAD assemblies. We introduce PartCoder, the first method for reconstructing multi-file CAD code assemblies directly from point clouds. PartCoder hierarchically decomposes a dense input point cloud into geometrically coherent parts, completes newly created interfaces, reconstructs each part using a generative CAD code backbone, and combines the components through a central assembly file. It retains only decompositions that improve the overall reconstruction, adaptively selecting the number of parts and enabling existing backbones to handle more complex objects without retraining. We introduce PrIMe3D, a benchmark of 1,000 mechanical objects generated through a prompt-to-image-to-mesh pipeline for evaluating the emerging task of converting AI-generated 3D assets into editable CAD models. Averaged across three backbones, PartCoder improves mean IoU over the corresponding monolithic baselines by 1.1, 7.7, and 11.1 percentage points on Fusion360, OmniCAD, and PrIMe3D, respectively, with larger gains on more complex datasets while producing modular CAD assemblies with independently editable parts.

1 INTRODUCTION

Contemporary 3D generative models, such as Trellis 2 (Xiang et al., 2025) and Hunyuan3D (Tencent, 2025), enable the rapid creation of detailed mesh prototypes from text and images. However, these meshes capture only surface geometry and lack the structured, editable construction logic of CAD models. Converting them into editable computer aided design (CAD) representations therefore constitutes an increasingly relevant CAD reverse engineering task. While point-cloud-to-CAD methods have traditionally focused on scans of real-world objects (Liu et al., 2024), point clouds are also well suited for representing generated geometry, including cases where the underlying meshes are non-watertight or have invalid connectivity. They thus provide a common input representation for reconstructing CAD models from both scanned and AI-generated geometry.

In parallel, LLM-based methods increasingly generate editable models as executable CAD code using frameworks such as CadQuery (cadquery, 2024). These frameworks represent CAD models as programs that follow standard programming paradigms and compile into CAD objects, making them well suited to LLM-based generation. Most methods condition this process on text, sketches, or images, which support open-ended design but underconstrain the exact target geometry. When a 3D prototype is available, its geometry provides a more explicit target and turns CAD generation into a faithful reconstruction problem. CAD-Recode (Rukhovich et al., 2025) and cadrille (Kolodiazhnyi et al., 2026) extend LLM-based CAD code generation to reverse engineering by generating CAD code directly from point clouds.

However, these methods represent the entire object as a single monolithic program, creating both generation and representation bottlenecks. The underlying model must reconstruct the complete geometry within a single CAD code script, while the output lacks explicit components that can be manipulated independently. These limitations are particularly pronounced for complex objects composed of multiple simpler geometric parts.

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

In this work, we introduce PartCoder, a framework for synthesizing modular, multi-file CAD code assemblies from point clouds. PartCoder hierarchically decomposes the input geometry into coherent parts, reconstructs each part using an existing CAD code generator, and combines the resulting programs through a central assembly file. Candidate decompositions are retained only when they improve the fidelity of the complete assembly. PartCoder thus transforms a difficult program synthesis problem into an adaptively selected set of simpler ones, extending the effective complexity range of existing generative backbones without modification or retraining. The resulting representation also better reflects industrial CAD practice and supports the independent inspection, removal, modification, and repositioning of individual parts.

As an additional contribution, we also develop PrIMe3D, a benchmark of 1,000 AI-generated meshes created through a rapid prompt-to-image-to-mesh prototyping pipeline. This setup reflects the growing need to convert generated 3D assets into editable CAD models and provides more complex geometries than typical simplified CAD reverse-engineering benchmarks, better capturing real-world challenges. In summary, our contributions are as follows:

- We introduce PartCoder, the first method for reconstructing multi-file CAD code assemblies directly from point clouds.
- We propose a hierarchical part-based pipeline that decomposes objects into modular CAD programs, reducing complexity and improving controllability.
- We design PrIMe3D, a CAD reverse-engineering benchmark targeting the emerging use case of converting AI-generated meshes into editable CAD models.
- We evaluate PartCoder on Fusion360, OmniCAD, and PrIMe3D, demonstrating state-of-the-art reconstruction quality while producing part-aware CAD representations.

Code and dataset are available at `https://anonymous.4open.science/r/PartCoder`.

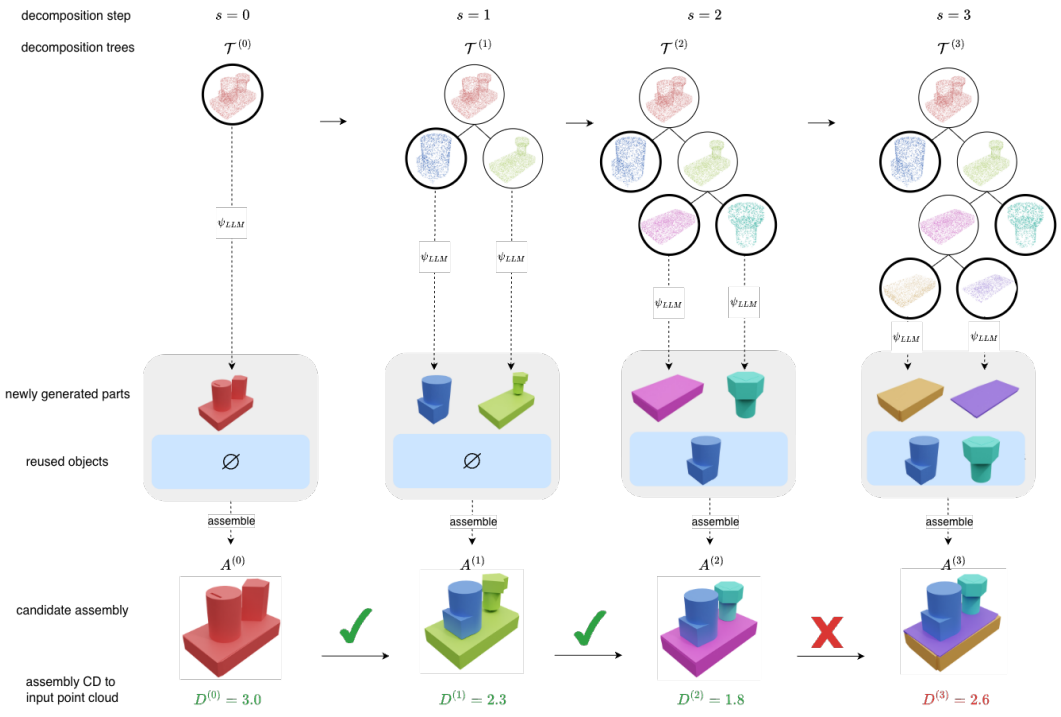


Figure 1: **Overview of PartCoder.** At each decomposition step s, one leaf (bold) of $\mathcal{T}^{(s)}$ is split into two child parts; only their CAD codes are generated, while unchanged parts are cached and reused. Candidate assemblies are evaluated against the input point cloud (red) using Chamfer distance. Non-improving decomposition steps are rejected, and the previous assembly is returned.

2 Related Work

2.1 CAD Reverse Engineering

CAD reverse engineering aims to recover editable CAD models from 3D representations such as point clouds (Rukhovich et al., 2025). Existing approaches primarily follow two directions: fitting predefined geometric primitives and reconstructing sketch-and-extrude operation sequences.

Primitive-fitting methods approximate the input geometry using predefined elements. Some recover boundary geometry by fitting surfaces (Li et al., 2019; Sharma et al., 2020; Yan et al., 2021; Guo et al., 2022; Liu et al., 2024) or edges (Wang et al., 2020; Liu et al., 2021; Zhu et al., 2023), whereas others reconstruct solids by combining volumetric Constructive Solid Geometry (CSG) primitives (Sharma et al., 2018; Du et al., 2018; Kania et al., 2020; Ren et al., 2021; Yu et al., 2022; 2023). Although CSG provides a compact CAD representation, it differs from modern CAD workflows, which are typically expressed as sequences of modeling operations.

Sequence-based methods more closely reflect these workflows. DeepCAD (Wu et al., 2021) introduced a widely adopted sketch-and-extrude representation whose sequential structure naturally aligns with transformer-based architectures, as employed by several methods (Wu et al., 2021; Dupont et al., 2024; Khan et al., 2024). Further advancements extend these generative frameworks by introducing self-supervised (Li et al., 2023) and unsupervised learning paradigms (Li et al., 2024), or by incorporating explicit sketch constraints (Karadeniz et al., 2025). CADFit (Nehme et al., 2026) instead reconstructs editable CAD sequences from watertight meshes through iterative parametric fitting and geometric validation.

Recently, LLM-based synthesis of CAD code has emerged as a third direction and has been extended to CAD reverse engineering. CAD-Recode (Rukhovich et al., 2025) trains a small language model to generate CAD code directly from point clouds, while cadrille (Kolodiazhnyi et al., 2026) extends this paradigm through multimodal conditioning and reinforcement learning. CADReasoner (Kabisov et al., 2026) employs iterative self-editing but relies on render-based visual feedback, requiring an input mesh. Despite these advances, existing code-based reverse-engineering methods reconstruct each object as a monolithic CAD program, leaving part-level assembly structure unaddressed.

2.2 LLM-based CAD Code Generation

Representing CAD models as executable code provides a natural interface for applying LLMs to CAD generation. OpenECAD (Yuan et al., 2024c) generates code using pythonOCC (Paviot, 2022), while Query2CAD (Badagabettu et al., 2024) generates executable macros for FreeCAD (FreeCAD, 2026). Other approaches instead employ custom CAD languages and command sequences (Xu et al., 2024; Li et al., 2025a; Zhang et al., 2025; Wang et al., 2025; Li et al., 2026b). More recently, CadQuery (cadquery, 2024) has become a common choice due to its Pythonic structure and close alignment with standard code-synthesis tasks (Alrashedy et al., 2025).

Initial approaches directly synthesize CAD code from text, sketches, or images using LLMs and VLMs (Picard et al., 2023; Yuan et al., 2024c; Li et al., 2025b; Doris et al., 2026; Xie & Ju, 2025). Subsequent approaches introduce more sophisticated pipelines, including visual self-refinement (Yuan et al., 2024b; Badagabettu et al., 2024; Chen et al., 2025; Li et al., 2026b), question answering (Alrashedy et al., 2025), and evolutionary optimization (Preintner et al., 2025). Other approaches employ agentic systems, including multi-agent frameworks (Ocker et al., 2025; Barkley et al., 2026), proactive agents that ask user feedback (Yuan et al., 2026), and tool-using agents (Mallis et al., 2025; Gong et al., 2026).

Complementary work improves the underlying generator through supervised fine-tuning (Guan et al., 2026) and reinforcement learning (Li et al., 2026a; Niu et al., 2026b;a). Further training strategies introduce primitive-aware tokens (Wang et al., 2026) or augment training data using observed generation failures (Giannone et al., 2026).

However, these approaches primarily generate single-part objects. While ArtiCAD (Shui et al., 2026) and AssemCAD (Dong et al., 2026) construct multi-part designs, they rely on ambiguous text or image inputs and thus target plausible prototypes rather than faithful reconstruction.

CADTalk (Yuan et al., 2024a) also operates on CAD code assemblies, but assumes them as input and uses them to generate descriptive part comments.

Prior work therefore either reconstructs monolithic CAD programs from geometry or generates multi-part CAD models from ambiguous text and image inputs. PartCoder combines explicit 3D conditioning with structured, multi-file output by reconstructing CAD code assemblies directly from point clouds. Rather than requiring a more capable generator, it decomposes complex reconstruction into simpler part-level synthesis problems that can be solved using existing backbones.

3 PARTCODER

We introduce PartCoder, the first approach for reconstructing multi-file CAD code assemblies from point clouds using LLMs. Rather than reconstructing the complete object as a single program, PartCoder adaptively divides the task into simpler part-level problems. Its three-stage pipeline hierarchically decomposes a dense input point cloud, independently generates CAD code for each resulting part, and combines the parts through a central assembly file, continuing decomposition only while the assembled reconstruction improves. This adaptive strategy extends the effective complexity range of existing generators without retraining. Algorithm 1 outlines the pipeline, while Figure 1 provides a visual overview. Further methodological details and algorithms for each stage are provided in Appendix A.

3.1 PROBLEM DEFINITION

We represent a dense point cloud with N points as $\boldsymbol{P} \in \mathbb{R}^{N\times 3}$. Let $\mathcal{C}_{\text{part}}$ and $\mathcal{C}_{\text{asm}}$ denote the spaces of valid part and assembly code, respectively. Both use the same CAD framework, but assembly code references and places the part files. We define a CAD assembly as

$$A = (C_{\text{asm}}, \{C_i\}_{i\in \boldsymbol{I}}) \in \mathcal{A},$$

where $C_i \in \mathcal{C}_{\text{part}}$ are independently editable part programs and $C_{\text{asm}} \in \mathcal{C}_{\text{asm}}$ scales, positions, and combines them. Compiling the assembly and its part files yields $\phi(A) \in \mathcal{O}$, where $\mathcal{O}$ denotes the space of compiled CAD geometries. The decomposition is selected for reconstruction fidelity and need not correspond to the original manufacturing components. We thus seek to construct a mapping

$$\Psi : \mathbb{R}^{N\times 3} \to \mathcal{A}, \qquad \Psi(\boldsymbol{P}) = A,$$

that recovers an assembly whose geometry matches $\boldsymbol{P}$ as closely as possible.

3.2 HIERARCHICAL PART DECOMPOSITION

Dynamic decomposition. Since the optimal number m^* of assembly parts is unknown a priori, we employ a dynamic hierarchical decomposition strategy that progressively reduces part-level geometric complexity rather than committing to a fixed number of parts. We represent the decomposition as a binary tree $\mathcal{T}$ with at most $m_{\max}$ leaves, corresponding to the assembly parts **(line 1)**. Each node v_k is associated with a point cloud $\boldsymbol{P}_k \subseteq \boldsymbol{P}$. The tree satisfies the following properties:

(1) The root node represents the complete point cloud, $\boldsymbol{P}_0 = \boldsymbol{P}$.

(2) At decomposition step $s \in \{1, \ldots, m_{\max} - 1\}$, exactly one current leaf is split into two children **(line 6)**.

(3) For the point cloud of every parent v_k with children v_{k_1} and v_{k_2} it holds $\boldsymbol{P}_k = \boldsymbol{P}_{k_1} \dot{\cup} \boldsymbol{P}_{k_2}$.

Let $\mathcal{T}^{(s)}$ denote the tree after s decomposition steps and $\boldsymbol{I}^{(s)}$ the indices of its leaf nodes, where $s = 0$ corresponds to the root-only tree **(line 3)**. If all $m_{\max} - 1$ splits are performed, the fully grown tree contains $m_{\max}$ leaves and $M_{\max} = 2m_{\max} - 1$ nodes.

To obtain spatially local and geometrically coherent parts, we apply PartField (Liu et al., 2025) with agglomerative clustering. We denote the resulting hierarchical segmentation procedure by f_{seg}, which produces the dendrogram $\mathcal{D} = f_{\text{seg}}(\boldsymbol{P}, m_{\max})$. Traversing $\mathcal{D}$ from the root downward yields the successive binary splits used to construct $\mathcal{T}$.

Interface Completion (line 7). Decomposing the dense point cloud $\boldsymbol{P}$ can create open interfaces when connected geometry is split. We therefore complete these artificial boundaries so that each part-level input resembles a complete CAD component. For every parent node v_k, we refer to its two child point clouds as siblings. Their interface corresponds to the boundary introduced by the split. We approximate their shared interface using nearest-neighbor distances between the two point clouds. For a point $\boldsymbol{p} \in \boldsymbol{P}_{k_1}$, define its distance to $\boldsymbol{P}_{k_2}$ as

$$d(\boldsymbol{p}, \boldsymbol{P}_{k_2}) = \min_{\boldsymbol{q} \in \boldsymbol{P}_{k_2}} \|\boldsymbol{p} - \boldsymbol{q}\|_2,$$

and symmetrically for $\boldsymbol{q} \in \boldsymbol{P}_{k_2}$. The interface set contains points from either sibling point cloud whose nearest point in the other cloud lies within distance τ.

$$\mathcal{B}_{k_1 k_2} = \{\boldsymbol{p} \in \boldsymbol{P}_{k_1} \mid d(\boldsymbol{p}, \boldsymbol{P}_{k_2}) \leq \tau\} \cup \{\boldsymbol{q} \in \boldsymbol{P}_{k_2} \mid d(\boldsymbol{q}, \boldsymbol{P}_{k_1}) \leq \tau\}.$$

We approximate the interface by fitting a plane to $\mathcal{B}_{k_1 k_2}$ using RANSAC (Fischler & Bolles, 1981), selecting the plane with the largest number of inliers under a point-to-plane distance threshold ϵ. We complete the interface by projecting the interface points onto the estimated plane, sampling additional points within their convex hull, and adding the sampled points to both sibling point clouds. This yields closure-augmented part point clouds $\bar{\boldsymbol{P}}_{k_1}$ and $\bar{\boldsymbol{P}}_{k_2}$, which are used as inputs for subsequent part-level CAD generation.

3.3 Part-Level CAD Generation

Part-level code generation proceeds iteratively and can use existing generative backbones without additional training, avoiding the associated overhead and enabling straightforward integration of different models. At decomposition step s, we generate CAD code only for the two newly created child parts **(line 8)**, while reusing the CAD codes of all unchanged leaf nodes. Following the inference-time sampling strategy of CAD-Recode (Rukhovich et al., 2025), we independently downsample each closure-augmented dense part point cloud $\bar{\boldsymbol{P}}_k$ ten times to n points and process each sample with a task-specific, fine-tuned LLM,

$$\psi_{\mathrm{LLM}} : \mathbb{R}^{n \times 3} \to \mathcal{C}_{\mathrm{part}},$$

such as CAD-Recode (Rukhovich et al., 2025) or cadrille (Kolodiazhnyi et al., 2026), yielding ten candidate CAD codes per part. Each candidate is first compiled into a CAD model. Successfully compiled candidates are evaluated against the corresponding part point cloud $\bar{\boldsymbol{P}}_k$ using Chamfer distance, and the candidate with the lowest distance is selected as the final part-level CAD code C_k.

3.4 Assembly and Adaptive Stopping

Part alignment and assembly. After the individual parts are generated independently, they must be organized into a structured assembly representing the complete object. To this end, the assembly stage constructs the assembly code $C_{\mathrm{asm}}^{(s)}$ using the current part-level CAD codes and point clouds $\{C_k, \boldsymbol{P}_k\}_{k \in \boldsymbol{I}^{(s)}}$ **(line 9)**. To align each newly generated part with its corresponding point cloud $\boldsymbol{P}_k$ within the input point cloud $\boldsymbol{P}$, we determine a scaling matrix $\boldsymbol{S}_k \in \mathbb{R}^{3 \times 3}$ and a translation vector $\mathbf{t}_k \in \mathbb{R}^3$. Part orientation is preserved during generation.

The transformation parameters are obtained by aligning the compiled CAD model $\phi(C_k)$ with its corresponding part point cloud $\boldsymbol{P}_k$. The assembly code then imports the individual part files and applies their respective scaling and translation transformations, yielding $C_{\mathrm{asm}}^{(s)}$.

Stopping Criterion. To determine whether an additional decomposition step improves the overall reconstruction, we sample N points from the compiled current assembly $\phi(A^{(s)})$, yielding $\hat{\boldsymbol{P}}^{(s)}$, and compute its Chamfer distance to the original input point cloud $\boldsymbol{P}$ **(line 10)**. For $s = 0$, the reconstruction error is computed before any decomposition **(line 4)**. At each subsequent step, the current error is compared with that of the previous step **(line 11)**. A decomposition is accepted if

$$\mathrm{CD}(\boldsymbol{P}, \hat{\boldsymbol{P}}^{(s)}) < \mathrm{CD}(\boldsymbol{P}, \hat{\boldsymbol{P}}^{(s-1)}),$$

in which case the current split improves the reconstruction and decomposition continues. Otherwise, decomposition stops and $A^{(s-1)}$ is returned **(line 12)**. If no stopping occurs after $s = m_{\max} - 1$, the assembly from the finest decomposition, $A^{(m_{\max}-1)}$, is retained as the final output **(line 13)**.

Algorithm 1 PartCoder

Input: Dense point cloud $\boldsymbol{P}$, maximum number of parts $m_{\max}$
Output: Final CAD assembly

$\mathcal{T}, \{\boldsymbol{I}^{(s)}\}_{s=0}^{m_{\max}-1} \leftarrow \textsc{DecompositionTree}(\boldsymbol{P}, m_{\max})$
$C_0 \leftarrow \textsc{GenerateCadCode}(\boldsymbol{P}, \psi_{\text{LLM}})$
$A^{(0)} \leftarrow \textsc{InitializeRootAssembly}(C_0)$
$D^{(0)} \leftarrow \text{CD}(\boldsymbol{P}, A^{(0)})$
for $s = 1, \ldots, m_{\max} - 1$ **do**
  $\boldsymbol{P}_{k_1}, \boldsymbol{P}_{k_2} \leftarrow \textsc{SplitLeafPart}(\boldsymbol{P}, \mathcal{T})$
  $\bar{\boldsymbol{P}}_{k_1}, \bar{\boldsymbol{P}}_{k_2} \leftarrow \textsc{CompleteInterface}(\boldsymbol{P}_{k_1}, \boldsymbol{P}_{k_2})$
  $C_{k_1}, C_{k_2} \leftarrow \textsc{GenerateCadCode}(\bar{\boldsymbol{P}}_{k_1}, \bar{\boldsymbol{P}}_{k_2}, \psi_{\text{LLM}})$
  $A^{(s)} \leftarrow \textsc{Assemble}(\{C_k, \boldsymbol{P}_k\}_{k \in \boldsymbol{I}^{(s)}})$
  $D^{(s)} \leftarrow \text{CD}(\boldsymbol{P}, A^{(s)})$
  if $D^{(s)} \geq D^{(s-1)}$ **then**
    return $A^{(s-1)}$
return $A^{(m_{\max}-1)}$

4 Experiments

In this section, we describe the experimental setup, benchmarks and baselines, followed by quantitative and qualitative evaluations.

4.1 Experimental Setup

PartCoder configurations. We instantiate PartCoder with CAD-Recode (Rukhovich et al., 2025) and cadrille (Kolodiazhnyi et al., 2026) as generative backbones, evaluating both the supervised fine-tuned (SFT) and reinforcement learning (RL) cadrille variants.

Although PartCoder does not require a fixed dense point cloud size, we follow CAD-Recode and cadrille by using $N = 8{,}192$ and $n = 256$ in all experiments to ensure direct comparability. We set the maximum number of assembly parts to $m_{\max} = 8$. An ablation on this limit is presented in Appendix B.1. For interface completion, we use $\tau = \epsilon = 5\bar{d}_{\text{NN}}$, where $\bar{d}_{\text{NN}}$ denotes the mean nearest-neighbor distance of the normalized input point cloud and serves as a reference length scale that adapts the thresholds to the point cloud's density.

Baselines. We compare PartCoder against the original CAD-Recode and cadrille models. For Cadrille, we evaluate both the supervised fine-tuned (SFT) and reinforcement-learning (RL) variants. For the stronger RL variant, we evaluate all available input configurations: point cloud (PC), multi-view images (MV), and multimodal input (MV+PC). For CAD-Recode, cadrille, and PartCoder, we use the same inference-time sampling strategy introduced by CAD-Recode. Multi-view images are rendered from ground truth meshes following cadrille's settings. Since such images are unavailable in point-cloud-only settings, we include these variants solely for additional comparison.

Using the same images, we further evaluate general-purpose LLMs by prompting them to reconstruct the depicted object as CadQuery code, including GPT-5.4 (OpenAI, 2026), Claude Sonnet 4.6 (Anthropic, 2026), and Gemini 3.5 Flash (Google DeepMind, 2026). Moreover, we include CADReasoner (Kabisov et al., 2026) as a recent CAD reverse-engineering baseline.

4.2 Evaluation data

Fusion 360 and OmniCAD. To evaluate performance across increasing benchmark difficulty, we benchmark on the Fusion 360 Gallery (Willis et al., 2021) test set, containing 1,725 human-designed objects, and OmniCAD (Xu et al., 2024), an extension of DeepCAD (Wu et al., 2021) with more complex and realistic geometries.

To focus on challenging OmniCAD samples, we filter its test set by extrusion count, a recognized proxy for difficulty (Xu et al., 2024), retaining objects with at least 10 extrusions. For each selected

object, we include its construction states from the tenth extrusion onward. After excluding non-watertight states to ensure valid volumetric evaluation, the resulting test set contains 658 samples.

PrIMe3D. To evaluate even more complex objects and the emerging use case of converting rapidly generated 3D assets into CAD models, we introduce PrIMe3D (Prompt-to-Image-to-Mesh). PrIMe3D contains 1,000 mechanical objects generated from randomly sampled prompts based on a mechanical engineering vocabulary. The prompts are first used to synthesize images with GPT-image-1-mini (OpenAI, 2025), which are then converted into high-quality 3D mesh prototypes using Trellis 2 (Xiang et al., 2025).

To enable volumetric evaluation, non-watertight PrIMe3D meshes are processed using CGAL's 3D Alpha Wrapping (Alliez et al., 2026) to obtain watertight evaluation meshes. The original meshes are retained for sampling point cloud inputs and rendering images. PrIMe3D therefore provides both challenging geometries and a benchmark for CAD reverse engineering of AI-generated 3D assets.

4.3 METRICS

We evaluate geometric fidelity by compiling the generated CAD code into a 3D object $\hat{O}$ and comparing it with the ground-truth object O using Chamfer distance (CD) for surface similarity and intersection over union (IoU) for volumetric overlap. Unlike the intermediate evaluations, final CD uses 100,000 surface points sampled from each object, yielding point clouds $\boldsymbol{Q}$ and $\hat{\boldsymbol{Q}}$ for O and $\hat{O}$, respectively. We denote the Euclidean distance between $q \in \boldsymbol{Q}$ and $\hat{q} \in \hat{\boldsymbol{Q}}$ by $d(q, \hat{q})$. IoU is computed following established practice of Wu et al. (2021). We use this protocol for all methods.

$$CD(\boldsymbol{Q}, \hat{\boldsymbol{Q}}) = \frac{1}{2|\boldsymbol{Q}|} \sum_{q \in \boldsymbol{Q}} \min_{\hat{q} \in \hat{\boldsymbol{Q}}} d(q, \hat{q}) + \frac{1}{2|\hat{\boldsymbol{Q}}|} \sum_{\hat{q} \in \hat{\boldsymbol{Q}}} \min_{q \in \boldsymbol{Q}} d(q, \hat{q}) \tag{1}$$

$$IoU(O, \hat{O}) = \frac{|O \cap \hat{O}|}{|O \cup \hat{O}|} \tag{2}$$

Following CAD-Recode and cadrille, we report median CD and mean IoU over successfully compiled reconstructions, since CD is sensitive to outliers and geometric metrics are undefined for invalid outputs. Code validity is measured by the invalidity ratio (IR), defined as the percentage of non-compilable samples (Wu et al., 2021). The main paper reports CD, IoU, and IR, while Hausdorff distance (HD), Dice coefficient (DSC), and per-sample distributions for all methods and datasets are provided in Appendix E.

4.4 RESULTS

Improved reconstruction quality. Table 1 shows that all PartCoder variants improve or match every reported metric relative to their corresponding monolithic backbones across all datasets. For CAD-Recode on Fusion360, CD and IoU remain unchanged, while PartCoder substantially reduces IR from 8.3 to 2.9. In every other matched comparison, PartCoder improves both CD and IoU, while IR either decreases or remains at zero. PartCoder-cadrille-RL performs best overall, achieving the highest IoU on all three datasets, the lowest CD on OmniCAD and PrIMe3D, and zero IR throughout, only its CD on Fusion360 is second-best.

General-purpose LLMs remain substantially behind specialized CAD reconstruction methods. cadrille's multimodal variants improve individual metrics but do not consistently outperform the point-cloud-only variant. Since PartCoder improves the same point-cloud backbones without additional input modalities, these gains can be attributed to its decomposition-and-assembly strategy rather than richer input conditioning.

Increasing benefits with benchmark difficulty. PartCoder's advantage grows with reconstruction difficulty. Averaged across the three backbone variants, PartCoder improves IoU over the corresponding monolithic baselines by 1.1 percentage points on Fusion360, 7.7 points on OmniCAD, and 11.1 points on PrIMe3D. This trend suggests that decomposing reconstruction into simpler part-level synthesis problems becomes increasingly beneficial as geometric complexity increases.

Adaptive decomposition behavior. Table 2 shows that the adaptive stopping criterion produces finer decompositions as benchmark difficulty increases. Averaged across the three backbones, the

Method	Fusion360			OmniCAD			PrIMe3D		
	CD↓	IoU↑ (%)	IR↓ (%)	CD↓	IoU↑ (%)	IR↓ (%)	CD↓	IoU↑ (%)	IR↓ (%)
Gemini-Flash 3.5	8.60	44.9	24.4	14.2	23.7	38.0	15.8	32.9	48.1
Claude Sonnet 4.6	24.9	36.7	5.1	37.2	17.8	10.9	25.5	25.1	7.8
GPT 5.4	17.9	39.3	12.9	34.9	18.3	18.5	28.9	27.0	24.8
CADReasoner	**0.06**	86.5	0.06	1.89	46.6	**0**	3.53	48.2	0.1
cadrille-RL (mv)	0.14	86.3	0.06	4.03	55.7	**0**	7.92	48.2	0.1
cadrille-RL (mv+pc)	0.08	90.1	0.1	1.25	62.7	**0**	3.19	54.9	**0**
CAD-Recode	0.10	86.6	8.3	1.57	52.3	4.7	4.79	39.2	0.1
cadrille-SFT (pc)	0.09	86.6	0.1	1.41	54.2	2.1	3.96	44.9	**0**
cadrille-RL (pc)	0.08	89.8	**0**	1.17	60.4	**0**	2.79	55.5	**0**
PartCoder-CAD-Recode	0.10	86.6	2.9	0.77	59.2	0.2	1.73	52.9	**0**
PartCoder-cadrille-SFT	0.08	88.3	**0**	0.65	63.1	**0**	1.46	57.0	**0**
PartCoder-cadrille-RL	0.08	**91.3**	**0**	**0.54**	**67.7**	**0**	**1.32**	**62.9**	**0**

Table 1: Comparison of PartCoder and baselines on Fusion360, OmniCAD, and PrIMe3D. Monolithic baselines (middle block) and corresponding PartCoder variants (bottom block) appear in matching order. Best results are **bold** and second-best results are underlined. Each PartCoder variant achieves significantly better CD and IoU than its monolithic baseline under one-sided paired Wilcoxon signed-rank tests ($\alpha = 0.01$); see Appendix C.1.

Method	Average number of parts		
	Fusion360	OmniCAD	PrIMe3D
PartCoder-CAD-Recode	1.37	2.05	2.95
PartCoder-cadrille-SFT	1.30	2.28	2.94
PartCoder-cadrille-RL	1.33	2.28	2.68

Table 2: **Selected number of parts.** Average number of parts used in the final reconstruction for each method and dataset. More complex datasets tend to benefit from finer decompositions.

selected part count rises from 1.33 on Fusion360 to 2.20 on OmniCAD and 2.86 on PrIMe3D, with the same trend observed for every backbone. Together with the increasing IoU gains in Table 1, this suggests that PartCoder introduces additional parts where decomposition is most beneficial. Figure 4 in the Appendix provides the corresponding distributions of selected part counts. An ablation of $m_{\max}$ further shows that allowing finer decompositions generally improves reconstruction quality across model-dataset combinations; full results and a discussion are provided in Appendix B.

Runtime comparison. Table 3 compares average runtimes for PartCoder-cadrille-RL, monolithic cadrille-RL, and CADReasoner on 100 uniformly sampled objects per dataset, using a single NVIDIA RTX A6000 GPU. Averaged across datasets, the monolithic backbone takes 5.0 seconds per object versus 30.7 for PartCoder, due to additional part-level generation and assembly evaluation. Despite this overhead, PartCoder remains 3.9–6.9 times faster than CADReasoner while improving reconstruction quality and producing structured multi-file CAD assemblies. Increasing monolithic sampling to 50 candidates does not close the quality gap on OmniCAD or PrIMe3D (Appendix C.3).

Method	Average runtime (s)		
	Fusion360	OmniCAD	PrIMe3D
CADReasoner	80.5	234.7	248.8
PartCoder-cadrille-RL	20.9	35.2	35.9
cadrille-RL	4.6	5.1	5.3

Table 3: **Average runtime.** Average runtime required to generate a CAD code reconstruction.

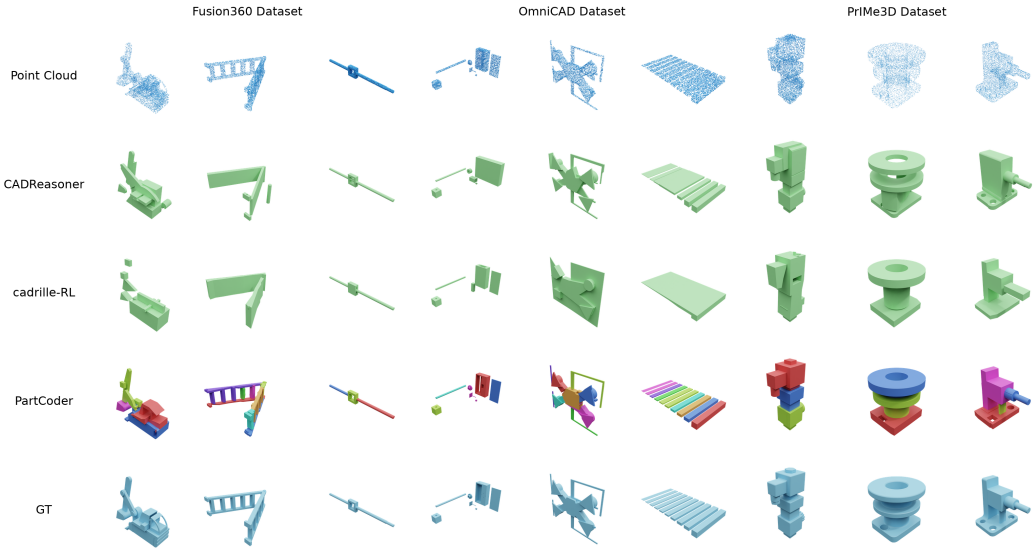


Figure 2: **Qualitative results.** All methods reconstruct the same targets. PartCoder and cadrille-RL use the shown point clouds, whereas CADReasoner requires mesh input. PartCoder outputs are colored by part to highlight assembly structure; baseline outputs use a single color.

Qualitative results. Figure 2 illustrates how PartCoder's decomposition-and-assembly strategy improves reconstruction while exposing explicit part structure. The benefits are particularly evident for disconnected components (columns 4 and 6), objects composed of several simpler parts (columns 2, 3, and 8), and geometrically complex compositions (columns 1, 5, 7 and 9). In these cases, PartCoder preserves both global arrangement and local geometry, whereas the monolithic reconstructions produced by cadrille-RL and CADReasoner often omit components, merge distinct structures, or oversimplify individual elements.

5 Conclusion

We presented PartCoder, the first method for generating multi-file CAD code assemblies directly from point clouds. By hierarchically decomposing objects, reconstructing each part independently, and retaining only decompositions that improve the assembled reconstruction, PartCoder transforms a difficult monolithic synthesis problem into an adaptively selected set of simpler part-level problems. Across Fusion360, OmniCAD, and PrIMe3D, PartCoder achieves state-of-the-art reconstruction quality, with increasing gains as object complexity grows, while producing modular and independently controllable CAD representations. We further introduced PrIMe3D, a new benchmark for converting AI-generated 3D assets into editable CAD models.

A current limitation is the interface completion step, which assumes approximately planar part interfaces. Although this assumption is reasonable for many CAD objects, it is not guaranteed and may fail for curved or complex contact surfaces. Another limitation is that CD serves as both the decomposition stopping criterion and a final evaluation metric.

Although our hierarchical decomposition adaptively selects a suitable number of assembly parts, predicting the optimal number m^* a priori could streamline the process and improve its efficiency. Developing such a predictor is a challenging but promising direction for future work, as m^* depends not only on the object geometry but also on the generation capabilities of the underlying model.

AI use statement

In this work, we used generative AI tools to create a synthetic benchmark dataset and help to interpret results. We have not used generative AI tools for critical ingredients, design and conduct

experiments, to create figures and result images and the rest of the required disclosure tasks are not applicable to this work. Additionally, we used generative AI tools for aid and polish writing, format tables, create or edit software code, summarize or analyse existing literature, sourcing/searching for information, edit a research paper to improve readability, identify relevant literature, suggest a structure for a research paper, propose a title or keywords for a research paper. We have reviewed all AI-assisted work. All content was manually checked and corrected. We checked LLM-generated literature proposals and summaries through a manual literature search. LLM-generated code was verified and tested for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

Reproducibility statement

To support reproducibility, we provide the complete PartCoder implementation and evaluation code at `https://anonymous.4open.science/r/PartCoder`. The repository README documents environment setup, data preprocessing, inference, and evaluation, and includes per-sample metric results for PartCoder and all baselines as CSV files. Further methodological and implementation details are provided in Appendix A. We also release PrIMe3D and the filtered and augmented OmniCAD dataset at the following two Zenodo uploads: PrIMe3D and OmniCAD. If either of the two underlined dataset links is inaccessible, links to both datasets are also available in the anonymous PartCoder repository above. Fusion360 is available from its official website.

References

Pierre Alliez, David Cohen-Steiner, Michael Hemmer, Cédric Portaneri, and Mael Rouxel-Labbé. 3D alpha wrapping. In *CGAL User and Reference Manual*. CGAL Editorial Board, 6.2 edition, 2026. URL `https://doc.cgal.org/6.2/Manual/packages.html#PkgAlphaWrap3`.

Kamel Alrashedy, Pradyumna Tambwekar, Zulfiqar Haider Zaidi, Megan Langwasser, Wei Xu, and Matthew Gombolay. Generating CAD code with vision-language models for 3d designs. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=BLWaTeucYX`.

Anthropic. Claude Sonnet 4.6 System Card. Technical report, Anthropic, 2026. URL `https://www-cdn.anthropic.com/78073f739564e986ff3e28522761a7a0b4484f84.pdf`.

Akshay Badagabettu, Sai Sravan Yarlagadda, and Amir Barati Farimani. Query2cad: Generating cad models using natural language queries. *arXiv preprint arXiv:2406.00144*, 2024.

Jesse Barkley, Rumi Loghmani, and Amir Barati Farimani. Cadsmith: Multi-agent cad generation with programmatic geometric validation. *arXiv preprint arXiv:2603.26512*, 2026.

cadquery. Cadquery 2.4.0, 1 2024. URL `https://doi.org/10.5281/zenodo.10513848`.

Jiali Chen, Xusen Hei, HongFei Liu, Yuancheng Wei, Zikun Deng, Jiayuan Xie, Yi Cai, and Qing Li. Cadreview: Automatically reviewing cad programs with error detection and correction. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9909–9927, 2025.

Yurui Dong, Shu Zou, Siqi Li, Nianchen Deng, Hongbin Zhou, Xuemeng Yang, Pinlong Cai, Licheng Wen, Xinyu Cai, and Botian Shi. Assemcad: Production-ready cad assembly generation from natural language. *arXiv preprint arXiv:2607.05123*, 2026.

Anna C Doris, Ferdous Alam, Amin Heyrani Nobari, and Faez Ahmed. Cad-coder: An open-source vision-language model for computer-aided design code generation. *Journal of Mechanical Design*, 148(7):071702, 2026.

Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. Inversecsg: Automatic conversion of 3d models to csg trees. *ACM Transactions on Graphics (TOG)*, 37(6):1–16, 2018.

Elona Dupont, Kseniya Cherenkova, Dimitrios Mallis, Gleb Gusev, Anis Kacem, and Djamila Aouada. Transcad: A hierarchical transformer for cad sequence inference from point clouds. In *European Conference on Computer Vision*, pp. 19–36. Springer, 2024.

Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24 (6):381–395, 1981. doi: 10.1145/358669.358692.

FreeCAD. FreeCAD: An open-source parametric 3d modeler. https://github.com/FreeCAD/FreeCAD, 2026.

Giorgio Giannone, Anna C. Doris, Amin Heyrani Nobari, Kai Xu, Akash Srivastava, and Faez Ahmed. GIFT: Bootstrapping image-to-CAD program synthesis via geometric feedback. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=LShQYeeop5.

Yifei Gong, Xing Wu, Wenda Liu, et al. Toolcad: Exploring tool-using large language models in text-to-cad generation with reinforcement learning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 23161–23188, 2026.

Google DeepMind. Gemini 3.5 Flash: Model Card. Google DeepMind, 2026. URL https://deepmind.google/models/model-cards/gemini-3-5-flash/.

Yandong Guan, Xilin Wang, Ximing Xing, Jing Zhang, Dong Xu, and Qian Yu. Cad-coder: Text-to-cad generation with chain-of-thought and geometric reward. *Advances in Neural Information Processing Systems*, 38:59765–59789, 2026.

Haoxiang Guo, Shilin Liu, Hao Pan, Yang Liu, Xin Tong, and Baining Guo. Complexgen: Cad reconstruction by b-rep chain complex generation. *ACM Transactions on Graphics (TOG)*, 41(4): 1–18, 2022.

Soslan Kabisov, Vsevolod Kirichuk, Andrey Volkov, Marina Barannikov, Gennadiy Savrasov, Anton Konushin, Andrey Kuznetsov, and Dmitrii Zhemchuzhnikov. Cadreasoner: Iterative program editing for cad reverse engineering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6143–6153, 2026.

Kacper Kania, Maciej Zieba, and Tomasz Kajdanowicz. Ucsg-net-unsupervised discovering of constructive solid geometry tree. *Advances in neural information processing systems*, 33:8776–8786, 2020.

Ahmet Serdar Karadeniz, Dimitrios Mallis, Danila Rukhovich, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. MiCADangelo: Fine-grained reconstruction of constrained CAD models from 3d scans. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=GoFYIRUVAz.

Mohammad Sadil Khan, Elona Dupont, Sk Aziz Ali, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. Cad-signet: Cad language inference from point clouds using layer-wise sketch instance guided attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4713–4722, 2024.

Maksim Kolodiazhnyi, Denis Tarasov, Dmitrii Zhemchuzhnikov, Alexander Nikulin, Ilya Zisman, Anna Vorontsova, Anton Konushin, Vladislav Kurenkov, and Danila Rukhovich. cadrille: Multi-modal cad reconstruction with reinforcement learning, 2026. URL https://arxiv.org/abs/2505.22914.

Jiahao Li, Weijian Ma, Xueyang Li, Yunzhong Lou, Guichun Zhou, and Xiangdong Zhou. Cad-llama: leveraging large language models for computer-aided design parametric 3d model generation. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 18563–18573. IEEE, 2025a.

Jiahao Li, Yusheng Luo, Yunzhong Lou, and Xiangdong Zhou. Recad: Reinforcement learning enhanced parametric cad model generation with vision-language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 6190–6198, 2026a.

Lingxiao Li, Minhyuk Sung, Anastasia Dubrovina, Li Yi, and Leonidas J Guibas. Supervised fitting of geometric primitives to 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2652–2660, 2019.

Pu Li, Jianwei Guo, Xiaopeng Zhang, and Dong-Ming Yan. Secad-net: Self-supervised cad reconstruction by learning sketch-extrude operations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16816–16826, 2023.

Pu Li, Jianwei Guo, Huibin Li, Bedrich Benes, and Dong-Ming Yan. Sfmcad: Unsupervised cad reconstruction by learning sketch-based feature modeling operations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4671–4680, 2024.

Xingang Li, Yuewan Sun, and Zhenghui Sha. Llm4cad: Multimodal large language models for three-dimensional computer-aided design generation. *Journal of Computing and Information Science in Engineering*, 25(2):021005, 2025b.

Xueyang Li, Jiahao Li, Yu Song, Yunzhong Lou, and Xiangdong Zhou. Seek-cad: A self-refined generative modeling for 3d parametric cad using local inference via deepseek. In *International Conference on Learning Representations*, volume 2026, pp. 94278–94298, 2026b.

Minghua Liu, Mikaela Angelina Uy, Donglai Xiang, Hao Su, Sanja Fidler, Nicholas Sharp, and Jun Gao. Partfield: Learning 3d feature fields for part segmentation and beyond. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9704–9715. IEEE, 2025.

Yujia Liu, Stefano D'Aronco, Konrad Schindler, and Jan Dirk Wegner. Pc2wf: 3d wireframe reconstruction from raw point clouds. In *International Conference on Learning Representations (ICLR 2021)*. OpenReview, 2021.

Yujia Liu, Anton Obukhov, Jan Dirk Wegner, and Konrad Schindler. Point2cad: Reverse engineering cad models from 3d point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 3763–3772, 2024.

Dimitrios Mallis, Ahmet Serda Karadeniz, Sebastian Cavada, Danila Rukhovich, Niki Foteinopoulou, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. Cad-assistant: Tool-augmented vllms as generic cad task solvers. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 7284–7294. IEEE, 2025.

Ghadi Nehme, Eamon Whalen, and Faez Ahmed. Cadfit: Precise mesh-to-cad program generation with hybrid optimization. *International Conference of Machine Learning (ICML)*, 2026.

Ke Niu, Haiyang Yu, Zhuofan Chen, Zhengtao Yao, Weitao Jia, Xiaodong Ge, Jingqun Tang, Benlei Cui, Bin Li, and Xiangyang Xue. Cme-cad: Heterogeneous collaborative multi-expert reinforcement learning for cad code generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 39272–39281, 2026a.

Ke Niu, Haiyang Yu, Zhuofan Chen, Mengyang Zhao, Teng Fu, Bin Li, and Xiangyang Xue. From intent to execution: Multimodal chain-of-thought reinforcement learning for precise cad code generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 8160–8167, 2026b.

Felix Ocker, Stefan Menzel, Ahmed Sadik, and Thiago Rios. From idea to cad: A language model-driven multi-agent system for collaborative design. *arXiv preprint arXiv:2503.04417*, 2025.

OpenAI. gpt-image-1-mini model. OpenAI API Documentation, 2025. URL `https://developers.openai.com/api/docs/models/gpt-image-1-mini`.

OpenAI. GPT-5.4 Model. OpenAI API Documentation, 2026. URL `https://developers.openai.com/api/docs/models/gpt-5.4`.

Thomas Paviot. pythonocc (7.7. 2). *Zenodo. Accessed April*, 25:2024, 2022.

Cyril Picard, Kristen M Edwards, Anna C Doris, Brandon Man, Giorgio Giannone, Md Ferdous Alam, and Faez Ahmed. From concept to manufacturing: Evaluating vision-language models for engineering design. *arXiv preprint arXiv:2311.12668*, 2023.

Tobias Preintner, Weixuan Yuan, Adrian König, Thomas Bäck, Elena Raponi, and Niki Van Stein. Evocad: Evolutionary cad code generation with vision language models. In *2025 IEEE 37th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 504–511. IEEE, 2025.

Daxuan Ren, Jianmin Zheng, Jianfei Cai, Jiatong Li, Haiyong Jiang, Zhongang Cai, Junzhe Zhang, Liang Pan, Mingyuan Zhang, Haiyu Zhao, et al. Csg-stump: A learning friendly csg-like representation for interpretable shape parsing. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 12478–12487, 2021.

Danila Rukhovich, Elona Dupont, Dimitrios Mallis, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. Cad-recode: Reverse engineering cad code from point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9801–9811, 2025.

Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. Csgnet: Neural shape parser for constructive solid geometry. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Gopal Sharma, Difan Liu, Subhransu Maji, Evangelos Kalogerakis, Siddhartha Chaudhuri, and Radomír Měch. Parsenet: A parametric surface fitting network for 3d point clouds. In *European conference on computer vision*, pp. 261–276. Springer, 2020.

Yuan Shui, Yandong Guan, Zhanwei Zhang, Juncheng Hu, Jing Zhang, Dong Xu, and Qian Yu. Articad: Articulated cad assembly design via multi-agent code generation. *arXiv preprint arXiv:2604.10992*, 2026.

Tencent. Hunyuan3d 2.5: Towards high-fidelity 3d assets generation with ultimate details, 2025. URL https://arxiv.org/abs/2506.16504.

Ruiyu Wang, Shizhao Sun, Weijian Ma, and Jiang Bian. Cad-tokenizer: Towards text-based cad prototyping via modality-specific tokenization. In *International Conference on Learning Representations*, volume 2026, pp. 147972–147992, 2026.

Siyu Wang, Cailian Chen, Xinyi Le, Qimin Xu, Lei Xu, Yanzhou Zhang, and Jie Yang. Cad-gpt: Synthesising cad construction sequence with spatial reasoning-enhanced multimodal llms. In *Proceedings of the AAAI conference on artificial intelligence*, volume 39, pp. 7880–7888, 2025.

Xiaogang Wang, Yuelang Xu, Kai Xu, Andrea Tagliasacchi, Bin Zhou, Ali Mahdavi-Amiri, and Hao Zhang. Pie-net: Parametric inference of point cloud edges. *Advances in neural information processing systems*, 33:20167–20178, 2020.

Karl DD Willis, Yewen Pu, Jieliang Luo, Hang Chu, Tao Du, Joseph G Lambourne, Armando Solar-Lezama, and Wojciech Matusik. Fusion 360 gallery: A dataset and environment for programmatic cad construction from human design sequences. *ACM Transactions on Graphics (TOG)*, 40(4): 1–24, 2021.

Rundi Wu, Chang Xiao, and Changxi Zheng. Deepcad: A deep generative network for computer-aided design models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6772–6782, 2021.

Jianfeng Xiang, Xiaoxue Chen, Sicheng Xu, Ruicheng Wang, Zelong Lv, Yu Deng, Hongyuan Zhu, Yue Dong, Hao Zhao, Nicholas Jing Yuan, et al. Native and compact structured latents for 3d generation. *arXiv preprint arXiv:2512.14692*, 2025.

Haoyang Xie and Feng Ju. Text-to-cadquery: A new paradigm for cad generation with scalable large model capabilities. *arXiv preprint arXiv:2505.06507*, 2025.

Jingwei Xu, Chenyu Wang, Zibo Zhao, Wen Liu, Yi Ma, and Shenghua Gao. Cad-mllm: Unifying multimodality-conditioned cad generation with mllm, 2024.

Siming Yan, Zhenpei Yang, Chongyang Ma, Haibin Huang, Etienne Vouga, and Qixing Huang. Hpnet: Deep primitive segmentation using hybrid representations. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 2753–2762, 2021.

Fenggen Yu, Zhiqin Chen, Manyi Li, Aditya Sanghi, Hooman Shayani, Ali Mahdavi-Amiri, and Hao Zhang. Capri-net: Learning compact cad shapes with adaptive primitive assembly. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11768–11778, 2022.

Fenggen Yu, Qimin Chen, Maham Tanveer, Ali Mahdavi Amiri, and Hao Zhang. D^2 csg: Unsupervised learning of compact csg trees with dual complements and dropouts. *Advances in Neural Information Processing Systems*, 36:22807–22819, 2023.

Bo Yuan, Zelin Zhao, Petr Molodyk, Bin Hu, and Yongxin Chen. Clarify before you draw: Proactive agents for robust text-to-CAD generation. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=1WJ283sbfs.

Haocheng Yuan, Jing Xu, Hao Pan, Adrien Bousseau, Niloy J Mitra, and Changjian Li. Cadtalk: An algorithm and benchmark for semantic commenting of cad programs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3753–3762, 2024a.

Zeqing Yuan, Haoxuan Lan, Qiang Zou, and Junbo Zhao. 3d-premise: Can large language models generate 3d shapes with sharp features and parametric control? *arXiv preprint arXiv:2401.06437*, 2024b.

Zhe Yuan, Jianqi Shi, and Yanhong Huang. Openecad: An efficient visual language model for editable 3d-cad design. *Computers & Graphics*, 124:104048, 2024c.

Zhanwei Zhang, Shizhao Sun, Wenxiao Wang, Deng Cai, and Jiang Bian. Flexcad: Unified and versatile controllable cad generation with fine-tuned large language models. In *International Conference on Learning Representations*, volume 2025, pp. 3204–3227, 2025.

Xiangyu Zhu, Dong Du, Weikai Chen, Zhiyou Zhao, Yinyu Nie, and Xiaoguang Han. Nerve: Neural volumetric edges for parametric curve extraction from point cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13601–13610, 2023.

A Extended Methodological Details

This section provides further details on the three methodological stages introduced in Section 3. Algorithm 2 describes the decomposition and interface completion procedure, Algorithm 3 presents the part generation stage, and Algorithm 4 details the assembly process.

A.1 Hierarchical Decomposition and Interface Completion

The DecompositionTree function takes the input point cloud and the maximum allowed number of parts (line 1) and returns a hierarchical decomposition tree containing all intermediate decompositions and their associated point clouds (line 9). First, the input point cloud is hierarchically clustered into up to $m_{\max}$ parts, yielding the dendrogram $\mathcal{D}$ (line 2). The root node, its associated point cloud, and the initial leaf-index set are initialized (lines 4-5). In each subsequent iteration, $\mathcal{D}$ determines which leaf point cloud $\boldsymbol{P}_k$ to split next and specifies its two child clusters (line 6). The tree structure and corresponding leaf-index set are subsequently updated by adding the resulting child nodes (lines 7-8).

After each split, the CompleteInterface function identifies the interface set $\mathcal{B}_{k_1k_2}$ between the resulting sibling point clouds (line 11), as detailed in Subsection 3.2. For disconnected components, no shared interface exists, and therefore the set is empty and no completion is performed. Otherwise, it contains points from each sibling that lie sufficiently close to the other point cloud. A plane Π is then fitted to the interface set using RANSAC, with the selected plane corresponding to the candidate with the largest number of inliers under the point-to-plane distance threshold ϵ (line 12). To ensure that the sampled points $\boldsymbol{P}_{k_1k_2}$ lie inside the open interface, the inliers are projected onto the plane Π, their convex hull is computed, and new points are sampled on the plane within this hull (line 14). Finally, the sampled points are added to both sibling point clouds, completing the interface exposed by the split (lines 15–16).

Algorithm 2 Hierarchical Part Decomposition and Interface Completion

1: **function** DecompositionTree($\boldsymbol{P}, m_{\max}$)
2: $\quad\mathcal{D} \leftarrow f_{\text{seg}}(\boldsymbol{P}, m_{\max})$ `// agglomerative clustering of` $\boldsymbol{P}$
3: $\quad$Initialize $\mathcal{T}^{(0)}$ with root v_0
4: $\quad\boldsymbol{I}^{(0)} \leftarrow \{0\}$, $\boldsymbol{P}_0 \leftarrow \boldsymbol{P}$
5: $\quad$**for** $s = 1, \ldots, m_{\max} - 1$ **do**
6: $\quad\quad(k, \boldsymbol{P}_{k_1}, \boldsymbol{P}_{k_2}) \leftarrow$ next cut of $\mathcal{D}$, splitting v_k's points $\boldsymbol{P}_k$
7: $\quad\quad\mathcal{T}^{(s)} \leftarrow \mathcal{T}^{(s-1)} \cup \{v_{k_1}, v_{k_2}\}$
8: $\quad\quad\boldsymbol{I}^{(s)} \leftarrow \big(\boldsymbol{I}^{(s-1)} \setminus \{k\}\big) \cup \{k_1, k_2\}$
9: $\quad\mathcal{T} \leftarrow \mathcal{T}^{(m_{\max}-1)}$
10: $\quad$**return** $\mathcal{T}, \{\boldsymbol{I}^{(s)}\}_{s=0}^{m_{\max}-1}$

11: **function** CompleteInterface($\boldsymbol{P}_{k_1}, \boldsymbol{P}_{k_2}, \tau, \epsilon$)
12: $\quad\mathcal{B}_{k_1k_2} \leftarrow \{\boldsymbol{p} \in \boldsymbol{P}_{k_1} \mid d(\boldsymbol{p}, \boldsymbol{P}_{k_2}) \le \tau\} \cup \{\boldsymbol{q} \in \boldsymbol{P}_{k_2} \mid d(\boldsymbol{q}, \boldsymbol{P}_{k_1}) \le \tau\}$
13: $\quad\Pi \leftarrow$ RANSACPlane($\mathcal{B}_{k_1k_2}, \epsilon$)
14: $\quad\mathcal{H} \leftarrow$ ProjectedConvexHull($\mathcal{B}_{k_1k_2}, \Pi$)
15: $\quad\boldsymbol{P}_{k_1k_2} \leftarrow$ SamplePoints($\Pi, \mathcal{H}$)
16: $\quad\bar{\boldsymbol{P}}_{k_1} \leftarrow \boldsymbol{P}_{k_1} \cup \boldsymbol{P}_{k_1k_2}$
17: $\quad\bar{\boldsymbol{P}}_{k_2} \leftarrow \boldsymbol{P}_{k_2} \cup \boldsymbol{P}_{k_1k_2}$
18: $\quad$**return** $\bar{\boldsymbol{P}}_{k_1}, \bar{\boldsymbol{P}}_{k_2}$

A.2 CAD Code Generation

Algorithm 3 performs R inference-time sampling iterations (line 1). In each iteration, the dense, closure-augmented point cloud is downsampled to n points and passed to the code-generation LLM, producing a candidate $\widehat{C}_{k,r}$ (line 3). The candidate is then compiled into a 3D object, from whose surface a dense point cloud is sampled (line 4). The Chamfer distance between this point cloud and the dense input point cloud is then computed (line 5). After all R candidates have been evaluated, the candidate with the lowest Chamfer distance is selected and returned (lines 6–8).

Algorithm 3 Part-Level CAD Code Generation

Input: Closure-augmented point cloud $\bar{\boldsymbol{P}}_k$, generation model ψ_{LLM}, number of downsampled points n, number of inference time sampling candidates R
Output: CAD code C_k

for $r = 1, \ldots, R$ **do**
  $\widetilde{\boldsymbol{P}}_{k,r} \leftarrow \textsc{DownSample}(\bar{\boldsymbol{P}}_k, n)$
  $\widehat{C}_{k,r} \leftarrow \psi_{\text{LLM}}(\widetilde{\boldsymbol{P}}_{k,r})$
  $\widehat{\boldsymbol{P}}_{k,r} \leftarrow \textsc{SamplePointCloud}(\phi(\widehat{C}_{k,r}))$
  $d_{k,r} \leftarrow \text{CD}(\bar{\boldsymbol{P}}_k, \widehat{\boldsymbol{P}}_{k,r})$
$r^* \leftarrow \arg\min_{r \in \{1,\ldots,R\}} d_{k,r}$
$C_k \leftarrow \widehat{C}_{k,r^*}$
return C_k

A.3 Assembly Code Construction

Algorithm 4 performs the final assembly stage following decomposition and part code generation. For the newly generated parts resulting from the current split s, the scale matrices and translation vectors are computed and cached (lines 1–3). For all unchanged parts, the transformations cached at previous splits are reused (line 5). These transformations are then used to construct the assembly code (line 6), which, together with the generated part codes, defines the final assembly (line 7).

Algorithm 4 Assembly Code Construction

Input: Current CAD part codes and point clouds $\{(C_k, \boldsymbol{P}_k)\}_{k \in \boldsymbol{I}^{(s)}}$, cached transformations
Output: CAD code assembly $A^{(s)}$

`// Compute transformations for the newly generated children`
for $k \in \boldsymbol{I}^{(s)} \setminus \boldsymbol{I}^{(s-1)}$ **do**
  $\boldsymbol{S}_k, \mathbf{t}_k \leftarrow \textsc{ComputeTransformation}(\phi(C_k), \boldsymbol{P}_k)$
  $\textsc{CacheTransformation}(k, \boldsymbol{S}_k, \mathbf{t}_k)$

`// Reuse transformations of unchanged parts`
for $k \in \boldsymbol{I}^{(s)} \cap \boldsymbol{I}^{(s-1)}$ **do**
  $\boldsymbol{S}_k, \mathbf{t}_k \leftarrow \textsc{CachedTransformation}(k)$
$C^{(s)}_{\text{asm}} \leftarrow \textsc{WriteAssemblyCode}(\{(C_k, \boldsymbol{S}_k, \mathbf{t}_k)\}_{k \in \boldsymbol{I}^{(s)}})$
$A^{(s)} \leftarrow \left(C^{(s)}_{\text{asm}}, \{C_k\}_{k \in \boldsymbol{I}^{(s)}}\right)$
return $A^{(s)}$

B Analysis of Decomposition Granularity

B.1 Ablation on the Maximum Number of Parts

Figure 3 evaluates the effect of the maximum number of parts $m_{\max}$. Across most model–dataset combinations, increasing $m_{\max}$ improves reconstruction quality, as reflected by higher IoU and lower Chamfer distance.

It is important to note that results across different values of $m_{\max}$ are not directly comparable because the geometric metrics are computed only for successfully reconstructed samples. Consequently, the evaluated subset of samples may vary with $m_{\max}$, introducing a validity-induced selection effect. To reduce sensitivity to changes in the evaluated subset, we report median CD and IoU values in this ablation, mitigating but not eliminating the underlying selection effect. This effect is most pronounced for PartCoder with CAD-Recode on Fusion360, where the IR improves from 8.3% at $m_{\max} = 1$ to 2.9% at $m_{\max} = 8$, corresponding to approximately 93 additional valid reconstructions. At $m_{\max} = 1$, more difficult objects are more likely to yield non-compilable code and are

therefore excluded from the evaluation. As $m_{\max}$ increases, finer decomposition enables some of these objects to be reconstructed successfully. Their inclusion, however, may lower the aggregate geometric scores, partially offsetting the gains obtained from finer decomposition and explaining the trend in the top-left plot.

For the remaining settings, this selection effect is weaker because the IR generally varies less across values of $m_{\max}$. Overall, despite this limitation, the predominant trend indicates that allowing finer decompositions improves reconstruction quality.

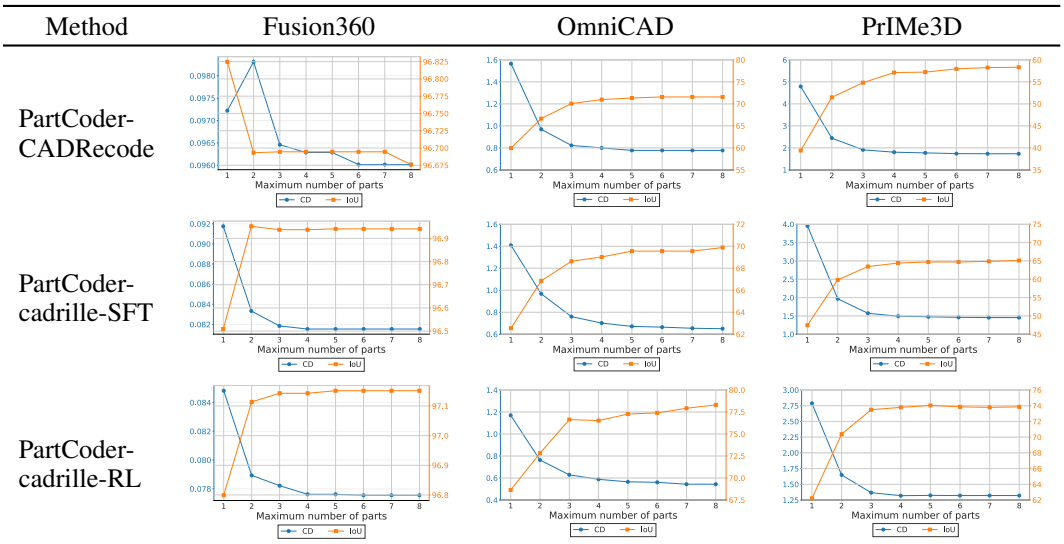


Figure 3: **Ablation on the maximum number of parts.** Median CD (↓, solid) and IoU (↑, dashed) across backbone variants, datasets, and values of $m_{\max}$. Overall, reconstruction quality tends to improve with $m_{\max}$ despite the validity-induced expansion of the evaluated subset.

B.2 Distribution of Selected Part Counts

Figure 4 shows the distribution of final part counts across objects. Most Fusion360 objects remain single-part reconstructions, whereas OmniCAD and particularly PrIMe3D exhibit broader distributions shifted toward higher part counts. This is consistent with the averages in Table 2, suggesting that finer decompositions are selected more frequently for the more challenging benchmarks. These dataset-specific patterns remain consistent across all three backbone variants, suggesting that the selected granularity is strongly influenced by the input geometry. Moreover, the maximum of eight parts is rarely selected, indicating that adaptive decomposition typically terminates before reaching the predefined limit.

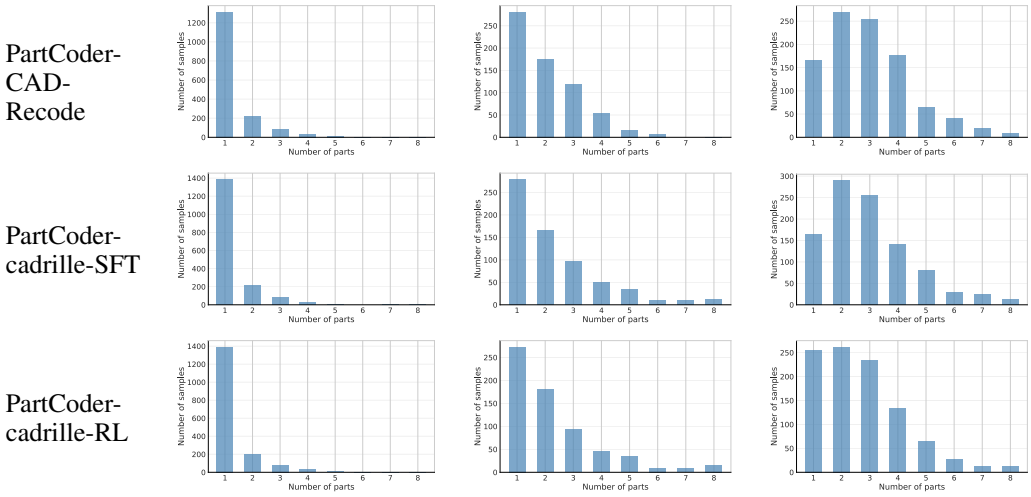


Figure 4: **Distribution of selected part counts.** Number of objects reconstructed with each final part count across backbone variants and datasets. More complex datasets favor finer decompositions, with consistent trends across backbones.

C More Quantitative Evaluation

C.1 Wilcoxon Signed-Rank Tests

To assess the statistical significance of PartCoder's geometric improvements, we perform one-sided paired Wilcoxon signed-rank tests between each PartCoder variant with its monolithic baseline. The alternative hypotheses test whether PartCoder achieves lower CD and higher IoU. Each test includes only samples that are successfully compiled by both methods, ensuring that the comparison is performed on the same set of objects. This isolates geometric quality among jointly valid reconstructions, while differences in compilation success are measured separately by IR. Table 4 reports the resulting p-values. All values are below the significance threshold $\alpha = 0.01$, providing strong statistical evidence against the respective null hypotheses and in support of the observed geometric improvements.

Method	**Fusion360**		**OmniCAD**		**PrIMe3D**	
	CD p-value	IoU p-value	CD p-value	IoU p-value	CD p-value	IoU p-value
CAD-Recode	1.48×10^{-45}	1.34×10^{-19}	9.22×10^{-59}	1.16×10^{-26}	2.25×10^{-136}	8.76×10^{-76}
cadrille-sft	1.59×10^{-56}	5.68×10^{-19}	1.01×10^{-61}	9.58×10^{-27}	1.33×10^{-134}	2.59×10^{-76}
cadrille-rl	1.90×10^{-56}	6.03×10^{-27}	6.04×10^{-64}	5.20×10^{-36}	2.62×10^{-111}	1.68×10^{-92}

Table 4: One-sided paired Wilcoxon signed-rank test p-values comparing the selected PartCoder reconstructions with their respective monolithic baselines on samples successfully compiled by both methods. The results provide strong statistical evidence for these improvements.

C.2 Evaluation on Jointly Valid Subsets

Table 5 reports results on the jointly valid samples used for the Wilcoxon tests. For each PartCoder variant and its monolithic baseline, metrics are computed on the same subset, with corresponding rows placed consecutively for direct comparison. Since the evaluated samples are identical, we additionally report mean CD. Across all pairs, PartCoder achieves lower CD and higher IoU than its corresponding baseline.

Method	Fusion360			OmniCAD			PrIMe3D		
	med. CD↓	mean CD↓	IoU↑ (%)	med. CD↓	mean CD↓	IoU↑ (%)	med. CD↓	mean CD↓	IoU↑ (%)
CAD-Recode	0.10	1.47	86.6	1.57	3.86	52.4	4.79	5.90	39.2
PartCoder-CAD-Recode	**0.09**	**0.58**	**88.8**	**0.74**	**1.70**	**60.5**	**1.73**	**2.36**	**52.9**
cadrille-sft	0.09	0.79	86.6	1.41	2.87	54.2	3.96	4.70	44.9
PartCoder-cadrille-sft	**0.08**	**0.51**	**88.4**	**0.68**	**1.48**	**62.7**	**1.46**	**2.07**	**57.0**
cadrille-rl	0.08	0.64	89.8	1.17	2.50	60.4	2.79	3.60	55.5
PartCoder-cadrille-rl	**0.08**	**0.41**	**91.3**	**0.54**	**1.28**	**67.7**	**1.32**	**1.88**	**62.9**

Table 5: **Joint-subset evaluation.** Results for each PartCoder variant and its monolithic baseline on samples successfully compiled by both methods, enabling direct comparison, including mean CD.

C.3 Increased Inference-Time Sampling

PartCoder generates $R = 10$ candidates for each reconstructed part, requiring more backbone calls than a monolithic reconstruction with $R = 10$. To test whether additional sampling alone accounts for its gains, Table 6 evaluates the monolithic backbones with $R \in 10, 30, 50$ candidates per object. PartCoder matches or improves CD and IoU in every comparison except CAD-Recode on Fusion360 at $R = 30$ and 50. In that case, CAD-Recode achieves lower CD and higher IoU, but has a substantially higher invalidity ratio: 6.9% at $R = 50$, compared with 2.9% for PartCoder. Because CD and IoU are computed only for valid reconstructions, those geometric scores reflect different subsets of objects. Overall, increasing the monolithic models sampling budget does not reproduce PartCoder's gains across the benchmarks.

Method	Fusion360			OmniCAD			PrIMe3D		
	CD↓	IoU↑ (%)	IR↓ (%)	CD↓	IoU↑ (%)	IR↓ (%)	CD↓	IoU↑ (%)	IR↓ (%)
CAD-Recode (R=10)	0.10	86.6	8.3	1.57	52.3	4.7	4.79	39.2	0.1
CAD-Recode (R=30)	**0.09**	87.4	7.1	1.08	55.6	3.5	3.87	41.2	**0**
CAD-Recode (R=50)	**0.09**	**87.5**	6.9	0.94	55.7	3.0	3.59	42.5	**0**
PartCoder-CAD-Recode	0.10	86.6	**2.9**	**0.77**	**59.2**	**0.2**	**1.73**	**52.9**	**0**
cadrille-SFT (pc) (R=10)	0.09	86.6	0.1	1.41	54.2	2.1	3.96	44.9	**0**
cadrille-SFT (pc) (R=30)	0.09	86.8	0.1	1.06	56.8	1.5	3.34	47.4	**0**
cadrille-SFT (pc) (R=50)	0.09	87.6	**0**	0.90	57.2	1.5	3.04	47.0	**0**
PartCoder-cadrille-SFT	**0.08**	**88.3**	**0**	**0.65**	**63.1**	**0**	**1.46**	**57.0**	**0**
cadrille-RL (pc) (R=10)	**0.08**	89.8	**0**	1.17	60.4	**0**	2.80	55.5	**0**
cadrille-RL (pc) (R=30)	**0.08**	90.3	**0**	0.89	62.5	**0**	2.47	56.0	**0**
cadrille-RL (pc) (R=50)	**0.08**	90.4	**0**	0.82	62.8	**0**	2.35	56.7	**0**
PartCoder-cadrille-RL	**0.08**	**91.3**	**0**	**0.54**	**67.7**	**0**	**1.32**	**62.9**	**0**

Table 6: Comparison with increased inference-time sampling. Monolithic backbones use $R \in 10, 30, 50$ candidates per object; PartCoder uses $R = 10$ per reconstructed part.

D MORE QUALITATIVE RESULTS

Figures 5, 6, and 7 present additional qualitative results on Fusion360, OmniCAD, and PrIMe3D, respectively.

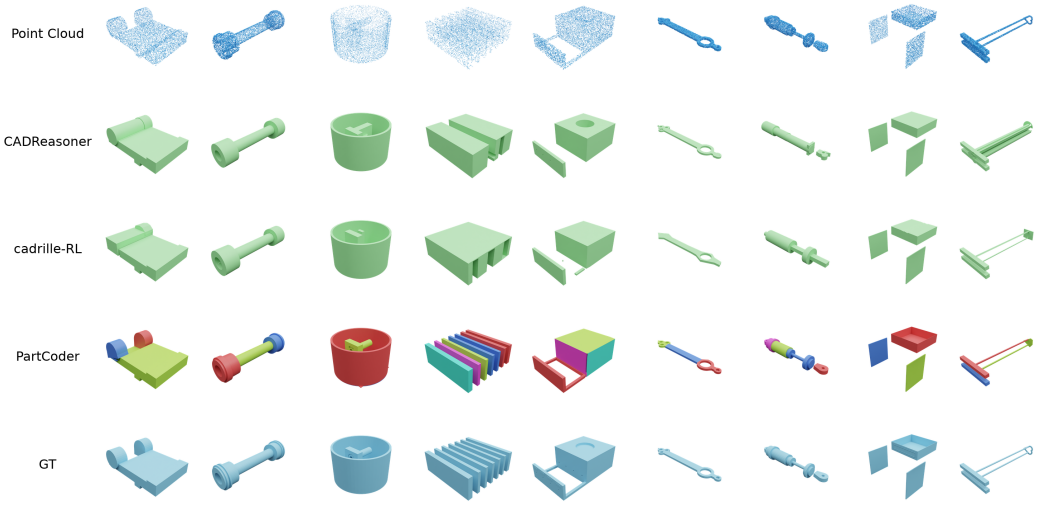


Figure 5: Additional qualitative results on **Fusion360**.

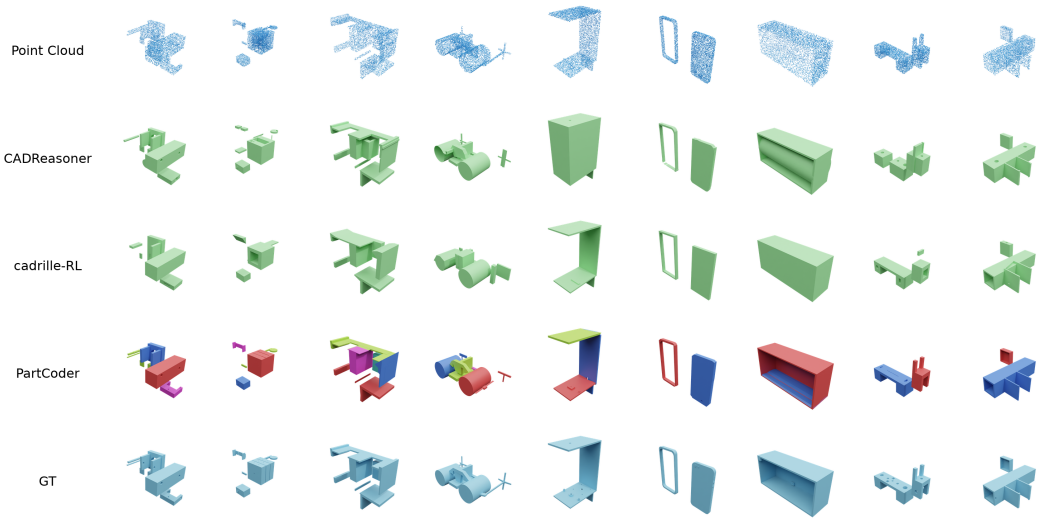


Figure 6: Additional qualitative results on **OmniCAD**.

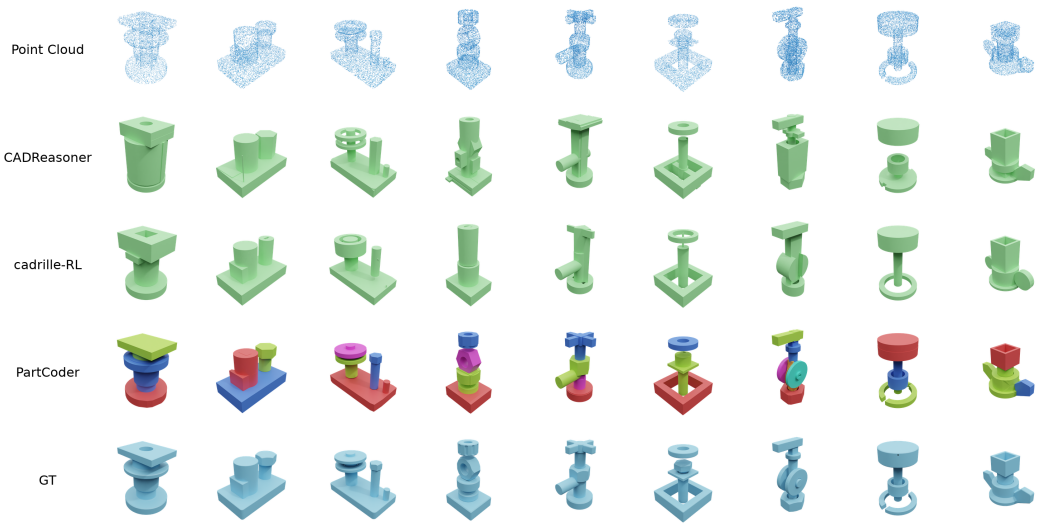


Figure 7: Additional qualitative results on **PrIMe3D**.

E METRIC DISTRIBUTION PLOTS

Figures 8, 9, and 10 show the per-sample metric distributions underlying the aggregate results in Tab. 1. In addition to CD and IoU, we report the Hausdorff distance (HD) and Dice similarity coefficient (DSC).

Each histogram shows the metric values on the x-axis and the number of samples per bin on the y-axis, using 50 uniformly spaced bins. The mean and median are indicated in each plot. Within each dataset-metric pair, the x-axis limits and bin widths are fixed across methods to enable direct comparison. Because the general-purpose LLM baselines span substantially different value ranges, we use separate x-axis limits and bin widths to capture their full distributions while keeping these settings consistent across the LLM baselines.

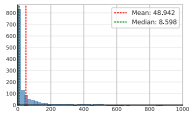
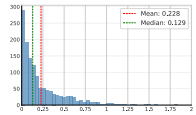
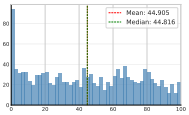
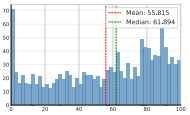
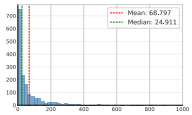
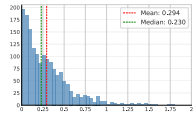
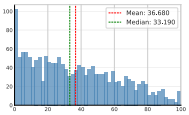
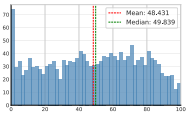
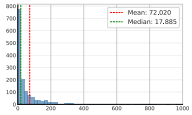
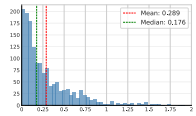
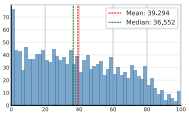
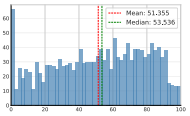
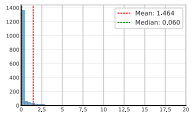
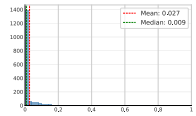
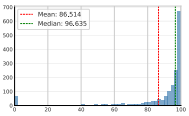
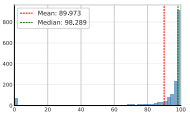
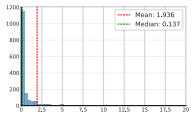
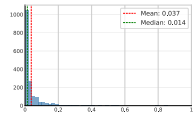
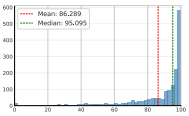
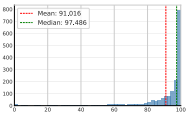
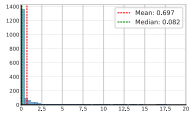
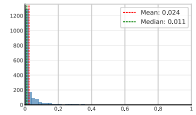
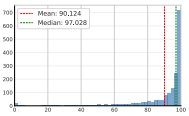
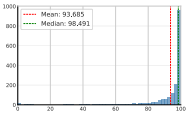
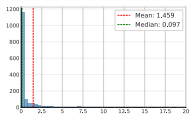
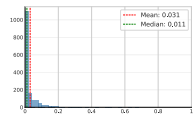
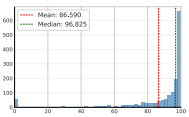
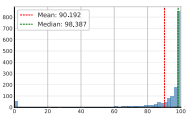
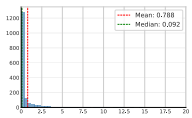
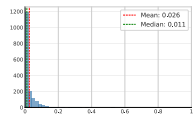
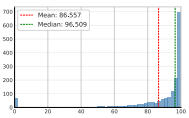
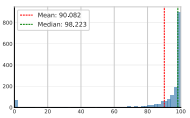
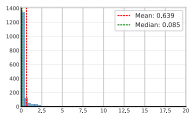
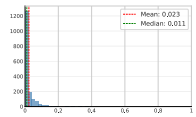
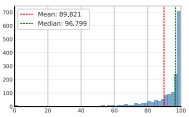
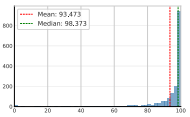
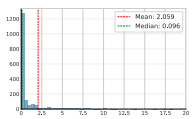
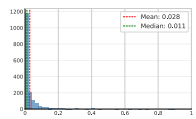
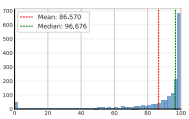
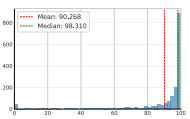
Method	CD↓	HD↓	IoU↑(%)	Dice↑(%)
Gemini-Flash 3.5				
Claude Sonnet 4.6				
GPT 5.4				
CADReasoner				
cadrille-RL (mv)				
cadrille-RL (mv+pc)				
CAD-Recode				
Cadrille-SFT (pc)				
cadrille-RL (pc)				
PartCoder-CAD-Recode				
PartCoder-cadrille-SFT				
PartCoder-cadrille-RL				

Figure 8: Per-sample performance distributions for the results reported in Table 1, together with the additional Hausdorff distance and Dice score metrics, on the **Fusion360** dataset.

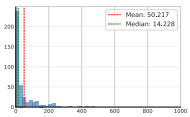
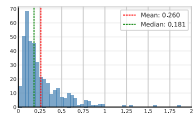
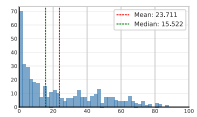
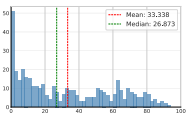
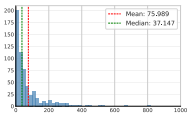
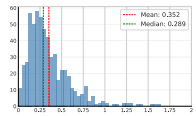
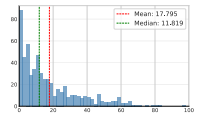
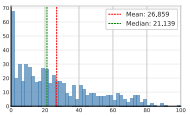
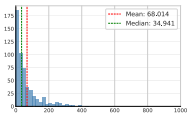
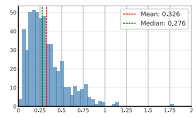
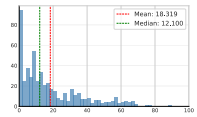
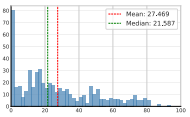
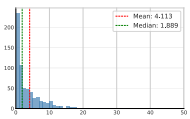
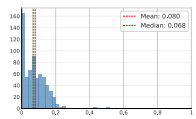
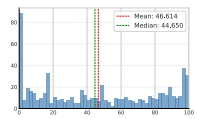
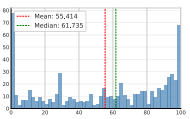
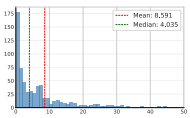
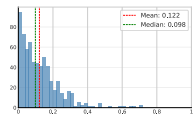
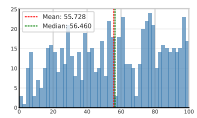
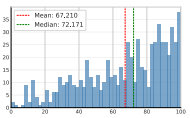
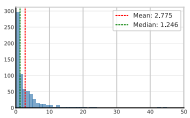
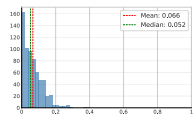
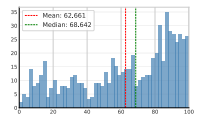
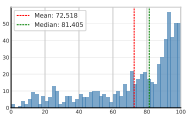
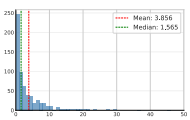
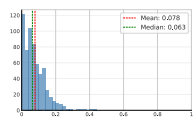
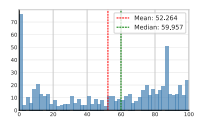
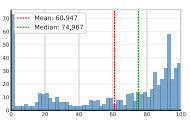
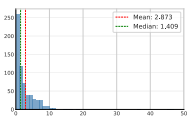
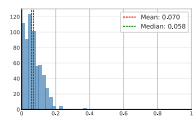
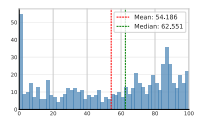
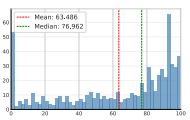
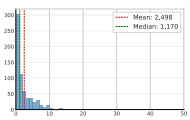
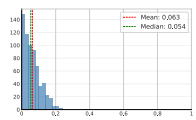
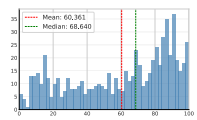
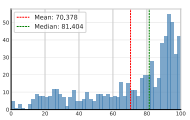
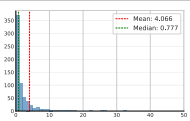
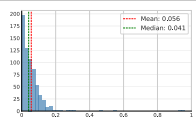
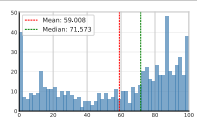
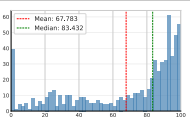
Method	CD↓	HD↓	IoU↑(%)	Dice↑(%)
Gemini-Flash 3.5				
Claude Sonnet 4.6				
GPT 5.4				
CADReasoner				
cadrille-RL (mv)				
cadrille-RL (mv+pc)				
CAD-Recode				
Cadrille-SFT (pc)				
cadrille-RL (pc)				
PartCoder-CAD-Recode				
PartCoder-cadrille-SFT				
PartCoder-cadrille-RL				

Figure 9: Per-sample performance distributions for the results reported in Table 1, together with the additional Hausdorff distance and Dice score metrics, on the **OmniCAD** dataset.

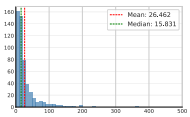
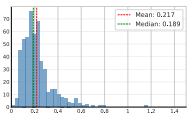
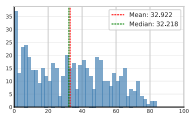
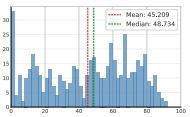
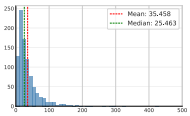
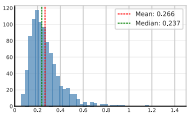
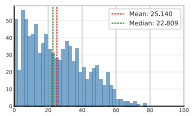
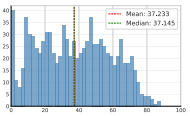
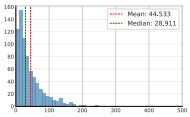
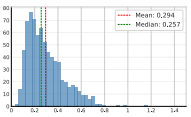
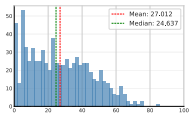
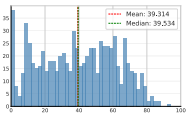
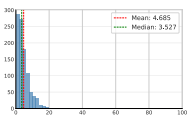
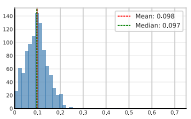
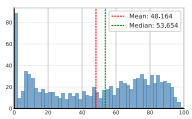
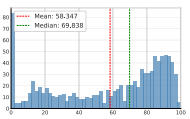
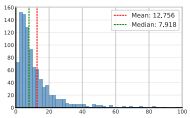
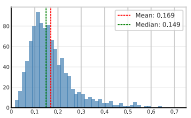
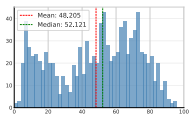
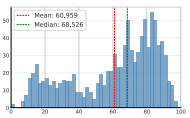
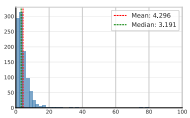
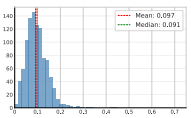
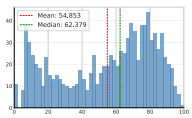
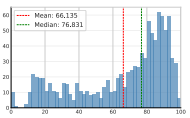
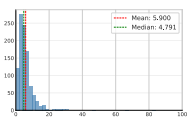
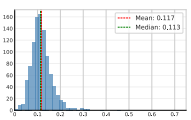
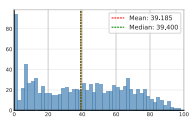
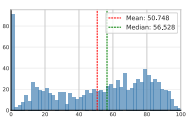
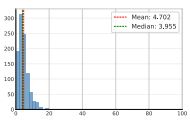
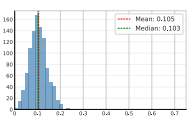
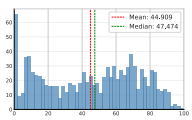
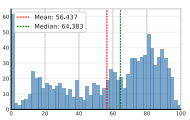
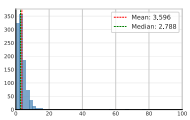
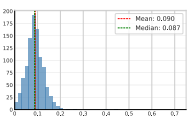
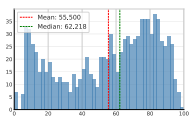
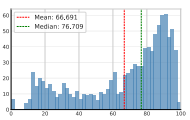
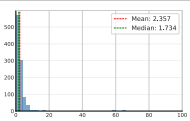
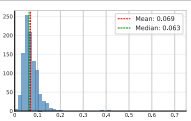
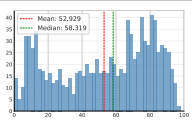
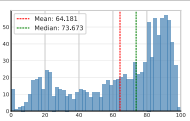
Method	CD↓	HD↓	IoU↑(%)	Dice↑(%)
Gemini-Flash 3.5				
Claude Sonnet 4.6				
GPT 5.4				
CADReasoner				
cadrille-RL (mv)				
cadrille-RL (mv+pc)				
CAD-Recode				
Cadrille-SFT (pc)				
cadrille-RL (pc)				
PartCoder-CAD-Recode				
PartCoder-cadrille-SFT				
PartCoder-cadrille-RL				

Figure 10: Per-sample performance distributions for the results reported in Table 1, together with the additional Hausdorff distance and Dice score metrics, on the **PrIMe3D** dataset.