BrepCoder: A Unified Multimodal Large Language Model for Multi-task B-rep Reasoning

Mingi Kim, Yongjun Kim, Jungwoo Kang, and Hyungki Kim*

Chungnam National University, Daejeon, Republic of Korea
mingi@o.cnu.ac.kr, kimit@g.cnu.ac.kr
mercurii713@o.cnu.ac.kr, hk.kim@cnu.ac.kr

Abstract. Recent advancements in deep learning have actively addressed complex challenges within the Computer-Aided Design (CAD) domain. However, most existing approaches rely on task-specific models requiring structural modifications for new tasks, and they predominantly focus on point clouds or images rather than the industry-standard Boundary Representation (B-rep) format. To address these limitations, we propose BrepCoder, a unified Multimodal Large Language Model (MLLM) that performs diverse CAD tasks from B-rep inputs. By leveraging the code generation capabilities of Large Language Models (LLMs), we convert CAD modeling sequences into Python-like code and align them with B-rep. We then adopt a two-stage training strategy: First, pre-training on reverse engineering to learn geometric features and design logic. Second, effectively extending the model to various downstream tasks such as completion, error correction, and CAD-QA. Consequently, by interpreting B-rep as structural code, BrepCoder achieves superior generalization across diverse tasks, demonstrating its potential as a general-purpose CAD agent.

Keywords: Computer-Aided Design · Boundary Representation · Multimodal Large Language Model

1 Introduction

Computer-Aided Design (CAD) is a fundamental technology in modern manufacturing and engineering, essential for defining geometric shapes and designing product prototypes. Driven by the increasing demand for maximizing design efficiency and establishing intelligent automation, recent research has actively leveraged deep learning techniques to address complex CAD challenges using multi-modal inputs such as text, images, point clouds, and Boundary Representations (B-rep) [7, 20, 41, 46]. Notably, extensive efforts have been dedicated to solving critical industrial tasks, including reverse engineering to recover design history [19, 43], completion of incomplete modeling sequences [18], automatic

* Corresponding author.

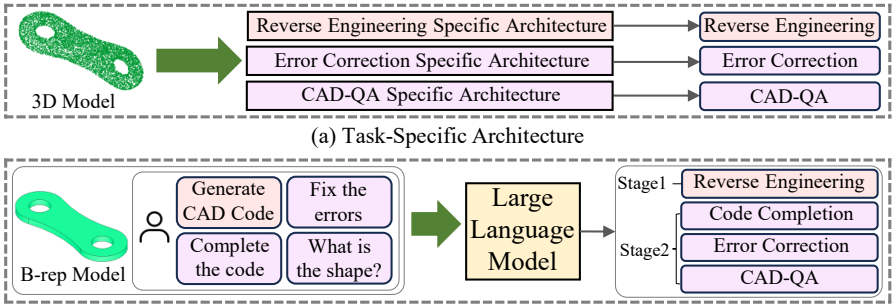


Fig. 1: Comparison of CAD Learning Frameworks. (a) Task-Specific Architecture: Models that adopt different architectures depending on the task. (b) BrepCoder: A unified MLLM framework that employs a two-stage training strategy.

error correction [6], and question answering to facilitate the semantic understanding of design intent in 3D models.

Despite these advancements, the majority of existing deep learning models in the CAD domain remain confined to task-specific architectures designed for individual tasks [12], as illustrated in Fig. 1(a). This approach is inefficient in terms of scalability, as it necessitates structural modifications whenever a new task or design environment is introduced. Consequently, these task-specific models fail to share mutual knowledge across different tasks throughout the design process, lacking the generalization capabilities needed to flexibly address the complex requirements of real-world industrial applications.

The advent of Multimodal Large Language Models (MLLMs) presents a promising avenue to address these challenges. Leveraging the extensive pre-trained knowledge of Large Language Models (LLMs), MLLMs have demonstrated versatile reasoning and generalization capabilities [1–3, 45]. They can integrate and process diverse modalities, such as images, point clouds, and B-reps, alongside natural language instructions [39]. Consequently, combining the general-purpose reasoning of LLMs with modality-specific information enables the model to learn the correlations between CAD geometry and design logic [11, 22]. This synergy facilitates the construction of a unified framework capable of performing a wide range of CAD tasks within a single MLLM architecture.

From the perspective of modality selection, traditional deep learning research in CAD has predominantly focused on representations such as images and point clouds [5, 21, 44]. Although recent studies have begun exploring B-reps for CAD sequence recovery [19, 43] or as inputs for MLLMs [9], the scope of this research remains limited compared to image or point cloud based methodologies. However, given that the vast majority of industrial CAD models are stored and shared in the B-rep format, developing methodologies that directly utilize them is of great practical significance [19, 46]. Despite this importance, a general-purpose

MLLM designed to comprehensively address diverse CAD tasks using B-rep data is still lacking. To bridge this gap, we propose a unified framework that adopts B-rep as the core input modality to solve various tasks within the CAD domain.

While there have been attempts to integrate B-rep with LLMs, existing B-rep-based MLLMs have been largely confined to tasks that describe geometric characteristics or identify shape categories by aligning B-reps with natural language captions [9]. Although this approach may help the model grasp the external geometric features, it merely provides an understanding of the final output geometry, failing to capture the specific procedural steps required to construct it. Consequently, without a concrete understanding of both geometry and design logic, these models face fundamental limitations in executing diverse CAD tasks that demand high level engineering reasoning, such as generating modeling sequences or correcting errors.

To address these challenges, we propose BrepCoder, illustrated in Fig. 1(b). Our framework aligns B-rep data with CAD sequences to leverage the generalization capabilities of LLMs for complex tasks. Specifically, we first convert modeling sequences into Python-like CAD code [18], allowing the LLM to utilize its robust programming reasoning for effective interpretation. Next, after aligning B-rep with the converted CAD code, we implement a two-stage training strategy for progressive learning. Stage 1 focuses on a reverse engineering task, where the model internalizes the fundamental correlations between geometric features and design logic. In Stage 2, the model leverages these learned correlations to address various downstream tasks, including completion, error correction, and CAD-QA. Our main contributions are summarized as follows:

1. We propose BrepCoder, a unified framework adopting B-rep as its core input modality. Aligning these geometric representations with Python-like CAD code enables the LLM to directly interpret complex design logic.
2. We introduce a Two-Stage Training Strategy to learn the correlations between B-reps and CAD code. Pre-training on reverse engineering allows the model to internalize design logic, securing strong generalization capabilities for various downstream tasks.
3. Leveraging the acquired design logic, BrepCoder demonstrates higher performance across complex CAD domain tasks requiring high-level engineering reasoning, including reverse engineering, completion, error correction, and CAD-QA.

2 Related Work

Deep learning for 3D CAD reconstruction. Recent 3D CAD generation research favors parametric modeling incorporating design history. DeepCAD [33] pioneered formulating this process as sequential operations generated via a Transformer-based auto-encoder [4,30], a paradigm subsequently extended across diverse modalities. For instance, MultiCAD [21] improved reconstruction performance by employing a contrastive learning strategy to capture the relationship between point clouds and CAD sequences. Img2CAD [7] proposed a two-stage

approach, utilizing GPT-4V [34] to predict the global structure while employing a separate module for parameter estimation. Furthermore, Brep2Seq [43] interpreted B-reps as graph structures to predict modeling sequences via a Transformer encoder-decoder, and CADCL [19] enhanced reconstruction fidelity by introducing a contrastive learning framework to align B-rep and CAD sequence embeddings. However, existing methods predominantly rely on rigid, task-specific architectures trained from scratch for isolated reconstruction tasks.

Large language models for CAD. Recently, CAD design has increasingly adopted a linguistic perspective, leveraging the code generation capabilities of LLMs [42]. Text2CAD [16] pioneered CAD generation from natural language prompts, and CAD-Llama [18] translated CAD sequences into Structured Parametric CAD code (SPCC) to perform diverse tasks such as text-to-CAD, completion, addition, and deletion. Beyond natural language instruction, research on MLLMs that incorporate visual information is progressing. CAD-GPT [31] reconstructed design histories from image and text inputs. CAD-MLLM [36] integrated point cloud data alongside text and images for history reconstruction, while CADReview [6] proposed an error correction task to rectify flawed CAD programs using image inputs. More recently, efforts have emerged to extend LLM comprehension beyond images and point clouds to B-rep data. BrepLLM [9] demonstrated that LLMs can interpret B-reps through a training strategy that aligns them with natural language captions. However, its capabilities are strictly confined to descriptive tasks, such as classification and captioning.

3 Method

Fig. 2 illustrates the BrepCoder framework. Phase 1 aligns B-rep representations with the corresponding CAD code. Phase 2 integrates these aligned modalities into an LLM, training the unified architecture through a two-stage strategy.

3.1 Phase 1: Multimodal Alignment

CAD code representation. Conventional methods like DeepCAD [33] represent the sequential modeling operations inherent to CAD generation using custom-defined integer tokens. Recently, however, translating these sequences into code formats has gained traction to leverage the pre-trained programming knowledge of LLMs [6,11,18]. To maximize the reasoning capabilities of the LLM, we adopt the Python-like CAD code format proposed by CAD-Llama [18]. Fig. 3 compares the simple integer token sequence used in DeepCAD with our code-based representation. As illustrated in Fig. 3(b), the proposed method adheres to object-oriented programming syntax and exhibits the following structural characteristics. (1) Objectification: Each 2D sketch is declared and managed as a unique variable (*e.g.*, `sketch_0 = []`). (2) Method Invocation: Geometric commands such as `Line`, `Arc`, and `Circle` are executed as method calls associated with the corresponding sketch object, with their specific parameters passed as arguments. (3) Explicit Reference: The `Extrude` operation, which generates the

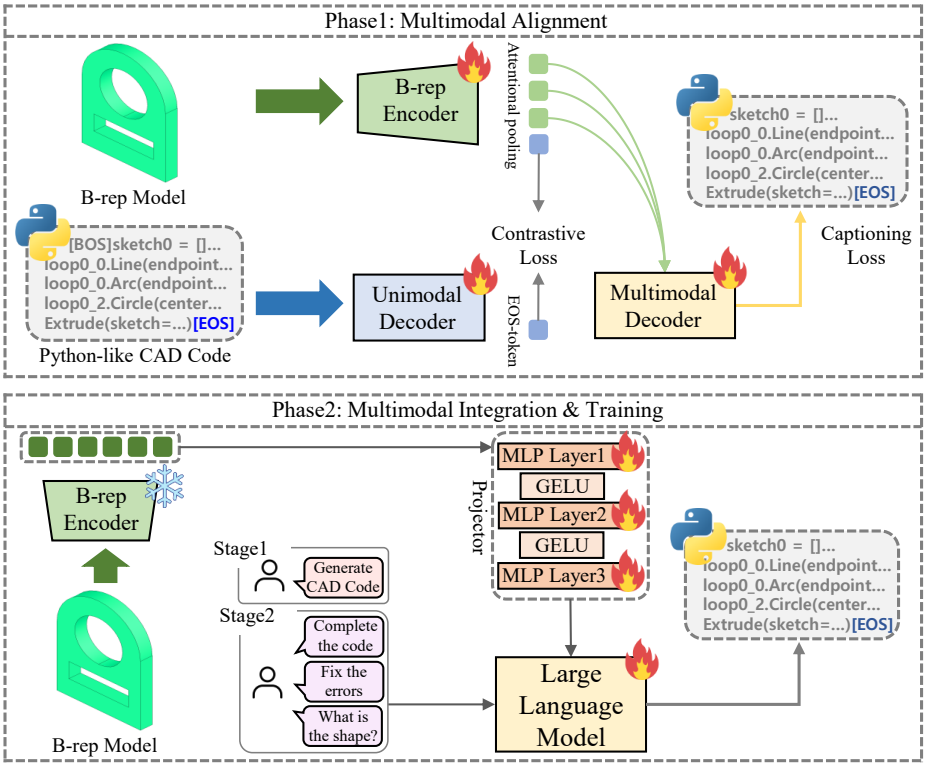


Fig. 2: The overall framework of BrepCoder. In Phase 1, B-rep representations are aligned with CAD code. In Phase 2, the frozen B-rep encoder is integrated with the LLM via a projector. Employing a Two-Stage Training strategy, Stage 1 captures the correspondence between geometric features and code through a reverse engineering task, while Stage 2 fine-tunes the model for diverse downstream tasks.

3D shape, takes the previously defined sketch variable as an explicit argument (*e.g.*, `sketch=sketch_0`), along with its specific extrusion parameters.

B-rep feature encoding. Unlike discrete 3D representations such as point clouds and voxels, B-rep defines 3D shapes through continuous geometries and their topological connections. To encode this representation, some methodologies, such as BrepNet [17], utilize heterogeneous graphs that treat points, curves, and surfaces as individual nodes. However, this approach significantly increases graph complexity and degrades computational efficiency. Therefore, we adopt UV-Net [13] as our encoder, which simplifies the topological structure by configuring surfaces as single nodes. Following the methodology of UV-Net, we convert the B-rep data into a face-adjacency graph, where nodes represent surfaces and edges denote the adjacency between connected surfaces. The geometric information of surfaces and curves is projected onto 2D and 1D grids, respectively. These projections are extracted as local geometric features via 2D and 1D convolution

B-rep Model

[SOL] 2 176 128 48 5 128 128 128 128 128 128 38 224 128 0 0 [EOS]

(a) DeepCAD Representation

```
sketch0 = []
loop0_0 = Loop()
loop0_0.Circle(center=(48, 0), radius=48)
sketch0.add(loop0_0)
Extrude(sketch=sketch0, orientation_degrees=(0, 0, 0), origin=(0, 0,
0), scale=0.27, extrude_distance=(96, 0), boolean_type='New',
extent_type='One-sided')
```

(b) Python-like CAD Code Representation

Fig. 3: Comparison of CAD Representations. (a) DeepCAD representation composed of integer tokens. (b) Python-like CAD code representation that explicitly describes design logic.

operations and subsequently integrated across the entire topological structure through a Graph Neural Network (GNN) [35]. Specifically, for a B-rep input with N surfaces, UV-Net generates a local feature embedding $E_{node} \in \mathbb{R}^{N\times 128}$ for each node and a global feature embedding $E_{shape} \in \mathbb{R}^{1\times 128}$ obtained through pooling. Finally, we concatenate these two embeddings to obtain the final B-rep feature, $z_{brep} \in \mathbb{R}^{N\times 256}$. Consequently, z_{brep} encapsulates both the fine-grained geometric details of each individual surface and the holistic global context of the entire 3D model.

CoCa-based multimodal alignment. Conventional contrastive learning-based methodologies such as CLIP [26] excel at performing global alignment between two modalities and are effective for tasks like retrieval and classification. Consequently, contrastive learning is an effective approach for distinguishing the inherent geometric differences among various B-rep models and aligning the unique features of each shape in a one-to-one manner with the corresponding CAD code. However, the ultimate objective of this study is not limited to classification. Our goal is to accurately generate CAD code based on B-rep features when integrated with an LLM. Therefore, the encoder must be explicitly guided to capture not only the global features but also the fine-grained geometric details of the shapes. To achieve this, we adopt the CoCa [40] framework, which integrates a contrastive loss for representation alignment and a captioning loss for code generation.

First, we apply the cascaded attentional pooling mechanism of CoCa to the feature z_{brep} obtained from the B-rep encoder. This approach initially extracts features for code generation and subsequently summarizes them to derive features for contrastive learning. Specifically, $z_{brep} \in \mathbb{R}^{N\times 256}$ is first passed through a linear layer to project it into $z'_{brep} \in \mathbb{R}^{N\times 512}$. To facilitate fine-grained code generation, we then define $Q_{gen} \in \mathbb{R}^{256\times 512}$, which consists of $n_{query} = 256$ randomly initialized learnable queries with a hidden dimension of $d = 512$. Finally, we extract the generation feature z_{gen} via Multi-Head Attention by utilizing z'_{brep} as both the key and the value:

$$z_{gen} = \text{MultiHeadAttn}(Q_{gen}, z'_{brep}, z'_{brep}) \in \mathbb{R}^{256\times 512}. \tag{1}$$

Subsequently, for contrastive learning, we define a randomly initialized learnable query $Q_{con} \in \mathbb{R}^{1\times 512}$ with $n_{query} = 1$. Using z_{gen} generated from Eq. (1) as both the key and the value, we then extract the global feature z_{con}, which encapsulates the entire B-rep geometry:

$$z_{con} = \text{MultiHeadAttn}(Q_{con}, z_{gen}, z_{gen}) \in \mathbb{R}^{1\times 512}. \tag{2}$$

The extracted features are jointly trained with a Unimodal Decoder and a Multimodal Decoder. The Unimodal Decoder processes the CAD code input exclusively and shares an identical architecture with a Transformer Encoder. The Multimodal Decoder features an architecture identical to a Transformer Decoder and generates the CAD code by interacting with the extracted B-rep feature z_{gen}. Under this framework, the model is optimized using two distinct loss functions. First, we introduce a contrastive loss $\mathcal{L}_{con}$ to establish a global alignment between the B-rep and the CAD code. This loss is computed using the `[EOS]` token embedding t_{eos} of the CAD code processed through the Unimodal Decoder and the global B-rep feature z_{con} derived in Eq. (2). The exact formulation is defined as follows:

$$\mathcal{L}_{con} = -\frac{1}{2}\sum_{i}\left[\log\frac{e^{\text{sim}(z_{con}^{(i)},t_{eos}^{(i)})/\eta}}{\sum_j e^{\text{sim}(z_{con}^{(i)},t_{eos}^{(j)})/\eta}} + \log\frac{e^{\text{sim}(t_{eos}^{(i)},z_{con}^{(i)})/\eta}}{\sum_j e^{\text{sim}(t_{eos}^{(i)},z_{con}^{(j)})/\eta}}\right], \tag{3}$$

where $sim(\cdot,\cdot)$ denotes the cosine similarity function, and η represents the temperature parameter. Next, we apply a captioning loss $\mathcal{L}_{cap}$ for code generation. During this process, the Multimodal Decoder generates the CAD code by taking the B-rep feature z_{gen} as the key and value, and the global CAD code feature t_{eos} as the query. The formulation is defined as follows:

$$\mathcal{L}_{cap} = -\sum_{k=1}^{L}\log P(t_k \mid t_{<k}, z_{gen}), \tag{4}$$

where t_k denotes the ground truth CAD code token and $t_{<k}$ represents the preceding CAD code tokens. Finally, the model is optimized through a weighted sum of the two loss functions. The total loss is formulated as follows:

$$\mathcal{L}_{total} = \lambda_{con}\cdot\mathcal{L}_{con} + \lambda_{cap}\cdot\mathcal{L}_{cap}, \tag{5}$$

where λ_{con} and λ_{cap} denote the weights for each respective loss. We adopt the optimal configuration from the CoCa, setting $\lambda_{con} = 1$ and $\lambda_{cap} = 2$.

3.2 Phase 2: Multimodal Integration & Training

Multimodal integration. Through the alignment training, the B-rep encoder is capable of extracting geometric features z_{brep} that are aligned with the CAD code features. To preserve this aligned feature information, we freeze the B-rep encoder and integrate it with the LLM to construct our MLLM architecture.

During this process, we insert a projector module to resolve the discrepancy between the feature space extracted by the encoder and the embedding space of the LLM. The projector takes the encoder output $z_{brep} \in \mathbb{R}^{N\times 256}$ as input and maps it into the LLM space to obtain z_{proj}.

$$z_{proj} = \text{Projector}(z_{brep}) \in \mathbb{R}^{N\times d}, \tag{6}$$

where Projector$(\cdot)$ consists of three linear layers and a GELU activation function. Finally, z_{proj} is concatenated with the text prompt embeddings and fed into the LLM.

Stage 1: Structural logic acquisition via reverse engineering. The first stage involves training the LLM to acquire reverse engineering capabilities, enabling it to interpret B-rep geometries and translate them into explicit modeling sequences. We train the model to generate the corresponding Python-like CAD code by utilizing z_{proj} and the text prompt embeddings as inputs. The loss function for Stage 1, $\mathcal{L}_{stage1}$, is defined as follows:

$$\mathcal{L}_{stage1} = -\sum_{k=1}^{L} \log P(t_k \mid t_{<k}, z_{proj}; \text{prompt}_{reverse}), \tag{7}$$

where t_k denotes the ground truth CAD code token, $t_{<k}$ represents the preceding CAD code tokens, and $\text{prompt}_{reverse}$ indicates the text prompt instructing the model to reconstruct the CAD code from the B-rep input. Through this process, the model internalizes the design logic and procedural knowledge embedded within the geometric information.

Stage 2: Multi-task adaptation for downstream applications. The second stage leverages the design logic capabilities acquired in Stage 1 to address diverse tasks within the CAD domain. Using the pre-trained weights as initialization, we fine-tune the model across three downstream tasks: (1) Completion: The model receives a B-rep input alongside a CAD code sequence with its latter portion masked, and reconstructs the complete CAD code. (2) Error Correction: Given a CAD code with permuted command sequences or incorrectly configured parameters, the model identifies logical contradictions against the B-rep geometry to output the corrected code. (3) CAD-QA: Presented with a B-rep and a question regarding its properties or design intent, the model interprets the geometric information to select the correct answer from four multiple-choice options.

4 Experiments

4.1 Experimental Setups

Datasets. We utilize the DeepCAD [33] dataset, comprising 170K sequences. We convert the original sequences of DeepCAD into the Python-like CAD code format and split into 90:5:5 for training, validation, and testing. For Phase 1 alignment and Stage 1 reverse engineering, we employ B-rep and complete CAD

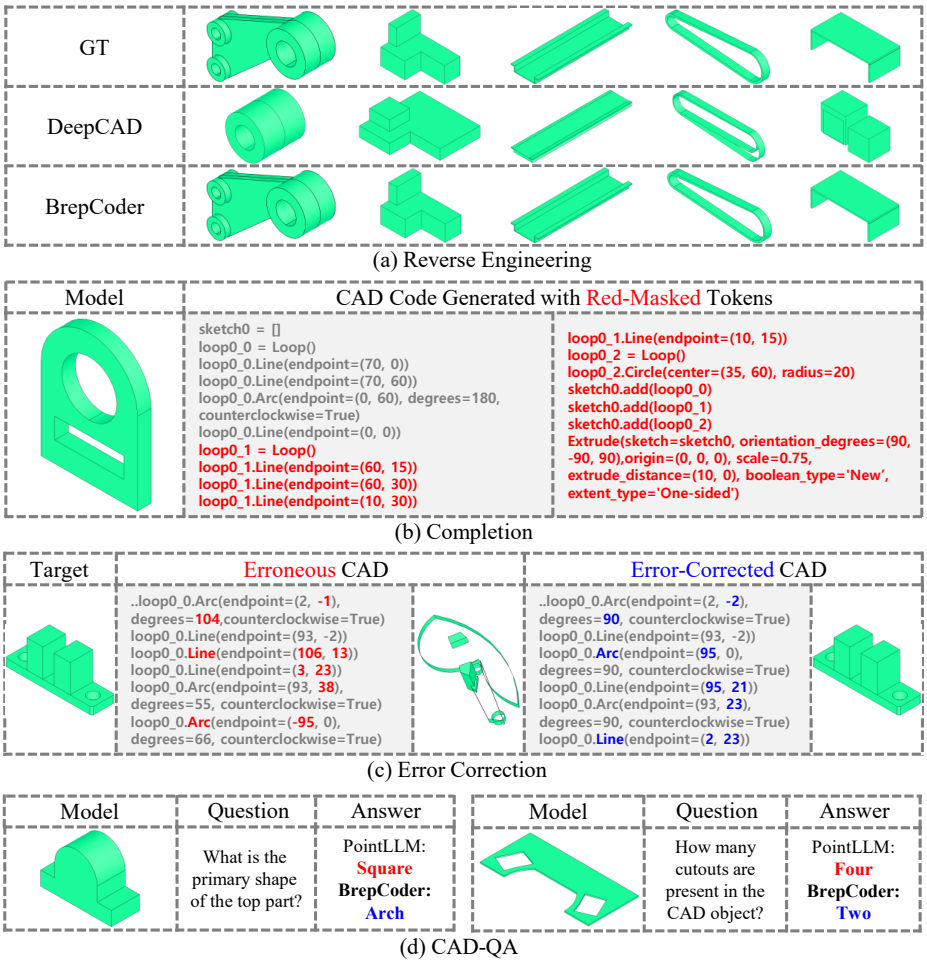


Fig. 4: Qualitative results of BrepCoder across various CAD domain tasks: (a) Reverse engineering, (b) Completion, (c) Error correction, and (d) CAD-QA.

code pairs. Stage 2 data is processed specifically for each task. For completion, we retain the first 30% to 50% of the sequence and mask the rest. For error correction, we permute command orders within loops or inject parameter noise, applying a 50% to 80% ratio. For CAD-QA, we use the SGP-Bench [25] dataset containing 1K models and multiple-choice questions split 80:10:10.

Metrics. To evaluate the alignment between the generated CAD code and the ground truth sequence, we utilize ACC_{cmd} and ACC_{param}. ACC_{cmd} measures the proportion of correctly predicted commands. ACC_{param} calculates the proportion of parameters where the absolute error is within a tolerance δ, considering

Table 1: Comparison of reverse engineering performance on the DeepCAD dataset. Chamfer Distance (CD) is scaled by 10^{-3}. The best results are highlighted in **bold**.

Method	ACC_{cmd} ↑	ACC_{param} ↑	Med. CD ↓	IR ↓	VLM ↑
DeepCAD [33]	85.95	74.22	10.30	12.08	4.08
Brep2Seq [43]	84.45	69.38	31.5	11.52	3.41
CADCL [19]	**90.31**	81.52	0.972	13.98	-
BrepCoder	89.34	**82.01**	**0.464**	**0.86**	**4.41**

only correctly predicted commands.

$$\text{ACC}_{cmd} = \frac{1}{N_c} \sum_{i=1}^{N_c} \mathbb{I}(\hat{c}_i = c_i), \tag{8}$$

$$\text{ACC}_{param} = \frac{1}{K} \sum_{i=1}^{N_c} \sum_{j=1}^{N_p} \mathbb{I}(|\hat{p}_{i,j} - p_{i,j}| < \delta) \cdot \mathbb{I}(\hat{c}_i = c_i), \tag{9}$$

where N_c denotes the total number of commands, and K represents the total number of parameters. $\mathbb{I}(\cdot)$ is an indicator function outputting either 0 or 1. The variables c_i and $\hat{c}_i$ indicate the ground truth and predicted values for the i-th command, respectively. Finally, $p_{i,j}$ and $\hat{p}_{i,j}$ denote the ground truth and predicted values for the j-th parameter of the i-th command.

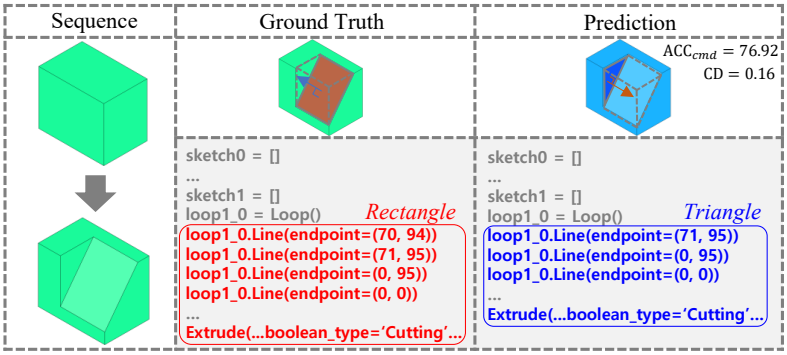
For 3D shape quality, we employ the Chamfer Distance (CD) by randomly sampling 8,096 points and the Invalid Ratio (IR) to verify the structural validity of the generated models. For visual quality assessment, we rendered the generated CAD codes into isometric view images and instructed the VLM (GPT-5) to compare against GT images on a scale of 0 to 5.

Implementation details. We use Qwen 2.5-1.5B-Base [3,38] as our LLM backbone. The Phase 1 unimodal and multimodal decoders each consist of 3 layers, and together with the UV-Net encoder, are trained from scratch. Specific hyperparameter configurations are detailed in the Appendix.

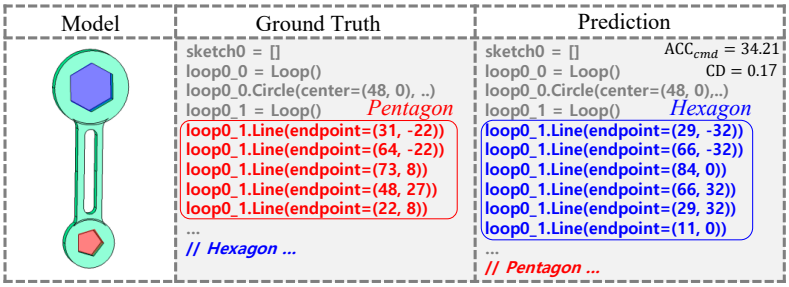
Baselines. We evaluate BrepCoder against state-of-the-art baselines for each task. For reverse engineering, we select CADCL [19] which generates CAD sequences from B-rep inputs. For completion, we compare against CAD-Llama [18] using textual annotations and initial codes. Error correction is evaluated against commercial MLLMs including GPT [28] and Gemini [2, 8]. Additionally, we retrain DeepCAD [33] and Brep2Seq [43] with B-rep inputs for completion and error correction, reconfiguring both to reconstruct CAD sequences by fusing B-rep embeddings with partial or erroneous CAD sequence embeddings. For CAD-QA, we train and compare several point cloud-based MLLMs such as ShapeLLM [24], MiniGPT-3D [29], and PointLLM [37] under identical conditions.

4.2 Reverse Engineering

This section evaluates the reverse engineering capabilities of BrepCoder acquired in Stage 1. Table 1 presents the quantitative evaluation results. First, regarding



(a) Different command counts using a triangle sketch instead of a rectangle.



(b) Different command orders processing the hexagon before the pentagon.

Fig. 5: Analysis of Modeling Ambiguity. In both cases, the model generates geometrically equivalent shapes despite different command counts and orders.

the Chamfer Distance metric, which indicates geometric shape similarity, our model achieves 0.464×10^{-3}. This demonstrates an approximate 52.26% performance improvement over the 0.972 recorded by the previous state-of-the-art model, CADCL [19]. Furthermore, the model records the highest value of 82.01% for the ACC_{param} metric, demonstrating its capability in capturing fine-grained geometric features and translating them into precise numerical parameters. A particularly notable aspect is the reduction in the Invalid Ratio. While existing methodologies exhibit high generation failure rates, BrepCoder records a value of 0.86%, successfully generating executable CAD models in the vast majority of cases. This suggests that the robust programming language comprehension and reasoning capabilities inherent in the pre-trained LLM contribute to ensuring the accuracy of the Python-like CAD code. Additionally, BrepCoder achieves the highest VLM score of 4.41, outperforming DeepCAD (4.08) and Brep2Seq (3.41). This confirms that BrepCoder generates geometrically accurate CAD models that are visually faithful to the ground truth. Qualitative results demonstrating this can be observed in Fig. 4(a). Conversely, regarding the ACC_{cmd}, BrepCoder yields a slightly lower performance of 89.34% compared to the 90.31% achieved by CADCL. However, the CAD modeling process is not unique. Identical 3D shapes can be defined by diverse sequences depending on

the designer's intent or methodology [14, 21, 23]. Fig. 5 illustrates examples of this phenomenon. The first case, depicted in Fig. 5(a), involves a discrepancy in the number of commands. The ground truth sequence sketches a rectangle with four commands to perform a cutting operation that chamfers the edge of a cuboid. In contrast, BrepCoder sketches a triangle with only three commands to produce a geometrically identical shape. In this scenario, the ACC_{cmd} is measured lower strictly due to the difference in the command count relative to the ground truth. The second case, shown in Fig. 5(b), presents a variation in the command order. While the ground truth generates the pentagon first, BrepCoder generates the hexagon first. This permutation in the command sequence also results in a lower ACC_{cmd}. Nevertheless, the generated shapes match the ground truth in both cases, maintaining a low CD. Statistical analysis further supports this ambiguity by showing that 14.14% and 11.97% of results with a CD ≤ 0.01 exhibit an ACC_{cmd} below 90.00 and 80.00, respectively. Consequently, despite the lower ACC_{cmd}, BrepCoder reconstructs reasonably accurate 3D geometries. More examples about modeling ambiguity are detailed in the Appendix.

4.3 Downstream Tasks

Completion. The completion task reconstructs the remaining code from an initial code. Baseline models, including CAD-Llama [18], receive the initial code alongside a textual annotation describing the geometry. For instance, this text might describe: "A central cylinder surrounded by other components..." We also compare against B-rep-based baselines, DeepCAD [33] and Brep2Seq [43], retrained for this task. As presented in Table 2(a), BrepCoder achieves the highest performance across all baselines. Compared to CAD-Llama, natural language text can only roughly approximate geometry and exhibits limitations in encoding precise numerical values and topological information. Furthermore, BrepCoder outperforms the B-rep-based baselines, demonstrating that the effective B-rep-code alignment established in Phase 1 and the design logic acquired through Stage 1 pre-training both contribute to the completion performance. Qualitative results are provided in Fig. 4(b).

Error correction. The error correction task rectifies flawed CAD codes. For Gemini and GPT, which cannot process B-rep inputs directly, we rendered 2D images from a diagonal upper viewpoint (elevation 20°, azimuth 45°). ReCAD [6] processes images from a best view and uses a custom OpenSCAD-based dataset, making direct comparison challenging. We also compare against B-rep-based baselines, DeepCAD [33] and Brep2Seq [43], as in the completion task. As shown in Table 2(a), Gemini and GPT achieve high ACC_{cmd} but suffer from low ACC_{param} and high CD, highlighting the difficulty of inferring precise spatial values, such as absolute coordinates (*e.g.*, `endpoint=(140, 80)`) or exact dimensions (*e.g.*, `radius=48`), from a single 2D image. Although disparate datasets make direct comparison challenging, our model achieves a lower CD than ReCAD, as even an optimal viewpoint cannot overcome the inherent limitations of 2D images such as occlusion. Furthermore, BrepCoder outperforms the B-rep-based baselines. This consistently validates the effectiveness of our

Table 2: Performance comparison on downstream tasks: (a) Completion & Error correction, and (b) CAD-QA.

Task	Method	ACC_{cmd} ↑	ACC_{param} ↑	Med. CD ↓	IR ↓	VLM ↑
Completion	CAD-Llama [18]	73.87	57.14	-	-	-
	DeepCAD [33]	92.40	86.41	1.06	12.48	4.17
	Brep2Seq [43]	89.67	79.85	9.33	10.64	3.78
	BrepCoder	**92.69**	**87.94**	**0.441**	**0.71**	**4.52**
Error Correction	Gemini-2.5 [8]	71.53	42.30	350.30	22.53	-
	GPT-5 [28]	82.83	43.41	399.78	25.84	-
	ReCAD [6]	-	-	1.43	**0.00**	-
	DeepCAD [33]	95.94	88.23	0.747	6.28	4.22
	Brep2Seq [43]	91.06	79.21	5.30	10.23	3.97
	BrepCoder	**99.08**	**93.74**	**0.335**	0.32	**4.67**

(a) Completion & Error Correction

Method	$ACC_{CAD\text{-}QA}$ ↑
ShapeLLM(7B) [24]	73%
MiniGPT-3D(2.7B) [29]	78%
PointLLM(7B) [37]	**81%**
BrepCoder(1.5B)	79%

(b) CAD-QA

proposed framework. Qualitative results are provided in Fig. 4(c).

CAD-QA. Finally, the CAD-QA task evaluates the question-answering capabilities regarding CAD models by utilizing the SGP-Bench [25] dataset. As presented in the experimental results in Table 2(b), BrepCoder achieves an accuracy of 79%. This represents a performance merely 2% lower than the 81% achieved by PointLLM [37]. While PointLLM is a large 7B parameter model pre-trained on massive 3D-text descriptions and QA data, BrepCoder relies on a compact 1.5B parameter model and is pre-trained solely on the code-based reverse engineering task. Despite these conditions, our model achieves highly competitive performance. Qualitative results are provided in Fig. 4(d).

4.4 Ablation Studies

Table 3(a) presents the ablation study results for the reverse engineering task in Stage 1. First, training the model end-to-end without Phase 1 alignment leads to a significant performance drop across all metrics. This underscores the necessity of B-rep-code alignment prior to LLM integration. Second, in Phase 1, training the encoder from scratch achieves higher performance than initializing it with segmentation pre-trained weights. While the pre-trained encoder is biased toward a segmentation-oriented feature space, training from scratch enables the model to jointly learn representations aligned with both B-rep geometry and CAD code. Third, replacing UV-Net [13] with AAGNet [32] yields a higher CD. We attribute this to its coarser UV sampling resolution (5×5 vs. 10×10), underscoring the importance of fine-grained geometric encoding for accurate parameter prediction. Fourth, aligning the features by utilizing the CoCa

Table 3: Ablation studies on (a) Reverse engineering and (b) Downstream tasks.

Method	ACC_{cmd} ↑	ACC_{param} ↑	Med. CD ↓	IR ↓
End-to-end	77.58	61.19	124.18	2.13
Pre-trained Encoder	89.39	81.93	0.499	0.92
AAGNet	**92.25**	81.14	3.99	**0.48**
CLIP	88.24	78.09	0.788	1.09
Encoder Unfreeze	89.38	81.30	0.516	0.78
Qwen-2.5 0.5B	88.51	79.77	0.612	1.01
Llama-3.2 1B	88.51	**82.52**	0.467	0.92
2-Layer	88.81	80.15	0.588	0.98
1-Layer	89.13	80.66	0.565	0.92
Ours	89.34	82.01	**0.464**	0.86

(a) Stage 1: Reverse engineering

Tasks	Metric	*No Stage 1*	**Ours**
Completion	ACC_{cmd} ↑	92.53	**92.96**
	ACC_{param} ↑	87.91	**87.94**
Error Correction	ACC_{cmd} ↑	**99.43**	99.08
	ACC_{param} ↑	93.53	**93.74**
	Med. CD ↓	0.405	**0.335**
	IR ↓	0.40	**0.32**
CAD-QA	$ACC_{CAD\text{-}QA}$ ↑	71%	**79%**

(b) Stage 2: Downstream tasks

framework demonstrates higher performance compared to the CLIP [26] method. This confirms the critical role of the captioning loss in generating complex CAD codes. Fifth, freezing the B-rep encoder yields higher performance than leaving it unfrozen. This is likely because joint training of the encoder degrades the pre-trained topological information during the alignment process. Sixth, a comparison between the 1.5B and 0.5B architectures confirms the significance of the LLM scale. The performance degradation observed in the 0.5B model implies an insufficient model capacity. Seventh, Llama-3.2 1B performs comparably to Qwen-2.5 1.5B. This suggests that BrepCoder is robust to the choice of LLM backbone. Finally, regarding the projector, the 3-layer configuration exhibits higher performance than both the 1-layer and 2-layer. This is attributed to its enhanced expressive power in mapping the B-rep features into the LLM textual space.

Table 3(b) compares the impact of the Stage 1 pre-training on the downstream tasks. Our fully trained model demonstrates overall performance improvements across all tasks compared to the baseline omitting this initial stage, denoted as *No Stage 1*. This suggests that the model effectively internalizes the structural logic between the 3D geometry and the CAD code during Stage 1, which subsequently benefits the downstream applications. Although the *No Stage 1* scores marginally higher in the ACC_{cmd} metric for the error correction task, this phenomenon can be interpreted in a similar context as the modeling ambiguity previously discussed in Fig. 5.

5 Conclusion

We proposed BrepCoder, a multimodal framework integrating B-rep representations with Large Language Models for diverse CAD tasks. The first stage of our two-stage strategy establishes semantic alignment through reverse engineering pre-training. Experimental results verify that this foundation enables strong performance across downstream tasks, including completion, error correction, and CAD-QA. For future work, we plan to extend BrepCoder to point cloud inputs, seamlessly encompassing reverse engineering tasks such as CAD-Recode [27], CAD-SIGNet [15], and TransCAD [10]. We expect BrepCoder will advance research in general-purpose CAD agents and intelligent design automation.

Acknowledgements

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (RS-2026-25480008).

References

1. Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., et al.: GPT-4 technical report. arXiv preprint arXiv:2303.08774 (2023). https://doi.org/10.48550/arXiv.2303.08774
2. Anil, R., Borgeaud, S., Alayrac, J.B., Yu, J., Soricut, R., Schalkwyk, J., et al.: Gemini: A family of highly capable multimodal models. arXiv preprint arXiv:2312.11805 (2023). https://doi.org/10.48550/arXiv.2312.11805
3. Bai, J., Bai, S., Chu, Y., Cui, Z., Dang, K., Deng, X., et al.: Qwen technical report. arXiv preprint arXiv:2309.16609 (2023). https://doi.org/10.48550/arXiv.2309.16609
4. Bank, D., Koenigstein, N., Giryes, R.: Autoencoders. In: Machine Learning for Data Science Handbook: Data Mining and Knowledge Discovery Handbook, pp. 353–374. Springer (2023). https://doi.org/10.1007/978-3-031-24628-9_16
5. Chen, C., Wei, J., Chen, T., Zhang, C., Yang, X., Zhang, S., Yang, B., Foo, C.S., Lin, G., Huang, Q., Liu, F.: CadCrafter: Generating computer-aided design models from unconstrained images. In: CVPR. pp. 11073–11082 (2025). https://doi.org/10.1109/cvpr52734.2025.01034
6. Chen, J., Hei, X., Liu, H., Wei, Y., Deng, Z., Xie, J., Cai, Y., Qing, L.: 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). https://doi.org/10.18653/v1/2025.acl-long.489
7. Chen, T., Yu, C., Hu, Y., Li, J., Xu, T., Cao, R., et al.: Img2cad: Conditioned 3-D CAD model generation from single image with structured visual geometry. IEEE Transactions on Industrial Informatics (2025). https://doi.org/10.1109/tii.2025.3584476
8. Comanici, G., Bieber, E., Schaekermann, M., Pasupat, I., Sachdeva, N., Dhillon, I., et al.: Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. arXiv preprint arXiv:2507.06261 (2025). https://doi.org/10.48550/arXiv.2507.06261

9. Deng, L., Guo, H., Bai, Y., Dai, Y., Huang, H., Shi, Y.: BrepLLM: Native boundary representation understanding with large language models. arXiv preprint arXiv:2512.16413 (2025). https://doi.org/10.48550/arXiv.2512.16413
10. Dupont, E., Cherenkova, K., Mallis, D., Gusev, G., Kacem, A., Aouada, D.: TransCAD: A hierarchical transformer for CAD sequence inference from point clouds. In: ECCV. pp. 19–36. Springer Nature Switzerland, Cham (2024). https://doi.org/10.1007/978-3-031-73030-6_2
11. Guan, Y., Wang, X., Xing, X., Zhang, J., Xu, D., Yu, Q.: CAD-Coder: Text-to-CAD generation with chain-of-thought and geometric reward. arXiv preprint arXiv:2505.19713 (2025). https://doi.org/10.48550/arXiv.2505.19713
12. Heidari, N., Iosifidis, A.: Geometric deep learning for computer-aided design: A survey. IEEE Access (2025). https://doi.org/10.1109/access.2025.3587121
13. Jayaraman, P.K., Sanghi, A., Lambourne, J.G., Willis, K.D.D., Davies, T., Shayani, H., Morris, N.: UV-Net: Learning from boundary representations. In: CVPR. pp. 11703–11712 (2021). https://doi.org/10.1109/cvpr46437.2021.01153
14. Jung, M., Kim, M., Kim, J.: ContrastCAD: Contrastive learning-based representation learning for computer-aided design models. arXiv preprint arXiv:2404.01645 (2024). https://doi.org/10.48550/arXiv.2404.01645
15. Khan, M.S., Dupont, E., Ali, S.A., Cherenkova, K., Kacem, A., Aouada, D.: CAD-SIGNet: CAD language inference from point clouds using layer-wise sketch instance guided attention. In: CVPR. pp. 4713–4722 (2024). https://doi.org/10.1109/cvpr52733.2024.00451
16. Khan, M.S., Sinha, S., Sheikh, T.U., Stricker, D., Ali, S.A., Afzal, M.Z.: Text2CAD: Generating sequential CAD designs from beginner-to-expert level text prompts. In: NeurIPS. vol. 37, pp. 7552–7579 (2024). https://doi.org/10.52202/079017-0242
17. Lambourne, J.G., Willis, K.D.D., Jayaraman, P.K., Sanghi, A., Meltzer, P., Shayani, H.: B-repNet: A topological message passing system for solid models. In: CVPR. pp. 12773–12782 (2021). https://doi.org/10.1109/cvpr46437.2021.01258
18. Li, J., Ma, W., Li, X., Lou, Y., Zhou, G., Zhou, X.: CAD-Llama: Leveraging large language models for computer-aided design parametric 3D model generation. In: CVPR. pp. 18563–18573 (2025). https://doi.org/10.1109/CVPR52734.2025.01730
19. Liang, J., He, F., Fan, R., Chu, Y., Yan, X.: CADCL: Reconstruct parametric CAD models from B-rep via contrastive learning. Journal of Computational Design and Engineering **12**(10), 176–184 (2025). https://doi.org/10.1093/jcde/qwaf102
20. Liu, Y., Obukhov, A., Wegner, J.D., Schindler, K.: Point2cad: Reverse engineering CAD models from 3D point clouds. In: CVPR. pp. 3763–3772 (2024). https://doi.org/10.1109/cvpr52733.2024.00361
21. Ma, W., Xu, M., Li, X., Zhou, X.: MultiCAD: Contrastive representation learning for multi-modal 3D computer-aided design models. In: Proceedings of the 32nd ACM International Conference on Information and Knowledge Management. pp. 1766–1776 (2023). https://doi.org/10.1145/3583780.3614982
22. Mews, M., Aynetdinov, A., Schiller, V., Eisert, P., Akbik, A.: Don't Mesh with Me: Generating constructive solid geometry instead of meshes by fine-tuning a code-generation LLM. arXiv preprint arXiv:2411.15279 (2024). https://doi.org/10.48550/arXiv.2411.15279
23. Para, W., Bhat, S., Guerrero, P., Kelly, T., Mitra, N., Guibas, L., Wonka, P.: Sketchgen: Generating constrained CAD sketches. In: NeurIPS. vol. 34, pp. 5077–5088 (2021), https://dl.acm.org/doi/10.5555/3540261.3540649

24. Qi, Z., Dong, R., Zhang, S., Geng, H., Han, C., Ge, Z., Yi, L.: ShapeLLM: Universal 3D object understanding for embodied interaction. In: ECCV. pp. 214–238. Springer Nature Switzerland, Cham (2024). https://doi.org/10.1007/978-3-031-72775-7_13
25. Qiu, Z., Liu, W., Feng, H., Liu, Z., Xiao, T.Z., Collins, K.M., Tenenbaum, J.B., Weller, A., Black, M.J., Schölkopf, B.: Can large language models understand symbolic graphics programs? arXiv preprint arXiv:2408.08313 (2024). https://doi.org/10.48550/arXiv.2408.08313
26. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., Sutskever, I.: Learning transferable visual models from natural language supervision. In: ICML. pp. 8748–8763 (2021), https://proceedings.mlr.press/v139/radford21a.html
27. Rukhovich, D., Dupont, E., Mallis, D., Cherenkova, K., Kacem, A., Aouada, D.: CAD-Recode: Reverse engineering CAD code from point clouds. In: ICCV. pp. 9801–9811 (2025), https://openaccess.thecvf.com/content/ICCV2025/html/Rukhovich_CAD-Recode_Reverse_Engineering_CAD_Code_from_Point_Clouds_ICCV_2025_paper.html
28. Singh, A., Fry, A., Perelman, A., Tart, A., Ganesh, A., El-Kishky, A., et al.: OpenAI GPT-5 system card. arXiv preprint arXiv:2601.03267 (2026). https://doi.org/10.48550/arXiv.2601.03267
29. Tang, Y., Han, X., Li, X., Yu, Q., Hao, Y., Hu, L., Chen, M.: MiniGPT-3D: Efficiently aligning 3D point clouds with large language models using 2D priors. In: Proceedings of the 32nd ACM International Conference on Multimedia. pp. 6617–6626 (2024). https://doi.org/10.1145/3664647.3681257
30. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: NeurIPS. vol. 30 (2017), https://dl.acm.org/doi/10.5555/3295222.3295349
31. Wang, S., Chen, C., Le, X., Xu, Q., Xu, L., Zhang, Y., Yang, J.: CAD-GPT: Synthesising CAD construction sequence with spatial reasoning-enhanced multimodal LLMs. In: AAAI. vol. 39, pp. 7880–7888 (2025). https://doi.org/10.1609/aaai.v39i8.32849
32. Wu, H., Lei, R., Peng, Y., Gao, L.: Aagnet: A graph neural network towards multi-task machining feature recognition. Robotics and Computer-Integrated Manufacturing **86**, 102661 (2024). https://doi.org/10.1016/j.rcim.2023.102661
33. Wu, R., Xiao, C., Zheng, C.: DeepCAD: A deep generative network for computer-aided design models. In: ICCV. pp. 6772–6782 (2021). https://doi.org/10.1109/iccv48922.2021.00670
34. Wu, T., Yang, G., Li, Z., Zhang, K., Liu, Z., Guibas, L., Lin, D., Wetzstein, G.: GPT-4V(ision) is a human-aligned evaluator for Text-to-3D generation. In: CVPR. pp. 22227–22238 (2024). https://doi.org/10.1109/cvpr52733.2024.02098
35. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems **32**(1), 4–24 (2020). https://doi.org/10.1109/tnnls.2020.2978386
36. Xu, J., Wang, C., Zhao, Z., Liu, W., Ma, Y., Gao, S.: CAD-MLLM: Unifying multimodality-conditioned CAD generation with MLLM. arXiv preprint arXiv:2411.04954 (2024). https://doi.org/10.48550/arXiv.2411.04954
37. Xu, R., Wang, X., Wang, T., Chen, Y., Pang, J., Lin, D.: PointLLM: Empowering large language models to understand point clouds. In: ECCV. pp. 131–147. Springer Nature Switzerland, Cham (2024). https://doi.org/10.1007/978-3-031-72698-9_8

38. Yang, A., Yang, B., Hui, B., Zheng, B., Yu, B., Zhou, C., et al.: Qwen2 technical report. arXiv preprint arXiv:2407.10671 (2024). https://doi.org/10.48550/arXiv.2407.10671
39. Yin, S., Fu, C., Zhao, S., Li, K., Sun, X., Xu, T., Chen, E.: A survey on multimodal large language models. National Science Review **11**(12), nwae403 (2024). https://doi.org/10.1093/nsr/nwae403
40. Yu, J., Wang, Z., Vasudevan, V., Yeung, L., Seyedhosseini, M., Wu, Y.: CoCa: Contrastive captioners are image-text foundation models. arXiv preprint arXiv:2205.01917 (2022). https://doi.org/10.48550/arXiv.2205.01917
41. Yuan, H., Xu, J., Pan, H., Bousseau, A., Mitra, N.J., Li, C.: CADTalk: An algorithm and benchmark for semantic commenting of CAD programs. In: CVPR. pp. 3753–3762 (2024). https://doi.org/10.1109/cvpr52733.2024.00360
42. Zhang, L., Le, B., Akhtar, N., Lam, S.K., Ngo, D.: Large language models for computer-aided design: A survey. ACM Computing Surveys (2025). https://doi.org/10.1145/3787499
43. Zhang, S., Guan, Z., Jiang, H., Ning, T., Wang, X., Tan, P.: Brep2Seq: A dataset and hierarchical deep learning network for reconstruction and generation of computer-aided design models. Journal of Computational Design and Engineering **11**(1), 110–134 (2024). https://doi.org/10.1093/jcde/qwae005
44. Zhang, Y., He, F., Fan, R., Fan, B.: View2cad: Parsing multi-view into CAD command sequences. In: Proceedings of the 2024 27th International Conference on Computer Supported Cooperative Work in Design (CSCWD). pp. 2949–2954. IEEE (2024). https://doi.org/10.1109/cscwd61410.2024.10580755
45. Zhao, W.X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., et al.: A survey of large language models. arXiv preprint arXiv:2303.18223 (2023). https://doi.org/10.48550/arXiv.2303.18223
46. Zhou, S., Tang, T., Zhou, B.: CADParser: A learning approach of sequence modeling for B-Rep CAD. In: IJCAI. pp. 1804–1812 (2023). https://doi.org/10.24963/ijcai.2023/200