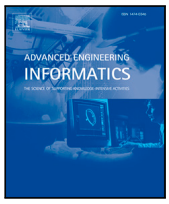
Advanced Engineering Informatics 76 (2026) 104990

Contents lists available at ScienceDirect

Advanced Engineering Informatics

journal homepage: www.elsevier.com/locate/aei



Review article

Learning-based 3D CAD model generation in mechanical engineering: A survey

Fabian Baumeister [a,*], Jakob Bönsch [a], Constantin Chaumet [b], Laura Dörr [a], Anne Meyer [a]

[a] Department of Mechanical Engineering, Karlsruhe Institute of Technology, Karlsruhe, Germany
[b] Lamarr Institute for Machine Learning and Artificial Intelligence, TU Dortmund University, Dortmund, Germany

ARTICLE INFO

Keywords:
Computer-aided design
Generative artificial intelligence
3D model generation
Mechanical engineering
Design automation

ABSTRACT

Computer-aided design (CAD) solid modeling lies at the core of product design in mechanical engineering, enabling designers to transform conceptual design ideas into precise, parametric, 3D CAD models. Automating solid modeling has the potential to accelerate engineering workflows, foster innovation, and lower barriers to design for non-expert users. Recently, increasing efforts are being devoted to leveraging learning-based methods to address this challenge. To the best of our knowledge, no existing survey provides a comprehensive overview of learning-based methods for 3D CAD model generation, nor addresses their applications in mechanical engineering. For that reason, we survey literature comprising state-of-the-art, learning-based methods for 3D CAD model generation, with particular emphasis on their relevance to mechanical engineering. Based on the type of input to the respective methods, we define three tasks and classify the literature accordingly: CAD Generation, CAD Reconstruction, and CAD Reverse Engineering. To provide a structured and comprehensive overview, we further organize the literature according to its types of output and the methodologies applied. Additionally, this survey highlights current research gaps, including limitations in fine-grained user control and geometric detail preservation, and discusses potential directions for future research. Finally, we present one potential use case for each of the aforementioned tasks in the mechanical engineering workflow, aiming to bridge the gap between existing methods and real-world applications. These use cases include design space exploration, quality control, and data enhancement.

Contents

* Corresponding author.
E-mail address: fabian.baumeister@kit.edu (F. Baumeister).

https://doi.org/10.1016/j.aei.2026.104990
Received 10 November 2025; Received in revised form 19 June 2026; Accepted 19 June 2026
Available online 25 June 2026

1. Motivation and contribution

Computer-aided design (CAD) solid modeling is at the core of product design in mechanical engineering. It enables designers to transform conceptual design ideas into precise, parametric, 3D CAD models, which facilitate a wide range of engineering-related downstream tasks such as simulation and computer-aided manufacturing (CAM). Traditionally, modeling is performed manually by expert users through a time-consuming process. Automating the modeling process has the potential to (1) accelerate and enhance engineering workflows, (2) foster innovation through rapid iteration, and (3) lower the barriers to design for non-expert users.

Generative models [1,2] have shown promising results across various domains, including robotics [3,4], image generation [5,6], natural language generation [7,8] and geometry generation [9]. While prior efforts in geometry generation have predominantly concentrated on voxels, polygonal meshes, and implicit shape representations, comparatively little attention has been directed toward generating precise parametric 3D CAD models necessary for mechanical engineering workflows. With the growing availability of large-scale CAD datasets [10,11], increasing efforts are being devoted to applying learning-based methods to 3D CAD model generation. Recent years have seen numerous approaches for learning expressive representations of 3D CAD models, whether represented as boundary representations (BREP) [12,13] or as procedural command sequences [10,14]. In parallel, several methods have been proposed to automatically construct 3D CAD models from diverse input modalities such as natural language descriptions [15,16], point clouds [17,18], and images [19,20]. These approaches leverage a wide spectrum of learning-based techniques, including transformer, diffusion, and foundation models, as well as neural–analytical methods that combine neural network predictions with analytical modeling principles.

Despite the remarkable progress achieved in recent years, many challenges remain. Existing datasets often lack the diversity and geometric complexity of real-world CAD models. In this context, it is important to consider the modeling paradigms underlying modern industrial CAD systems. These systems predominantly operate parametric and feature-based, representing models as sequences of parameterized operations and editable design histories. These paradigms form the backbone of CAD/CAM workflows in domains such as mechanical engineering and aerospace, and are essential for enabling downstream tasks such as simulation, manufacturing, and future design modification. Therefore, learning-based approaches to 3D CAD model generation must not only produce geometrically accurate shapes, but also adhere to these paradigms to ensure applicability in industry scenarios [21,22].

Furthermore, although recent research has focused on enhancing user control over the generation process, creating 3D CAD models that satisfy specific design requirements or functional constraints continues to be a major open problem. These challenges represent significant barriers to the adoption of current learning-based methods for 3D CAD model generation in practical mechanical engineering applications.

With the growing body of literature addressing these challenges, there is a need for comprehensive surveys that provide an overview of this emerging field. Existing surveys either do not cover the full spectrum of learning-based methods or are limited to specific input modalities considered in the surveyed literature [23,24]. Moreover, limited attention has been given to the practical application of these methods within real-world mechanical engineering workflows. To the best of our knowledge, this work presents the first comprehensive survey that encompasses both generative and neural–analytical methods for 3D CAD model generation. We review the state-of-the-art in this area, discuss research gaps, and present possible future research directions. Furthermore, to help bridge the gap between current research and practical deployment, we outline potential applications of the surveyed methods within the mechanical engineering domain. To structure this survey and link the surveyed methods to practical applications, we define three fundamental tasks based on the type of input and classify the literature accordingly. An overview of these tasks is shown in Fig. 1.

1. **CAD Generation** involves creating 3D CAD models from abstract inputs such as text, semantic class labels, or other forms of user guidance. The user may also provide partial models for editing or autocompletion.
2. **CAD Reconstruction** involves creating 3D CAD models from sensor-based inputs such as point clouds, voxels, or images that represent physical objects.
3. **CAD Reverse Engineering** involves deriving sequences of CAD modeling operations from existing 3D CAD models given as BREPS.

To summarize, our contributions are threefold: (1) We present a comprehensive overview and classification of state-of-the-art literature and their methods for learning-based 3D CAD model generation, including methods for 3D CAD representation learning. We also present

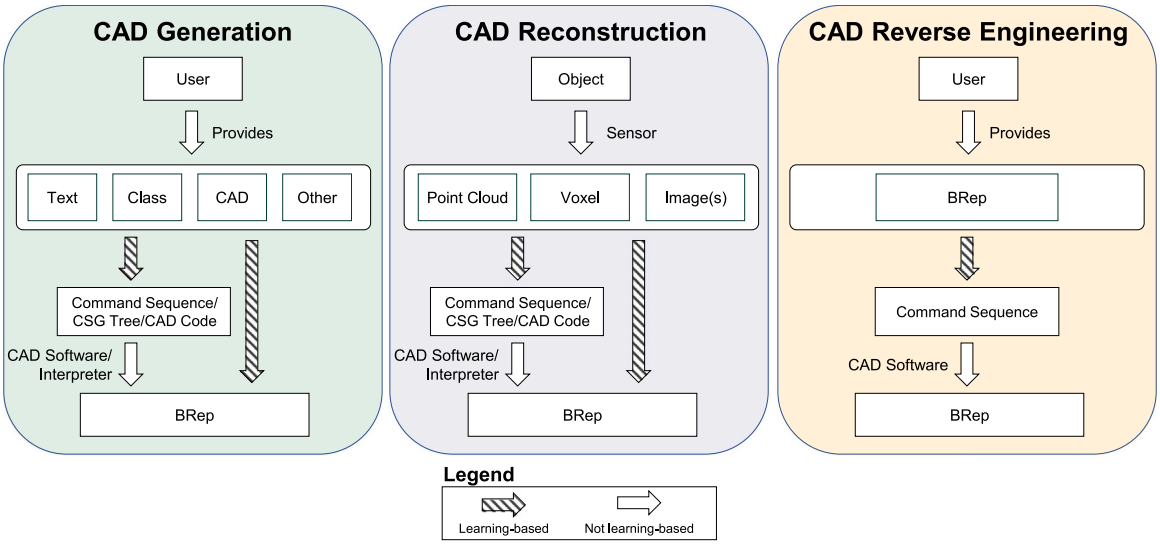


Fig. 1. The three main tasks addressed in the surveyed literature. (Left) **CAD Generation** from text, semantic class labels, or other user guidance. The user may also provide partial CAD models for editing or autocompletion. (Middle) **CAD Reconstruction** from sensor-based inputs such as point clouds, voxels, or images. (Right) **CAD Reverse Engineering** of CAD command sequences from BREP input.

Table 1
Scopus search settings.

Queries	Pub. year	Language	Doc. Type	Subject area
{'computer-aided design'} **AND** {'generative' **OR** 'learning' }	2021–2026	English	Article, Conference paper	Engineering, Computer science

the datasets used for training them, including datasets that follow the parametric and feature-based paradigm of industrial CAD systems. (2) We identify research gaps with respect to mechanical engineering and potential for future research. (3) We discuss potential mechanical engineering use cases, aiming to bridge the gap between research and real-world applications.

This survey is structured as follows: The literature selection process is described in Section 2. Preliminaries are presented in Section 3. Literature addressing 3D CAD representation learning is presented in Section 4. A comprehensive overview of literature comprising state-of-the-art, learning-based methods for 3D CAD model generation is presented in Section 5. Datasets used for training are presented in Section 6. A discussion of research gaps and future research directions is presented in Section 7. Potential use cases in the mechanical engineering workflow are presented in Section 8. Finally, a summary of key insights and findings, as well as limitations of this survey, are presented in Section 9.

2. Literature selection

Due to its large database of peer-reviewed, high-quality, scientific papers, we conducted the literature search in *Scopus*. The search was conducted on January 07, 2026. We searched article abstracts using the keywords shown in Table 1. We selected the start date to be 2021, as the publication of *DeepCAD* [10] marks the start of extensive research on learning-based 3D CAD model generation. We retrieved 844 publications from the database. During an initial scan of the article titles and abstracts, we identified **33** publications as relevant for review. We filtered out publications which either do not address learning-based 3D CAD model generation or specifically address challenges from domains other than mechanical engineering, e. g., medical engineering, architecture, or gaming. Subsequently, we conducted a thorough, automated forward–backward search on the resulting corpus of 33 publications, identifying an additional **38** publications that are relevant to this survey. During the review process, we identified **2** publications from before 2021 that addressed 3D CAD model generation, including them in the survey. To ensure that this survey covers state-of-the-art research, we included another **5** publications that were published at relevant machine learning and computer vision conferences (CVPR,[1] ICCV,[2] ICLR,[3] ICML,[4] AAAI[5]) but were not covered by our search procedure and that we identified as relevant for this survey. In total, we surveyed **78** publications. An overview of all search settings is given in Table 1. While alternative spellings of 'computer-aided design' exist, we restricted our initial search to this term since it is the most widely used. The search query 'computer-aided design' may have excluded certain publications that omit the hyphen. To mitigate this limitation, we conducted the forward–backward search to ensure the inclusion of relevant publications not retrieved by the initial query. We deliberately excluded the abbreviation 'CAD' from the search query, as its inclusion would have led to substantial overlap with medical literature on *coronary artery disease* and *computer-aided diagnosis*, thereby substantially increasing the number of publications initially retrieved from the database to about 4000, making manual filtering infeasible.

3. Preliminaries

This section provides an introduction to the key concepts of CAD solid modeling and selected learning-based methods, including transformer and diffusion models.

[1] The IEEE/CVF Conference on Computer Vision and Pattern Recognition.
[2] The IEEE/CVF International Conference on Computer Vision.
[3] International Conference on Learning Representations.
[4] International Conference on Machine Learning.
[5] AAAI Conference on Artificial Intelligence.

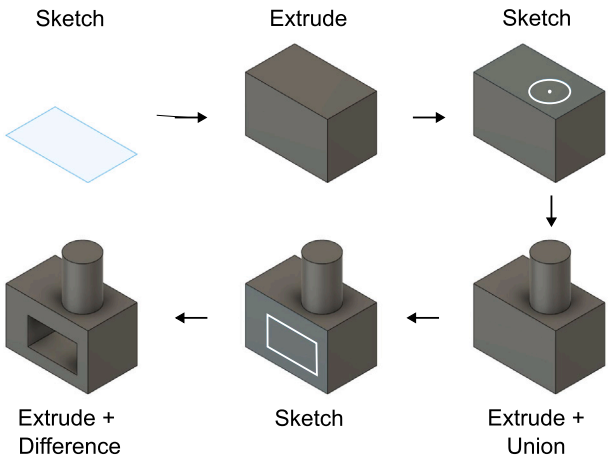


Fig. 2. Command sequence consisting of sequential sketch and extrude operations to construct a solid model.

3.1. CAD solid modeling

CAD solid modeling refers to the construction and representation of 3D CAD models, also referred to as solid models. These solid models are defined to have a well-defined volume, enclosed by continuous, closed surfaces without open faces or free edges [25]. In the following, we introduce the core concepts for constructing and representing these solid models. Furthermore, we refer to a *solid* as a 3D object with the same properties as a solid model, used as a building block in the construction of a solid model. The term *solid model* is reserved for the outcome of this construction process.

Feature-based parametric modeling is a procedural approach to constructing solid models by combining parameterized 2D sketches and 3D operations. The parameters of the operations are used in tandem with geometric constraints to define relationships between elements. This allows changes in one part of the model to propagate to other parts while preserving geometric consistency. Sketches are defined on planar surfaces and consist of geometric 2D primitives such as lines, arcs, and circles, which form one or more closed loops. Each loop encloses a region called a *profile*, and one or more profiles together constitute a *sketch*. A common operation to transform a sketch into a solid is *sweeping*, which creates a solid by translating the sketch along a defined trajectory. The special case of sweeping along a straight line perpendicular to the sketch plane is known as *extrusion*. Solids generated through these operations can be combined using Boolean operations, including union, difference, intersection, and XOR, to form more complex geometries. Modern CAD software provides a variety of additional feature operations, such as fillet, chamfer, and revolve, that allow the creation of more detailed and diverse designs. Further, a sequence of such modeling operations is referred to as a *command sequence*, which can be executed, modified, and interpreted within CAD software environments to construct solid models. The mapping from a solid model to its originating command sequence is generally non-unique, as multiple sequences can yield the same model. An example of a command sequence consisting of alternating sketch and extrude operations is shown in Fig. 2.

Constructive solid geometry (CSG) provides an alternative approach to constructing and representing solid models by expressing them as compositions of simple primitive solids through Boolean operations. In this survey, we also consider literature that represents primitives as combinations of 2D sketches and CAD operations (e.g., extrusion) to construct the corresponding solids. In constructive solid geometry (CSG), a solid model is represented as a binary tree, where the leaf nodes correspond to primitives and the internal nodes represent operators. These operators can denote either rigid transformations or Boolean

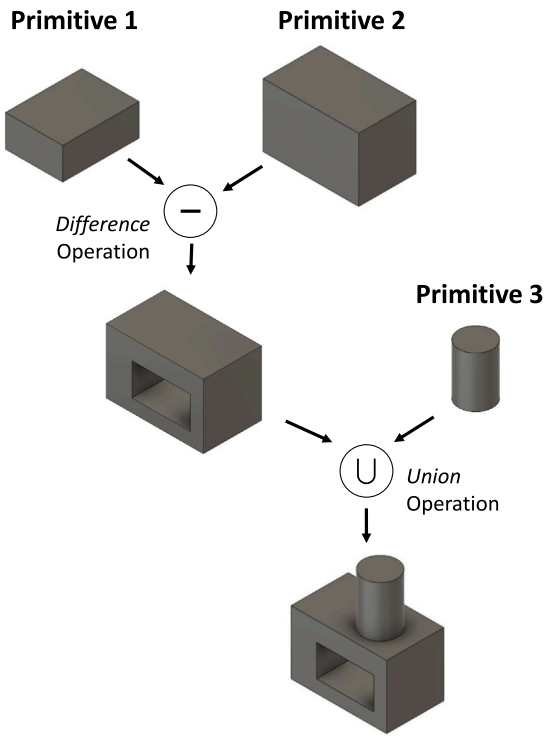


Fig. 3. Constructive solid geometry (CSG) tree with primitives and Boolean operations combining them to construct a solid model.

operations such as union, intersection, or difference. Each internal node combines the solids defined by its child nodes according to the specified operation, resulting in a hierarchical and interpretable representation of complex geometries. We refer to this tree representation as *CSG tree* [26]. An example of a CSG tree is shown in Fig. 3.

The boundary representation (BREP) is the most widely used representation of solid models in modern CAD systems. It defines a solid by explicitly representing its enclosing surfaces rather than its interior volume. The boundary of a solid is decomposed into a finite number of oriented surface patches called *faces*. Each face is bounded by a set of *edges*, which connect at *vertices* [26]. Faces correspond to subsets of parametric surfaces and are required to form closed loops to define valid areas. To represent adjacency and topological relationships between entities, directed *coedges* are used. A coedge stores references to its next and previous coedges within a loop, its adjacent coedge on a neighboring face, and its parent face and edge [13]. An example BREP is shown in Fig. 4.

CAD code encompasses the use of programming languages such as Python to construct solid models programmatically. Various open-source frameworks facilitate the generation of CAD code, leveraging a wide range of common CAD modeling commands, including sketch-based modeling and CSG. The generated code is executed by a suitable interpreter to construct the solid model. A common framework is CadQuery [27], which is built on the OpenCascade Technology CAD kernel and provides a Python interface for programmatic solid modeling. Listing 1 shows an example of CadQuery code alongside the resulting solid model.

3.2. Transformer and diffusion models

The transformer is an encoder–decoder neural network based on the attention mechanism initially employed for sequence modeling [1]. The encoder consists of a stack of layers, each comprising a self-attention module and a position-wise multi-layer perceptron (Mlp). Given a sequence of input vectors, the self-attention module computes

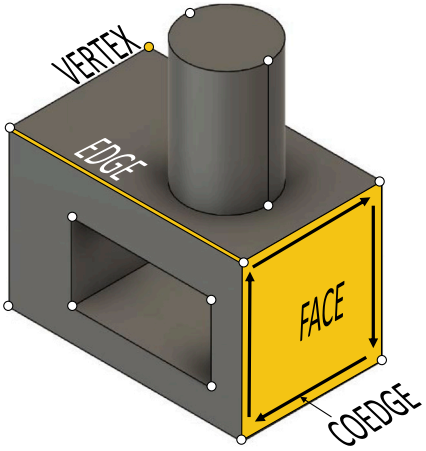


Fig. 4. BREP consisting of faces, edges, vertices, and coedges.

```python
import cadquery as cq

# Parameter Definitions
block_length = 47
block_width = 29
block_height = 31

# Further Parameter Definitions
...

# Step 1: Create Base Block
result = cq.Workplane("XY").box(
    block_length,
    block_width,
    block_height
)

# Step 2: Add Cylinder
result = (
    result.faces(">Z")
    .workplane()
    .center(cyl_x_offset, cyl_y_offset)
    .circle(cyl_diameter / 2)
    .extrude(cyl_height)
)

# Step 3: Cut Rectangular Section
result = (
    result.faces("<Y")
    .workplane(centerOption="CenterOfMass")
    .center(cut_x_offset, cut_z_offset)
    .rect(cut_width, cut_height)
    .cutBlind(-cut_depth)
)
```

Listing 1: CadQuery code for the construction of a solid model similar to that in Fig. 4.

a query, key, and value vector for each input vector through a linear transformation. For each position in the input sequence, the attention mechanism calculates a weighted sum over all value vectors. The weights correspond to the similarity between the query vector of the respective position and the key vector of every other position in the input sequence, obtained by computing the dot product. Higher similarity results in larger weights, while lower similarity results in smaller weights. The self-attention module also employs a residual connection and layer normalization to facilitate model training. The encoder outputs one vector per input position. The self-attention mechanism enables the exchange of information between different positions in the sequence. The decoder generates the output sequence given the encoder output vectors. In an autoregressive setting, it generates one output vector at a time, using previously generated vectors as input. Self-attention is applied to these vectors, and a cross-attention module integrates information from the encoder output vectors. In cross-attention, queries are computed from the decoder input, i.e., previously generated vectors, while keys and values are computed from the encoder output vectors, allowing the decoder to incorporate encoded information into sequence generation. Alternatively, the decoder can operate in a fully feed-forward manner, taking a set of embedding vectors as input and generating the entire sequence in a single forward pass.

Diffusion models are a class of generative models inspired by the diffusion process in thermodynamics [28] and initially gained significant attention for their capabilities in image generation. They operate through a two-step process, consisting of a forward process and a reverse process. In the forward process, random noise, which is typically sampled from a prior distribution (e.g., standard Gaussian distribution), is iteratively added to the input data until it is fully corrupted and follows the prior distribution. In the reverse process, the model learns to invert this noising process, ultimately reconstructing the input data from random noise. Neural networks are commonly employed to learn this reverse process [2]. At inference time, a random vector is sampled from the prior distribution and denoised to generate new data. Diffusion models can operate on different data domains, including raw input data (e.g., images), continuous latent representations of the input data, or discrete input data. This flexibility enables their application across a wide range of tasks, such as image generation [29], audio synthesis [30], and molecule generation [31].

4. 3D CAD representation learning

Learning feature-rich representations of CAD models is crucial for a wide range of CAD-related downstream tasks, including CAD generation, CAD reconstruction, and CAD reverse engineering. A major challenge lies not only in capturing the overall geometry of a model but also in representing fine-grained features such as chamfers, fillets, and through-holes. Equally important is the ability to encode the underlying design intent, which reflects the reasoning and constraints that a human designer applies during the construction process. In this section, we present 3D CAD representation learning approaches. *UV-Net* [12], *BRep-Net* [13], and *DeepCAD* [10] are reused by methods from the surveyed literature in Section 5. Table 2 provides an overview of the surveyed methods on 3D CAD representation learning, indicating the inputs and whether contrastive learning is applied. Contrastive learning is a dedicated approach for learning robust representations of input data by constructing pairs of samples that are either similar or dissimilar. The model is trained in a self-supervised manner by minimizing the distance between representations of similar pairs and maximizing the distance between those of dissimilar pairs [32]. We group the methods addressing BREP and command sequence representation learning, respectively. The distinction between these methods and those in Section 5 remains fluid, as their adaptations or extensions could also fall within the scope of that section. In the following, the terms representation, latent vector, and embedding are used interchangeably.

4.1. BRep representation learning

Learning expressive representations of BREPS constitutes a fundamental step for the methods presented in Section 5.4. These approaches encode a given BREP into a latent representation from which a sequence of CAD modeling operations can be inferred to reconstruct the original design.

UV-Net [12] represents the BREP as a face adjacency graph, in which nodes represent BREP faces and edges represent their connectivity. Each

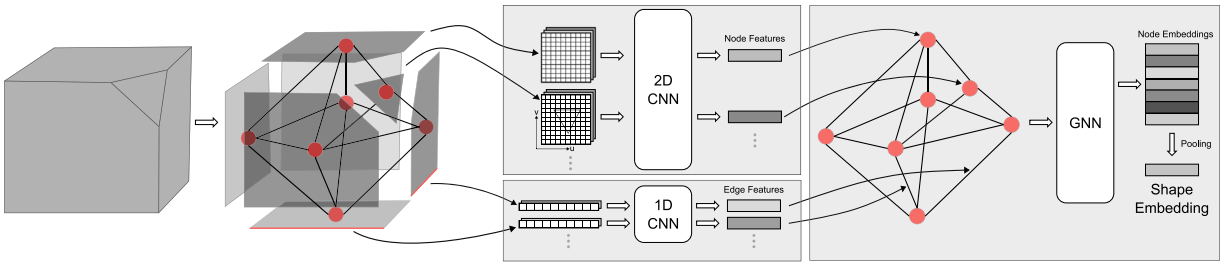


Fig. 5. *UV-Net* architecture [12]. The input Brep is represented as a graph for which node and edge features can be extracted. A Gnn processes the graph and outputs a set of node embeddings, which are pooled to obtain the final shape embedding.

Table 2
Overview of methods for 3D Cad representation learning. Literature grouped by input and sorted by year of publication.

Name	Input				
	BRep	Sequence	Point cloud	Image(s)	Contrastive
Boundary representation					
UV-Net [12]	✓	–	–	–	(✓)
BRepNet [13]	✓	–	–	–	–
SSRL [33]	✓	–	–	–	–
BRep-BERT [34]	✓	–	–	–	✓
BRT [35]	✓	–	–	–	–
UniRep [36]	✓	–	–	–	–
Command sequence					
DeepCAD [10]	–	✓	–	–	–
ContrastCAD [14]	–	✓	–	–	✓
MultiCAD [37]	–	✓	✓	–	✓
DH-CAD [38]	–	✓	✓	–	–
CF-CAD [39]	–	✓	–	✓	✓

face and edge contains a parametric surface and curve, respectively, which can be represented as a regular grid in the parameter domain. Each point of this grid is assigned geometric features, i.e., point coordinates and normal or tangent vector, evaluated from the surface or curve, respectively. These 1D/2D grids are used as edge and vertex features. Subsequently, 1D and 2D convolutional neural networks (Cnns) are employed to compute node and edge features, which are further processed by a Gnn. The resulting set of node embeddings is pooled to obtain the final shape embedding. Fig. 5 shows the *UV-Net* architecture. The method is trained and evaluated on a shape classification and a face segmentation task. Additionally, the authors propose a contrastive learning strategy, facilitating self-supervised training.

BRepNet [13] proposes a coordinate-free approach for representing Brep, granting translation and rotation invariance. Central to the approach is the concept of a topological walk, a sequence of instructions to traverse a Brep from a starting coedge to a destination coedge. In comparison to *UV-Net*, the features extracted for the Brep faces, edges, and coedges do not contain any coordinate information. The proposed neural network comprises a stack of convolutional units that aggregate face, edge, and coedge features using convolutional kernels. These convolutional kernels are defined by topological walks relative to the coedges of the Brep. Aggregating face features after the last convolution unit by, e.g., pooling, produces a final shape embedding. The method is trained and evaluated on a face segmentation task, for which the authors present the *Fusion 360 Gallery segmentation dataset*, which builds on the *Fusion 360 Gallery dataset* [40].

SSRL [33] addresses the lack of labeled Brep data and presents a neural network to learn a hybrid implicit/explicit surface representation for Brep. The network comprises an encoder [41] to compute embeddings for each topological entity of the Brep, followed by a decoder for face reconstruction. The decoder is implemented as a conditional neural field [42], explicitly capturing the parametric surfaces, and implicitly representing the corresponding boundaries. Therefore, the neural field learns to map coordinates from the 2D parametric domain of a face's surface to the corresponding 3D position, given the corresponding face embeddings. For the surface's boundaries, the neural field is trained to map the same coordinates to single values, measuring the signed distances to the boundary. Initial training is performed in a self-supervised manner. The trained encoder can be leveraged to facilitate few-shot learning on downstream tasks such as face classification or segmentation.

BRep-BERT [34] extends the BERT [43] architecture from the language domain to Cad, enabling self-supervised pre-training on Brep data. Training is separated into two stages. During the first stage a Gnn tokenizer is trained to compute discrete Brep entity tokens from the Brep graph [12]. In the second stage these tokens are used as supervision for training a BERT-like transformer model. A masked language modeling objective is used, where the model is trained to reconstruct randomly masked tokens given the surrounding context. Additionally, the method employs contrastive learning objectives to promote the acquisition of rich geometric and topological representations. After self-supervised pre-training, *BRep-BERT* can be applied to downstream tasks such as solid classification, surface segmentation, and few-shot learning.

BRT [35] proposes a novel Brep representation designed to address the discrete nature of input tokens to transformer-based models. In contrast to prior representation learning approaches that discretize parametric edges and surfaces, *BRT* decomposes edges into Bézier curves [44] and faces into triangular Bézier patches [45]. Transformer-based encoders are applied at the vertex, edge, and face levels to obtain respective embeddings. These are subsequently hierarchically aggregated from vertices to edges, loops, faces, and ultimately to the shell level using multiple dedicated neural networks. This process produces a sequence of face-level tokens, which is subsequently processed by a transformer encoder to generate final embeddings suitable for downstream tasks. The authors train and evaluate the proposed method on part classification and feature recognition tasks.

UniRep [36] extends prior Brep graph representations in which the adjacency matrix encodes only face–edge connectivity by additionally incorporating edge–vertex and face–vertex relationships. Moreover, adjacency matrix entries are weighted based on loop information and the topological directions of edges. A feature matrix combines face, edge, and vertex information. The feature matrix and the weighted adjacency matrix define a weighted graph representation of the Brep. Building on this formulation, the authors introduce a novel graph autoencoder which is trained to learn a latent representation of the weighted graph. The learned latent representation is evaluated across multiple downstream tasks, including Brep generation using a latent diffusion model, part classification, and part retrieval.

4.2. Command sequence representation learning

Representation learning of command sequences forms the foundation for methods addressing command sequence generation and reconstruction, as discussed in Section 5.2.1 and Section 5.3.1. Since different

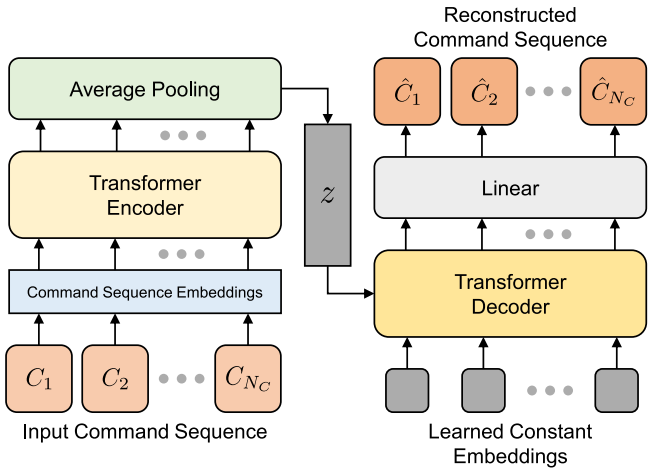


Fig. 6. *DeepCAD* architecture [10]. Command sequence embeddings are computed and processed by the transformer encoder, producing the latent vector z. The transformer decoder takes a set of learned constant embeddings as input and attends to the latent vector to reconstruct the original command sequence in a feed-forward manner.

command sequences can yield identical model geometries, the learned representations must capture not only the syntactic structure of the sequence but also the geometric characteristics of the resulting CAD model implicitly encoded within it.

DeepCAD [10] marks a milestone in the field of learning-based 3D CAD model generation. The authors introduce a new, learning-friendly representation of command sequences. A sketch profile S is described by a list of N loops $S = [Q_1, \dots, Q_N]$. Each loop $Q_i = [\texttt{<SOL>}, C_1, \dots, C_{n_i}]$ starts with an indicator command <SOL> followed by a series of n_i curve commands. An individual curve command $C_j = (t_j, \mathbf{p}_j)$ is described by its curve type $t_j \in \{\texttt{<SOL>}, \text{L}, \text{A}, \text{R}\}$ and its parameters $\mathbf{p}_j$. The authors consider *line*, *circle*, and *arc* curves. The extrusion command always refers to the directly previously predicted sketch profile. With that, a CAD model M is described as a command sequence $M = [C_1, \dots, C_{N_c}]$ with a total of N_c commands. Table 3 gives an overview of the individual commands and their parameters. The authors propose discretizing continuous parameters within a fixed range to facilitate more stable model training. To date, this representation is adopted by most learning-based methods on command sequences. The corresponding neural network architecture is shown in Fig. 6. The transformer-based autoencoder encodes the input command sequence M into a latent vector $\mathbf{z}$. Given a fixed number of learned embeddings, the decoder reconstructs the input command sequence, incorporating the latent vector via cross-attention. The decoder is followed by two separate linear layers, predicting command type and command parameters, respectively, for each command. The method is trained in a self-supervised manner, using the reconstruction loss between the ground truth sequences and the reconstructed sequences. Evaluation is performed on an unconditional generation task and a point cloud reconstruction task. As *DeepCAD* is a foundational work for representing and learning with command sequences, we include it in this section and also reference it in Section 5.2.

ContrastCAD [14] learns expressive representations of CAD command sequences using a contrastive learning strategy. For a batch of command sequences, an encoder [10] computes the respective latent vectors. The decoder reconstructs the input command sequences given the latent vectors. Additionally, contrastive learning is employed by duplicating and masking the individual latent vectors. For each command sequence in the input batch, two augmented latent vectors are obtained by applying two separate masking operations to the initial latent vector. The contrastive loss encourages latent vectors of the same sequence to be closer to each other, while enforcing latent vectors of the other input sequences to be further away. The method is trained in a self-supervised manner, using the reconstruction loss between the ground truth sequences and the reconstructed sequences, as well as a contrastive loss. Evaluation is performed on an unconditional generation task.

Table 3
Learning-friendly command sequence representation proposed in *DeepCAD* [10]. The table shows the represented CAD commands and their parameters. <SOL> marks the start of a loop; <EOS> marks the end of the sequence.

Commands	Parameters	
<SOL>	∅	
L (Line)	x, y :	line end-point
A (Arc)	x, y :	arc end-point
	α :	sweep angle
	f :	counter-clockwise flag
R (Circle)	x, y :	center
	r :	radius
E (Extrude)	θ, ϕ, γ :	sketch plane orientation
	p_x, p_y, p_z :	sketch plane origin
	s :	scale of sketch profile
	e_1, e_2 :	extrude distances
	b :	Boolean type
	u :	extrude type
<EOS>	∅	

Several approaches leverage the multimodal characteristics of CAD models by incorporating both geometric shape information and the knowledge encoded in corresponding command sequences into the learning process. In the following, we present the approaches that consider sequential and geometric information for representation learning.

MultiCAD [37] presents a contrastive learning framework to align command sequence embeddings and corresponding point cloud embeddings. A two-stage training process is employed, first learning sequence embeddings under the guidance of geometry, followed by aligning the geometry embeddings with the learned sequence embeddings. During the first stage, contrastive learning is employed to align the embeddings of both inputs via a multimodal contrastive loss. Simultaneously, a decoder is trained to reconstruct input command sequences from the command sequence embeddings. In the second stage, the geometry encoder [46] is trained under the supervision of both the geometric and sequence embeddings. This strategy facilitates self-supervised training. Evaluation is performed on a point cloud reconstruction task and a point cloud classification task.

DH-CAD [38] employs a transformer encoder including cross-attention to fuse command sequence and point cloud embeddings. The resulting latent vectors are passed to a context-aware transformer decoder for command sequence reconstruction. The decoding process includes an Mlp for command type prediction as well as command-specific Mlps for parameter prediction. The method is trained in a self-supervised manner, using the reconstruction loss between the ground truth sequences and the reconstructed sequences. Evaluation is performed on an unconditional generation task and a retrieval task.

CF-CAD [39] employs a shared codebook along with a contrastive training strategy to align command sequence embeddings and corresponding image embeddings. A codebook is a finite set of learnable embedding vectors used to discretize a continuous embedding space. Embeddings obtained from the encoders are mapped to their nearest codebook entries, further also referred to as codes, thereby transforming continuous representations into discrete ones. This discretization has shown to offer several advantages over purely continuous embeddings [47]. In *CF-CAD* command sequence embeddings and image embeddings are mapped to their closest embedding vector in the shared codebook and aligned using a contrastive loss. Furthermore, command sequence embeddings are augmented using dropout operations, enabling intra-modal alignment through the contrastive loss.

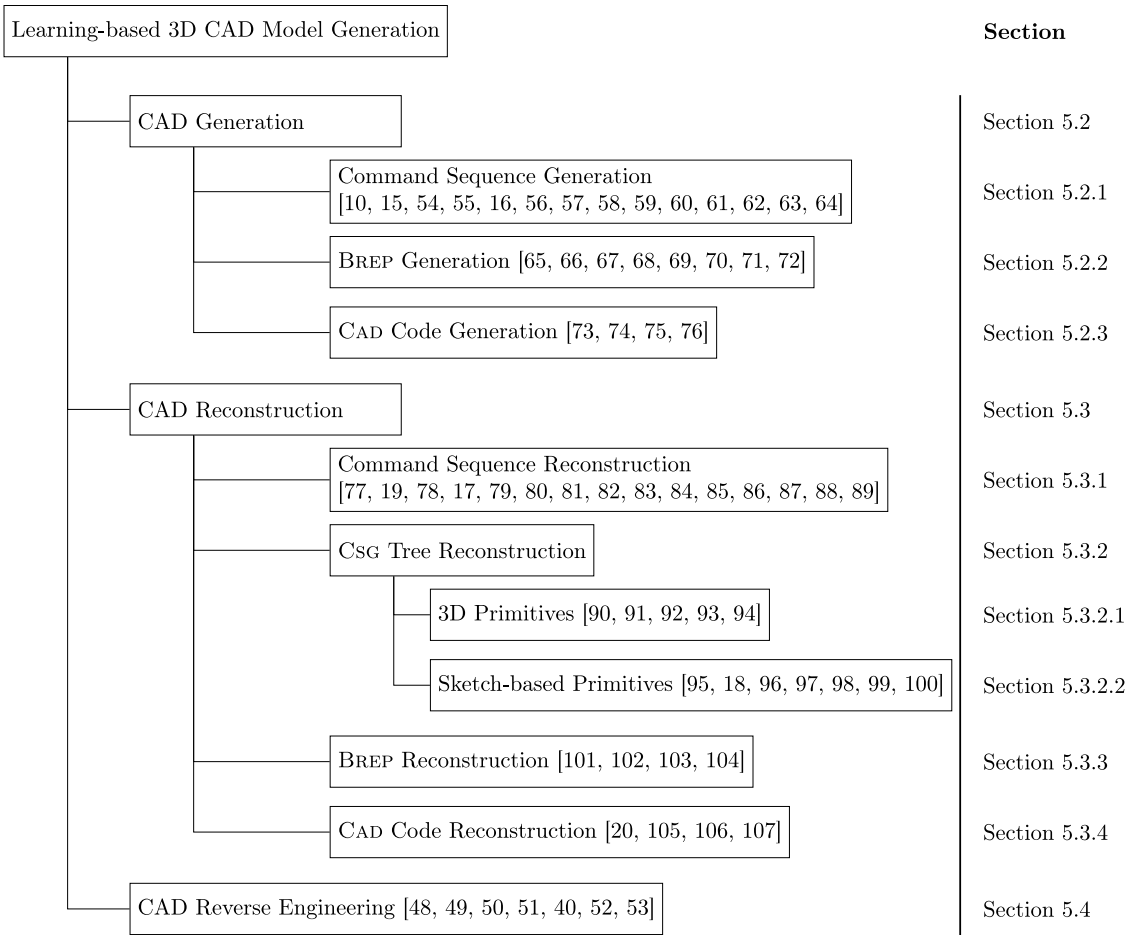


Fig. 7. Taxonomy and overall structure of the surveyed literature in Section 5.

Command sequence embeddings are passed to a transformer decoder to reconstruct the input command sequence. The method is trained in a self-supervised manner, using the reconstruction loss between the ground truth sequences and the reconstructed sequences, as well as the contrastive loss. Evaluation is performed on an unconditional generation task.

5. Learning-based 3D CAD model generation

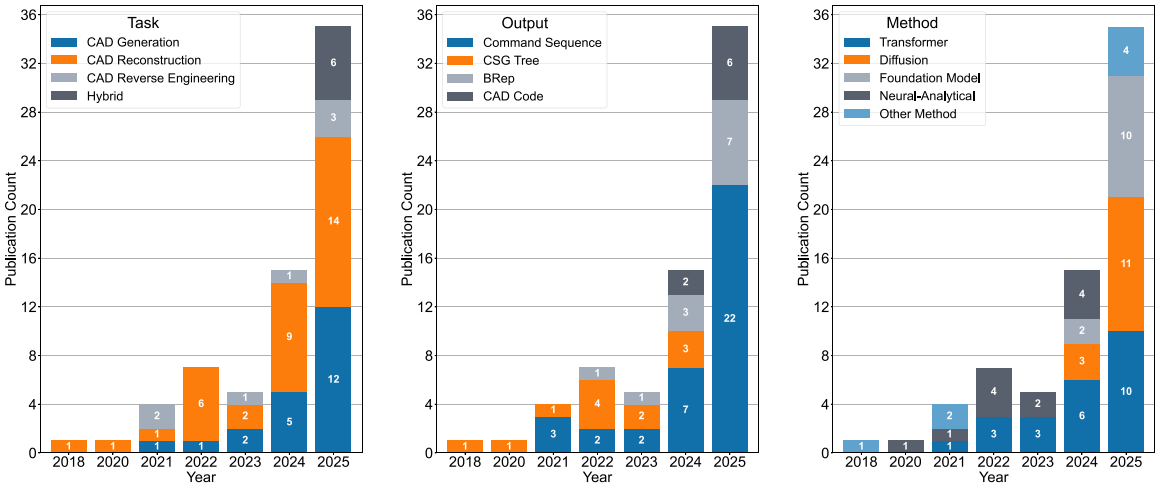
Since the publication of *DeepCAD* [10], learning-based approaches for 3D CAD model generation are receiving increased attention within the research community. Fig. 7 illustrates the proposed taxonomy, which also defines the structure of this section. Fig. 8(a) shows trends in the surveyed literature, grouped by the tasks defined and presented in Section 1. *Hybrid* denotes methods that can be employed for both CAD generation and CAD reconstruction. Early work predominantly focused on CAD reconstruction. More recently, the emphasis has shifted toward CAD generation, which by 2025 occurs at a frequency comparable to CAD reconstruction. CAD reverse engineering remains the least explored task in the literature.

Independent of the tasks, we identified four different types of output that are produced in the surveyed literature:

1. **Command Sequence**: Sequence of CAD modeling operations, mimicking the workflow of a human designer in CAD software. These command sequences can be executed directly in CAD software to construct the corresponding CAD model.
2. **CSG Tree**: Geometric primitives and Boolean operations to combine the primitives to construct the CAD model. We further distinguish by how geometric primitives are represented:
   - **3D Primitives**: Parametric 3D primitives such as cuboids, spheres, or cones.
   - **Sketch-based Primitives**: Primitives represented as 2D sketches and corresponding CAD operations (e.g., extrusion) to construct the 3D primitives.
3. **BRep**: Boundary representation (BREP) of a CAD model.
4. **CAD Code**: Executable code, e.g., Python code, to construct the CAD model programmatically.

Fig. 8(b) shows trends in the literature, grouped by the aforementioned output types. The construction of CSG trees has been consistently addressed by individual studies over the years, though no new work appeared in 2025. Since the publication of *DeepCAD* [10], there has been a growing interest in producing command sequences, which has become the most extensively explored output type to date. In contrast, directly constructing BREP remains less explored. More recently, CAD code has emerged as a novel and promising output type for learning-based 3D CAD model generation. Further, we identified five types of methods in the surveyed literature:

- **Transformer**-based approaches
- **Diffusion**-based approaches
- **Foundation models** including pre-trained large language models (LLMs) or vision-language models (VLMs).
- **Neural–analytical** approaches combining learning-based and analytical methods
- **Other** approaches including recurrent neural networks (RNNs), neurally-guided search, state space models, and reinforcement learning (RL).



(a) Literature grouped by task. (b) Literature grouped by output. (c) Literature grouped by method.

Fig. 8. Trends in the surveyed literature, grouped by task (left), output (middle), and method (right).

Fig. 8(c) shows trends in the literature, grouped by the aforementioned methods. Early work explored neural–analytical methods and transformer-based neural networks. Over the past two years, diffusion-based neural networks have attracted growing attention. Most recently, methods built on pre-trained foundation models are emerging and are explored as often as transformer and diffusion models for 3D CAD model generation in the past year.

In the following, we introduce the most commonly used evaluation metrics and subsequently review state-of-the-art learning-based methods for 3D CAD model generation, organized according to the previously defined tasks. Hybrid methods are presented in one of the respective sections. Additionally, we indicate where they can be applied to other tasks. For each task, we further group the literature by the type of output and sort by method and year of publication.

5.1. Evaluation metrics

In this section, we provide an overview of the most commonly used evaluation metrics for 3D CAD model generation. Most evaluation metrics can be computed independently of the generated output representation. Based on their application, we categorize the evaluation metrics identified in this survey into four groups.

Geometric metrics evaluate how well the geometry of two CAD models aligns. The two most commonly used metrics are:

- *Chamfer Distance* (CD) measures the similarity between two point clouds sampled from the surfaces of two CAD models. Lower is better.
- *Intersection over Union* (IoU) measures geometric overlap between two CAD models as the ratio of their intersecting volume to the total volume covered by both shapes. Higher is better.

Distribution metrics evaluate how well a generative model learns a ground truth (training) data distribution:

- *Coverage* (COV) measures the diversity of generated CAD models by computing the proportion of ground truth CAD models having at least one match after each generated shape is assigned to its nearest neighbor of the ground truth models based on Chamfer distance. Higher is better.
- *Minimum Matching Distance* (MMD) measures geometric similarity by computing the average Chamfer distance between each ground truth CAD model and the geometrically closest generated model. Lower is better.
- *Jensen–Shannon Divergence* (JSD) measures the similarity between the ground truth data distribution and the learned distribution. Lower is better.

CAD metrics evaluate the robustness and uniqueness of the CAD generation process:

- *Valid* measures the proportion of generated CAD models that are valid. Depending on the output representation, validity may refer to command sequences that can be executed in CAD software without errors, watertight BREP models with no open surfaces or invalid intersections, or CAD code that can be executed without errors to successfully construct a CAD model. Higher is better.
- *Novel* measures the proportion of generated CAD models that are not present in the training dataset. Higher is better.
- *Unique* measures the proportion of generated CAD models that appear only once during evaluation. Higher is better.

Command sequence metrics evaluate how well generated command sequences match the ground truth sequences. Assuming that each command consists of a command type and corresponding parameters, accuracy can be computed for both components:

- *Command Accuracy* (ACC_{cmd}) measures the proportion of correctly predicted command types. Higher is better.
- *Parameter Accuracy* (ACC_{param}) measures the proportion of correctly predicted command parameters in case the corresponding command type has been predicted correctly. Higher is better.

While these are the most commonly used evaluation metrics, we note that, even when identical metrics are employed, direct comparison of methods based on reported experimental results is generally not feasible due to differences in experimental setups. In addition, a variety of further metrics tailored to specific inputs, methods, and outputs are used across the literature. Owing to their domain-specific nature, a detailed discussion of these metrics is beyond the scope of this survey.

5.2. CAD generation

Automated CAD generation enables designers to efficiently translate early design concepts into precise parametric CAD models suitable for downstream applications such as simulation and manufacturing. A central challenge lies in providing mechanisms that allow the designer to control the modeling process, ensuring that the resulting CAD model

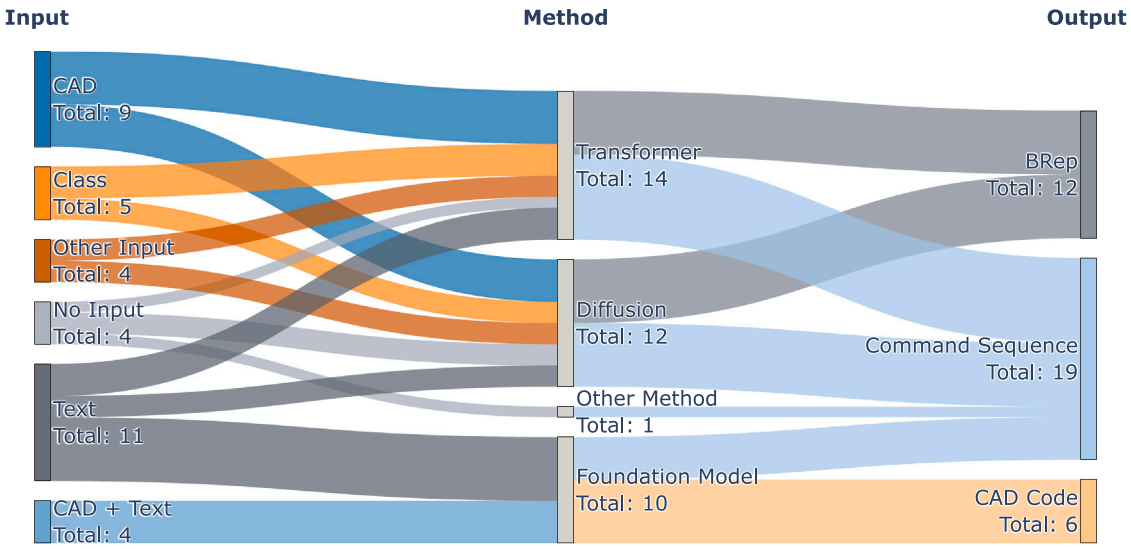


Fig. 9. Sankey diagram of the literature addressing CAD generation with inputs, methods, and outputs.

satisfies predefined design requirements. Such control can be achieved through different forms of input. We distinguish the following types of input for CAD generation:

- **Text**: A natural language input.
- **CAD**: Command sequences, BREPS, or CAD code.
- **CAD + Text**: A CAD model and natural language input.
- **Class**: A semantic class label defining the target object category.

Depending on the input, we distinguish different types of generation:

- **Unconditional generation** producing CAD models sampled from the learned data distribution.
- **Controllable generation** from natural language descriptions, semantic class labels specifying an object category (e.g., chair or table), or other mechanisms for constraining the geometry or topology of the resulting model.
- **Autocompletion** of partial CAD models provided by the user.
- **Editing** of CAD models provided by the user. This includes the generation of random edits, edits of parts specified by the user, or edits from natural language descriptions.
- **Interpolation** between two CAD models provided by the user.

Table 4 gives an overview of the surveyed literature for CAD generation, summarizing the used methods, the types of input, the used training datasets, the generation types, and the used evaluation metrics. We deliberately refrain from reporting quantitative metric values, as doing so would imply comparability across methods, which is not justified given differences in datasets, data preprocessing pipelines, metric formulations, and inputs. Nevertheless, to provide the reader with an intuition for the order of magnitude of the reported performance, we summarize the range of metric values for unconditional generation methods trained on the *DeepCAD* dataset [10] in Table 5.

Fig. 9 shows the Sankey diagram of the literature presented in this section with inputs, methods, and outputs. The individual nodes are arranged to minimize link intersections.

5.2.1. Command sequence generation

Command sequence generation has been extensively studied in the surveyed literature, with approaches utilizing almost exclusively transformer, diffusion, and foundation models. In the following, we group the surveyed literature by method and sort it by year of publication.

Transformer for Command Sequence Generation

DeepCAD [10] is the first work to address command sequence generation and introduces a learning-friendly representation that has been adopted by many subsequent methods operating on command sequences. A more detailed description of the *DeepCAD* approach and the learning-friendly command sequence representation is provided in Section 4.

SkexGen [15] facilitates the generation of command sequences from discrete codes, allowing the user to control the topology and the geometry of the generated model. The method trains a transformer-based autoencoder to encode topological, geometrical, and extrude variations within command sequences into three distinct codebooks [47]. A separate autoregressive transformer decoder is trained to select combinations of codes from the three codebooks. The selected codes are subsequently passed to the decoder part of the autoencoder to generate command sequences. A user can influence the topology and the geometry of the generated command sequence by selecting codes from the respective codebooks and conditioning further code selection of the transformer decoder on these codes. Beyond this, *SkexGen* enables interpolation between CAD models.

HNC-CAD [48] represents command sequences as hierarchical trees comprised of solid, profile, and loop levels, where each node is represented as a code from a codebook. In the first training stage, a transformer-based autoencoder is trained to encode loop, profile, and solid variations of the command sequences into three distinct codebooks [47]. In a subsequent stage, an autoregressive transformer is trained to generate a new tree by selecting codes from the learned codebooks. The tree generation can be conditioned on the embeddings of a partial command sequence. A second autoregressive transformer is trained to generate the complete CAD command sequence given the code tree and the embeddings of the partial command sequence. Besides the autocompletion of partial sequences, the code tree representation allows the user to edit the sequence at different levels and perform design-preserving editing by, e.g., only changing specific codes of a given code tree while leaving others unchanged.

CAD Translator [49] enables command sequence generation from natural language descriptions. A transformer-based autoencoder is combined with a contrastive learning strategy to align command sequence and text embeddings [71]. Additionally, the embeddings of both modalities are mixed [72] to form a hybrid embedding. The decoder is trained to reconstruct the input sequence given both the text embedding and the hybrid embedding. At inference time, the decoder generates command sequences solely from the text embeddings.

Table 4
Overview of the surveyed literature for CAD generation. Literature sorted by (1) method, and (2) year of publication.

Name	Method	Input					Training dataset	Generation type					Evaluation metrics					
		Text	CAD	CAD + Text	Class	Other		Uncond.	Controll.	Autocomp.	Editing	Interpolation	COV	MMD	JSD	Novel	Unique	Valid
Command sequence																		
DeepCAD [10]	Transformer	–	–	–	–	–	[10]	✓	–	–	–	–	✓	✓	✓	–	–	–
SkexGen [15]	Transformer	–	✓	–	–	✓	[10]	✓	✓	–	–	✓	✓	✓	✓	✓	✓	–
HNC-CAD [48]	Transformer	–	✓	–	–	–	[10]	✓	–	✓	✓	–	✓	✓	✓	✓	✓	–
CAD Translator [49]	Transformer	✓	–	–	–	–	[49]	–	✓	–	–	–	–	✓	✓	–	–	–
Text2CAD [16]	Transformer	✓	–	–	–	–	[10]	–	✓	–	–	–	–	–	–	–	–	–
CADTrans [50]	Transformer	–	✓	–	–	✓	[10]	✓	✓	✓	✓	–	✓	✓	✓	✓	✓	–
VQ-CAD [51]	Diffusion	–	–	–	–	–	[10]	✓	–	–	–	–	✓	✓	✓	✓	✓	–
Diffusion-CAD [52]	Diffusion	–	✓	–	✓	✓	[10]	✓	✓	✓	–	–	✓	✓	✓	✓	✓	✓
RECAD [53]	Diffusion	–	✓	–	–	✓	[10]	✓	✓	✓	✓	–	✓	✓	✓	✓	✓	✓
FlexCAD [54]	Foundation	–	–	✓	–	–	[10]	–	–	–	✓	–	–	–	–	–	–	–
CADFusion [55]	Foundation	✓	–	–	–	–	[10]	–	✓	–	–	–	✓	✓	✓	–	–	✓
CAD-Editor [56]	Foundation	–	–	✓	–	–	[10]	–	–	–	✓	–	–	–	✓	–	–	✓
CAD-GPT[a] [57]	Foundation	✓	–	–	–	–	[57]	–	✓	–	–	–	–	–	–	–	–	✓
Mamba-CAD [58]	Other	–	–	–	–	–	[58]	✓	–	–	–	–	✓	✓	✓	✓	✓	–
Boundary representation																		
SolidGen [59]	Transformer	–	–	–	✓	–	[10,59]	✓	✓	–	–	–	–	–	–	✓	✓	✓
Stitch-A-Shape [60]	Transformer	–	✓	–	✓	–	[10,11,61]	✓	✓	✓	–	–	✓	✓	✓	✓	✓	✓
BrepGPT[a] [62]	Transformer	✓	✓	–	✓	–	[10,11,61]	✓	✓	✓	–	✓	✓	✓	✓	✓	✓	✓
BrepGen [61]	Diffusion	–	✓	–	✓	–	[10,11,61]	✓	✓	✓	–	✓	✓	✓	✓	✓	✓	✓
CMT[a] [63]	Diffusion	✓	–	–	–	–	[10,11,63]	✓	✓	–	–	–	✓	✓	✓	✓	✓	✓
BRepGiff [64]	Diffusion	–	–	–	–	–	[10]	✓	–	–	–	–	✓	✓	✓	✓	✓	✓
BRepDiff [65]	Diffusion	–	✓	–	–	–	[10,11]	✓	–	✓	–	✓	✓	✓	–	–	–	–
HoLa[a] [66]	Diffusion	✓	–	–	–	–	[10]	✓	✓	–	–	–	✓	✓	✓	–	–	✓
CAD code																		
LLM4CAD [67]	Foundation	✓	–	–	–	–	training-free	–	✓	–	–	–	–	–	–	–	–	✓
CADCodeVerify [68]	Foundation	✓	–	–	–	–	training-free	–	✓	–	–	–	–	–	–	–	–	✓
CAD-Llama [69]	Foundation	✓	–	✓	–	–	[10]	✓	✓	✓	✓	–	✓	✓	✓	✓	–	✓
ReCAD[a] [70]	Foundation	✓	–	✓	–	–	[10]	–	✓	–	✓	–	–	–	–	–	–	✓

[a] Also applicable to CAD reconstruction.

Table 5
Range of reported values for unconditional generation on the *DeepCAD* dataset [10]. These values are provided solely to give an intuition for the order of magnitude of the metrics and should not be used for quantitative comparison between methods, as differences in experimental setups preclude direct comparability.

	Metrics					
	COV ↑	MMD ↓	JSD ↓	Novel ↑	Unique ↑	Valid ↑
Range	45.03–89.54	0.77–1.62	0.54–3.76	80.64–99.90	97.10–99.90	62.90–99.90

Text2CAD [16] facilitates command sequence generation from natural language descriptions of different levels of abstraction. A pre-trained text encoder [43] computes text embeddings, which are further adapted to the CAD-specific domain through an additional trainable transformer encoder block. A transformer decoder is trained to autoregressively reconstruct the ground truth command sequences given the text embeddings. To facilitate training, the authors design a multi-stage annotation pipeline that combines an LLM [73] and a VLM [74] to generate diverse, semantically rich descriptions of CAD models from the *DeepCAD* dataset. The generated descriptions cover different levels of abstraction, including both high-level and fine-grained parametric details of the models.

CADTrans [50] follows the ideas of *SkexGen* [15] and *HNC-CAD* [48] by representing command sequences as a hierarchical tree of discrete codes at the loop, profile, and solid levels, facilitating command sequence generation and different types of user control. The method combines codebook [47] and adversarial learning [75] to encode loop-, profile-, and solid-level information contained in the command sequences into three separate codebooks. In a second training stage, a transformer decoder is trained to generate sequences of discrete codes. The resulting hierarchical code tree then serves as guidance for a subsequent transformer model that generates the final parameterized command sequences. This approach enables interactive editing, autocompletion of partial models, and user-controlled generation by specifying different code tree structures.

Diffusion for Command Sequence Generation

VQ-CAD [51] presents the first diffusion-based framework for command sequence generation. The method is built on the hierarchical tree representation of *HNC-CAD* [48]. Separate adapted variational autoencoders (VAEs) [47] are trained to encode command sequence variations at the loop, profile, and solid levels into distinct codebooks. In the second training stage, ground truth command sequences are converted into corresponding code trees, which are then used to train a transformer decoder to reconstruct the command sequences. Finally, a discrete diffusion model [76] is trained to generate code trees, which are subsequently decoded into command sequences using the decoder trained in the previous stage. The authors propose to generate code trees from object images by incorporating their embeddings into the diffusion process via cross-attention [77,78].

Diffusion-CAD [52] facilitates command sequence generation from semantic class labels as well as user control over command types, command parameters, and geometric constraints. The method leverages a latent diffusion model, which allows the user to control the generation process in various ways. Input command sequences are first mapped into a continuous embedding space, where Gaussian noise is

iteratively added during the forward diffusion process. The denoising network [43] is trained to recover the original embeddings, from which an Mlp predicts command types and parameters. At inference time, Gaussian noise is sampled and denoised to produce embeddings that are subsequently decoded into command sequences. For command-type control, command-parameter control, and autocompletion, the embeddings of the corresponding commands, parameters, or partial sequences are computed and used to define a modified noise distribution, from which the initial noise vector is sampled alongside the standard Gaussian distribution. For geometric constraint control, equations representing the desired constraints, e.g., perpendicularity, are formulated and inserted into the sequence at each denoising step. For class-conditional generation, a classifier-guided approach is employed.

RECAD [53] introduces a two-stage diffusion-based framework for command sequence generation, facilitating autocompletion, control over extrusion boxes, and editing of sketch images. The authors propose a novel representation in which sketches are represented as binary images, paired with parameterized extrusion boxes that define the corresponding 3D operations. The generation process is decomposed into two stages. A transformer-based denoising network generates a set of extrusion boxes. Given the extrusion boxes, a UNet-based denoising network generates the associated sketch images.

Foundation Models for Command Sequence Generation

FlexCAD [54] facilitates the generation of design variations of command sequences provided by the user. The method first converts the command sequences into a structured text representation suitable for processing by a pre-trained LLM [79]. For model fine-tuning, hierarchical masking is applied at different construction levels of the command sequences, i.e., sketch, loop, and extrude levels. The LLM is fine-tuned to generate the unmasked sequence, given the masked sequence as well as natural language instructions. At inference time, the user selects parts of an input sequence to be masked, and the LLM generates multiple alternatives to replace the masks with.

CADFusion [55] allows the generation of command sequences from natural language descriptions. The method employs a two-stage training strategy. In the *sequential learning stage*, a LLM [79] is fine-tuned on pairs of textual descriptions and corresponding command sequences. In the *visual feedback stage*, the LLM is further optimized via direct preference optimization [80]. Based on the textual instructions, a VLM [81] provides feedback by scoring rendered images of the Cad models. An alternating training schedule is employed to balance sequential learning with visual feedback. To facilitate training, the authors construct a dataset of text–command sequence pairs by leveraging a VLM. The Cad model descriptions generated by the VLM are further refined through human feedback.

CAD-Editor [56] enables command sequence editing from natural language editing instructions. The method employs a two-stage training strategy. In the *locating stage*, an LLM [79] is fine-tuned to generate masked command sequences from the original unmasked sequences and textual editing instructions, where mask tokens denote positions requiring modification. In the *infilling stage*, a second LLM [79] is fine-tuned to generate the edited command sequences from the original sequences, the editing instructions, and the masked sequences generated in the first stage. To support training, the authors design an automated data synthesis pipeline that leverages foundation models [79,82] to generate triplets of data from the *DeepCAD* dataset, each consisting of an original command sequence, a textual editing instruction, and a corresponding edited sequence.

CAD-GPT [57] enhances a pre-trained large language model [83] with spatial reasoning capabilities to generate command sequences from text or image inputs. To this end, the authors extend the model's vocabulary with special tokens representing sketch plane orientations, 3D point coordinates, and 2D sketch coordinates. These tokens are further augmented with learnable positional embeddings to better encode spatial information. For fine-tuning, the authors construct a novel dataset consisting of image–text–command sequence triplets. To incorporate visual information, a pre-trained vision transformer is used in conjunction with an adaptive Mlp that aligns visual features with the language model's embedding space.

Other Methods for Command Sequence Generation

Mamba-CAD [58] addresses the limitation of short command sequences by leveraging state space models to generate substantially longer command sequences than prior methods. Specifically, the method employs Mamba [84], a discrete state space model that has demonstrated strong capability in modeling long-range dependencies. Similar to *DeepCAD* [10], the approach first performs pre-training of an encoder–decoder architecture to learn an expressive latent representation of command sequences. The encoder consists of four Mamba blocks followed by a compression block that reduces the dimensionality of the latent space. The decoder uses transposed convolution layers followed by a prediction head to infer command types and associated parameters. Subsequently, a 1D latent-GAN [85] is trained to generate latent vectors from Gaussian noise, which are then passed through the decoder to generate complete command sequences. While the method is limited to random generation, it demonstrates the effectiveness of state space models for producing longer command sequences compared to prior approaches.

5.2.2. BRep generation

Both transformer-based and diffusion-based approaches have been explored for Brep generation. In the following, we group the surveyed literature by method and sort it by year of publication.

Transformer for BRep Generation

SolidGen [59] facilitates Brep generation from semantic class labels, leveraging three autoregressive transformer models. The models sequentially generate the vertices, edges, and faces of the Brep. For edge and face generation, two additional pointer networks [86] are employed, enabling edges to refer to the previously generated vertices, and faces to refer to previously generated edges, ultimately generating a coherent Brep. Learned class embeddings are passed as start-of-sequence embeddings to the individual transformer models, enabling class-conditional generation.

Stitch-A-Shape [60] introduces a transformer-based Brep generative framework that separates geometry and topology generation. The method sequentially generates vertices, predicts their connectivity, identifies and classifies loops, and subsequently predicts curve and face geometry. Compared to prior Brep generative approaches, *Stitch-A-Shape* supports the generation of open surfaces and compound solids. Beyond random generation, the method enables autocompletion from partial face inputs as well as class-conditional generation.

BrepGPT [62] presents a novel Brep representation to facilitate Brep generation. The model is represented as a sequence of vertex features where each vertex feature contains information on its 3D coordinates, topological relationships to other vertices, as well as geometric and topological information derived from connected half-edges, termed the *Voronoi-half patch representation*. Vertex features are tokenized and subsequently passed to a GPT-style transformer, which is trained to autoregressively generate new vertex tokens. These tokens are subsequently detokenized to reconstruct the Brep. Beyond random generation, the method supports class-conditional generation as well as generation from point clouds, images, or text. In addition, *BrepGPT* enables Brep autocompletion and shape interpolation.

Diffusion for BRep Generation

BrepGen [61] enables both Brep generation from semantic class labels as well as the autocompletion of partial Breps and shape interpolation. The authors propose a structured latent geometry representation, transforming Brep into a hierarchical tree structure. As in *UV-Net* [12], position and local geometry are extracted for faces, edges, and vertices, respectively, and encoded as tree node features [29]. During the forward diffusion process, Gaussian noise is iteratively added to the node features, generating corrupted Brep geometry and

topology. Subsequently, four transformer-based denoisers sequentially remove the added noise in the tree, from faces to edges and then to vertices. Finally, a set of heuristics is used to find duplicated nodes in the tree and explicitly decode the generated Brep topology.

CMT [63] facilitates Brep generation from a variety of different inputs, including natural language descriptions, point clouds, or images. The method leverages two distinct VAEs [87] to obtain sequences of tokens representing the edges and the surfaces of the ground truth Brep. A unified multimodal encoder [88] computes text, point cloud, or image embeddings. Random masking is applied to both the sequences of edge and surface tokens using learnable mask tokens. Given the input embeddings, two consecutive diffusion-based networks [89] generate the unmasked edge and surface token sequences. A topology predictor employs cross-attention [1] to predict the topological relationships between the generated edge and surface tokens. To facilitate training, the authors introduce the *mmABC* dataset which contains over 1.3M Cad models annotated with multi-view images, textual descriptions, and point cloud information.

BRepGiff [64] proposes a 3D graph diffusion approach for Brep generation. For training, the Brep is converted into a 3D graph, where nodes correspond to face centroids in 3D space. Node features are computed similar to *UV-Net* [12]. In the forward diffusion process, graph edges are selectively removed and node features are progressively noised. For the denoising process, a graph attention network is trained to reconstruct the original, uncorrupted graph, which is subsequently decoded into a valid Brep. The method does not support guided generation. Instead, it focuses on the efficient random generation of valid, complex Brep, achieving lower computational costs compared to related approaches.

BRepDiff [65] employs a single-stage diffusion-based approach for Brep generation, enabling autocompletion, merging, and interpolation of Breps. The method introduces a novel Brep representation, explicitly representing each face as grid of points sampled on each faces' surface, along with a visibility mask [12]. During the forward diffusion process, Gaussian noise is iteratively added to the explicit point coordinates, producing a noisy model representation. A diffusion transformer model [90] is trained to reconstruct the denoised representation. Post-processing yields the final Brep.

HoLa [66] addresses the limitation of prior Brep generation methods that rely on multi-stage training pipelines and learn separate latent spaces for different Brep entities, i.e., vertices, edges, and faces. Instead, the method trains a VAE to learn a unified holistic latent space. The encoder employs a Gnn and a self-attention module to obtain holistic per-surface latent vectors. Given pairs of surface latent vectors a neural intersection module is trained to recover the latent vectors of their intersecting curve. A subsequent decoder takes pairs of surface latents together with the generated curve latents to reconstruct the input Brep. In a second training stage, a latent diffusion model is trained to generate a fixed set of surface latent vectors conditioned on an external vector. The generated surface latents are then processed by the neural intersection module and the decoder to generate the complete Brep. The conditioning vector may be derived from different modalities, including text, point clouds, or images.

5.2.3. CAD code generation

To date, only foundation models have been investigated for Cad code generation. In the following, we sort these approaches by year of publication.

LLM4CAD [67] represents one of the earliest efforts to leverage pre-trained LLMs for Cad generation. The method employs different GPT-4 models [82] in a zero-shot setting to generate CadQuery code [27] from text or text–image pairs. A self-debugging mechanism allows the LLM to iteratively refine the generated code in the presence of syntax errors until executable code is produced. For evaluation, the authors construct a synthetic dataset comprising five classes of mechanical components, i.e., shafts, nuts, flanges, springs, and gears.

Both *CADCodeVerify* [68] and *CAD-Llama* [69] facilitate Cad code generation from natural language descriptions. *CADCodeVerify* [68] proposes an iterative refinement loop comprising code generation from natural language descriptions, as well as the automated generation of feedback to refine the generated code. Initially, a VLM generates CadQuery code [27] from a natural language description. The generated code is executed by the Python interpreter. In case of syntax errors, the error message is used as feedback for the VLM. This process is repeated until the code can be executed and a Cad model can be rendered. Further, a feedback loop is used to refine the code and address discrepancies in the generated design. The feedback is generated in a two-step process. First, the VLM generates a set of *yes/no* verification questions given the natural language description of the object and a few example questions. Next, the VLM answers these questions using reference images of the rendered Cad model, using Chain-of-Thought [91], where each answer is accompanied by supporting reasoning. The question–answer pairs are passed to the VLM to generate feedback for code refinement. The generated feedback, previously generated code, natural language instructions, and, optionally, images of the rendered Cad model are passed to the VLM to generate the refined Cad code.

CAD-Llama [69] introduces a Python-like Cad code representation referred to as Structured Parametric CAD Code (SPCC), which is derived from command sequences. To construct the SPCC corpus, the authors leverage GPT-4o [92] to generate detailed structural text descriptions, which are integrated into the corresponding sections of the Cad code. A LLM [79] is then pre-trained on this SPCC corpus. To enhance contextual understanding, a pre-trained CLIP [32] model is employed to group similar Cad models, enabling the LLM to focus on subtle geometric differences during training. The pre-trained model is subsequently fine-tuned on several Cad-centric instructional datasets [69] derived from the SPCC corpus, each designed to address specific downstream Cad tasks, including autocompletion and editing.

ReCAD [70] combines pre-trained VLMs with RL to generate parameterized Cad code. Cad models are represented as hierarchical compositions of primitives, including loops, faces, sketches, and sketch–extrude operations, expressed in a structured code representation. Training proceeds in two stages: supervised fine-tuning on multimodal image–text–code data, followed by RL using group-relative policy optimization [93] to refine geometric accuracy without a value network. Curriculum learning is employed to progressively increase modeling complexity. The generation process is formulated as a question-answering task guided by the parameterized code representation, encouraging outputs that respect geometric and engineering constraints.

5.3. CAD reconstruction

Cad reconstruction facilitates the transformation of physical objects into precise, digital models, hence allowing for design modifications and automated quality control. In the surveyed literature, Cad models are reconstructed from diverse geometric inputs including (1) point clouds, (2) voxel representations, (3) image(s), (4) text and image(s) or (5) other inputs such as hand-drawn sketches.

Table 6 gives an overview of the surveyed methods for Cad reconstruction, summarizing the used methods, the types of input, the used training datasets, and the used evaluation metrics. We deliberately refrain from reporting quantitative metric values, as doing so would imply comparability across methods, which is not justified given differences in datasets, data preprocessing pipelines, metric formulations, and inputs. Instead, we refer the reader to the experimental sections of the respective papers. Fig. 10 shows the Sankey diagram of the literature presented in this section with inputs, methods, and outputs. The individual nodes are arranged to minimize link intersections.

Table 6
Overview of the surveyed literature for CAD reconstruction. Literature sorted by (1) method and (2) year of publication.

Name	Method	Input					Training dataset	Evaluation metrics					
		Point cloud	Voxel	Image(s)	Image(s) + Text	Other		ACC_{cmd}	ACC_{param}	CD	IoU	Valid	Other
Command sequence													
Free2CAD [94]	Transformer	–	–	–	–	✓	[94]	–	–	–	–	–	✓
View2CAD [19]	Transformer	–	–	✓	–	–	[19]	✓	✓	✓	–	✓	–
CAD-SIGNet [95]	Transformer	✓	–	–	–	–	[10]	–	–	✓	–	✓	–
TransCAD [17]	Transformer	✓	–	–	–	–	[10]	–	–	✓	–	✓	✓
Drawing2CAD [96]	Transformer	–	–	–	–	✓	[96]	✓	✓	✓	–	✓	–
Img2CAD [97]	Transformer	–	–	✓	–	–	[97]	✓	✓	✓	–	✓	–
PFCAD [98]	Transformer	✓	✓	–	–	✓	[10]	✓	✓	✓	–	✓	–
RenCAD [99]	Transformer	✓	–	–	–	–	[10]	✓	✓	✓	–	✓	–
CAD-Diffuser [100]	Diffusion	✓	–	–	–	–	[10]	✓	✓	✓	✓	✓	–
CADCrafter [101]	Diffusion	–	–	✓	–	–	[10]	✓	✓	✓	–	✓	–
GenCAD [102]	Diffusion	–	–	✓	–	–	[10]	–	–	–	–	–	✓
GenCAD-Self-Repairing [103]	Diffusion	–	–	✓	–	–	[103]	–	–	–	–	✓	✓
GenCAD-3D [104]	Diffusion	✓	–	–	–	✓	[10,104]	✓	✓	✓	✓	✓	✓
Sketch2Seq [105]	Other	–	–	–	–	✓	[105]	–	–	–	–	–	✓
DAFU-CAD [106]	Other	–	–	–	–	✓	[106]	–	–	–	–	–	✓
CSG tree													
CSG *Tree with 3D Primitives*													
UCSG-Net [107]	Neural–Analytical	–	✓	–	–	–	[108]	–	–	✓	–	–	–
CSG-Stump [109]	Neural–Analytical	✓	–	–	–	–	[108]	–	–	✓	–	–	–
CAPRI-Net [110]	Neural–Analytical	–	✓	–	–	–	[11,108]	–	–	✓	–	–	✓
D2CSG [111]	Neural–Analytical	✓	–	–	–	–	training-free	–	–	✓	–	–	✓
CSG-Net [112]	Other	–	✓	–	–	–	[112]	–	–	✓	✓	–	–
CSG *Tree with Sketch-based Primitives*													
Point2skh [113]	Transformer	✓	–	–	–	–	[10]	✓	✓	✓	–	–	✓
ExtrudeNet [18]	Neural–Analytical	✓	–	–	–	–	[108]	–	–	✓	✓	–	✓
ReconPris [114]	Neural–analytical	–	✓	–	–	–	[10]	–	–	–	✓	✓	✓
Point2Cyl [115]	Neural–Analytical	✓	–	–	–	–	[10,40]	–	–	–	–	–	✓
SECAD-Net [116]	Neural–Analytical	–	✓	–	–	–	[11,40]	–	–	✓	–	–	✓
SfmCAD [117]	Neural–Analytical	–	✓	–	–	–	[11,108,117]	–	–	✓	–	–	✓
MV2Cyl [118]	Neural–Analytical	–	–	✓	–	–	[10,40]	–	–	–	–	–	✓
Boundary representation													
ComplexGen [119]	Transformer	✓	–	–	–	–	[11]	–	–	–	–	–	✓
CADDreamer [120]	Diffusion	–	–	✓	–	–	[10,11]	–	–	✓	–	–	✓
Point2CAD [121]	Neural–Analytical	✓	–	–	–	–	[11]	–	–	✓	–	–	✓
Split-and-fit [122]	Neural–Analytical	✓	–	–	–	–	[11]	–	–	✓	–	–	✓
CAD code													
CAD-Recode [20]	Foundation model	✓	–	–	–	–	[20]	–	–	✓	✓	✓	–
OpenECAD [123]	Foundation model	–	–	–	✓	–	[123]	–	–	–	–	–	✓
CAD-Assistant[a] [124]	Foundation model	✓	–	✓	–	✓	training-free	–	–	–	–	–	✓
CAD-Coder [125]	Foundation model	–	–	–	✓	–	[125]	–	–	–	✓	✓	–

[a] Also applicable to CAD generation.

5.3.1. Command sequence reconstruction

Both transformer and diffusion-based approaches have been explored for command sequence reconstruction. In the following, we group the surveyed literature by method and sort it by year of publication.

Transformer for Command Sequence Reconstruction

Free2CAD [94] introduces a framework for generating command sequences from hand-drawn 3D sketches. The central challenge is to group the individual strokes of the input sketch according to the corresponding CAD commands. To address this, the authors combine a transformer-based network for stroke grouping with a Cnn-based module that reconstructs operations by predicting command parameters for each stroke group and refining those of previously predicted groups. This design enables the autoregressive generation and refinement of command sequences.

View2CAD [19] enables command sequence reconstruction from multi-view images. A pre-trained image encoder [126] extracts image features, which are passed to a preliminary transformer decoder to generate intermediate command sequence embeddings. These embeddings are processed by a subsequent autoregressive transformer decoder, generating the final command sequence.

Both *CAD-SIGNet* [95] and *TransCAD* [17] facilitate command sequence reconstruction from point clouds. *CAD-SIGNet* [95] extracts point cloud features using a local feature aggregation module [127]. Given the point cloud features, a transformer decoder autoregressively generates sequences of sketch and extrude tokens. In contrast to the conventional design workflow, where sketch commands precede their corresponding extrusions, *CAD-SIGNet* first predicts extrusion tokens, followed by the associated sketch tokens. The generated extrusion tokens guide the selection of a relevant subset of the point cloud for subsequent sketch generation. The user may provide a partial command sequence, which is autoregressively completed based on the given point cloud.

TransCAD [17] is comprised of a point cloud encoder [46], a loop-extrusion decoder, and parallel loop and extrusion decoders. The loop-extrusion decoder takes a set of learned embeddings as input and generates a sequence of high-level embeddings. The decoder integrates

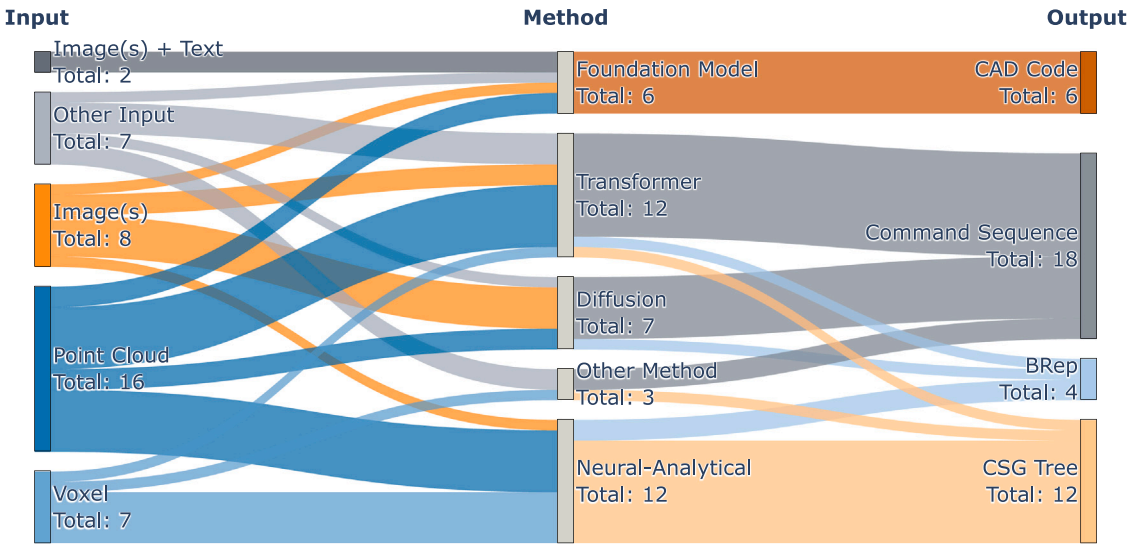


Fig. 10. Sankey diagram of the literature addressing CAD reconstruction with inputs, methods, and outputs.

point cloud features via its cross-attention module. Subsequently, each embedding is classified as a loop or as an extrusion embedding. Based on the classification, the individual embeddings are processed by either the loop or the extrusion decoder, which predicts the respective command parameters. In the last step, a loop refiner predicts the error of the predicted quantized loop parameters, improving parameter quality.

Drawing2CAD [96] presents the first approach to enable the reconstruction of command sequences directly from 2D technical drawings represented as scalable vector graphics (SVG). To this end, the method introduces a network-friendly representation of SVG drawings. The input drawings are embedded using an Mlp followed by a transformer encoder. Based on the embedding vector, two separate transformer decoders are employed to predict command types and command parameters, respectively. Since each command type requires a specific set of parameters, the method conditions the parameter prediction on the predicted command type, thereby enhancing parameter prediction. To facilitate training, the authors introduce the *CAD-VGDrawing dataset*, derived from the *DeepCAD dataset*, which comprises engineering drawings paired with their corresponding command sequences.

Img2CAD [97] proposes a transformer-based architecture augmented with additional geometric feature extraction to reconstruct command sequences from single-image inputs. Inspired by prior work [128], the method derives a vectorized wireframe representation from the input image that captures line segments and their associated endpoints. A transformer decoder incorporates both wireframe features and image features extracted by a pre-trained vision transformer, to autoregressively generate the output command sequence. For training and evaluation, the authors introduce two new datasets: (1) *ABC-mono*, an extension of the *ABC dataset* that includes command sequences paired with rendered images or sketches, and (2) *KOCAD*, which contains around 100 command sequences aligned with real-world images of the corresponding manufactured objects captured under varying conditions.

PFCAD [98] introduces a dual-transformer architecture for reconstructing command sequences from discrete 3D representations, including point clouds, meshes, and voxels. A global transformer decoder first processes geometry features produced by a suitable encoder to generate a sequence of global CAD features. These features along with a target mask are then passed to a second, autoregressive transformer decoder, which predicts a sequence of local CAD features. An attention-based parameter adaptor processes global and local features to predict the final command types and associated command parameters of the command sequence.

RenCAD [99] proposes an end-to-end autoregressive framework for reconstructing command sequences from point clouds. A central component is a group-based point cloud tokenization module that captures local geometric structures by partitioning the point cloud into local groups using farthest point sampling followed by k-nearest neighbors. Each group is can be seen as a token and is embedded using a leightweight PointNet++ [46]. The resulting embeddings are further processed by a transformer-based point cloud encoder to obtain the final point cloud embeddings. Point cloud and command sequence embeddings, are fed into a transformer decoder that autoregressively generates latent command embeddings. A final generator with three MLP-based sub-decoders predicts the command type, command parameters, and extrusion parameters, while applying command-specific masks to exclude invalid parameter predictions.

Diffusion for Command Sequence Reconstruction

CAD-Diffuser [100] presents a novel, multimodal discrete diffusion approach to reconstruct command sequences from point clouds. The method leverages dedicated point cloud and command sequence tokenizers to combine both modalities at a token level, yielding a joint token sequence for processing by the diffusion model. During the forward diffusion process, a volume-based noise schedule is employed, iteratively masking tokens in the input sequence until the sequence is fully masked. The denoiser [43] is trained to recover the unmasked command sequence from the masked command sequence. During inference, the command sequence is reconstructed, given the tokenized input point cloud.

CADCrafter [101] presents a latent diffusion framework for reconstructing command sequences from both single-view and multi-view images of an object. The diffusion model operates in the latent space of command sequences learned by a transformer-based autoencoder [10]. In the forward diffusion process, Gaussian noise is iteratively added to the latent vector. The denoising network is trained to reconstruct the original latent vector. A pre-trained geometry extractor [129], DINO-V2 [130], and a transformer-based geometry encoder extract image features that serve as conditioning for the denoising network. Knowledge from the multi-view encoder is distilled to a single-view encoder by reducing the distance between the respective image features. In the last step, the diffusion model is fine-tuned via direct preference optimization [80] using feedback from the Python interpreter, which indicates whether generated command sequences are executable or not. Training on synthetic textureless images enhances the model's applicability to real-world image inputs.

GenCAD [102] introduces a four-stage diffusion-based framework for reconstructing command sequences from images. The method employs CAD–image contrastive pre-training to align image embeddings

with command sequence embeddings, obtained by training a transformer-based encoder–decoder in a prior stage. Subsequently, an image-conditioned latent diffusion model [29] with a ResNet–Mlp denoising neural network [131] is trained to generate command sequence embeddings from image input. During inference, the diffusion model generates a command sequence embedding from noise conditioned on the image embedding, which is then decoded into a command sequence using the transformer-based decoder trained in the initial stage.

GenCAD-Self-Repairing [103] extends the work of *GenCAD* [102] by explicitly targeting the generation of invalid command sequences, i.e., sequences that cannot be executed by a Cad kernel. The authors introduce a guided latent diffusion framework that biases generation toward latent vectors which result in valid command sequences. Guidance is provided by a latent space classifier that estimates the feasibility of a latent vector, and a self-supervised regression model that is trained to predict latent vectors associated with valid execution. Further, the method employs a self-repair mechanism. When execution of a generated command sequence fails, the corresponding latent vector is passed through the regression model to predict a new latent vector, ultimately increasing the rate of valid command sequences compared to *GenCAD*. To train the latent classifier, the authors construct a dataset based on the *GenCAD dataset*, in which latent vectors are paired with validity labels.

GenCAD-3D [104] extends *GenCAD* [102] by enabling command sequence generation from meshes, point clouds, and point clouds augmented with normal vectors. To this end, modality-specific encoders are trained, and their latent embeddings are aligned with command sequence embeddings via contrastive learning. In addition, the authors introduce a dataset that augments *DeepCAD* [10] with synthetically generated models to address sequence-length imbalance.

Other Methods for Command Sequence Reconstruction

Sketch2Seq [105] is closely related to the work by *Free2CAD* [94] and proposes a framework for command sequence reconstruction from single hand-drawn sketches. To this end, the authors combine a feature extraction module with a graph attention network to segment the input sketch according to modeling operations, i.e., extrude, cut, fillet, and chamfer. Given the segmented sketch, multiple feasible candidate sequences of modeling operations are generated, executed, and optimized using the input sketch. The sequence that minimizes the discrepancy between the input sketch and the sketch of the generated model is selected. For training, the authors derive a novel feature-based Cad sketch segmentation dataset from the *Fusion 360 segmentation* dataset [13].

DAFU-CAD [106] extends prior sketch-to-Cad generation methods by introducing a depth-assisted feature-disentanglement network for classifying sketches into corresponding modeling operations. The network is jointly trained on sketches and depth maps, disentangling sketch features into geometry features shared with the depth domain and style features specific to the sketch domain, which improves robustness to unseen drawing styles. After operation classification, operation-specific parameters are predicted and refined using the input sketch and the rendered Cad model. To enable training across diverse operations, the authors introduce a dataset covering multiple drawing styles and a range of modeling operations, including spirals, pyramids, and spheres, which are challenging for prior methods. However, the approach is limited to single operations and does not generate sequences of modeling operations.

5.3.2. CSG tree reconstruction

As previously discussed, we divided Csg tree reconstruction into approaches that directly use 3D primitives and those that decompose 3D primitives into 2D sketches and corresponding Cad operations. Some of the presented methods may not directly produce outputs that are compatible with integration into Cad software, such as those based on implicit shape representations like signed distance functions [42]. In such cases, post-processing is required to convert the results into parametric sketch curves or 3D primitives. In the following, the term shape embedding refers to the feature vector computed from the input point cloud, voxel grid, or image using a suitable encoder.

5.3.2.1. 3D primitives.

With the exception of a single work, exclusively neural–analytical approaches have been employed for Csg tree reconstruction using 3D primitives. In the following, we group the surveyed literature by method and sort it by year of publication.

Neural–Analytical Methods for CSG Tree Reconstruction with 3D Primitives

The following approaches employ Mlp-based neural networks to predict the parameters of 3D primitives and the Boolean operations to combine them. Implicit shape representations are obtained by computing signed distance or occupancy values [42,132] for sampled 3D points, enabling unsupervised training through reconstruction losses between predicted and ground truth occupancies.

UCSG-Net [107] facilitates reconstruction from a voxelized input shape. An Mlp predicts the parameters of a fixed set of cuboid and sphere primitives from the shape embedding. Subsequent trainable Csg-layers dynamically perform Boolean operations, yielding test point occupancy values [132] of the reconstructed shape. Each layer is informed about the current reconstruction state by a gated recurrent unit (GRU) [133], enabling a dynamic selection of the shapes to combine.

CSG-Stump [109] proposes a novel, three-layer reformulation of the classic Csg tree representation, referred to as *CSG-Stump*, facilitating reconstruction from point clouds. The nodes at the three different levels perform complement, intersection, and union operations, respectively. The method employs a 3D Cnn [134] and an Mlp to predict the parameters of a fixed number of cuboid, sphere, cylinder, and cone primitives. Three distinct linear layers predict connection matrices, determining which primitives to select for the operations at the respective *CSG*-Stump layers. Given the connection matrix, test point occupancy values [132] of the reconstructed shape are obtained using the *CSG-Stump* formulation.

CAPRI-Net [110] facilitates reconstruction from a voxelized input shape and represents primitives as parameterized convex quadric surfaces. Given the shape embedding, an Mlp predicts the parameters of a fixed set of quadric surface primitives. A trainable selection matrix selects a subset of primitives for combination using sequential intersection, union, and difference operations, yielding test point occupancy values [132] of the reconstructed shape.

D2CSG [111] operates on a learnable feature vector rather than directly on the input shape, enabling reconstruction from point clouds. For each input point cloud, this feature vector and the neural network parameters are jointly optimized to reconstruct the input shape. The authors argue that such per-shape optimization is particularly suitable for Cad data due to the high structural and topological diversity of Cad models. Given the feature vector, an Mlp predicts two sets of primitives, each containing convex and inverse-convex primitives. These sets are processed by two separate branches, each equipped with trainable Csg-layers that perform intersection and union operations. The resulting intermediate shapes are then combined via a difference operation to obtain test point occupancy values [132] of the reconstructed shape. *D2CSG* is capable of modeling complex concavities in the reconstructed shapes, which were not achievable in previous methods.

Other Methods for CSG Tree Reconstruction using 3D Primitives

CSG-Net [112], employs an RNN to generate sequences of Csg instructions from voxelized input shapes. These instructions specify both the generation of 3D primitives (sphere, cuboid, cylinder) and their combination through Boolean operations (union, difference, intersection). The network is trained either in a supervised setting, using ground truth instruction sequences, or via RL, where the geometric reconstruction loss between the input shape and the generated shape serves as a reward signal.

5.3.2.2. Sketch-based primitives.

Similar to reconstruction with 3D primitives, the majority of existing work employs neural–analytical methods for Csg tree reconstruction using sketch-based primitives, with only a single study investigating

the use of a transformer-based model. In the following, we group the surveyed literature by method and sort it by year of publication.

Transformer for CSG Tree Reconstruction with Sketch-based Primitives

Point2skh [113] facilitates CSG tree reconstruction from point clouds. The individual primitives are represented as sequences of CAD commands, as in [10]. In contrast to neural–analytical methods that implicitly represent sketches using signed distance functions, *Point2skh* directly predicts explicit sketch primitive parameters, enabling more accurate and faithful sketch reconstruction. A point cloud segmentation network [46] decomposes the input point cloud into subsets of points corresponding to individual extrusion operations [115]. Each point subset is then processed by a denoising transformer that predicts the parameters of the associated sketch commands, producing a sequence of sketch operations. Additionally, a base–barrel point segmentation is performed to facilitate the estimation of extrusion parameters, yielding the complete sketch-and-extrude primitive information. The final model is reconstructed by combining the individual extrusion primitives.

Neural–Analytical Methods for CSG Tree Reconstruction with Sketch-based Primitives

ExtrudeNet [18] facilitates reconstruction from point clouds. Sketches are represented as rational cubic Bézier curves [135]. Given the shape embedding, a neural network predicts sketch curve parameters, the parameters of the associated sketch plane, and the extrusion height, for a fixed number of primitives. These predictions are used to construct the corresponding 3D primitives. A second neural network predicts connection matrices as in *CSG-Stump* [109], which are used to combine the primitives using the *CSG-Stump* formulation [109], yielding test point occupancy values [132] of the reconstructed shape. The neural network is trained in an unsupervised fashion, using a reconstruction loss between the predicted and ground truth occupancy values.

ReconPris [114] leverages a set of predefined extrusion recipes, each consisting of a fixed number of primitives and Boolean operations derived from common extrusion patterns, enabling reconstruction from voxelized inputs. Primitives are represented as 2D sketch images and 1D arrays describing extrusion limits. Given the shape embedding, each recipe employs deconvolutional networks to predict sketch images and extrusion limits from which voxelized 3D primitives are computed. The primitives are combined with the Boolean operations in the respective recipe. The network is trained with both unsupervised geometric reconstruction losses and supervised losses using ground truth sketch images and extrusion limits. At inference time, a search procedure retrieves parametric sketches from the *DeepCAD dataset* [10] that best match the predicted sketch images.

Point2Cyl [115] facilitates reconstruction from point clouds and defines a primitive generated by the extrusion of a 2D sketch as an *extrusion cylinder*. Points lying in the planes at either extent of the extrusion cylinder are referred to as *base* points, while points located along its lateral surface are termed *barrel* points. A neural network is trained to assign each point from the input point cloud to an extrusion cylinder, classify it as either a base or barrel point, and estimate its normal vector. From these predictions, the extrusion center, extrusion axis, and extrusion extent of each cylinder are analytically recovered. For each cylinder, barrel points are projected onto the corresponding sketch plane along the extrusion axis. A sketch implicit encoder–decoder network is trained to predict the signed distance values [42] of the projected points to the ground truth curve. During inference, the marching squares algorithm [136] is applied to obtain closed sketch curves of each extrusion cylinder.

SECAD-Net [116] employs an extrusion box network to predict a fixed set of sketch plane and extrusion parameters from a voxelized input shape. 3D test points are projected onto the corresponding sketch planes, and sketch implicit neural networks are trained to predict their signed distance values [42] to the associated sketch curves. The network's predictions, together with the previously estimated extrusion heights, are used to compute occupancy values [132] for the resulting 3D primitives. These primitives are then combined using a union operation. During inference, closed B-spline curves are reconstructed from the predicted signed distance values of the corresponding curves [137, 138]. The neural network is trained in an unsupervised fashion, using a reconstruction loss between the predicted and ground truth occupancy values.

SfmCAD [117] employs a two-stage training strategy to reconstruct sweep primitives from voxelized input shapes. In the first stage, a neural network is trained to predict the parameters of a fixed set of sweeping paths, as well as the parameters of rigid boxes distributed along these paths. During the second stage, the paths learned in the first stage are used to refine the coarse sweep primitives. To achieve this, a sketch implicit neural network is trained to predict the signed distance values [42] of 2D sample points with respect to the primitives' sketch curves. Combined with the sweeping paths, the occupancy values [132] of the sweep primitives are then computed and merged using a union operation, yielding the test point occupancy values of the reconstructed shape. The framework also supports the use of loft operations in place of sweep operations.

MV2Cyl [118] proposes a framework for reconstructing extrusion cylinders from multi-view images. The method leverages a 2D segmentation network [139] to jointly optimize a density field and a semantic field. Given a point sampled from 3D space, these fields predict the likelihood of the point lying on a surface or a curve, determine the corresponding extrusion cylinder, and classify the point as a base point or a barrel point. Using these optimized fields, semantically enriched point clouds are generated. The subsequent reconstruction of extrusion cylinders from these point clouds follows a similar procedure to that of *Point2Cyl* [115].

5.3.3. BRep reconstruction

Transformer models, diffusion models, and neural–analytical approaches have been explored for BREP reconstruction. In the following, we group the surveyed literature by method and sort it by year of publication.

Transformer for BRep Reconstruction

ComplexGen [119] reconstructs a BREP chain complex [140] composed of vertices, edges, and faces, along with their topological relationships, from an input point cloud. Given point cloud features, three parallel transformer decoders generate vertices, edges, and faces, while simultaneously modeling their topological dependencies. During the generation process, each decoder attends to information from the neighboring decoders, promoting consistent BREP reconstruction. A subsequent global optimization step, followed by geometric refinement, ensures the validity and coherence of the resulting BREP.

Diffusion for BRep Reconstruction

CADDreamer [120] facilitates BREP reconstruction from single-view images. From the single image, a segmented 3D mesh is generated using a cross-domain diffusion model [141] and NeuS [142]. Each segment of the 3D mesh corresponds to a geometric primitive, i.e., plane, cylinder, cone, sphere, torus, or boundary line. Primitive parameters are initially estimated using RANSAC [143]. Subsequently, a geometric optimization algorithm recovers both geometric and topological relationships between the primitives. A topological representation is then extracted from the segmented mesh and employed to perform topology-preserving intersection operations between each primitive and its neighboring surfaces, resulting in the construction of BREP faces, edges, and vertices.

Neural–Analytical Methods for BRep Reconstruction

Point2CAD [121] introduces a pipeline for BREP reconstruction from point clouds, combining learning-based point cloud segmentation, surface fitting, and topological clipping. For surface fitting, the method explores both classical primitive fitting, i.e., planes, spheres, cylinders, and cones, and a novel fitting scheme for freeform surfaces based

on implicit neural representations [144]. The pipeline further applies surface clipping, surface intersection for edge detection, and pairwise edge intersections for vertex identification, ultimately reconstructing a complete Brep.

Split-and-fit [122] approaches Brep reconstruction from point clouds as a space partitioning problem. The method employs a UNet-like [139] model to transform point clouds into Voronoi diagrams [145]. Region growing is applied to extract Voronoi cells, and geometric primitives are fitted to their boundaries. The connectivity of the Voronoi cells further enables the inference of topological relations, yielding a consistent Brep.

5.3.4. CAD code reconstruction

The surveyed approaches on Cad code reconstruction rely exclusively on foundation models. In the following, we sort them by year of publication.

CAD-Recode [20] fine-tunes a pre-trained LLM [146] to reconstruct CadQuery code [27] from point cloud input. A point cloud projection module encodes the point cloud into a sequence of tokens, which is concatenated with tokenized CadQuery code and passed to the LLM for next-token prediction during training. At inference time, the model autoregressively generates CadQuery code from the point cloud tokens. To facilitate training, the authors introduce a procedurally generated dataset comprising one million synthetic Cad models represented as sketch-and-extrude sequences in CadQuery code.

OpenECAD [123] facilitates Cad code reconstruction from object images. The authors introduce a dedicated code format, *OpenECAD* code, along with an interface to the PythonOCC [147] APIs, enabling the construction of 3D models from code. The authors present several *OpenECAD* models that combine different pre-trained LLMs [8,148,149] with visual encoders [32,150] to achieve multimodal capabilities. Each model is trained using the LLaVA method and dataset [151], and subsequently fine-tuned to generate *OpenECAD* code from one or more images of an object. For fine-tuning, the authors construct multiple *OpenECAD* datasets containing image–code pairs, where the code is translated from command sequences in the *DeepCAD* dataset. Additionally, a VLM is leveraged to annotate the *OpenECAD* code with natural language comments describing individual modeling operations.

CAD-Assistant [124] is a general-purpose, training-free approach capable of processing diverse types of input, including text, images, or point clouds, facilitating diverse Cad-related tasks. These tasks include sketch autoconstraining [152], Cad question answering, and Cad reconstruction from point clouds or images. The system consists of a planner, an environment, and a set of Cad-specific tools. A VLM [82] functions as the planner, generating plans and corresponding actions, based on the user query and the current design state. The actions that are generated as Python code are executed within the environment and may invoke tools from the provided toolset via their Python APIs. The execution results are iteratively fed back to the planner, forming a closed decision loop that continues until the user's request is fulfilled.

CAD-Coder [125] facilitates the reconstruction of CadQuery code [27] from input images. Initially, an Mlp is trained to align the image features obtained from a pre-trained image encoder [32] with the word embeddings of the LLM [83]. During the second training stage the LLM is fine-tuned to generate CadQuery code from an input image and a natural language description of the task. For fine-tuning the authors present the *GenCAD-Code* dataset containing pairs of synthetic images and corresponding CadQuery code. The authors demonstrate the generalizability of the method to real-world images of simple parts. Furthermore, the approach enables users to iteratively modify the generated models through natural language instructions.

5.4. CAD reverse engineering

Cad reverse engineering enables the enhancement of Cad data, facilitating data exchange across different software platforms or organizations, where the design history of Cad models is frequently lost.

Table 7
Overview of the surveyed literature for Cad reverse engineering. Literature sorted by (1) method and (2) year of publication.

Name	Method	Training dataset	Evaluation metrics					
			ACC_{cmd}	ACC_{param}	CD	IoU	Valid	Other
CADParser [153]	Transformer	[153]	–	–	–	✓	–	–
BRep2Seq [154]	Transformer	[154]	✓	✓	✓	✓	✓	–
eCAD-Net [155]	Transformer	[10]	✓	✓	–	–	✓	✓
CADCL [156]	Transformer	[10, 157]	✓	✓	✓	–	✓	–
Fusion 360 Gallery [40]	Other	[40]	–	–	–	✓	–	✓
ZoneGraphs [158]	Other	[40]	–	–	–	✓	–	✓
RLCAD [159]	Other	[10,11]	–	–	✓	✓	–	✓

Table 7 gives an overview of the surveyed methods for Cad reverse engineering, summarizing the used methods, the used training datasets, and the used evaluation metrics. We deliberately refrain from reporting quantitative metric values, as doing so would imply comparability across methods, which is not justified given differences in datasets, data preprocessing pipelines, metric formulations, and inputs. Instead, we refer the reader to the experimental sections of the respective papers. Fig. 11 shows the Sankey diagram of the literature presented in this section with inputs, methods, and outputs. The surveyed approaches on Cad reverse engineering exclusively generate command sequences from Brep input. Although transformer architectures remain the predominant choice, several non-transformer approaches have also been explored. In the following, we group the surveyed literature by method and sort year of publication. We conclude this section by examining why Cad reverse engineering remains less explored than Cad generation and Cad reconstruction, and by outlining three major challenges that limit progress in this task.

Transformer for CAD Reverse Engineering

CADParser [153] leverages *BRepNet* [13] to extract face, edge, and coedge features, referred to as local Brep features. Average pooling is applied to the local features to obtain a global Brep feature vector. An autoregressive transformer decoder generates the command sequence, integrating the local features via cross-attention [1]. The global feature vector is concatenated with the input command sequence embeddings, providing global shape information. To facilitate training, the authors introduce a new dataset of command sequences, including sketch, extrusion, revolution, fillet, and chamfer operations.

BRep2Seq [154] leverages *UV-Net* [12] to extract face features. In addition, a graph distance matrix is constructed, containing the shortest distance between any two Brep faces. A transformer encoder processes both the face features and the distance matrix to compute a latent vector representing the input Brep. The decoder processes the latent vector to generate the command sequences. The network is trained on pairs of Breps, one representing the overall shape and the other containing detailed features. This setup enables training of two distinct encoders, capturing different levels of detail in the Brep. At inference time, the summed latent vectors of the individual encoders are passed to the decoder, which generates the command sequence.

eCAD-Net [155] leverages *UV-Net* [12] to obtain a Brep shape embedding. A transformer decoder generates a command sequence given the shape embedding. Additionally, the generated sequences undergo post-processing to fit the dimensions of the input Brep. This process includes scaling and fine-tuning of the command parameters.

CADCL [156] presents a transformer-based approach for reverse engineering, including a contrastive learning objective. The method follows a two-stage training strategy. In the first stage, a transformer encoder–decoder architecture, similar to *DeepCAD* [10], is trained on command sequences to learn expressive sequence embeddings. In the second stage, the Brep is converted into a graph representation as in

UV-Net [12]. A Gnn-based Brep encoder computes Brep embeddings form which ground truth command sequences are reconstructed by a transformer decoder, similar to that of the first training stage. Additionally, a contrastive learning is applied to align the command sequence embeddings obtained in the first stage with the corresponding Brep embeddings, encouraging the Brep encoder to capture information relevant to the sequential and structural characteristics of the command sequence. Further, the method employs a node-masking strategy during training of the Brep graph encoder, randomly masking out parts of the graph, promoting the network to focus on structural semantics rather than low-level feature variations.

Other Methods for Reverse Engineering

To reduce the search space and take advantage of the information provided by the Brep, *Fusion 360 Gallery* [40] and *ZoneGraphs* [158] focus on generating sequences of face extrude commands. Each command involves selecting a starting face, a parallel end face that defines the extrusion, and a Boolean operation including union, intersection, or difference.

Fusion 360 Gallery [40] introduces the *Fusion 360 Gym*, a RL environment for training agents to reconstruct target Breps. The environment state includes both the current reconstruction and the target model, represented as in *UV-Net* [12]. A neural network based on message passing networks (MPNs) [160] is employed to predict probabilities over candidate actions. The reward is based on the geometric similarity between the current reconstructed Brep and the target Brep. At inference time, agent predictions are employed to guide a search over face extrude sequences, enabling the exploration of multiple plausible command sequences through different search strategies.

ZoneGraphs [158] introduces the *zone graph*, a geometric representation in which *zones* are solid regions formed by partitioning space using the faces of the target Brep, extended into infinite surfaces. Nodes in the zone graph represent individual zones, while edges encode geometric adjacencies. Each zone is represented as a point cloud, with additional node features indicating whether it belongs to the current reconstruction, the target Brep, and the associated Boolean operation. A point cloud encoder [161] and a MPN are used to predict probabilities over possible next actions, guiding the search process. Unlike [40], the network is trained in a fully supervised manner.

RLCAD [159] extends Fusion 360 Gallery [40] by introducing a Cad-based RL environment that additionally supports revolution operations, enabling the reverse engineering of more complex command sequences. A UV-Net [12] is pre-trained on Brep face-adjacency graphs using contrastive learning. The method employs Proximal Policy Optimization (PPO) [162] to train a Cad policy that predicts Cad command actions. The environment state comprises the target Brep, the current reconstructed Brep, and the sequence of previously executed actions. In addition to geometric reward terms which exhibit limited sensitivity to subtle Cad features, the approach introduces a neural reward computed as the cosine similarity between the embeddings of the target and the current reconstructed Brep, obtained by the pre-trained UV-Net.

Challenges in Reverse Engineering

As previously noted, Cad reverse engineering remains one of the least explored tasks in the literature, as it requires recovering not only the final geometry but also a valid and interpretable command sequence, making it substantially more complex than Cad generation or reconstruction. The following challenges contribute to this difficulty:

- **Sequence Ambiguity and Design Intent:** In Cad reverse engineering, multiple distinct command sequences can produce identical final geometries, resulting in a many-to-one relationship between geometry and command sequence. Unlike Cad generation or reconstruction, where command sequences primarily serve as a means to obtain geometry, reverse engineering treats the command sequence as the central modeling artifact to be recovered. Consequently, a model must infer not only which operations reproduce the target Brep, but also why specific modeling decisions including command ordering, reference selection, and parameterization were made. Since such design intent is not explicitly encoded in the final geometry, geometrically valid sequences may differ substantially in robustness, interpretability, and editability.
- **Dataset Limitations:** Existing datasets typically cover a limited set of modeling operations, such as basic sketches and extrusions, and do not adequately reflect the complex, feature-based workflows used in professional Cad design. Consequently, models trained on these datasets lack exposure to realistic command sequences, parametric constraints, and editing strategies, which limits their applicability to real-world reverse engineering tasks.
- **Persistent Naming Problem:** Reverse engineering is further complicated by the persistent naming problem [163], whereby geometric entities such as faces or edges are assigned identifiers that may change as the model evolves. Small deviations early in a reconstructed command sequence can alter intermediate topology, causing later operations to reference non-existent entities and fail. As a result, errors often lead to complete breakdowns of downstream operations rather than minor geometric inaccuracies, making commands that refer to specific geometric entities a major challenge for learning-based Cad reverse engineering.

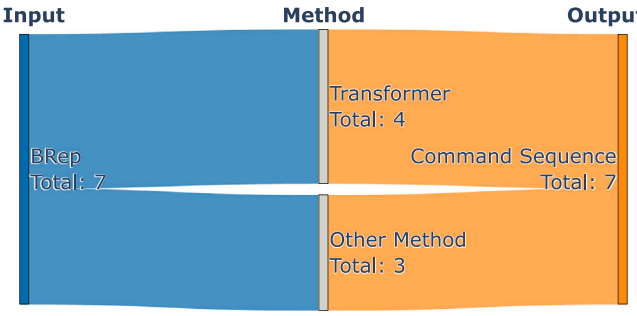


Fig. 11. Sankey diagram of the literature addressing Cad reverse engineering with inputs, methods, and outputs.

In Section 7 we discuss further challenges and outline possible future directions to address them.

6. Datasets

Learning-based methods for 3D Cad model generation require large-scale, high-quality datasets containing Cad data, and, depending on the specific task, additional annotations. In the following, we present the datasets used for training or introduced in the surveyed literature. We group the datasets into five categories:

1. **Original datasets**: Datasets created by gathering new Cad data from different sources.
2. **Extended datasets**: Datasets created by extending, transforming, or reconstructing Cad data of existing datasets.
3. **Annotated datasets**: Datasets created by annotating Cad models of existing datasets with different kinds of modalities.
4. **Synthetic datasets**: Datasets created by synthetically creating Cad models through the random, or rule-based execution of Cad commands.
5. **Industry-Level Datasets**: Datasets that capture the parametric, feature-based paradigm of industrial Cad systems.

Beyond this categorization, the datasets are ordered by how frequently they were used in the surveyed literature, and by dataset size in cases of equal usage. Table 8 provides an overview of the datasets, summarizing the contained Cad data, annotations, the type of Cad models, the number of samples in the dataset, and the reference to their source

Table 8
Datasets used for training or introduced in the surveyed literature. Sorted by how often the respective datasets were used in the surveyed literature. Sorted by dataset size in case the datasets were used equally often.

Name	CAD data				Annotations						Type	#Models	Source geometry
	BRep	Sequence	CAD code	Mesh	Text	Image(s)	Point cloud	Voxel	Class	Other			
Original datasets													
ABC [11]	✓	–	–	–	–	–	–	–	–		Mechanical	1M+	–
ShapeNet [108]	–	–	–	✓	–	–	–	✓	✓		Common	3M+	–
Fusion 360 Gallery [40]	✓	✓	–	–	–	–	–	–	–		Mechanical	9k	–
Furniture [61]	✓	–	–	–	–	–	–	–	✓		Furniture	6k	–
CADParser [153]	✓	✓	–	–	–	–	–	–	–		Mechanical	40k	–
CADPrompt [68]	–	–	✓	–	✓	–	–	–	–		Mechanical	200	–
Extended datasets													
DeepCAD [10]	✓	✓	–	–	–	–	–	–	–		Mechanical	178k	[11]
Mamba-CAD [58]	–	✓	–	–	–	–	–	–	–		Mechanical	77k	[10,11]
Annotated datasets													
ABC-mono [97]	–	✓	–	–	–	–	✓	–	–	3D sketches	Mechanical	7M+	[10]
mmABC [63]	✓	–	–	–	✓	✓	✓	–	–		Mechanical	1.3M	[11]
GenCAD [102]	–	✓	–	–	–	✓	–	–	–	3D sketches	Mechanical	841k	[10]
OpenECAD [123]	–	–	✓	–	–	✓	–	–	–		Mechanical	200k	[10]
Text2CAD [49]	–	✓	–	–	✓	–	–	–	–		Mechanical	178k	[10]
GenCAD-3D [104]	–	✓	–	✓	–	–	✓	–	–	Surface normals	Mechanical	169k	[10]
GenCAD-Code [125]	–	–	✓	–	–	✓	–	–	–		Mechanical	164k	[10]
CAD-VGDrawing [96]	–	✓	–	–	–	–	–	–	–	Engineering drawings	Mechanical	161k	[10]
GenCAD-SR [103]	–	✓	–	–	–	–	–	–	–	Validity labels	Mechanical	133k	[10]
View2CAD [19]	–	✓	–	–	–	✓	–	–	–		Mechanical	130k	[10]
Sketch2Seq [105]	–	✓	–	–	–	–	–	–	–	Hand-drawn sketches	Mechanical	112k	[13]
CADDreamer [120]	✓	–	–	–	–	✓	–	–	–		Mechanical	30k	[10,11]
CAD-GPT [57]	–	✓	–	–	✓	✓	–	–	–		Mechanical	18k	[10]
KOCAD [97]	–	✓	–	–	–	✓	–	–	–		Mechanical	300	[10]
RealCAD [101]	–	✓	–	–	–	✓	–	–	–		Mechanical	150	[10]
Synthetic datasets													
CAD-Recode [20]	–	–	✓	–	–	–	✓	–	–		Random	1M	–
BRep2Seq [154]	✓	✓	–	–	–	–	–	–	–		Random	1M	–
SynthBal-1M [104]	–	✓	–	–	–	✓	✓	–	–	Surface normals	Mechanical	1M	[10]
CSGNet [112]	–	✓	–	–	–	–	–	✓	–		Random	520k	–
PVar [59]	✓	–	–	–	–	–	–	–	✓		Random	120k	–
Free2CAD [94]	–	✓	–	–	–	✓	–	–	–	Hand-drawn sketches	Random	82k	–
Tree [117]	–	–	–	–	–	–	–	✓	–		Random	5k	–
LLM4CAD [67]	–	–	✓	–	✓	✓	–	–	–	3D sketches	Mechanical	3.3k	–
DAFU-CAD [106]	–	✓	–	–	–	–	–	–	–	Hand-drawn sketches	Random	N/A	–
Industry-level datasets													
WHUCAD [157]	✓	✓	–	–	–	–	–	–	–		Mechanical	144k	[10,11]

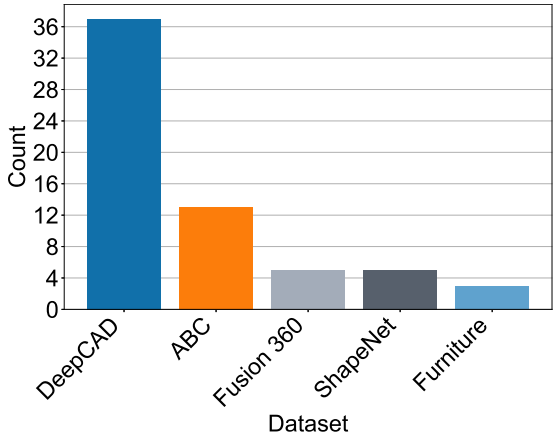


Fig. 12. Datasets used for training in the surveyed literature, sorted by the number of times they are employed. *Fusion 360* refers to the *Fusion 360 Gallery* dataset [40].

geometry. Fig. 12 shows how often the respective datasets were used for training in the surveyed literature, considering only datasets used more than once.

6.1. Original datasets

Original datasets are created by gathering new CAD data from different sources. The **ABC** dataset [11] is one of the earliest and largest CAD datasets. The dataset contains more than one million BREPS of human-created CAD models, mostly mechanical parts, retrieved from the *Onshape online model collection* [164].

The **ShapeNet** dataset [108] contains over three million meshed shapes of everyday objects. Even though the dataset only contains mesh data, it is a valuable resource for research in CAD model generation, as the shapes can be used by unsupervised methods that aim to reconstruct the geometry of an input shape.

The **Fusion 360 Gallery** dataset [40] contains BREPS and corresponding sequences of sketch and extrude commands of mechanical parts, obtained from human designs submitted to the *Autodesk Online Gallery* [165]. Due to the restricted set of modeling operations, the resulting CAD models have limited complexity.

The **Furniture** dataset [61] contains around 6k BREPS of 10 common furniture categories as well as corresponding class labels.

The **CADParser** dataset [153] contains a total of 40k CAD models and corresponding command sequences of manufactured parts designed in *SolidWorks* and collected via the *SolidWorks* APIs. To overcome the limitations of the *Fusion 360 Gallery* dataset, the *CADParser* dataset introduces additional command types, including revolution, fillet, and chamfer.

The **CADPrompt** dataset [68] comprises 200 examples of expert-generated CadQuery code [27], intended for the evaluation of CAD code generation methods.

6.2. Extended datasets

Extended datasets are created by extending, transforming, or reconstructing CAD data of existing datasets. The **DeepCAD** dataset [10] was created by leveraging *Onshape*'s developer API [166] to derive command sequences from CAD models in the *ABC* dataset. These command sequences are limited to line, arc, circle, and extrusion operations. In total, the dataset contains around 178k command sequences. To date, the *DeepCAD* dataset is the largest dataset of command sequences, making it the most widely used benchmark for learning-based methods for CAD generation.

The **Mamba-CAD** dataset [58] contains command sequences that are longer than those in the *DeepCAD* dataset. To this end, the authors select models from the *DeepCAD* dataset and discard sequences shorter than 10 commands. In addition, they translate models from the *ABC* dataset into command sequences with lengths of up to 128 commands. Similar to the *DeepCAD* dataset, the *Mamba-CAD* dataset is limited to primitive sketch and extrusion operations.

6.3. Annotated datasets

Annotated datasets are created by annotating CAD models of existing datasets with different kinds of modalities. The *ABC-mono* dataset [97] contains images and 3D sketches rendered from multiple viewpoints of CAD models drawn from the *DeepCAD* dataset, as well as additional self-collected data. The *mmABC* dataset [63] adds multimodal annotations to the *ABC* dataset, i.e., text, image, and point cloud annotations, and performs deduplication as well as data augmentation, resulting in over 1.3M annotated BREPS. The *GenCAD* dataset [102] annotates each CAD model from the *DeepCAD* dataset with five grayscale images rendered from different viewpoints and further converts each image into corresponding 3D sketches. The datasets introduced in *OpenECAD* [123] contain between 100k, 150k, and 200k pairs of synthetically generated images and the corresponding *OpenECAD* code [123], derived from the *DeepCAD* dataset. The *Text2CAD* dataset [49] adds natural language descriptions to the CAD models of the *DeepCAD* dataset. The *GenCAD-3D* dataset [104] adds point cloud, mesh, and surface normal information to the *DeepCAD* dataset. The *GenCAD-Code* dataset [125] is constructed by converting each command sequence in the *GenCAD* dataset [102] into a CadQuery script [27], resulting in a dataset of paired CAD code and multi-view images. The *CAD-VGDrawing* dataset [96] pairs command sequences with corresponding engineering drawings. To this end, models are selected from the *DeepCAD* dataset and automatically converted into engineering drawings, which are exported in Scalable Vector Graphics (SVG) format. For each CAD model, three orthographic views and one isometric view are generated. The *GenCAD-SR* dataset [103] builds upon the *GenCAD* approach and dataset [102] by pairing command sequences and images with latent vectors. Image latent vectors correspond to command sequences that can either be validly executed to produce a CAD model or fail to do so. Additionally, the ground truth latent vectors associated with the corresponding command sequences are stored. The *View2CAD* dataset [19] removes duplicates from the *DeepCAD* dataset and adds synthetically generated multi-view image annotations using the rendered views of the command sequences. The *Sketch2Seq* dataset [105] collects models from the *Fusion 360 segmentation* dataset [13] and converts them into 3D sketches, segmented according to the CAD command responsible for generating each line. The *CADDreamer* dataset [120] contains models from the *DeepCAD* and *ABC* datasets and enhances them with textures and backgrounds. Each model is subsequently rendered into multi-view images. The *CAD-GPT* dataset [57] is based on models from the *DeepCAD* dataset and includes ten distinct natural language instructions (e.g., "*Please create a CAD model based on the provided image*". [57]) that guide a LLM to generate a CAD model from a given image. To facilitate text-to-CAD finetuning, the authors additionally leverage InstructGPT [167] to generate approximately 18k natural language descriptions of CAD models. The *KOCAD* dataset [97] consists of approximately 300 real-world images captured under varying lighting conditions and backgrounds. To construct this dataset, around 100 CAD models were selected from the *ABC-mono* dataset [97] and subsequently 3D printed. For the *RealCAD* dataset [101] 150 CAD models from the *DeepCAD* dataset were selected and 3D-printed. The 3D-printed parts were used to collect real-world images, which can be used to evaluate the transferability of methods trained on synthetically created images to real-world images.

6.4. Synthetic datasets

Synthetics datasets are created by synthetically creating CAD models through the random, or rule-based execution of CAD commands. The *CAD-Recode* dataset [20] guides a random generation process using heuristics to create a dataset of 1M CAD models represented as CadQuery [27] code. The *BRep2Seq* dataset [154] builds on previous research on CAD model feature classification [168,169] and was created by randomly generating CAD models, considering general spatial geometric constraints such as concentricity or tangency. Further, machining features, such as through holes, pockets, or chamfers, are added to the CAD models. The *SynthBal-1M* dataset [104] addresses the imbalance in command sequence lengths present in the *DeepCAD* dataset. It is generated through data augmentation of existing command sequences, producing synthetic sequences and resulting in a dataset that combines real and synthetic samples with a more uniform sequence length distribution. This balanced distribution helps mitigate model bias toward simpler CAD models. The datasets introduced in *CSG-Net* [112] are generated by randomly sampling from the grammar introduced in the paper to obtain between 130k and 520k pairs of CSG programs and corresponding voxel shape representations. The *PVar* dataset [59] was created by generating extrusion variations of 60 sketches selected from the *Sketch-Graphs* dataset [170], as well as corresponding class labels. The variations obtained from one specific sketch belong to the same class. The *Free2CAD* dataset [94] was generated by randomly sampling sequences of sketch and extrude commands. For these command sequences, images of sketches segmented by the operations they were created from, are obtained. To facilitate the training and evaluation of sweep-based CAD reconstruction approaches, the *Tree* dataset [117] is constructed using procedural modeling to generate a dataset of voxel representations of tree-like shapes. The *LLM4CAD* dataset [92] comprises mechanical components from five categories: flanges, nuts, shafts, gears, and springs. The models are automatically generated using CadQuery [27]. For each component, a rendered image and a corresponding 3D sketch are provided. In addition, the authors leverage the online crowdsourcing platform *Amazon Mechanical Turk* [171] to collect natural language descriptions, resulting in approximately 3k high-quality textual annotations. *DAFU-CAD* [106] introduces a synthetic dataset of hand-drawn sketches exhibiting diverse drawing styles, generated using a range of basic CAD operations such as extrusion and sweeping.

Table 9
Architecture comparison between the *DeepCAD* [10] and *WHUCAD* [157] datasets.

Operation	# Params	DeepCAD	WHUCAD
Primitive operations			
Line	2	✓	✓
Arc	4	✓	✓
Circle	3	✓	✓
Spline	–	–	✓
SCP[a]	2	–	✓
Extrude	12	✓	✓
Selection operations to support advanced operations			
Selection	4	–	✓
Advanced operations			
Revolve	10	–	✓
Pocket	11	–	✓
Groove	9	–	✓
Topo	–	–	✓
Shell	2	–	✓
Chamfer	2	–	✓
Fillet	1	–	✓
Hole	6	–	✓
Mirror	–	–	✓
Mirror topo	–	–	✓
Draft	1	–	✓

[a] Spline Control Point.

6.5. Industry-level datasets

Industry-level datasets capture the native parametric, feature-based modeling paradigm of modern industrial CAD systems, such as CATIA, Siemens NX, and SolidWorks. These models comprise advanced primitives, parametric feature operations, and a persistent naming and selection mechanism. To the best of our knowledge, the **WHUCAD** dataset [157] is currently the only publicly available dataset in this category.

WHUCAD captures the native parametric, feature-based components characteristic of real-world CAD modeling, namely advanced primitives, parametric feature operations, and a persistent naming and selection mechanism. These components are not represented in the command sequences of the *DeepCAD* dataset [10] or the BREP of the *ABC* dataset [11]. As described in the original paper [157], *WHUCAD* draws its source geometry from the *DeepCAD* and *ABC* datasets. Its distinguishing contribution lies in reconstructing this geometry into a native parametric, feature-based representation that mirrors the modeling conventions of industrial CAD systems.

The architecture comparison between the *DeepCAD* and *WHUCAD* datasets, covering primitive, selection, and advanced operations, is shown in Table 9.

At the primitive level, *WHUCAD* additionally supports spline primitives, which are not represented in *DeepCAD* or its derivatives. Splines are a fundamental primitive in modern geometric modeling for parametric CAD, as they enable the representation of the complex, free-form geometries frequently required in industrial mechanical engineering applications.

Taken together, these properties make *WHUCAD*, to the best of our knowledge, the only publicly available dataset that combines advanced primitives, parametric feature-based operations, and a persistent-naming and selection mechanism, more closely reflecting the modeling capabilities of industrial CAD systems than prior datasets.

7. Research gaps and future directions

Despite the recent progress in leveraging learning-based methods to automate the generation, reconstruction, or reverse engineering process of CAD models, significant challenges remain. In this section, we outline key research gaps and challenges and highlight promising directions for future work toward addressing these gaps.

7.1. CAD generation control

Current methods for CAD generation offer limited forms of user control, mainly focusing on the generation from natural language descriptions. Other studies provide mechanisms to specify the semantic class of the desired design, the intended commands and parameters of the command sequence, or high-level descriptions of its topology and geometry. However, these mechanisms remain fairly coarse and do not enable users to influence fine-grained details of the generated designs. Moreover, existing control approaches are largely restricted to the geometric aspects of the model. To date, no methods allow users to guide other crucial aspects of the generation process. In mechanical engineering, however, a part is not defined solely by its geometry or aesthetics but is also designed to fulfill specific functions and meet technical requirements. These requirements can be quite diverse and may include structurally connecting neighboring parts, enabling specific movements, withstanding predefined loads, meeting weight constraints, or fitting within a predefined design space. A promising direction for future research is the generation of mechanical parts tailored to fulfill a defined functional purpose. Addressing this challenge first requires reliable methods for evaluating part functionality and performance characteristics beyond pure geometry. Certain aspects of functionality, such as the ability to withstand mechanical or thermal loads, could be assessed through simulation, enabling the creation of annotated datasets. Other forms of functionality, such as performing specific motions, cannot be easily captured analytically and may instead be evaluated within physics-based simulation environments. Because such evaluations are difficult to integrate into the end-to-end training of neural networks, alternative approaches, such as RL, could be explored, where physics simulations provide a reward signal during training.

7.2. Unsupervised CAD reconstruction quality

Many methods address the challenge of reconstructing editable CAD models from geometric representations such as point clouds, images, or voxels. Approaches that reconstruct command sequences typically rely on ground truth information. However, this information is difficult and costly to obtain. To overcome this limitation, several methods perform unsupervised training, generating CSG trees and using implicit geometry representations. These methods calculate a geometric reconstruction loss between occupancy values of the reconstructed shape and the input shape. While unsupervised methods reduce the need for annotated datasets, focusing solely on geometric reconstruction can limit the quality of the reconstructed models. This is particularly important for mechanical parts, where certain fine-grained features such as holes, chamfers, or slots play a critical role in ensuring proper functionality. Pure geometric losses often underweight these small, but functionally essential, features, resulting in reconstructions that may fail to meet design requirements. Future work should address this limitation by developing methods that explicitly account for functionally important, geometrically small features. Possible directions include introducing additional metrics or loss functions that go beyond pure geometry to prioritize the accurate reconstruction of these features. Another promising approach is multi-scale reconstruction loss, which can weight fine-grained parts of the model as heavily as larger features, ensuring that all elements critical to part functionality are faithfully captured.

7.3. Command sequence ambiguity

Formulating CAD model generation as a sequence modeling problem is a widely used approach applied to CAD generation, CAD reconstruction, and CAD reverse engineering. However, a key challenge arises from the fact that many different parameterized command sequences can produce the same CAD model geometry. In the continuous parameter

space, there is an infinite number of command sequences that yield identical geometry, and even in a purely discrete space, numerous distinct sequences can lead to the same result. Moreover, even when the operations and their parameters are fixed, their execution order can often be rearranged without changing the resulting CAD model geometry. Although some approaches have attempted to address this issue by employing multimodal representation learning to jointly embed command sequences and their resulting geometries, there is no guarantee that different command sequences producing the same CAD model will map to the same embedding. This inconsistency increases the difficulty of model training. A promising direction for future research is the development of a representation of command sequences that is invariant to the specific sequence used, such that all sequences producing the same CAD model share a common representation. As a first step, future work could focus on designing permutation-invariant representations of command sequences. Rather than training models directly on raw command sequences, they could instead be trained on this unified representation. A model could, for instance, be trained to generate this canonical representation, from which all valid command sequences can be derived. Such an approach could deepen the understanding of CAD modeling operations and improve performance across the three aforementioned tasks.

7.4. Dataset availability and complexity

Learning-based methods for 3D CAD model generation fundamentally rely on the availability of large-scale, high-quality datasets. Depending on the specific approach and desired output, both BREP and command sequence data are often required. The most widely used dataset in this domain, the *DeepCAD* dataset [10], contains paired BREP and command sequence representations. It is derived from the *ABC* dataset [11] and includes a substantial number of duplicate models. Many datasets released after *DeepCAD* are themselves derived from it, frequently focusing on duplicate removal or the addition of further annotations such as images, point clouds, or text descriptions. The *Fusion 360 Gallery* dataset [40] also provides BREP and command sequence data, but is considerably smaller in scale compared to the *DeepCAD* dataset. Despite these advances, most CAD models in existing datasets remain relatively simple and generic, lacking the structural and functional complexity characteristic of real-world mechanical parts. Moreover, datasets that include command sequences typically support only a limited subset of modeling operations, primarily sketching and extrusion. The *WHUCAD* dataset constitutes a substantial advance toward a more human-like, parametric, and feature-based design process, incorporating more advanced feature operations and interactions with geometric entities. To facilitate the generation of more realistic and application-oriented CAD models, future efforts should focus on collecting datasets containing more complex designs with richer mechanical features, similar to those in the *WHUCAD* dataset.

8. Applications in mechanical engineering

In this section, we discuss how the methods presented in the surveyed literature could be incorporated into real-world engineering applications and how they could enhance mechanical engineering workflows. Therefore, we present one application for each of the three previously introduced tasks, i.e., CAD generation, CAD reconstruction, and CAD reverse engineering.

8.1. CAD generation for design space exploration in prototyping

The traditional design process is inherently iterative and often constrained by the experience and creativity of the design engineer, as well as the time and cost required to model and evaluate each idea. By shifting the engineer's role from low-level design to high-level specification of design requirements, learning-based methods can automatically generate multiple design variations simultaneously. These approaches populate the design space with a diverse set of candidates, enabling engineers to leverage their expertise to assess trade-offs and select the most promising options for further refinement. During prototyping, learning-based methods have the potential to uncover solutions that may not arise through conventional design workflows, accelerating ideation and expanding creative possibilities [172].

The current state-of-the-art enables designers to generate CAD models from natural language descriptions, autocomplete or edit partial CAD models, generate models belonging to a specified semantic class, or to exert control over geometry and topology during generation. Among these, natural language-based generation is particularly promising, as it allows designers to directly specify models that match their requirements. However, current methods largely limit control to geometric aspects of the part, without addressing broader considerations such as functionality, weight constraints, or load requirements.

8.2. CAD reconstruction for quality control

Mechanical components are manufactured under tight tolerances and require precise dimensional and form accuracy to ensure long service life. However, all production processes inherently exhibit some degree of deviation. For instance, additive manufacturing builds parts layer by layer through filament extrusion, whereas injection molding melts raw pellets and injects the material into a high-pressure mold. Both methods enable the creation of complex, composite, and hybrid structures with levels of precision and control unattainable in traditional manufacturing [173]. Nevertheless, deviations from the intended design can arise due to thermal distortions, tool wear, or suboptimal process parameters. Controlling these deviations is therefore essential, as detailed deviation data are required to fine-tune process settings, reduce scrap rates, and improve overall product quality [174]. Traditionally, deviations are measured manually using calipers or coordinate measuring machines, a time-consuming and labor-intensive process. Reconstructing exact CAD models from scans or images of manufactured parts enables automated comparison with the nominal design. Such comparisons can automatically detect geometric deviations, thereby providing comprehensive diagnostic information. Beyond deviation detection, the reconstruction of an explicit CAD model from measurement data enables analyses not feasible with raw point cloud representations. These include high-fidelity simulations of structural, thermal, or fluid-dynamic behavior to evaluate whether observed deviations affect the functional performance or integrity of the component. Moreover, the reconstructed CAD model can be integrated into digital twin frameworks, supporting continuous monitoring, predictive maintenance, and performance assessment throughout the product lifecycle. Automating this reconstruction process would not only enhance the efficiency and robustness of quality assurance but also enable closed-loop manufacturing paradigms. In such systems, deviations identified during inspection can inform adaptive process control, allowing dynamic adjustment of tool paths, feed rates, or environmental parameters to compensate for observed errors. This feedback capability reduces material waste, minimizes rework, and shortens production cycles, ultimately contributing to the realization of self-optimizing, data-driven manufacturing ecosystems.

Several of the surveyed methods can be applied for quality control as they enable CAD generation from point cloud scans, images, or voxel representations. While voxel-based approaches discretize geometry and therefore do not capture it exactly, they require modifications to handle point cloud or image inputs. Overall, reconstruction methods demonstrate promising results with high fidelity on benchmark datasets. However, most have been trained and evaluated on synthetically generated point clouds or images. To be applied reliably in real-world scenarios, especially for manufactured parts, these methods must be further developed to improve robustness and capture

fine-grained geometric features, which are critical for mechanical engineering applications and quality control, specifically. Well-established methods could then not only tackle the analysis of individual parts but also revolutionize part matching for tolerance compensation in assemblies.

8.3. CAD reverse engineering for data enhancement and collaborative design

A persistent challenge in collaborative engineering is the transmission of geometry information. Not to mention the still very prominent role of 2D technical drawings, just exchanging CAD data between different software systems or even between different versions of the same software system is error-prone. The industry standard for this exchange is the use of neutral file formats like STEP[6] [175] or IGES[7] [176]. However, these formats are fundamentally limited as they only store the final BREP of the model and discard the rich parametric history of how that model was created. When a user imports a STEP file, they receive a static object with no editable features, parameters, or design history [177]. Even within a company, design history can become lost, for example, through successive model revisions, data conversions, or the exchange of geometry between teams. This limitation prevents modifications after data exchange, as altering a CAD model without its parametric command sequence oftentimes requires difficult and unreliable direct geometric manipulation.

Several of the surveyed approaches address this challenge by enabling the reverse engineering of parametric command sequences from BREP. These methods demonstrate promising results on existing benchmarks and can already be applied to relatively simple cases of command sequence reconstruction. However, most methods have been tested only on datasets with a limited set of CAD operations. Real-world mechanical engineering applications will require approaches that support a wider range of commands and can handle more complex parts.

9. Summary

This survey examined recent research on learning-based methods for 3D CAD model generation, analyzing a total of **78 publications**. We first present methods addressing representation learning on CAD data, which form the foundation for subsequent approaches focused on CAD model generation. The surveyed literature is subsequently organized into three distinct tasks identified in our analysis:

1. **CAD Generation** which involves unconditional generation of 3D CAD models as well as the generation of 3D CAD models from inputs such as text, semantic class labels, or other forms of user guidance. The user may also provide CAD models for autocompletion, editing, or interpolation.
2. **CAD Reconstruction** which involves creating 3D CAD models from sensor-based inputs such as point clouds, voxels, or images that represent physical objects.
3. **CAD Reverse Engineering** which involves deriving command sequences from existing 3D CAD models given as BREPS.

Depending on the specific method employed, the different approaches produce various types of outputs, including command sequences, CSG trees, BREPS, and executable CAD code. The generation of these outputs can be broadly attributed to four methodological directions identified in this survey: transformer-based approaches, diffusion-based approaches, foundation models, and neural–analytical methods. Both transformer-based and diffusion-based approaches have been applied to CAD generation and CAD reconstruction tasks, demonstrating their ability to generate command sequences as well as BREPS directly. Transformer models have additionally been employed for the reverse engineering of command sequences, whereas diffusion-based approaches have not yet been explored for this task. Foundation models represent the latest development, leveraging knowledge obtained through pre-training on large-scale datasets. Within this context, research has primarily focused on generating executable CAD code and utilizing language modeling capabilities to produce command sequences. Neural–analytical methods, which integrate neural network predictions with analytical components, have been particularly effective for CSG tree reconstruction, as they enable differentiable Boolean operations that facilitate the computation of unsupervised reconstruction losses. Additionally, analytical techniques have been incorporated into BREP reconstruction, where geometric primitives are analytically fitted to segmented point clouds. Individual approaches have also explored the application of state space models [84], RNNs [112], neurally-guided search [40,158], and RL [159].

Subsequently, we discuss methodological and data-related research gaps which give rise to multiple possible future research directions: (i) providing increased user control over the generation process of 3D CAD models and enabling generation under technical constraints and requirements, (ii) improving the quality of CAD reconstruction from point clouds, images, or voxels, with a focus on accurately recovering fine-grained mechanical features, (iii) resolving the inherent ambiguity of command sequences, and (iv) increasing both the availability and complexity of large-scale, high-quality datasets. Addressing these gaps could significantly advance the application of learning-based methods for 3D CAD model generation in practical mechanical engineering contexts.

The survey concludes by connecting current research directions with practical engineering tasks. It outlines potential applications of the surveyed methods in industrial settings, including design space exploration in prototyping, quality control, and data enhancement.

Even though a broad range of literature on learning-based 3D CAD model generation has been reviewed, this study is subject to several limitations. The literature search was restricted to Scopus and major machine learning and computer vision conferences. Although we aimed to cover all relevant research, some studies may have been missed due to the rapid evolution of this field, and preprint papers were not included. Furthermore, real-world applications were discussed only on a theoretical level and should be further examined and validated through practical test studies to assess their feasibility. Finally, non-academic approaches, including commercial tools that already enable automated 3D CAD model generation, potentially using learning-based methods, were not considered.

CRediT authorship contribution statement

Fabian Baumeister: Writing – review & editing, Writing – original draft, Conceptualization. **Jakob Bönsch:** Writing – review & editing, Conceptualization. **Constantin Chaumet:** Writing – review & editing, Conceptualization. **Laura Dörr:** Writing – review & editing, Conceptualization. **Anne Meyer:** Writing – review & editing, Supervision, Conceptualization.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work, the authors used OpenAI ChatGPT and Google Gemini to refine the writing and improve the clarity and wording of the text. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the content of this publication.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

[6] *Standard for the exchange of product model data.*
[7] *Initial graphics exchange specification.*

Acknowledgment

This work is associated with the research project IntWertL, 19S22003Q. Funded by the Federal Republic of Germany and the European Union. Funding bodies: Federal Ministry for Economic Affairs and Energy based on a resolution of the German Bundestag, and the European Union.

Data availability

No data was used for the research described in the article.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems, Vol. 30, 2017, pp. 5998–6008, http://dx.doi.org/10.48550/arXiv.1706.03762.

[2] J. Ho, A.N. Jain, P. Abbeel, Denoising diffusion probabilistic models, in: Advances in Neural Information Processing Systems, Vol. 33, 2020, pp. 6840–6851, http://dx.doi.org/10.48550/arXiv.2006.11239.

[3] N. Rudin, D. Hoeller, P. Reist, M. Hutter, Learning to walk in minutes using massively parallel deep reinforcement learning, in: Proceedings of the 5th Conference on Robot Learning, Vol. 164, 2022, pp. 91–100, http://dx.doi.org/10.48550/arXiv.2109.11978.

[4] M.J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E.P. Foster, P.R. Sanketi, Q. Vuong, T. Kollar, B. Burchfiel, R. Tedrake, D. Sadigh, S. Levine, P. Liang, C. Finn, OpenVLA: An open-source vision-language-action model, in: Proceedings of the 8th Conference on Robot Learning, Vol. 270, 2025, pp. 2679–2713, http://dx.doi.org/10.48550/arXiv.2406.09246.

[5] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, S. Zagoruyko, End-to-end object detection with transformers, in: European Conference on Computer Vision, 2020, pp. 213–229, http://dx.doi.org/10.1007/978-3-030-58452-8_13.

[6] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A.C. Berg, W.-Y. Lo, P. Dollar, R. Girshick, Segment anything, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2023, pp. 4015–4026, http://dx.doi.org/10.1109/ICCV51070.2023.00371.

[7] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, G. Lample, LLaMA: Open and efficient foundation language models, 2023, http://dx.doi.org/10.48550/arXiv.2302.13971, arXiv:2302.13971.

[8] Gemma Team, T. Mesnard, C. Hardin, R. Dadashi, S. Bhupatiraju, S. Pathak, L. Sifre, M. Rivière, M.S. Kale, J. Love, et al., Gemma: Open models based on Gemini research and technology, 2024, http://dx.doi.org/10.48550/arXiv.2403.08295, arXiv:2403.08295.

[9] Y. Siddiqui, A. Alliegro, A. Artemov, T. Tommasi, D. Sirigatti, V. Rosov, A. Dai, M. Nießner, MeshGPT: Generating triangle meshes with decoder-only transformers, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024, pp. 19615–19625, http://dx.doi.org/10.1109/CVPR52733.2024.01855.

[10] R. Wu, C. Xiao, C. Zheng, DeepCAD: A deep generative network for computer-aided design models, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 6772–6782, http://dx.doi.org/10.1109/ICCV48922.2021.00670.

[11] S. Koch, A. Matveev, Z. Jiang, F. Williams, A. Artemov, E. Burnaev, M. Alexa, D. Zorin, D. Panozzo, ABC: A big CAD model dataset for geometric deep learning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 9601–9611, http://dx.doi.org/10.1109/CVPR.2019.00983.

[12] P.K. Jayaraman, A. Sanghi, J.G. Lambourne, K.D.D. Willis, T. Davies, H. Shayani, N. Morris, UV-Net: Learning from boundary representations, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 11703–11712, http://dx.doi.org/10.1109/CVPR46437.2021.01153.

[13] J.G. Lambourne, K.D.D. Willis, P.K. Jayaraman, A. Sanghi, P. Meltzer, H. Shayani, BRepNet: A topological message passing system for solid models, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 12773–12782, http://dx.doi.org/10.1109/CVPR46437.2021.01258.

[14] M. Jung, M. Kim, J. Kim, ContrastCAD: Contrastive learning-based representation learning for computer-aided design models, 2024, http://dx.doi.org/10.48550/arXiv.2404.01645, arXiv:2404.01645.

[15] X. Xu, K.D.D. Willis, J.G. Lambourne, C.-Y. Cheng, P.K. Jayaraman, Y. Furukawa, SkexGen: Autoregressive generation of CAD construction sequences with disentangled codebooks, in: Proceedings of the 39th International Conference on Machine Learning, Vol. 162, 2022, pp. 24698–24724.

[16] M.S. Khan, S. Sinha, T.U. Sheikh, D. Stricker, S.A. Ali, M.Z. Afzal, Text2CAD: Generating sequential CAD designs from beginner-to-expert level text prompts, in: Advances in Neural Information Processing Systems, Vol. 37, 2024, pp. 7552–7579, http://dx.doi.org/10.52202/079017-0242.

[17] E. Dupont, K. Cherenkova, D. Mallis, G. Gusev, A. Kacem, D. Aouada, TransCAD: A hierarchical transformer for CAD sequence inference from point clouds, in: Computer Vision – ECCV 2024, Vol. 15119, 2024, pp. 19–36, http://dx.doi.org/10.1007/978-3-031-73030-6_2.

[18] D. Ren, J. Zheng, J. Cai, J. Li, J. Zhang, ExtrudeNet: Unsupervised inverse sketch-and-extrude for shape parsing, in: Computer Vision – ECCV 2022, Vol. 13662, 2022, pp. 482–498, http://dx.doi.org/10.1007/978-3-031-20086-1_28.

[19] Y. Zhang, F. He, R. Fan, B. Fan, View2CAD: Parsing multi-view into CAD command sequences, in: International Conference on Computer Supported Cooperative Work in Design, 2024, pp. 2949–2954, http://dx.doi.org/10.1109/CSCWD61410.2024.10580755.

[20] D. Rukhovich, E. Dupont, D. Mallis, K. Cherenkova, A. Kacem, D. Aouada, CAD-Recode: Reverse engineering CAD code from point clouds, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2025, pp. 9801–9811.

[21] M. Mäntylä, An Introduction to Solid Modeling, Computer Science Press, 1988.

[22] J.J. Shah, M. Mäntylä, Parametric and Feature-based CAD/CAM: Concepts, Techniques, and Applications, John Wiley & Sons, 1995.

[23] N. Heidari, A. Iosifidis, Geometric deep learning for computer-aided design: A survey, IEEE Access 13 (2025) 119305–119334, http://dx.doi.org/10.1109/ACCESS.2025.3587121.

[24] R. Lin, Y. Ji, W. Ding, T. Wu, Y. Zhu, M. Jiang, A survey on deep learning in 3D CAD reconstruction, Appl. Sci. 15 (12) (2025) 6681, http://dx.doi.org/10.3390/app15126681.

[25] S.J. Schoonmaker, The CAD Guidebook: A Basic Manual for Understanding and Improving Computer-Aided Design, CRC Press, 2002.

[26] A.G. Requicha, Representations for rigid solids: Theory, methods, and systems, ACM Comput. Surv. 12 (4) (1980) 437–464, http://dx.doi.org/10.1145/356827.356833.

[27] CadQuery contributors, CadQuery, 2026, http://dx.doi.org/10.5281/zenodo.18633916.

[28] J. Sohl-Dickstein, E.A. Weiss, N. Maheswaranathan, S. Ganguli, Deep unsupervised learning using nonequilibrium thermodynamics, in: Proceedings of the 32nd International Conference on Machine Learning, Vol. 37, 2015, pp. 2256–2265.

[29] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, B. Ommer, High-resolution image synthesis with latent diffusion models, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 10674–10685, http://dx.doi.org/10.1109/CVPR52688.2022.01042.

[30] L. Ruan, Y. Ma, H. Yang, H. He, B. Liu, J. Fu, N.J. Yuan, Q. Jin, B. Guo, MM-Diffusion: Learning multi-modal diffusion models for joint audio and video generation, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023, pp. 10219–10228, http://dx.doi.org/10.1109/CVPR52729.2023.00985.

[31] C. Vignac, I. Krawczuk, A. Siraudin, B. Wang, V. Cevher, P. Frossard, DiGress: Discrete denoising diffusion for graph generation, in: International Conference on Learning Representations, 2023.

[32] A. Radford, J.W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, I. Sutskever, Learning transferable visual models from natural language supervision, in: Proceedings of the 38th International Conference on Machine Learning, Vol. 139, 2021, pp. 8748–8763.

[33] B.T. Jones, M. Hu, M. Kodnongbua, V.G. Kim, A. Schulz, Self-supervised representation learning for CAD, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023, pp. 21327–21336, http://dx.doi.org/10.1109/CVPR52729.2023.02043.

[34] Y. Lou, X. Li, H. Chen, X. Zhou, BRep-BERT: Pre-training boundary representation BERT with sub-graph node contrastive learning, in: Proceedings of the 32nd ACM International Conference on Information and Knowledge Management, 2023, pp. 1657–1666, http://dx.doi.org/10.1145/3583780.3614795.

[35] Q. Zou, L. Zhu, Bringing attention to CAD: Boundary representation learning via transformer, Comput.-Aided Des. 189 (2025) 103940, http://dx.doi.org/10.1016/j.cad.2025.103940.

[36] S. Duan, J. Feng, Y. Qi, Uniform representation of parametric CAD models for generative application, in: Artificial Neural Networks and Machine Learning – ICANN 2025, Vol. 16071, 2025, pp. 239–250, http://dx.doi.org/10.1007/978-3-032-04555-3_20.

[37] W. Ma, M. Xu, X. Li, X. Zhou, 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, 2023, pp. 1766–1776, http://dx.doi.org/10.1145/3583780.3614982.

[38] M. Xu, Y. Lou, W. Ma, X. Li, X. Zhou, Parametric CAD primitive retrieval via multi-modal fusion and deep hashing, in: Proceedings of the 2024 International Conference on Multimedia Retrieval, 2024, pp. 1061–1069, http://dx.doi.org/10.1145/3652583.3658041.

[39] X. Li, H. Chen, Y. Lou, X. Zhou, CF-CAD: A contrastive fusion network for 3D computer-aided design generative modeling, in: Database Systems for Advanced Applications, Vol. 14852, 2024, pp. 435–450, http://dx.doi.org/10.1007/978-981-97-5555-4_31.

[40] K.D.D. Willis, Y. Pu, J. Luo, H. Chu, T. Du, J.G. Lambourne, A. Solar-Lezama, W. Matusik, Fusion 360 gallery: A dataset and environment for programmatic CAD construction from human design sequences, ACM Trans. Graph. 40 (4) (2021) 54:1–54:24, http://dx.doi.org/10.1145/3450626.3459818.

[41] B.T. Jones, D. Hildreth, D. Chen, I. Baran, V.G. Kim, A. Schulz, AutoMate: A dataset and learning approach for automatic mating of CAD assemblies, ACM Trans. Graph. 40 (6) (2021) 227:1–227:18, http://dx.doi.org/10.1145/3478513.3480562.

[42] J.J. Park, P.R. Florence, J. Straub, R.A. Newcombe, S. Lovegrove, DeepSDF: Learning continuous signed distance functions for shape representation, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 165–174, http://dx.doi.org/10.1109/CVPR.2019.00025.

[43] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), 2019, pp. 4171–4186, http://dx.doi.org/10.18653/v1/N19-1423.

[44] G. Farin, Algorithms for rational Bézier curves, Comput.-Aided Des. 15 (2) (1983) 73–77, http://dx.doi.org/10.1016/0010-4485(83)90171-9.

[45] R.N. Goldman, D.J. Filip, Conversion from Bézier rectangles to Bézier triangles, Comput.-Aided Des. 19 (1) (1987) 25–27, http://dx.doi.org/10.1016/0010-4485(87)90149-7.

[46] C.R. Qi, L. Yi, H. Su, L.J. Guibas, PointNet++: Deep hierarchical feature learning on point sets in a metric space, in: Advances in Neural Information Processing Systems, Vol. 30, 2017, pp. 5099–5108, http://dx.doi.org/10.48550/arXiv.1706.02413.

[47] A. van den Oord, O. Vinyals, K. Kavukcuoglu, Neural discrete representation learning, in: Advances in Neural Information Processing Systems, Vol. 30, 2017, pp. 6306–6315, http://dx.doi.org/10.48550/arXiv.1711.00937.

[48] X. Xu, P.K. Jayaraman, J.G. Lambourne, K.D.D. Willis, Y. Furukawa, Hierarchical neural coding for controllable CAD model generation, in: Proceedings of the 40th International Conference on Machine Learning, Vol. 202, 2023, pp. 38443–38461.

[49] X. Li, Y. Song, Y. Lou, X. Zhou, CAD Translator: An effective drive for text to 3D parametric computer-aided design generative modeling, in: Proceedings of the 32nd ACM International Conference on Multimedia, 2024, pp. 8461–8470, http://dx.doi.org/10.1145/3664647.3681549.

[50] X. Guo, X. Dong, J. Cao, Z. Chen, CADTrans: A code tree-guided CAD generative transformer model with regularized discrete codebooks, Graph. Model. 139 (2025) 101262, http://dx.doi.org/10.1016/j.gmod.2025.101262.

[51] H. Wang, M. Zhao, Y. Wang, W. Quan, D.-M. Yan, VQ-CAD: Computer-aided design model generation with vector quantized diffusion, Comput. Aided Geom. Design 111 (2024) 102327, http://dx.doi.org/10.1016/j.cagd.2024.102327.

[52] A. Zhang, W. Jia, Q. Zou, Y. Feng, X. Wei, Y. Zhang, Diffusion-CAD: Controllable diffusion model for generating computer-aided design models, IEEE Trans. Vis. Comput. Graphics 31 (12) (2025) 10188–10199, http://dx.doi.org/10.1109/TVCG.2025.3535797.

[53] P. Li, W. Zhang, J. Guo, J. Chen, D.-M. Yan, Revisiting CAD model generation by learning raster sketch, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 39, 2025, pp. 4869–4877, http://dx.doi.org/10.1609/aaai.v39i5.32515.

[54] Z. Zhang, S. Sun, W. Wang, D. Cai, J. Bian, FlexCAD: Unified and versatile controllable CAD generation with fine-tuned large language models, in: International Conference on Learning Representations, 2025.

[55] R. Wang, Y. Yuan, S. Sun, J. Bian, Text-to-CAD generation through infusing visual feedback in large language models, 2025, http://dx.doi.org/10.48550/arXiv.2501.19054, arXiv:2501.19054.

[56] Y. Yuan, S. Sun, Q. Liu, J. Bian, CAD-Editor: A locate-then-infill framework with automated training data synthesis for text-based CAD editing, in: Proceedings of the 42nd International Conference on Machine Learning, Vol. 267, 2025, pp. 73588–73603.

[57] S. Wang, C. Chen, X. Le, Q. Xu, L. Xu, Y. Zhang, J. Yang, CAD-GPT: Synthesising CAD construction sequence with spatial reasoning-enhanced multimodal LLMs, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 39, 2025, pp. 7880–7888, http://dx.doi.org/10.1609/aaai.v39i8.32849.

[58] X. Li, Y. Lou, Y. Song, X. Zhou, Mamba-CAD: State space model for 3D computer-aided design generative modeling, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 39, (5) 2025, pp. 5013–5021, http://dx.doi.org/10.1609/aaai.v39i5.32531.

[59] P.K. Jayaraman, J.G. Lambourne, N. Desai, K.D.D. Willis, A. Sanghi, N.J.W. Morris, SolidGen: An autoregressive model for direct B-rep synthesis, Trans. Mach. Learn. Res. (2023).

[60] P. Li, W. Zhang, J. Chen, D.-M. Yan, Stitch-A-Shape: Bottom-up learning for B-Rep generation, in: SIGGRAPH Conference Papers '25, 2025, pp. 1–12, http://dx.doi.org/10.1145/3721238.3730661.

[61] X. Xu, J.G. Lambourne, P.K. Jayaraman, Z. Wang, K.D.D. Willis, Y. Furukawa, BrepGen: A B-rep generative diffusion model with structured latent geometry, ACM Trans. Graph. 43 (4) (2024) 1–14, http://dx.doi.org/10.1145/3658129.

[62] P. Li, W. Zhang, W. Quan, B. Zhang, P. Wonka, D.-M. Yan, BrepGPT: Autoregressive B-rep generation with voronoi half-patch, ACM Trans. Graph. 44 (6) (2025) 1–18, http://dx.doi.org/10.1145/3763323.

[63] J. Wu, Y. Wang, X. Yue, X. Ma, J. Guo, D. Zhou, W. Ouyang, S. Tang, CMT: A cascade MAR with topology predictor for multimodal conditional CAD generation, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2025, pp. 7014–7024.

[64] H. Guo, X. Huang, J. Hao, Y. Bai, H. Gan, Y. Shi, BrepGiff: Lightweight generation of complex B-rep with 3D GAT diffusion, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2025, pp. 26587–26596.

[65] M. Lee, D. Zhang, C. Jambon, Y.M. Kim, BrepDiff: Single-stage B-rep diffusion model, in: Proceedings of the SIGGRAPH Asia 2025 Conference Papers, 2025, pp. 1–11, http://dx.doi.org/10.1145/3721238.3730698.

[66] Y. Liu, D. Xu, X. Yu, X. Xu, D. Cohen-Or, H. Zhang, H. Huang, HoLa: B-Rep generation using a holistic latent representation, ACM Trans. Graph. 44 (4) (2025) 1–25, http://dx.doi.org/10.1145/3730842.

[67] X. Li, Y. Sun, Z. Sha, LLM4cad: Multimodal large language models for three-dimensional computer-aided design generation, J. Comput. Inf. Sci. Eng. 25 (2) (2025) 021005, http://dx.doi.org/10.1115/1.4067085.

[68] K. Alrashedy, P. Tambwekar, Z.H. Zaidi, M. Langwasser, W. Xu, M.C. Gombolay, Generating CAD code with vision-language models for 3D designs, in: International Conference on Learning Representations, 2025.

[69] J. Li, W. Ma, X. Li, Y. Lou, G. Zhou, X. Zhou, CAD-Llama: Leveraging large language models for computer-aided design parametric 3D model generation, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2025, pp. 18563–18573.

[70] J. Li, Y. Luo, Y. Lou, X. Zhou, ReCAD: Reinforcement learning enhanced parametric CAD model generation with vision-language models, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 40, 2026, pp. 6190–6198, http://dx.doi.org/10.1609/aaai.v40i8.37544.

[71] J. Yu, Z. Wang, V. Vasudevan, L. Yeung, M. Seyedhosseini, Y. Wu, CoCa: Contrastive captioners are image-text foundation models, Trans. Mach. Learn. Res. (2022).

[72] H. Zhang, M. Cissé, Y.N. Dauphin, D. Lopez-Paz, Mixup: Beyond empirical risk minimization, in: International Conference on Learning Representations, 2018.

[73] A.Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D.S. Chaplot, D.d.l. Casas, E.B. Hanna, F. Bressand, et al., Mixtral of experts, 2024, http://dx.doi.org/10.48550/arXiv.2401.04088, arXiv:2401.04088.

[74] H. Liu, C. Li, Y. Li, B. Li, Y. Zhang, S. Shen, Y.J. Lee, LLaVA-NeXT: Improved reasoning, OCR, and world knowledge, 2024, URL https://llava-vl.github.io/blog/2024-01-30-llava-next/.

[75] I.J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A.C. Courville, Y. Bengio, Generative adversarial nets, in: Advances in Neural Information Processing Systems, Vol. 27, 2014, pp. 2672–2680.

[76] S. Gu, D. Chen, J. Bao, F. Wen, B. Zhang, D. Chen, L. Yuan, B. Guo, Vector quantized diffusion model for text-to-image synthesis, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 10686–10696, http://dx.doi.org/10.1109/CVPR52688.2022.01043.

[77] A. Sanghi, H. Chu, J.G. Lambourne, Y. Wang, C.-Y. Cheng, M. Fumero, K.R. Malekshan, CLIP-Forge: Towards zero-shot text-to-shape generation, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 18603–18613.

[78] A. Sanghi, R. Fu, V. Liu, K.D. Willis, H. Shayani, A.H. Khasahmadi, S. Sridhar, D. Ritchie, CLIP-Sculptor: Zero-shot generation of high-fidelity and diverse shapes from natural language, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023, pp. 18339–18348.

[79] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, et al., The Llama 3 herd of models, 2024, http://dx.doi.org/10.48550/arXiv.2407.21783, arXiv:2407.21783.

[80] R. Rafailov, A. Sharma, E. Mitchell, C.D. Manning, S. Ermon, C. Finn, Direct preference optimization: Your language model is secretly a reward model, in: Advances in Neural Information Processing Systems, Vol. 36, 2023, pp. 53728–53741.

[81] B. Li, Y. Zhang, D. Guo, R. Zhang, F. Li, H. Zhang, K. Zhang, P. Zhang, Y. Li, Z. Liu, C. Li, LLaVA-OneVision: Easy visual task transfer, Trans. Mach. Learn. Res. (2025).

[82] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, et al., GPT-4 technical report, 2023, http://dx.doi.org/10.48550/arXiv.2303.08774, arXiv:2303.08774.

[83] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J.E. Gonzalez, I. Stoica, E.P. Xing, Vicuna: An open-source chatbot impressing GPT-4 with 90%* ChatGPT quality, 2023, URL https://lmsys.org/blog/2023-03-30-vicuna/.

[84] A. Gu, T. Dao, Mamba: Linear-time sequence modeling with selective state spaces, in: First Conference on Language Modeling, 2024.

[85] Z. Chen, H. Zhang, Learning implicit fields for generative shape modeling, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 5939–5948.

[86] O. Vinyals, M. Fortunato, N. Jaitly, Pointer networks, in: Advances in Neural Information Processing Systems, Vol. 28, 2015, pp. 2692–2700.

[87] D.P. Kingma, M. Welling, Auto-encoding variational bayes, 2013, http://dx.doi.org/10.48550/arXiv.1312.6114, arXiv:1312.6114.
[88] J. Han, K. Gong, Y. Zhang, J. Wang, K. Zhang, D. Lin, Y. Qiao, P. Gao, X. Yue, OneLLM: One framework to align all modalities with language, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024, pp. 26584–26595.
[89] T. Li, Y. Tian, H. Li, M. Deng, K. He, Autoregressive image generation without vector quantization, in: Advances in Neural Information Processing Systems, Vol. 37, 2024, pp. 56424–56445.
[90] W. Peebles, S. Xie, Scalable diffusion models with transformers, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2023, pp. 4195–4205.
[91] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E.H. Chi, Q.V. Le, D. Zhou, Chain-of-thought prompting elicits reasoning in large language models, in: Advances in Neural Information Processing Systems, Vol. 35, 2022, pp. 24824–24837.
[92] OpenAI, A. Hurst, A. Lerer, A.P. Goucher, A. Perelman, A. Ramesh, A. Clark, A.J. Ostrow, A. Welihinda, A. Hayes, A. Radford, et al., GPT-4o system card, 2024, http://dx.doi.org/10.48550/arXiv.2410.21276, arXiv:2410.21276.
[93] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y.K. Li, Y. Wu, et al., DeepSeekMath: Pushing the limits of mathematical reasoning in open language models, 2024, http://dx.doi.org/10.48550/arXiv.2402.03300, arXiv:2402.03300.
[94] C. Li, H. Pan, A. Bousseau, N.J. Mitra, Free2CAD: Parsing freehand drawings into CAD commands, ACM Trans. Graph. 41 (4) (2022) 93:1–93:16, http://dx.doi.org/10.1145/3528223.3530133.
[95] M.S. Khan, E. Dupont, S.A. Ali, K. Cherenkova, A. Kacem, D. 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, 2024, pp. 4713–4722, http://dx.doi.org/10.1109/CVPR52733.2024.00451.
[96] F. Qin, S. Lu, J. Hou, C. Wang, M. Fang, L. Liu, Drawing2CAD: Sequence-to-sequence learning for CAD generation from vector drawings, in: Proceedings of the 33rd ACM International Conference on Multimedia, 2025, pp. 10573–10582, http://dx.doi.org/10.1145/3746027.3755782.
[97] T. Chen, C. Yu, Y. Hu, J. Li, T. Xu, R. Cao, L. Zhu, Y. Zang, Y. Zhang, Z. Li, L. Sun, Img2CAD: Conditioned 3-D CAD model generation from single image with structured visual geometry, IEEE Trans. Ind. Inform. 21 (11) (2025) 8539–8549, http://dx.doi.org/10.1109/TII.2025.3584476.
[98] Z. Zong, F. He, R. Fan, Y. Liu, Reconstruct PFCAD models with dual-transformer and continuation attention, in: 2025 28th International Conference on Computer Supported Cooperative Work in Design, 2025, pp. 1056–1061, http://dx.doi.org/10.1109/CSCWD64889.2025.11033229.
[99] J. Lu, Y. Wang, Y. Wu, H. Li, Y. Shi, F. Ning, An autoregressive framework for reconstructing editable parametric computer-aided design models from point clouds, Eng. Appl. Artif. Intell. 163 (2026) 113107, http://dx.doi.org/10.1016/j.engappai.2025.113107.
[100] W. Ma, S. Chen, Y. Lou, X. Li, X. Zhou, Draw step by step: Reconstructing CAD construction sequences from point clouds via multimodal diffusion, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024, pp. 27144–27153, http://dx.doi.org/10.1109/CVPR52733.2024.02564.
[101] C. Chen, J. Wei, T. Chen, C. Zhang, X. Yang, S. Zhang, B. Yang, C.-S. Foo, G. Lin, Q. Huang, F. Liu, CADCrafter: Generating computer-aided design models from unconstrained images, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2025, pp. 11073–11082, http://dx.doi.org/10.1109/CVPR52734.2025.01034.
[102] M.F. Alam, F. Ahmed, GenCAD: Image-conditioned computer-aided design generation with transformer-based contrastive representation and diffusion priors, Trans. Mach. Learn. Res. (2025).
[103] C. Tsuji, E. Flores Medina, H. Gupta, M.F. Alam, GenCAD-Self-Repairing: Feasibility enhancement for 3D CAD generation, in: Proceedings of the ASME 2025 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Vol. 2A, 2025, V02AT02A055, http://dx.doi.org/10.1115/DETC2025-169378.
[104] N. Yu, M.F. Alam, A.J. Hart, F. Ahmed, GenCAD-Three-Dimensional: Computer-aided design program generation using multimodal latent space alignment and synthetic dataset balancing, J. Mech. Des. 148 (3) (2026) 031703, http://dx.doi.org/10.1115/1.4069276.
[105] Y. Sun, J. Li, Z. Xu, J. Zhang, X. Liu, D. Zhang, G. Lu, Sketch2Seq: Reconstruct CAD models from feature-based sketch segmentation, IEEE Trans. Vis. Comput. Graphics 31 (10) (2025) 8214–8230, http://dx.doi.org/10.1109/TVCG.2025.3566544.
[106] Y. Sun, X. Liu, Z. He, J. Zhang, C. Wu, G. Lu, J. Li, DAFU-CAD: Depth-assisted feature unraveling for sketch-based robust CAD modeling, in: Proceedings of the 33rd ACM International Conference on Multimedia, 2025, pp. 8008–8017, http://dx.doi.org/10.1145/3746027.3755252.
[107] K. Kania, M. Zieba, T. Kajdanowicz, UCSG-NET: Unsupervised discovering of constructive solid geometry tree, in: Advances in Neural Information Processing Systems, Vol. 33, 2020, pp. 8776–8786.
[108] A.X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, F. Yu, ShapeNet: An information-rich 3D model repository, 2015, http://dx.doi.org/10.48550/arXiv.1512.03012, arXiv:1512.03012.
[109] D. Ren, J. Zheng, J. Cai, J. Li, H. Jiang, Z. Cai, J. Zhang, L. Pan, M. Zhang, H. Zhao, S. Yi, CSG-Stump: A learning friendly CSG-like representation for interpretable shape parsing, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 12458–12467, http://dx.doi.org/10.1109/ICCV48922.2021.01225.
[110] F. Yu, Z. Chen, M. Li, A. Sanghi, H. Shayani, A. Mahdavi-Amiri, H. Zhang, CAPRI-Net: Learning compact CAD shapes with adaptive primitive assembly, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 11768–11778.
[111] F. Yu, Q. Chen, M. Tanveer, A. Mahdavi-Amiri, H. Zhang, D^{2}CSG: Unsupervised learning of compact CSG trees with dual complements and dropouts, in: Advances in Neural Information Processing Systems, Vol. 36, 2023.
[112] G. Sharma, R. Goyal, D. Liu, E. Kalogerakis, S. Maji, CSGNet: Neural shape parser for constructive solid geometry, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 5515–5523, http://dx.doi.org/10.1109/CVPR.2018.00578.
[113] C. Wang, W. Sun, X. Ma, F. Deng, Point2Skh: End-to-end parametric primitive inference from point clouds with improved denoising transformer, Comput.-Aided Des. 181 (2025) 103838, http://dx.doi.org/10.1016/j.cad.2024.103838.
[114] J.G. Lambourne, K.D.D. Willis, P.K. Jayaraman, L. Zhang, A. Sanghi, K. Rahimi Malekshan, Reconstructing editable prismatic CAD from rounded voxel models, in: SIGGRAPH Asia 2022 Conference Papers, 2022, pp. 53:1–53:9, http://dx.doi.org/10.1145/3550469.3555424.
[115] M.A. Uy, Y.-Y. Chang, M. Sung, P. Goel, J.G. Lambourne, T. Birdal, L.J. Guibas, Point2Cyl: Reverse engineering 3D objects from point clouds to extrusion cylinders, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 11840–11850, http://dx.doi.org/10.1109/CVPR52688.2022.01155.
[116] P. Li, J. Guo, X. Zhang, D.-M. 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, 2023, pp. 16816–16826, http://dx.doi.org/10.1109/CVPR52729.2023.01613.
[117] P. Li, J. Guo, H. Li, B. Benes, D.-M. 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, 2024, pp. 4671–4680, http://dx.doi.org/10.1109/CVPR52733.2024.00447.
[118] E. Hong, M.H. Nguyen, M.A. Uy, M. Sung, MV2cyl: Reconstructing 3D extrusion cylinders from multi-view images, in: Advances in Neural Information Processing Systems, Vol. 37, 2024.
[119] H. Guo, S. Liu, H. Pan, Y. Liu, X. Tong, B. Guo, ComplexGen: CAD reconstruction by B-rep chain complex generation, ACM Trans. Graph. 41 (4) (2022) 129:1–129:18, http://dx.doi.org/10.1145/3528223.3530078.
[120] Y. Li, C. Lin, Y. Liu, X. Long, C. Zhang, N. Wang, X. Li, W. Wang, X. Guo, CADDreamer: CAD object generation from single-view images, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2025, pp. 21448–21457, http://dx.doi.org/10.1109/CVPR52734.2025.01998.
[121] Y. Liu, A. Obukhov, J.D. Wegner, K. Schindler, Point2CAD: Reverse engineering CAD models from 3D point clouds, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024, pp. 3763–3772, http://dx.doi.org/10.1109/CVPR52733.2024.00361.
[122] Y. Liu, J. Chen, S. Pan, D. Cohen-Or, H. Zhang, H. Huang, Split-and-Fit: Learning B-Reps via structure-aware Voronoi partitioning, ACM Trans. Graph. 43 (4) (2024) 108:1–108:13, http://dx.doi.org/10.1145/3658155.
[123] Z. Yuan, J. Shi, Y. Huang, OpenECAD: An efficient visual language model for editable 3D-CAD design, Comput. Graph. (2024).
[124] D. Mallis, A.S. Karadeniz, S. Cavada, D. Rukhovich, N. Foteinopoulou, K. Cherenkova, A. Kacem, D. Aouada, CAD-Assistant: Tool-augmented VLLMs as generic CAD task solvers, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2025, pp. 7284–7294.
[125] A.C. Doris, M.F. Alam, A. Heyrani Nobari, F. Ahmed, CAD-Coder: An open-source vision-language model for computer-aided design code generation, in: Proceedings of the ASME 2025 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Vol. 89220, 2025, V03AT03A031, http://dx.doi.org/10.1115/DETC2025-169758.
[126] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2014, http://dx.doi.org/10.48550/arXiv.1409.1556, arXiv:1409.1556.
[127] Q. Hu, B. Yang, L. Xie, S. Rosa, Y. Guo, Z. Wang, N. Trigoni, A. Markham, Learning semantic segmentation of large-scale point clouds with random sampling, IEEE Trans. Pattern Anal. Mach. Intell. 44 (11) (2022) 8338–8354, http://dx.doi.org/10.1109/TPAMI.2021.3083288.
[128] N. Xue, T. Wu, S. Bai, F.-D. Wang, G.-S. Xia, L. Zhang, P.H.S. Torr, Holistically-attracted wireframe parsing: From supervised to self-supervised learning, IEEE Trans. Pattern Anal. Mach. Intell. 45 (12) (2023) 14727–14744, http://dx.doi.org/10.1109/TPAMI.2023.3312749.

[129] W. Yin, C. Zhang, H. Chen, Z. Cai, G. Yu, K. Wang, X. Chen, C. Shen, Metric3D: Towards zero-shot metric 3D prediction from a single image, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2023, pp. 9043–9053.
[130] M. Oquab, T. Darcet, T. Moutakanni, H.V. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, et al., DINOv2: Learning robust visual features without supervision, Trans. Mach. Learn. Res. (2024).
[131] Y. Gorishniy, I. Rubachev, V. Khrulkov, A. Babenko, Revisiting deep learning models for tabular data, in: Advances in Neural Information Processing Systems, Vol. 34, 2021, pp. 18932–18943.
[132] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, A. Geiger, Occupancy networks: Learning 3D reconstruction in function space, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 4460–4470.
[133] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, 2014, pp. 1724–1734, http://dx.doi.org/10.3115/v1/D14-1179.
[134] Y. Wang, Y. Sun, Z. Liu, S.E. Sarma, M.M. Bronstein, J.M. Solomon, Dynamic graph CNN for learning on point clouds, ACM Trans. Graph. 38 (5) (2019) 146:1–146:12, http://dx.doi.org/10.1145/3326362.
[135] G. Farin, Curves and Surfaces for CAGD: A Practical Guide, fifth ed., Morgan Kaufmann, 2001.
[136] W.E. Lorensen, H.E. Cline, Marching cubes: A high resolution 3D surface construction algorithm, ACM SIGGRAPH Comput. Graph. 21 (4) (1987) 163–169, http://dx.doi.org/10.1145/37401.37422.
[137] C.-H. Teh, R.T. Chin, On the detection of dominant points on digital curves, IEEE Trans. Pattern Anal. Mach. Intell. 11 (8) (1989) 859–872, http://dx.doi.org/10.1109/34.31447.
[138] P. Dierckx, Algorithms for smoothing data with periodic and parametric splines, Comput. Graph. Image Process. 20 (2) (1982) 171–184, http://dx.doi.org/10.1016/0146-664X(82)90144-7.
[139] O. Ronneberger, P. Fischer, T. Brox, U-Net: Convolutional networks for biomedical image segmentation, in: Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015, 9351, 2015, pp. 234–241, http://dx.doi.org/10.1007/978-3-319-24574-4_28.
[140] A. Hatcher, Algebraic Topology, Cambridge University Press, ISBN: 9780521795401, 2002.
[141] X. Long, Y.-C. Guo, C. Lin, Y. Liu, Z. Dou, L. Liu, Y. Ma, S.-H. Zhang, M. Habermann, C. Theobalt, W. Wang, Wonder3D: Single image to 3D using cross-domain diffusion, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024, pp. 9970–9980, http://dx.doi.org/10.1109/CVPR52733.2024.00951.
[142] P. Wang, L. Liu, Y. Liu, C. Theobalt, T. Komura, W. Wang, NeuS: Learning neural implicit surfaces by volume rendering for multi-view reconstruction, in: Advances in Neural Information Processing Systems, Vol. 34, 2021.
[143] M.A. Fischler, R.C. Bolles, Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography, Commun. ACM 24 (6) (1981) 381–395, http://dx.doi.org/10.1145/358669.358692.
[144] M.A. Kramer, Nonlinear principal component analysis using autoassociative neural networks, AIChE J. 37 (2) (1991) 233–243, http://dx.doi.org/10.1002/aic.690370209.
[145] F. Aurenhammer, Voronoi diagrams—a survey of a fundamental geometric data structure, ACM Comput. Surv. 23 (3) (1991) 345–405, http://dx.doi.org/10.1145/116873.116880.
[146] A. Yang, B. Yang, B. Hui, B. Zheng, B. Yu, C. Zhou, C. Li, C. Li, D. Liu, F. Huang, et al., Qwen2 technical report, 2024, http://dx.doi.org/10.48550/arXiv.2407.10671, arXiv:2407.10671.
[147] pythonOCC, URL https://dev.opencascade.org/project/pythonocc.
[148] S. Mehta, M.H. Sekhavat, Q. Cao, M. Horton, Y. Jin, C. Sun, I. Mirzadeh, M. Najibi, D. Belenko, P. Zatloukal, M. Rastegari, Openelm: An efficient language model family with open training and inference framework, 2024, http://dx.doi.org/10.48550/arXiv.2404.14619, arXiv:2404.14619.
[149] M. Abdin, J. Aneja, H. Behl, S. Bubeck, R. Eldan, S. Gunasekar, M. Harrison, R.J. Hewett, M. Javaheripi, P. Kauffmann, et al., Phi-4 technical report, 2024, http://dx.doi.org/10.48550/arXiv.2412.08905, arXiv:2412.08905.
[150] X. Zhai, B. Mustafa, A. Kolesnikov, L. Beyer, Sigmoid loss for language image pre-training, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2023, pp. 11941–11952, http://dx.doi.org/10.1109/ICCV51070.2023.01100.
[151] H. Liu, C. Li, Q. Wu, Y.J. Lee, Visual instruction tuning, in: Advances in Neural Information Processing Systems, Vol. 36, 2023, pp. 34892–34916.
[152] A. Seff, W. Zhou, N. Richardson, R.P. Adams, Vitruvion: A generative model of parametric CAD sketches, 2021, http://dx.doi.org/10.48550/arXiv.2109.14124, arXiv:2109.14124.
[153] S. Zhou, T. Tang, B. Zhou, CADParser: A learning approach of sequence modeling for B-Rep CAD, in: Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, 2023, pp. 1804–1812, http://dx.doi.org/10.24963/ijcai.2023/200.
[154] S. Zhang, Z. Guan, H. Jiang, T. Ning, X. Wang, P. Tan, Brep2Seq: A dataset and hierarchical deep learning network for reconstruction and generation of computer-aided design models, J. Comput. Des. Eng. 11 (1) (2024) 110–134, http://dx.doi.org/10.1093/jcde/qwae005.
[155] C. Zhang, A. Polette, R. Pinquié, G. Carasi, H. De Charnace, J.P. Pernot, eCAD-Net: Editable parametric CAD models reconstruction from dumb B-Rep models using deep neural networks, Comput.-Aided Des. 178 (2025) 103806, http://dx.doi.org/10.1016/j.cad.2024.103806.
[156] J. Liang, F. He, R. Fan, Y. Chu, X. Yan, CADCL: Reconstruct parametric CAD models from B-rep via contrastive learning, J. Comput. Des. Eng. 12 (10) (2025) 176–184, http://dx.doi.org/10.1093/jcde/qwaf102.
[157] R. Fan, F. He, Y. Liu, Y. Song, L. Fan, X. Yan, A parametric and feature-based CAD dataset to support human-computer interaction for advanced 3D shape learning, Integr. Comput.-Aided Eng. 32 (1) (2025) 75–96, http://dx.doi.org/10.3233/ICA-240744.
[158] X. Xu, W. Peng, C.-Y. Cheng, K.D.D. Willis, D. Ritchie, Inferring CAD modeling sequences using zone graphs, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 6058–6066, http://dx.doi.org/10.1109/CVPR46437.2021.00600.
[159] X. Yin, X. Lu, J. Shen, J. Ni, H. Li, R. Tong, M. Tang, P. Du, RLCAD: Reinforcement learning training gym for revolution involved CAD command sequence generation, 2025, http://dx.doi.org/10.48550/arXiv.2503.18549, arXiv:2503.18549.
[160] J. Gilmer, S.S. Schoenholz, P.F. Riley, O. Vinyals, G.E. Dahl, Neural message passing for quantum chemistry, in: Proceedings of the 34th International Conference on Machine Learning, Vol. 70, 2017, pp. 1263–1272.
[161] C.R. Qi, H. Su, K. Mo, L.J. Guibas, PointNet: Deep learning on point sets for 3D classification and segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 652–660, http://dx.doi.org/10.1109/CVPR.2017.16.
[162] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, 2017, http://dx.doi.org/10.48550/arXiv.1707.06347, arXiv:1707.06347.
[163] D. Marcheix, G. Pierra, A survey of the persistent naming problem, in: Proceedings of the Seventh ACM Symposium on Solid Modeling and Applications, 2002, pp. 13–22, http://dx.doi.org/10.1145/566282.566288.
[164] Onshape, Onshape | product development platform, URL https://www.onshape.com/en/.
[165] Autodesk, Autodesk community gallery, URL https://www.autodesk.com/community/gallery.
[166] Onshape, Welcome to the onshape developer documentation, URL https://onshape-public.github.io/docs/.
[167] Y. Wang, Y. Kordi, S. Mishra, A. Liu, N.A. Smith, D. Khashabi, H. Hajishirzi, Self-Instruct: Aligning language models with self-generated instructions, in: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, 2023, pp. 13484–13508, http://dx.doi.org/10.18653/v1/2023.acl-long.754.
[168] Z. Zhang, P. Jaiswal, R. Rai, FeatureNet: Machining feature recognition based on 3D convolution neural network, Comput.-Aided Des. 101 (2018) 12–22, http://dx.doi.org/10.1016/j.cad.2018.03.006.
[169] A.R. Colligan, T.T. Robinson, D.C. Nolan, Y. Hua, W. Cao, Hierarchical CADNet: Learning from B-Reps for machining feature recognition, Comput.-Aided Des. 147 (2022) 103226, http://dx.doi.org/10.1016/j.cad.2022.103226.
[170] A. Seff, Y. Ovadia, W. Zhou, R.P. Adams, SketchGraphs: A large-scale dataset for modeling relational geometry in computer-aided design, 2020, http://dx.doi.org/10.48550/arXiv.2007.08506, arXiv:2007.08506.
[171] Amazon Web Services, Amazon mechanical turk, URL https://www.mturk.com/.
[172] C. Chaumet, Design Space Exploration in Engineering Automation (Ph.D. thesis), TU Dortmund University, 2025, http://dx.doi.org/10.17877/DE290R-25805, URL https://eldorado.tu-dortmund.de/handle/2003/44037.
[173] S.A.M. Tofail, E.P. Koumoulos, A. Bandyopadhyay, S. Bose, L. O'Donoghue, C. Charitidis, Additive manufacturing: Scientific and technological challenges, market uptake and opportunities, Mater. Today 21 (1) (2018) 22–37, http://dx.doi.org/10.1016/j.mattod.2017.07.001.
[174] A. Bujarska, P. Zmarzły, P. Szczygieł, Application of optical measurements to assess form deviations of cylindrical parts made using FDM additive technology, Sensors 25 (18) (2025) 5855, http://dx.doi.org/10.3390/s25185855.
[175] M.J. Pratt, B.D. Anderson, T. Ranger, Towards the standardized exchange of parameterized feature-based CAD models, Comput.-Aided Des. 37 (12) (2005) 1251–1265, http://dx.doi.org/10.1016/j.cad.2004.12.005.
[176] R.N. Nagel, W.W. Braithwaite, P.R. Kennicott, Initial Graphics Exchange Specification IGES Version 1.0, Tech. Rep. NBS IR 80-1978, National Bureau of Standards, 1980, http://dx.doi.org/10.6028/NBS.IR.80-1978.
[177] J. Kim, M.J. Pratt, R.G. Iyer, R.D. Sriram, Standardized data exchange of CAD models with design intent, Comput.-Aided Des. 40 (7) (2008) 760–777, http://dx.doi.org/10.1016/j.cad.2007.06.014.