UNIVERSITÉ DU LUXEMBOURG

PhD-FSTM-2025-077
The Faculty of Science, Technology and Medicine

DISSERTATION

Defence held on 30 June 2025 in Luxembourg
to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG EN INFORMATIQUE

by

Elona DUPONT
Born on 14 April 1981 in Pontoise (France)

DESIGN INTENT AWARE CAD REVERSE ENGINEERING: DEEP NEURAL APPROACHES FOR RECOVERING FEATURE-BASED SEQUENCES FROM 3D SCANS

Dissertation defence committee

Dr. Djamila AOUADA, Dissertation Supervisor
Assistant Professor, SnT, Université du Luxembourg

Dr. Gilbert FRIDGEN, Chairman
Professor, SnT, Université du Luxembourg

Dr. Lourdes AGAPITO
Professor, University College London

Dr. Tolga BIRDAL
Assistant Professor, Imperial College London

Dr. Patrick VANDEWALLE
Associate Professor, KU Leuven

Affidavit / Statement of originality

I declare that this thesis:

- is the result of my own work. Any contribution from any other party, and any use of generative artificial intelligence technologies have been duly cited and acknowledged;
- is not substantially the same as any other that I have submitted, and;
- is not being concurrently submitted for a degree, diploma or other qualification at the University of Luxembourg or any other University or similar institution except as specified in the text.

With my approval I furthermore confirm the following:

- I have adhered to the rules set out in the University of Luxembourg's Code of Conduct and the Doctoral Education Agreement (DEA)[1], in particular with regard to Research Integrity.
- I have documented all methods, data, and processes truthfully and fully.
- I have mentioned all the significant contributors to the work.
- I am aware that the work may be screened electronically for originality.

I acknowledge that if any issues are raised regarding good research practices based on the review of the thesis, the examination may be postponed pending the outcome of any investigation of such issues. If a degree was conferred, any such subsequently discovered issues may result in the cancellation of the degree.

Approved on 2025-06-06

[1] If applicable (DEA is compulsory since August 2020)

Acknowledgements

I would like to express my sincere appreciation to Prof. Djamila Aouada for providing me with the opportunity to complete my doctoral studies under her supervision and for her valuable guidance throughout this journey.

My heartfelt thanks go to Anis for his infinite patience and unwavering belief in me. Learning from you and collaborating on papers has been both educational and enjoyable.

I extend my gratitude to Kseniya for her regular meetings, insightful advice, and the trust she placed in me to present her work at various conferences. Additionally, I wish to acknowledge Gleb and the Artec3D team for their consistent support during my thesis work.

My appreciation extends to all colleagues with whom I have collaborated on different publications throughout my time at CVI2.

I wish to recognize those who ensured I could work in a clean and safe environment. The maintenance staff, who often sacrifice time with their families to work at unconventional hours, deserve special mention for their essential contribution.

Finally, I must acknowledge the university administration and IT department for their initial acceptance over four years ago to implement a preferred name option. However, despite being designated as a “top priority project”, this feature remains unimplemented for PhD students and staff at the time of completing this thesis. The prolonged absence of a solution to this matter speaks volumes about the actual commitment to inclusion at the University of Luxembourg.

The work of this thesis is supported by the National Research Fund, Luxembourg under the BRIDGES2021/IS/16849599/FREE-3D and IF/17052459/CASCADES projects, and by Artec 3D.

To Adrian and Lia.

Index

List of Abbreviations

CAD . . . . . Computer Aided Design

B-Rep . . . . . Boundary Representation

CSG . . . . . Constructive Solid Geometry

LLM . . . . . Large-Language Model

CNN . . . . . Convolutional Neural Networks

APCS . . . . . mean Average Precision of CAD Sequence

GUI . . . . . Graphical User Interface

NURBS . . . . . Non-Uniform Rational B-splines

SDF . . . . . Signed Distance Functions

MLP . . . . . Multi-Layer Perceptron

LFA . . . . . Local Feature Aggregation

GNN . . . . . Graph Neural Networks

SA . . . . . Self-Attention

CA . . . . . Cross-Attention

op.type . . . . . CAD operation type

op.step . . . . . CAD operation step

RIoU . . . . . Relaxed Intersection over Union

mAcc . . . . . mean Accuracy

mIoU . . . . . mean Intersection over Union

API . . . . . Application Programming Interface

CD . . . . . Chamfer Distance

CSSS . . . . . CAD Sequence Similarity Score

IR . . . . . Invalidity Ratio

AR . . . . . Auto-regressive

NAR . . . . . Non auto-regressive

NLL . . . . . Negative Log-Likelihood

MLLMs . . . . . Multimodal Large Language Models

List of Figures

List of Tables

Abstract

Computer-Aided Design (CAD) modeling plays a fundamental role in modern industrial manufacturing processes, enabling precise design and modification of products across diverse sectors. However, when existing physical objects lack associated CAD models because of lost documentation, or legacy part replacement requirements, engineers must engage in the time-consuming and expertise-intensive process of CAD reverse engineering. This process involves translating 3D scanned data into parametric CAD models that not only capture geometric form but also preserve design intent. Despite significant advances in 3D scanning technologies, automated CAD reverse engineering faces several persistent challenges: scanning artifacts that corrupt geometric fidelity, the lack of unified CAD representation standards, the inherent ambiguity in recovering design intent, and the limited availability of high-quality datasets with construction history annotations.

This thesis presents a systematic progression of approaches to address these challenges, beginning with the recovery of elements of construction history from existing Boundary Representations through joint learning of operation types and steps. Moving closer to the 3D reverse engineering scenario, we then advance to predicting CAD history sequences directly from point clouds using hierarchical transformer architectures that eliminate the need for intermediate representations. To overcome the inherent ambiguity in reverse engineering, we develop geometry-guided search strategies that explore multiple design alternatives, mimicking the decision-making process of expert CAD designers. Finally, we introduce a paradigm shift by leveraging Large Language Models to generate executable Python code directly from point clouds, simultaneously addressing representation limitations and training

data constraints through procedural generation techniques. Throughout this progression, we contribute novel datasets that enable more realistic evaluation and establish evaluation methodologies specifically designed for the CAD reverse engineering domain. The resulting frameworks significantly improve reconstruction quality while enhancing practical usability, advancing the field toward fully automated yet flexible CAD reverse engineering that can seamlessly integrate into industrial design workflows.

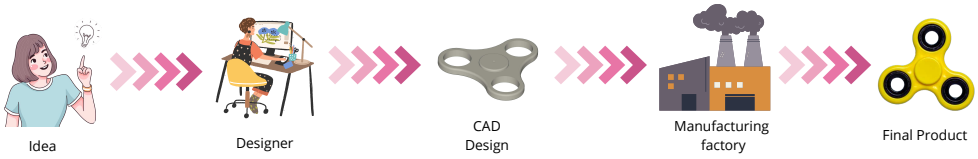
Chapter 1

Introduction

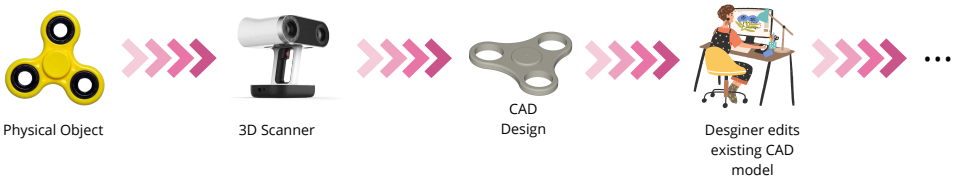
The digital transformation of industries represents one of the most profound technological shifts of the modern era. Across manufacturing, architecture, aerospace, and numerous other fields, the transition from analog to digital workflows has fundamentally altered how products are conceived, designed, and brought to market [1]. Within this broader revolution, few technologies have had as significant an impact as Computer-Aided Design (CAD) - a technology that allows engineers and designers to create, modify, and optimize product designs using computer systems rather than traditional manual drafting methods [2]. Before CAD, product design relied heavily on hand-drawn technical drawings, a process that was time-consuming, error-prone, and impractical for frequent iteration. CAD technology has transformed this landscape by providing software tools that enable the creation of precise digital models in three dimensions [3].

CAD models have revolutionized modern engineering and manufacturing by providing a comprehensive digital framework for product development. At their core, these models serve as precise digital representations that enable engineers and designers to craft and refine products before investing in physical production. The real power of CAD models lies in their versatility - they seamlessly transition from design to production, serving as direct inputs for manufacturing processes like CNC machining and 3D printing, ensuring accuracy and consistency in the final product [4]. CAD models have transformed how teams collaborate across the globe, creating a digital language that bridges communication gaps between

designers, engineers, and manufacturers [5, 6]. The iterative nature of CAD modeling allows engineers to rapidly test and optimize designs, performing complex analyses for factors like structural integrity and aerodynamics [7, 8]. These models also serve as living documentation, preserving exact technical specifications that prove invaluable for future modifications or maintenance.



(a) CAD design process



(b) CAD reverse engineering process

Figure 1.1: CAD models are created by skill designer in order to create a 3D model of an object that can then be manufactured as depicted in 1.1a. The 3D CAD reverse engineering process consists in recovering the CAD model of an existing physical object as shown in 1.1b.

.

Building upon the fundamental importance of CAD models in modern engineering, the field of 3D reverse engineering has emerged as both a critical necessity and a persistent technical challenge in contemporary industrial applications. 3D reverse engineering refers to the process of digitally reconstructing CAD models from existing physical objects—essentially working backwards from the final product to create its digital design representation [9]. Figure 1.1 depicts the differences between the forward design process and reverse engineering. Despite significant technological advances in data acquisition through 3D scanning technologies, the fully automated reconstruction of precise, parametric CAD models from physical objects remains an open problem [10]. The complexity lies not merely in capturing geometric data, but in the process of reproducing design intent from physical objects that allow for future edibility. The strategic value of reverse engineering capabilities extends throughout the industrial landscape: it enables the preservation and modernization of legacy components

whose original documentation has been lost, facilitates rapid competitive analysis through detailed understanding of existing products, and allows for the adaptation of proven designs to new requirements without starting from scratch [11]. The ongoing challenge of automating the translation from 3D data obtained from sensors to fully featured CAD models represents a significant opportunity for technological advancement, promising to dramatically reduce the time and expertise currently required for reverse engineering processes [12].

This thesis is focused on advancing the field of 3D CAD reverse engineering. In this chapter, we start by describing the motivation and scope of the present thesis. Then, we detail the challenges in 3D CAD reverse engineering that are addressed in this research work. In the following, the main objectives and contributions to the field of 3D CAD reverse engineering are outlined. Finally, a list of publications that have resulted from these PhD investigations is presented.

1.1 Motivation and Scope

Beyond the potential practical applications, CAD reverse engineering presents a rich research problem from a scientific perspective. The challenge extends far beyond merely reconstructing geometric shapes as it requires capturing and codifying the human ingenuity embedded in CAD models. Successfully reverse engineering these models requires not just capturing the final geometry, but also how it was created. This intersection of computational geometry, artificial intelligence, and design methodology creates a rich problem space where advances could enhance our understanding of both engineering practices and computational representation of human creative processes. Building on this perspective of CAD reverse engineering as a multifaceted research problem, this thesis addresses three specific problems in CAD reverse engineering that remain unresolved.

In the first part of the thesis, we tackle the problem of recovering elements of the construction history from CAD models represented as Boundary Representation (B-Rep), an aspect largely overlooked in existing research that has focused mainly on B-Rep object classification and face segmentation [13, 14, 15]. Specifically, our work focuses on the segmentation and

grouping of faces according to their position in the construction timeline, enabling the recovery of critical design information such as the sketches drawn by designers. This contribution represents a step toward recovering not just the geometry of CAD models, but also the underlying design intent and methodology.

Although our B-Rep analysis advances the recovery of design intent, it presupposes access to structured CAD data. To address more common reverse engineering scenarios that begin with unstructured point clouds, we developed our second approach. While existing deep learning methods [16, 17, 18, 19, 20, 21, 22] have made progress in identifying individual CAD elements from point clouds (such as parametric edges and faces), they typically fail to capture the critical design intent and construction sequence. The second part of the thesis focuses on the direct reconstruction of parametric, editable CAD models from raw point cloud data, representing a significant advancement beyond current methods that produce non-editable geometric representations [23, 24, 25]. For this research, we focus specifically on CAD models created exclusively through extrusion operations, which represent a substantial subset of CAD models [26] while allowing us to develop more robust reconstruction techniques.

Building on insights gained, we identified two critical areas that required further innovation: data scarcity and practical implementation challenges. Publicly available CAD datasets [26, 27, 28] provide limited control over both data quantity and design diversity, while the language-like CAD representation we developed requires substantial post-processing to integrate with existing software. The third part of this thesis addresses these limitations through a novel Large Language Model (LLM) based method that directly generates executable Python code. By training on a procedurally generated dataset of extrusion-based models, we gain precise control over training examples while eliminating the integration barrier. This approach significantly improves both reconstruction accuracy and practical usability, representing a substantial advancement toward automated CAD reverse engineering with direct industry applicability.

In summary, this thesis presents a progressive approach to CAD reverse engineering, advancing from B-Rep construction history analysis to parametric CAD reconstruction from

point clouds, and culminating in direct generation of executable CAD code from 3D scans. Throughout this research, our focus remains on recovering not just geometric shapes, but also the human design intent embedded within CAD models, thereby bridging the gap between computer vision advances and practical industrial applications in CAD reverse engineering.

1.2 Challenges

In this thesis, we aim to address the multifaceted set of technical challenges that span the entire CAD reverse engineering pipeline. These challenges begin with the imperfections of 3D scanned data, extend through the lack of unified CAD representation standards, encompass the complex task of recovering design intent, and include the fundamental data limitations that constrain machine learning approaches. In this section, we present a description of each of these challenges to provide context for the contributions made in this thesis.

1.2.1 3D Scanning Imperfections and Artifacts

Despite remarkable advances in 3D scanning technology, the digital representations acquired from these devices remain geometrical approximations of the physical objects they capture. This fundamental disparity in representation between the precise, parametric description of geometry and topology in CAD models versus the approximate nature of 3D scan meshes creates the first significant challenge in reverse engineering. Scanning artifacts appear in various forms that corrupt the geometric fidelity: surface noise that introduces random perturbations, erroneous protrusions that distort the shape, missing regions where data could not be captured, and edges that appear rounded when they should be sharp (see Figure 1.2). These imperfections stem from multiple sources, including the material properties of the scanned object (reflectivity, translucency), surface coloration that affects light absorption, and occlusions that prevent the scanner's light from reaching certain areas [30]. The cumulative effect of these artifacts significantly complicates the reverse engineering process, as design features must be distinguished from scanning anomalies while reconstructing a parametrically accurate CAD model.

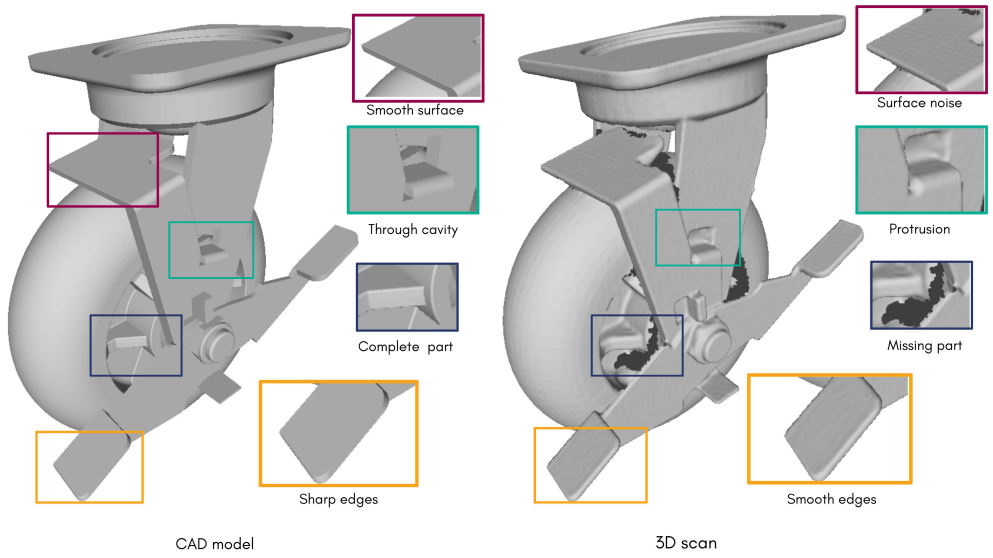


Figure 1.2: 3D scans differ from CAD objects as during the scanning process many details are lost. Scanning artifacts contain surface noise, protrusion, missing parts and smoothed edges. Those artifacts make the task of reverse engineering particularly challenging. [29]

1.2.2 Lack of Unified CAD Representation Standards

The evolution of CAD systems since their inception in the mid-1960s has produced a diverse and heterogeneous landscape of representations and formats that significantly complicates reverse engineering efforts [31]. CAD development has been predominantly driven by industrial needs and commercial interests, resulting in a proliferation of proprietary approaches. Major software companies have each developed their own geometry kernels, and data storage formats that are often incompatible with competitors' systems. This fundamental lack of a unified, universal representation for CAD models presents a substantial challenge for reverse engineering tasks.

The range of CAD representations has direct implications for developing suitable deep learning approaches. While other domains have benefited from standardized neural network architectures tailored to their data structures (*e.g.*, Convolutional Neural Networks for 2D images, Transformers for sequential data like text, and specialized architectures for unstructured 3D data such as MeshCNN [32] for meshes and PointNet [33] for point clouds),

the CAD domain lacks such consensus. Developing effective neural network architectures for CAD models consequently requires simultaneous innovation on two fronts: designing appropriate representations of the CAD data that capture both geometric and topological information, and creating compatible network architectures that can effectively learn from these representations. This dual challenge substantially increases the complexity of applying modern machine learning techniques to the reverse engineering process.

In this thesis, different CAD representations are considered with the aim of facilitating the integration of the predicted models within existing CAD software.

1.2.3 Beyond Geometry: Recovering Design Intent

Design intent represents a foundational concept in CAD modeling, yet it remains elusive in its precise definition, with ongoing research and debate within the engineering community [34]. For the scope of this thesis, we adopt the definition provided by the International Organization for Standardization, which characterizes design intent as the “intentions of the designer of a model with regard to how it may be instantiated or modified” [34]. This definition underscores a critical insight: a comprehensive CAD model must encapsulate more than just the geometric shape description as it must also preserve the underlying rationale of how that shape can be modified with respect to its functionality. Moreover, another significant challenge emerges from the variability in design approaches; multiple designers can achieve identical final geometries through substantially different construction sequences and modeling operations [35]. This non-uniqueness of solution paths introduces a fundamental ambiguity into the reverse engineering process. Recovering a CAD model from a scanned object is not merely a geometric reconstruction problem but a more complex challenge of inferring one valid construction pathway from potentially many valid alternatives, none of which can be definitively identified as the ”true” original without additional contextual information.

In this thesis, design intent will be addressed by developing methods that rely on learning patterns from existing models and also by identifying representations that facilitate editibility of the generated CAD models.

1.2.4 The CAD Data Challenge: Quality, Representation, and Availability

A fundamental challenge in applying machine learning to CAD reverse engineering stems from data requirements and limitations. The training of neural networks depends on the availability of high-quality data, yet CAD datasets present unique obstacles in this regard. The quality and diversity of training data directly influence prediction accuracy, with insufficient variation leading to poor generalization to novel designs. This data quality challenge is significantly compounded by the prevalence of proprietary models and modeling operations in industrial settings, where commercial interests restrict the public availability of comprehensive CAD examples.

Even when large-scale CAD datasets do exist, they frequently lack the representations necessary to capture design intent. For instance, the ABC dataset [28], while impressive with its one million models, only provides labels for parametric descriptions of surfaces, curves, and patch segmentation, information that falls short of describing the actual construction process employed by CAD designers. To address this issue, researchers have developed self-supervised learning approaches [24, 25], yet these methods typically struggle to fully capture the essential design intent embedded in CAD models. Moreover, their outputs often fail to satisfy the editibility requirement of reverse engineering.

Beyond quality and labeling issues, preparing CAD data for machine learning introduces additional complexity, as specialized preprocessing pipelines are required to transform raw CAD files into suitable training formats. The works of DeepCAD [26] made progress in this direction by parsing approximately $180\,$k CAD models into a vector sequence format directly usable for learning methods. However, subsequent analysis revealed a significant limitation: the dataset contains numerous duplicate models [36, 37], which can bias learning algorithms and reduce effective dataset size.

In this thesis, we consider the challenge related to CAD datasets from different perspectives, including using and evaluating existing datasets and proposing new datasets to address identified needs.

1.3 Objectives and Contributions

In response to the complex, interconnected challenges inherent to CAD reverse engineering, this thesis adopts a systematic approach to developing solutions across multiple dimensions of the problem. Rather than addressing each challenge in isolation, our research recognizes their fundamental interdependence and explores complementary methodological approaches that collectively advance the state-of-the-art. The following sections present the principal contributions developed during these doctoral studies.

1.3.1 Learning CAD Operation Types and Steps from B-Reps

Our first contribution addresses a fundamental component of the CAD reverse engineering challenge: the recovery of construction history from final geometry. In this work, we introduce CADOps-Net, a novel approach that demonstrates the feasibility of inferring crucial elements of a CAD model's creation process directly from its Boundary Representation (B-Rep). Specifically, we develop a deep neural network architecture that learns to identify the types of CAD operations (such as extrusions, revolutions, and fillets) through B-Rep face segmentation. This task is jointly learned with the grouping of the faces according to their design steps. This joint learning framework enables the decomposition of a complete B-Rep into semantically meaningful parts that correspond to specific operations performed at particular stages of the design process, hence providing essential information for design history recovery.

To support this research and advance the field, we propose the CC3D-Ops dataset, comprising over 37 k CAD models with annotations for operation types and construction steps. A distinguishing feature of CC3D-Ops is its inclusion of models with complexity and diversity more representative of industrial applications than existing public datasets. Our extensive experimental evaluation on both CC3D-Ops and the Fusion360 dataset [27] demonstrates that CADOps-Net achieves competitive performance compared to state-of-the-art methods while validating the advantages of jointly learning operation types and steps over treating them as separate tasks. The practical significance of our approach extends beyond classification performance. We demonstrate that the predictions from CADOps-Net can be leveraged to

recover the original 2D sketches used in extrusion operations which is a critical step toward complete design recovery. This work was published in [38].

1.3.2 A hierarchical Transformer for CAD Sequence Inference from Point Clouds

Building upon our initial work on B-Rep analysis, we next address the more challenging task of direct point cloud to CAD model, a problem that encompasses all four challenges identified earlier. While CADOps-Net explored recovering design history from an already-available B-Rep, TransCAD tackles the complete reverse engineering pipeline by inferring a fully parametric CAD model in the form of a sketch-extrude sequence. This contribution introduces an end-to-end feedforward transformer-based architecture that predicts a complete CAD construction sequence, thereby addressing both the representation challenge and the need to recover design intent within a unified framework. TransCAD leverages the structure of CAD sequences through a hierarchical learning strategy that mirrors the structure of CAD sketches and operations followed by designers. Furthermore, we introduce a specialized loop refiner module that regresses sketch primitive parameters in order to overcome the limitation of the cross-entropy loss used on quantized parameters.

Rigorous experimentation on the DeepCAD [26] and Fusion360 [27] datasets demonstrates that TransCAD achieves state-of-the-art results. To properly evaluate performance in the CAD reverse engineering domain, we introduce a new metric,the *mean Average Precision of CAD Sequence* with the goal of addressing the limitations of existing evaluation approaches. Finally, to provide results on a scenario that is closer to the scan to CAD reverse engineer scenario, we propose to evaluate TransCAD on noisy inputs with missing points. This work was published in [37].

1.3.3 Inference Geometry Guided Search for Auto and Non Auto-Regressive Scan-to-CAD Networks

While TransCAD demonstrated the feasibility of direct point cloud to parametric CAD reconstruction, our analysis revealed that the fundamental challenge of bridging the gap between noisy scan data and precise design intent remained unresolved. This limitation motivated our third contribution, which directly addresses the inherent ambiguity in scan-based reverse engineering by recognizing that multiple valid design interpretations can exist for a single input scan, hence mirroring how expert CAD designers typically explore various construction alternatives. Building upon our previous findings, we develop inference-time geometry-guided search strategies that significantly enhance reconstruction quality for both auto-regressive [39] and non-auto-regressive [37] approaches.

This work makes three interconnected contributions that address previously identified challenges. First, we provide a comprehensive comparative analysis of auto-regressive and non-auto-regressive architectures for CAD reconstruction, revealing their complementary strengths across different evaluation contexts. Second, we introduce novel geometry-guided inference search strategies that leverage geometric priors from the input point cloud to more effectively explore the neural network's output probability space. These strategies significantly improve reconstruction quality while reducing invalid CAD predictions, directly addressing the scanning artifacts challenge by using geometric constraints to disambiguate noisy input data. Third, we contribute the C3D-Recon dataset, the first collection of pairs of realistic 3D scans containing realistic scanning artifacts with their corresponding parametric CAD design sequences, hence directly addressing the data quality and representation challenges highlighted earlier.

Our extensive experimental evaluation demonstrates that the proposed search strategies substantially enhance performance across multiple datasets, with particularly notable improvements in challenging real-world scenarios featuring scanning noise and artifacts. This work was published in [40].

1.3.4 Reverse Engineering CAD Code from Point Clouds using LLM

Our previous contributions progressively addressed critical aspects of the CAD reverse engineering pipeline, from recovering design elements in existing B-Reps to generating parametric CAD sequences from point clouds with geometry-guided refinement. However, a fundamental limitation remained: the gap between research outputs and practical industry adoption due to the lack of integration with commercial CAD systems. Our final contribution, CAD-Recode, represents a paradigm shift that directly addresses this challenge while simultaneously tackling CAD representation and data quality issues identified earlier.

Unlike prior approaches that output intermediate representations requiring further processing, CAD-Recode translates point cloud data directly into executable Python code that, when run, reconstructs the complete CAD model in standard modeling environments. This innovation operates at three complementary levels: representation, network architecture, and training data. At the representation level, we reconceptualize CAD sketch-extrude sequences as Python code. For network design, we leverage the code-understanding capabilities of pre-trained Large Language Models (LLMs), combining a relatively small LLM decoder with a lightweight point cloud projector to create an efficient yet powerful architecture. Most significantly, we address the data scarcity challenge through a novel procedurally generated dataset of one million diverse CAD sequences, providing control over the amount of data and design features included during training.

Extensive evaluations demonstrate that CAD-Recode significantly outperforms existing methods across the DeepCAD [26], Fusion360 [27], and real-world CC3D [41] datasets, with particularly notable improvements on complex geometries with authentic scanning artifacts. Beyond quantitative performance gains, CAD-Recode offers two additional benefits: the direct integration with commercial CAD software through standard Python interfaces, eliminating the post-processing requirements of previous methods, and the interpretability of its output by off-the-shelf LLMs, enabling natural language-based CAD editing and CAD-specific question answering directly from point clouds. This work was published in [42].

1.4 Publications

JOURNAL

1. **Dupont, E.**, Rukhovich, D., Cherenkova, K., Mallis, D., Kacem, A., Aouada, D., 2025, "abraCADabra: Inference Geometry Guided Search for Auto and Non Auto-Regressive Scan-to-CAD Networks", In preparation for IEEE Transactions on Pattern Analysis and Machine Intelligence.

CONFERENCES

1. **Dupont, E.**, Cherenkova, K., Kacem, A., Aziz, A.S., Arzhannikov, I., Gusev, G., Aouada, D., 2022, "CADOps-Net: Jointly Learning CAD Operation Types and Steps from Boundary-Representations", International Conference on 3D Vision.

2. **Dupont, E.**, Cherenkova, K., Mallis, D., Gusev, G., Kacem, A., Aouada, D., 2024, "Transcad: A hierarchical transformer for cad sequence inference from point clouds", European Conference on Computer Vision.

3. Rukhovich, D., **Dupont, E.**, Mallis, D., Cherenkova, K., Kacem, A., Aouada, D., 2025, "CAD-Recode: Reverse Engineering CAD Code from Point Clouds", IEEE/CVF International Conference on Computer Vision.

PUBLICATIONS NOT INCLUDED IN THIS THESIS

1. Khan, M.S., **Dupont, E.**, Ali, S.K., Cherenkova, K., Kacem, A., Aouada, D., 2024, "CAD-SIGNet: CAD Language Inference from Point Clouds using Layer-wise Sketch Instance Guided Attention", Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.

2. Cherenkova, K., **Dupont, E.**, Kacem, A., Gusev, G., Aouada, D., 2024, "SpelsNet: Surface Primitive Elements Segmentation by B-Rep Graph Structure Supervision", The Thirty-eighth Annual Conference on Neural Information Processing Systems.

3. Mallis, D., Aziz, A.S., **Dupont, E.**, Cherenkova, K., Karadeniz, A.S., Khan, M.S., Kacem, A., Gusev, G., Aouada, D.,, 2023, "SHARP Challenge 2023: Solving CAD History and pArameters Recovery from Point clouds and 3D scans. Overview, Datasets, Metrics, and Baselines" Proceedings of the IEEE/CVF International Conference on Computer Vision.

4. Cherenkova, K., **Dupont, E.**, Kacem, A., Arzhannikov, I., Gusev, G., Aouada, D., 2023, "Sepicnet: Sharp edges recovery by parametric inference of curves in 3d shapes", Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.

5. **Dupont, E.**, Singh, I.P., Fuentes, L., Aziz, S.A., Kacem, A., Ghorbel, E., Aouada, D., 2023, "You Can Dance! Generating Music-Conditioned Dances on Real 3D Scans.", International Conference on Computer Vision Theory and Applications.

6. K., Karadeniz, Aziz, A.S., Kacem, A., **Dupont, E.**, Aouada, D., 2022, "Tscom-net: Coarse-to-fine 3d textured shape completion network", European Conference on Computer Vision.

1.5 Thesis Outline

This dissertation is organized as follows:

- **Chapter 2**: In this chapter, we provide the foundation of key concepts underpinning CAD reverse engineering, beginning with solid modeling paradigms and CAD representations. We then examine deep learning approaches relevant to reverse engineering, including point cloud encoders, graph neural networks, and transformer architectures.

- **Chapter 3**: This chapter introduces CADOps-Net, a neural network architecture for jointly learning CAD operation types and steps from Boundary Representations. We also present the CC3D-Ops dataset containing over 37k annotated B-Reps and demonstrate how this approach can be used to recover original 2D sketches from B-Rep faces.

- **Chapter 4**: Here we present TransCAD, a hierarchical transformer architecture for inferring CAD sequences directly from point clouds. This approach eliminates the need for intermediate B-Rep representations while introducing a specialized evaluation framework and demonstrating robustness to input perturbations.

- **Chapter 5**: This chapter explores abraCADabra, an extension of our previous work that implements geometry-guided search strategies for both auto-regressive and non-auto-regressive approaches. We also introduce the C3D-Recon dataset containing pairs of realistic 3D scans with their corresponding parametric CAD sequences, enabling more robust evaluation under real-world conditions.

- **Chapter 6**: In this chapter, we present CAD-Recode, an approach leveraging Large Language Models to translate point clouds directly into executable Python code. This paradigm shift addresses limitations in representation, model architecture, and training data through a procedurally generated dataset of one million CAD sequences.

- **Chapter 7**: The final chapter synthesizes our key contributions to CAD reverse engineering and their relationship to the challenges identified in the introduction. We also explore promising directions for future research, including interactive multimodal reconstruction systems and assembly-based approaches that could further advance practical applications of automated CAD reconstruction.

Chapter 2

Background

In this chapter, we provide the foundation of the key concepts underpinning CAD reverse engineering. We begin with an exploration of solid modeling paradigms and their evolution, including CAD-specific representations and various discrete 3D shape representations. The chapter then focuses on deep learning approaches relevant to reverse engineering, discussing point cloud encoders, graph neural networks for processing topological structures, and transformer architectures.

2.1 Solid Modeling

This section examines the evolution of solid modeling paradigms and the mathematical representations that enable the precise definition of three-dimensional objects in digital environments.

2.1.1 Introduction to 3D Shape Representation

Since the invention of the first CAD system, Sketchpad, by Ivan Sutherland in 1963, solid modeling has undergone five major paradigm shifts that have fundamentally transformed how engineers interact with digital design systems [43]. Figure 2.1 illustrates this evolution of CAD modeling approaches over time. The first generation of CAD systems enabled users to

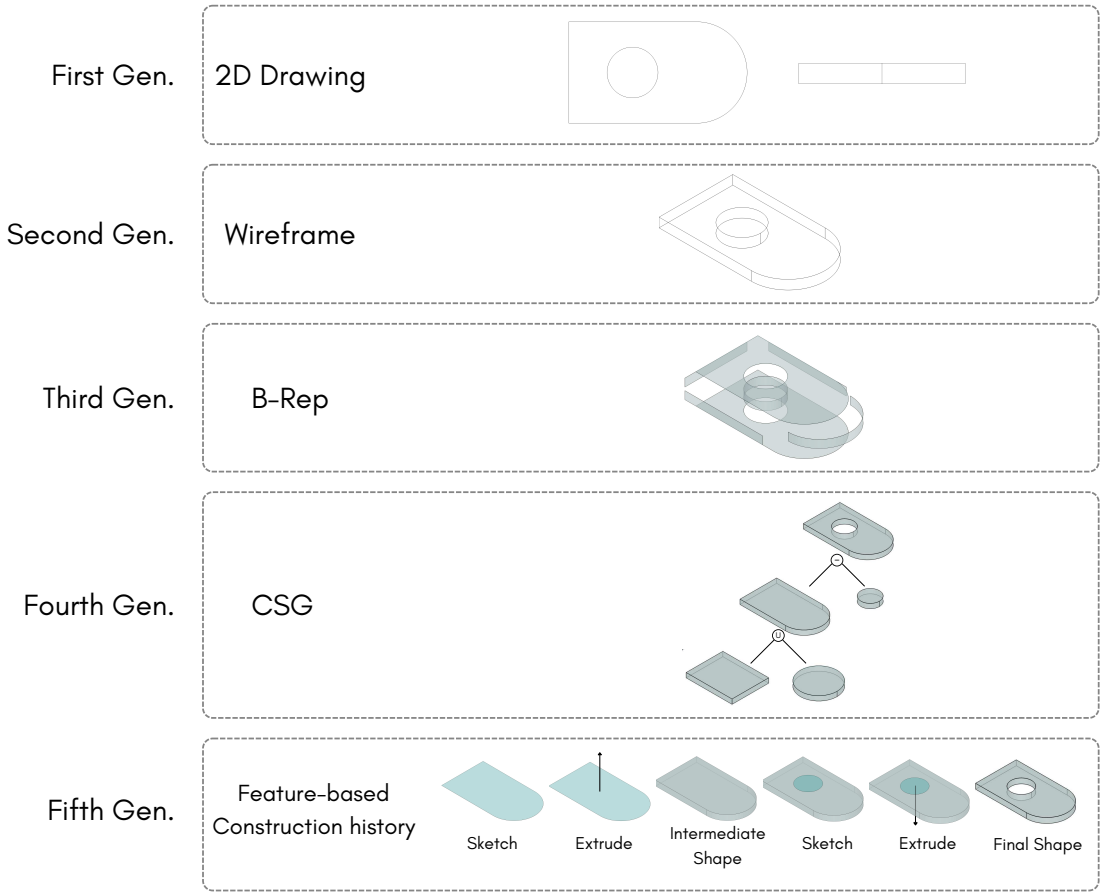
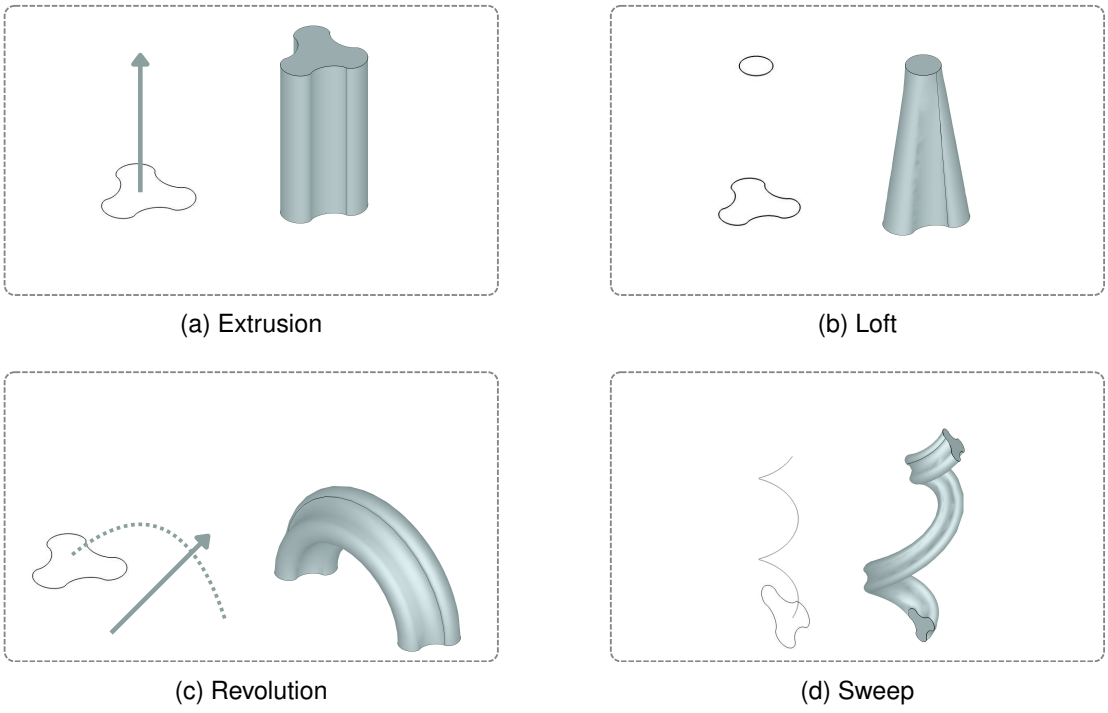


Figure 2.1: Illustration of the five CAD modeling paradigms. The first generation began with simple 2D drawings. The second generation introduced wireframe models. The third generation marked a significant advancement with Boundary Representation (B-Rep), where solids are defined as shells. The fourth generation introduced Constructive Solid Geometry (CSG), representing objects as hierarchical trees of boolean operations applied to primitive shapes. Finally, the fifth and current generation, known as feature-based modeling, describes shapes as construction histories.

create 2D projections of shapes using simple curves, primarily digitalizing traditional drafting practices. This foundation was extended in the second generation to incorporate wireframe models as collections of curves defining the edges of 3D objects, though these lacked the ability to distinguish between interior and exterior spaces. The third generation marked a significant advancement with the introduction of Boundary Representation (B-Rep), in which solids are described as shells of connected parametric faces, providing a mathematically complete representation of 3D geometry.



(a) Extrusion (b) Loft

(c) Revolution (d) Sweep

Figure 2.2: Examples of operations in feature-based modeling. One or more parametric sketches form a profile. An operation such as (a) extrusion, (b) loft, (c) revolution and (d) sweep can be applied on a profile to create a 3D shape.

The fourth generation sought to simplify the construction process through Constructive Solid Geometry (CSG), representing models as hierarchical construction trees of boolean operations (union, intersection, subtraction) applied to primitive shapes such as spheres, cylinders, and blocks. With the transition from specialized workstations to personal computers and the advent of Graphical User Interfaces (GUIs) in the 1990s, the fifth generation emerged,

feature-based modeling, also known as history-based CAD [44]. This paradigm revolutionized the design process by improving editability and design iteration efficiency. In feature-based modeling, designers create parametric 2D sketches that define profiles, then apply operations such as extrusion, revolution, or sweep to transform these sketches into 3D shapes. Figure 2.2 illustrates examples of these fundamental CAD operations. Through sequential application of such operations, complex shapes can be created while maintaining a complete construction history. This approach preserves the design intent by allowing parameters to be modified at any stage of the sequence, with changes propagating throughout the model. B-Rep and feature-based modeling have become the predominant CAD representation paradigms in modern engineering practice and, consequently, will be the primary focus of this thesis.

While the evolution of CAD systems has been marked by these paradigm shifts in modeling approaches, underlying these developments are fundamental differences in how 3D shapes are mathematically represented. These representations can be broadly categorized as explicit or implicit [45]. Explicit representations directly specify the geometric elements that constitute an object, such as points, edges, or surfaces, making them intuitive for direct manipulation and visualization. In explicit representations, curves and surfaces are often expressed as parametric functions where coordinates are directly computed from parameters. Basic curves such as lines, arcs, and circles can be represented by simple parametric functions, as can planar, spherical, and cylindrical surfaces. For more complex shapes, B-splines and Non-Uniform Rational B-Splines (NURBS) have become standard curve and surface representations in CAD modeling [46], alongside Bézier surfaces [47].

The Boundary Representation (B-Rep) exemplifies an explicit representation approach, containing direct descriptions of boundary surfaces, edges, and vertices of a 3D object. In B-Rep, both the geometry and topology (connectivity information between geometric elements) are explicitly stored, enabling precise control over surface properties. Polygon meshes and point clouds are two other important explicit representations commonly encountered in reverse engineering. Meshes are defined by vertex coordinates along with their connectivity information (typically triangles or quadrilaterals), while point clouds consist solely of surface point coordinates without explicit connectivity. These representations are particularly signifi-

cant because they form the natural output format of modern 3D scanning technologies, which capture visible surfaces as collections of discrete points or triangulated meshes. This creates a fundamental representation gap in reverse engineering: converting these explicit scanning outputs into the parameterized CAD models required for design modification.

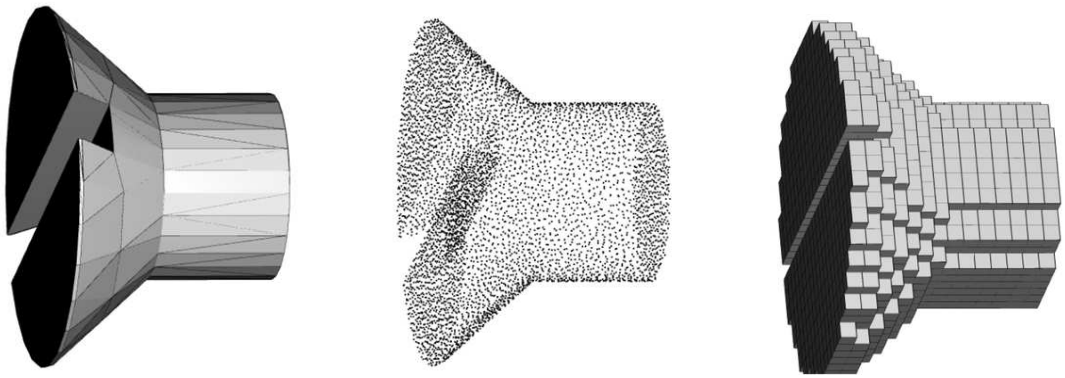
Implicit representations, conversely, define shapes through mathematical functions or equations whose solutions determine the geometry. Typically, an implicit surface is defined as the set of points satisfying an equation $f(x, y, z) = 0$, while an implicit solid comprises all points where $f(x, y, z) \leq 0$. This approach offers significant advantages for certain operations like boolean combinations and inside/outside testing. Examples of implicit representations include algebraic surfaces, such as a plane defined by $ax + by + cz + d = 0$, and Signed Distance Functions (SDFs) in which the function value represents the shortest distance to the nearest surface (negative inside, positive outside). Constructive Solid Geometry (CSG) leverages the implicit representation paradigm by defining complex shapes through boolean operations (union, intersection, subtraction) on simpler implicit primitives, creating a construction tree rather than explicitly storing the resulting geometry.

In the following sections, we formally describe the key representations that play central roles in CAD reverse engineering. We begin with the discrete representations that typically serve as input data, namely polygon meshes, voxel structures, and point clouds, detailing their mathematical foundations and properties. We then provide a formalization of B-Rep and construction history representations, which constitute key CAD representations. Understanding these representations and the gaps between them is essential for appreciating the technical challenges addressed throughout this thesis, particularly in transforming unstructured geometric data into parametric, editable CAD models that preserve design intent.

2.1.2 Discrete Representations

Discrete representations form the foundation of many 3D shape processing pipelines, particularly in reverse engineering where they often serve as input data from scanning technologies or intermediate formats during reconstruction. These representations approximate continuous surfaces through discrete elements, trading mathematical precision for computational

efficiency and flexibility. Figure 2.3 illustrates the three primary discrete representations discussed in this section, polygon meshes, point clouds, and voxel grids, each offering different trade-offs between surface fidelity, topological information, and volumetric completeness.



(a) Mesh (b) Point Cloud (c) Voxel Grid

Figure 2.3: Renderings of common 3D shape representations. (a) A mesh is a collections of connected triangles described by vertices. (b) A point cloud is a set of 3D surface points. (c) A voxel grid is a discretized 3D grid of cells.

Polygon Meshes

Polygon meshes provide a piecewise linear approximation of an object's surface through a collection of vertices, edges, and faces [48]. Unlike the exact mathematical surfaces used in CAD modeling, meshes offer a discrete approximation with accuracy depending on the density and distribution of vertices. This representation is ubiquitous in computer graphics, simulation, and as an intermediate format in reverse engineering pipelines [49]. Mathematically, a polygon mesh $\mathcal{M}$ can be defined as an ordered pair:

$$\mathcal{M} = (\mathcal{V}^M, \mathcal{F}^M)\,, \tag{2.1}$$

where $\mathcal{V}^M = \{v_1^M, v_2^M, \ldots, v_n^M\}$ is a set of vertices, with each vertex $v_i^M \in \mathbb{R}^3$ representing a point in three-dimensional space, and $\mathcal{F}^M = \{f_1^M, f_2^M, \ldots, f_m^M\}$ is a set of faces. Each face

f_j^M represents a polygon and is defined as an ordered list of vertex indices:

$$f_j^M = (i_1, i_2, \ldots, i_k)\,, \tag{2.2}$$

where each $i_l \in \{1, 2, \ldots, n\}$ references a vertex in $\mathcal{V}^M$. The ordering of indices in f_j^M determines the orientation of the face, which is essential for defining the interior and exterior of the object. While faces can theoretically be arbitrary polygons, triangular meshes are the most common, where each face consists of exactly three vertices:

$$f_j^M = (i_1, i_2, i_3)\,. \tag{2.3}$$

Triangular meshes offer computational advantages and are particularly prevalent in real-time applications due to their simplicity and efficiency in rendering algorithms. From the face definition, we can derive the edge set $\mathcal{E}^M$ of the mesh, where each edge e_k^M connects a pair of vertices:

$$\mathcal{E}^M = \{e_k^M = (v_i^M, v_j^M) \mid v_i^M, v_j^M \in \mathcal{V}^M \text{ and } v_i^M, v_j^M \text{ are adjacent in some face } f^M \in \mathcal{F}^M\}\,. \tag{2.4}$$

Beyond the basic geometric and topological structure, meshes often carry additional attributes at their vertices, edges, or faces. This may include vertex normals $\vec{n}_i^M$ providing surface orientation at each vertex v_i^M. In the context of reverse engineering, polygon meshes frequently serve as an intermediate representation, often created by processing point cloud data through surface reconstruction algorithms. The key advantage of meshes over point clouds is their explicit surface connectivity, which provides a watertight boundary representation of the object. However, this approximation introduces limitations. For example, curved surfaces are represented as collections of planar facets, perfectly sharp edges may be rounded, and the accuracy of the representation is limited by mesh resolution. These approximation errors can compound when attempting to derive precise engineering features from meshes. Furthermore, polygon meshes lack the parametric nature and design intent capture of CAD models, presenting a fundamental challenge in reverse engineering.

Point Clouds

Point clouds represent one of the simplest and most direct 3D shape representations, consisting solely of a collection of points in 3D space without explicit connectivity information. This representation forms the foundation of most reverse engineering processes, as it typically constitutes the raw output of 3D scanning technologies such as laser scanners, structured light systems, and photogrammetry [50]. Mathematically, a point cloud $\mathcal{X} \in \mathbb{R}^{n\times 3}$ can be defined as an unordered set of n points:

$$\mathcal{X} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\}, \tag{2.5}$$

where each point $\mathbf{p}_i \in \mathbb{R}^3$ represents a sample on the object's surface in three-dimensional space by its 3D Cartesian coordinates, $\mathbf{p}_i = [x_i, y_i, z_i]$. Unlike polygon meshes, point clouds lack explicit topological information about surface connectivity, making them a more primitive yet flexible representation.

Point clouds often carry additional attributes beyond spatial coordinates, such as normal vectors $\mathbf{n}_i$ that approximate the surface orientation at point $\mathbf{p}_i$. These normals can be estimated from local neighborhoods of points and provide valuable information for subsequent reconstruction steps. The density and distribution of points directly impact the cloud's ability to accurately represent the underlying surface, with denser sampling generally providing better approximation of detailed features but at the cost of increased computational requirements.

The unstructured nature of point clouds presents a fundamental challenge in reverse engineering workflows. While they effectively capture the geometric form of an object, they contain no explicit information about the geometric primitives, design features, or construction history that defined the original model. This lack of structural information creates a significant gap between the raw scanned data and the parametric CAD models required for design modification and manufacturing.

Voxel Representation

Voxel (volumetric pixel) representations offer a different approach to 3D shape modeling by discretizing space into a regular 3D grid of cells [48]. Similar to how pixels form a 2D grid to represent images, voxels extend this concept to the third dimension to represent volumes. Unlike meshes and point clouds that represent only the object's surface, voxel grids capture volumetric information about both the exterior and interior of the object. Mathematically, a voxel grid $\mathcal{G}$ can be defined as a three-dimensional array:

$$\mathcal{G} = \{g_{i,j,k} \mid i \in \{1, 2, \ldots, n_x\}, j \in \{1, 2, \ldots, n_y\}, k \in \{1, 2, \ldots, n_z\}\}, \tag{2.6}$$

where each voxel $g_{i,j,k}$ represents a cubic region of space, and its value can indicate occupancy (binary voxel grids). The parameters n_x, n_y, and n_z define the number of voxels along each coordinate axis, determining the grid's dimensions and resolution. An important consideration in voxel representations is that computational and memory requirements scale cubically with resolution, doubling the resolution along each axis results in an eightfold increase in the number of voxels. This cubic scaling relationship presents significant practical limitations for high-resolution representations of complex objects.

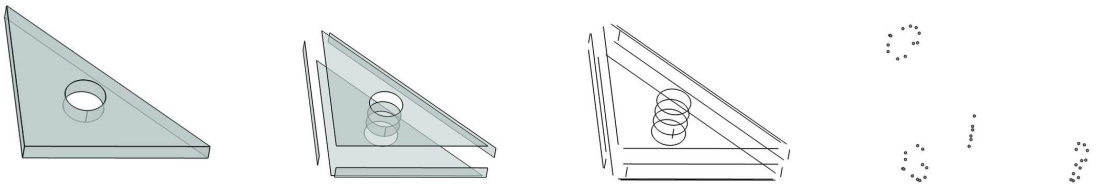
Voxel representations introduce several artifacts and limitations. Sharp features tend to appear "staircased" due to the cubic discretization, and the accuracy of curved surfaces depends directly on grid resolution. Furthermore, exact symmetries present in the original object are often not preserved in the voxelized version, as the discrete grid may not perfectly align with the symmetry axes. This loss of symmetry can be particularly problematic in reverse engineering applications where symmetrical features are both common and functionally important.

Voxel representations can be generated from point clouds or meshes through voxelization algorithms. The volumetric nature of voxel grids makes them well-suited for identifying internal structures and analyzing geometric properties. However, like other discrete representations, voxel grids lack the parametric definition and construction history required for CAD modeling, necessitating additional processing steps to convert them into editable, feature-based models.

2.1.3 CAD-Specific Representations

While discrete representations are essential for capturing geometric data from physical objects, modern CAD systems rely on specialized representations that offer greater precision and design flexibility. The following sections examine the two dominant CAD-specific representations: Boundary Representation (B-Rep) and construction history. Understanding these representations is crucial for reverse engineering tasks that aim to recover not only geometric form but also design intent.

B-Rep Representation



(a) CAD Model (b) Faces (c) Edges (d) Vertices

Figure 2.4: A B-Rep describes a (a) CAD model as a shell composed of connected faces (b). Each face is bounded by a loop of edges (c) and the edges are in turn bounded by vertices (d).

The Boundary Representation (B-Rep) is a data structure that describes the shell of a 3D model. In other words, it represents the boundary between the model and the non-model [51]. At its core, the B-Rep contains two main components: the topology and the geometry. The topology provides the connectivity between the different elements. The shell of the model can be defined by a set of connected faces. Each face is bounded by one or more loops each composed of edges. Each edge is bounded by vertices. Figure 2.4 shows the decomposition of a CAD model in the sets of faces, edges and vertices. The geometry of the model describes the parametrization of the faces into surfaces, edges into curves and vertices into points. Figure 2.5 depicts the relationship between topological and geometrical elements.

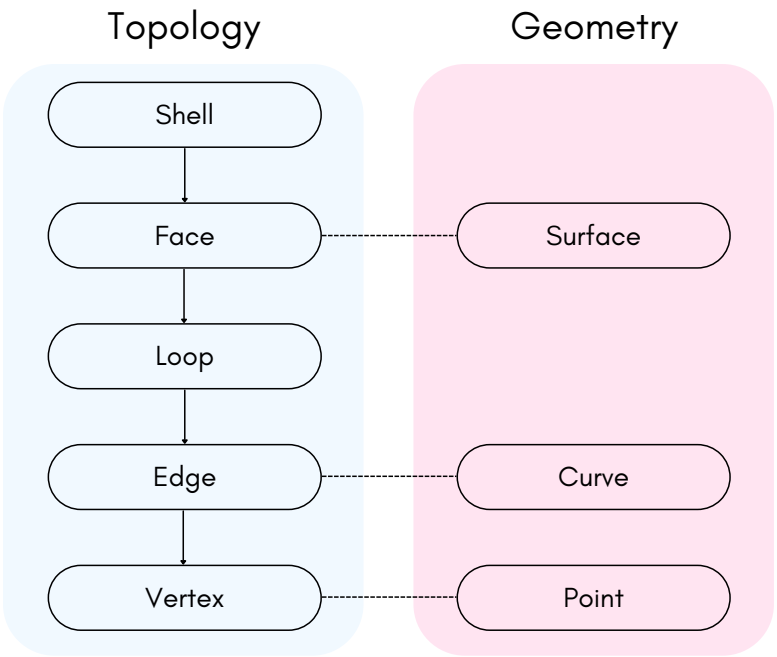


Figure 2.5: The B-Rep structure is defined by two main components. The topology (left) describes the connectivity between different elements and the geometry (right) provides the parametrization of the primitives.

Mathematically, a B-Rep $\mathcal{B}$ can be defined as a tuple of four sets of entities:

$$\begin{aligned}
&\mathcal{B} = (\mathcal{F}, \mathcal{E}, \mathcal{J}, \mathcal{V}), \text{ where:} \\
&\mathcal{F} = f_1, f_2, \ldots, f_{N_f} \text{ is a set of } N_f \text{ faces, ,} \\
&\mathcal{E} = e_1, e_2, \ldots, e_{N_e} \text{ is a set of } N_e \text{ edges,} \\
&\mathcal{J} = j_1, j_2, \ldots, j_{N_j} \text{ is a set of } N_j \text{ co-edges,} \\
&\mathcal{V} = v_1, v_2, \ldots, v_{N_v} \text{ is a set of } N_v \text{ vertices.}
\end{aligned} \tag{2.7}$$

Unlike the planar faces and straight edges in mesh representations, B-Rep entities can be more complex geometric elements. Each face $f_i \in \mathcal{F}$ corresponds to a parametric surface, each edge $e_j \in \mathcal{E}$ to a parametric curve, and each vertex $v_k \in \mathcal{V}$ to a point in three-dimensional space. The co-edge structure $\mathcal{J}$ provides orientation information and facilitates traversal of the model's topology, with each co-edge associated with an edge but having a specific direction and corresponding to a specific face usage. From a data structure perspective the B-Rep can be viewed as a connected list of co-edges.

While the B-Rep is an explicit representation that provides precise geometric definition and editability, it primarily describes the final state of the model rather than how it was created. The B-Rep does not capture the design intent or construction history that led to the final shape, making it insufficient for understanding the engineering rationale behind a model's features or for parametrically modifying the model according to its original design intent.

Construction History and Feature-Based Modeling

Although the B-Rep is an effective representation of a CAD model's final geometry, it lacks information on how the model was constructed. As a result, it can be difficult to edit and modify a CAD model from its B-Rep representation alone. The CAD construction sequence offers an alternative representation that encodes the complete design process and history. In modern CAD software, a designer typically starts by drawing a 2D sketch that is then projected onto a 3D plane. A CAD operation is then used to lift the sketch into a 3D volume. A wide set of CAD operations exists, including extrusion, revolution, chamfer, and fillet. This

sketch-operation process can be repeated several times, with each intermediate volume combined using boolean operations to create the final CAD model. An example of a CAD construction sequence is depicted in Figure 2.6. This representation of a CAD model captures design intent and greatly facilitates editing and parametric modification.

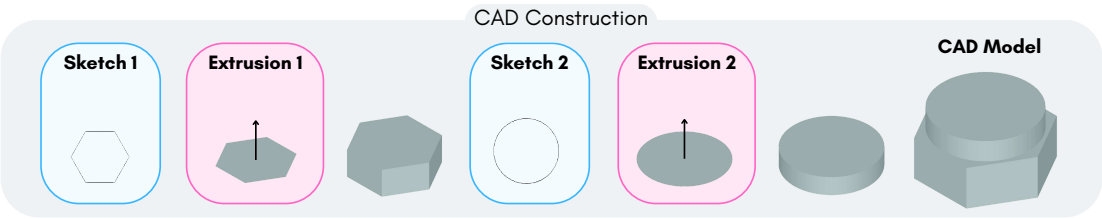


Figure 2.6: In feature-based modeling a CAD model is constructed as a sequence of 2D sketches and operation, such as extrusion.

Mathematically, we can formalize the construction history as follows. Let $\mathcal{C}$ represent the set of all possible CAD sequences. A specific CAD sequence $C \in \mathcal{C}$ can be modeled as a sequence of L design steps: $C = \{c_l\}_{l=1}^{L}$, where each step $c_l = (\mathcal{S}_l, \mathcal{O}_l)$ pairs a sketch $\mathcal{S}_l$ with a modeling operation $\mathcal{O}_l$. The sketch follows a hierarchical structure:

$$\mathcal{S}_l = \{\mathcal{F}_j\}_{j=1}^{N_f^l} \quad \text{(faces)}, \tag{2.8}$$

$$\mathcal{F}_j = \{\rho_k\}_{k=1}^{N_\rho^j} \quad \text{(loops)}, \tag{2.9}$$

$$\rho_k = \{\delta_m\}_{m=1}^{N_\delta^k} \quad \text{(primitives)}. \tag{2.10}$$

This hierarchy mirrors the organization of sketches in CAD systems. At the top level, a sketch $\mathcal{S}_l$ consists of one or more faces $\mathcal{F}_j$, which represent the distinct regions in the 2D sketch that will be affected by the subsequent modeling operation. In CAD terminology, these regions are often referred to as ”profiles”, the closed contours that define the shape to be operated upon. Each face or profile is bounded by one or more loops ρ_k, where the outer loop defines the profile boundary and any inner loops represent holes within that profile. Loops themselves are composed of primitives δ_m, which are the fundamental geometric elements (lines, arcs, and circles) that form the actual drawn content of the sketch. These primitives are connected end-to-end to create closed contours that define the loops. For a profile to be valid

for operations like extrusion or revolution, its loops must form closed, non-self-intersecting boundaries. This structured representation allows CAD systems to precisely interpret which regions of a sketch should be transformed into 3D features through subsequent modeling operations.

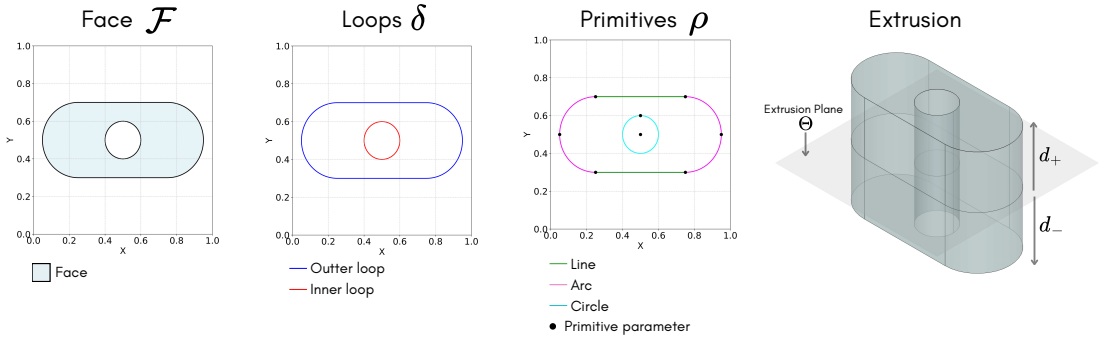


Figure 2.7: Figure depicting the hierarchical structure of sketches $\mathcal{S}$ and the extrusion operation $\mathcal{O}$.

At the lowest level, primitives δ_m can be represented through various parameterizations, typically involving a combination of points, distances, and angles. One possible formulation is a point-based representation where 2D coordinates $\mathbf{p} = [p_x, p_y]^\top$ define the key geometric elements:

$$\delta_m = \begin{cases} [\mathbf{p}_{\text{start}}, \mathbf{p}_{\text{end}}] & \text{for lines,} \\ [\mathbf{p}_{\text{start}}, \mathbf{p}_{\text{mid}}, \mathbf{p}_{\text{end}}] & \text{for arcs,} \\ [\mathbf{p}_{\text{center}}, \mathbf{p}_{\text{radius}}] & \text{for circles.} \end{cases} \tag{2.11}$$

where $\mathbf{p}_{\text{start}}$ and $\mathbf{p}_{\text{end}}$ represent the endpoints of a line or arc, $\mathbf{p}_{\text{mid}}$ is a point on the arc between its endpoints, $\mathbf{p}_{\text{center}}$ denotes the center point of a circle, and $\mathbf{p}_{\text{radius}}$ is a point on the circumference of the circle.

CAD systems typically implement multiple internal parameterizations to represent primitive geometry, optimized for different computational operations and user interaction paradigms. For instance, lines may be encoded using implicit equations ($ax + by + c = 0$), parametric forms, or vector representations; circles through center-radius pairs or implicit equations ($x^2 + y^2 = r^2$); and arcs via center, radius, and angular bounds or through rational Bézier

curve approximations. These representations are often interconverted within the software depending on the specific operation being performed, such as boolean operations, or display rendering. While these specialized representations offer computational advantages in specific contexts, the point-based formulation presented here provides a unified mathematical framework that facilitates consistent analysis across primitive types while preserving all geometric degrees of freedom necessary for complete reconstruction.

It is worth noting that modern CAD systems support a much wider variety of 2D primitives, including B-splines, NURBS curves, ellipses, parabolas, and hyperbolas, which enable the representation of more complex and freeform shapes. However, for the purposes of this thesis, we focus on the three of the most fundamental primitive types, namely lines, arcs, and circles, as they constitute the vast majority of elements in mechanical CAD models and provide sufficient expressivity for most engineering applications [52].

Each modeling operation $\mathcal{O}_l$ consists of common parameters Θ_l and operation-specific parameters Φ_l:

$$\mathcal{O}_l = \{\Theta_l, \Phi_l\}\,, \tag{2.12}$$

$$\Theta_l = \{\theta, \phi, \gamma, \tau_x, \tau_y, \tau_z, \sigma, \beta\}\,, \tag{2.13}$$

where (θ, ϕ, γ) defines the sketch plane orientation, (τ_x, τ_y, τ_z) specifies translation, σ controls scaling, and β denotes the boolean operation type. The operation-specific parameters Φ_l depend on the type of operation. In this work, we primarily consider the extrusion operation, where $\Phi_l = \{d_+, d_-\}$, with d_+ and d_- representing extrusion distances in the direction of the sketch-plane normal and the opposite direction, respectively. Figure 2.7 depicts the hierarchical structure of a CAD desing step $c_l = (\mathcal{S}_l, \mathcal{O}_l)$.

This sequential, parametric formulation of the construction history represents a powerful hybrid of explicit and implicit representation approaches. The sequence itself explicitly captures the ordered steps of model creation, while each operation implicitly defines volumes through generation rules rather than direct boundary specification. Unlike the B-Rep, which only represents the final geometric state, the construction sequence provides a complete

chronological description of how each feature was created, modified, and related to other features in the model. This temporal dimension is crucial for capturing design intent. Importantly, while a construction history can be evaluated to generate the corresponding B-Rep through sequential application of operations, the inverse process, deriving the original construction sequence from a B-Rep, is fundamentally ambiguous and often impossible without additional information. This one-way relationship underscores a central challenge in reverse engineering: multiple different construction sequences can produce geometrically identical B-Reps, yet each sequence may imply different design intent and modification capabilities. The hybrid nature of this representation makes it particularly valuable for reverse engineering tasks, as it enables not just geometric reconstruction but also recovery of the underlying design methodology and feature relationships that define the model elements that are essential for meaningful model modification and reuse.

2.2 Deep Learning Methods for CAD Reverse Engineering

Recent advances in deep learning have created new opportunities to address long-standing challenges in CAD reverse engineering. This section explores three key neural network architectures that form the foundation of modern approaches to automated CAD reconstruction: point cloud encoders that transform raw scan data into structured features, graph neural networks that capture topological relationships in 3D structures, and transformer architectures that are well suited for sequential representations such as CAD construction histories.

2.2.1 Point Cloud Encoder

A point cloud encoder is a crucial component for processing raw 3D scan data in CAD reverse engineering pipelines. Its primary function is to transform the unordered, potentially noisy set of 3D points into a structured, latent representation that captures the geometric features of the scanned object. Given a point cloud $\mathcal{X} \in \mathbb{R}^{n_p \times 3}$ composed of n_p points, a point cloud encoder can be modeled as a function Φ_p, such that:

$$\begin{aligned} \Phi_p : \mathbb{R}^{n_p \times 3} &\rightarrow \mathbb{R}^{n_f \times d_f} \ , \\ \Phi_p(\mathcal{X}) &= \mathbf{F}_p \ , \end{aligned} \tag{2.14}$$

where $\mathbf{F}_p \in \mathbb{R}^{n_f \times d_f}$ is the set of n_f feature vectors each of dimension d_f. The relationship between the number of input points n_p and output feature vectors n_f depends on the specific encoder architecture. In point-wise encoders, we typically have $n_f = n_p$, where each input point is transformed into a corresponding feature vector, preserving spatial correspondence. In global feature encoders, $n_f = 1$, producing a single feature vector that represents the entire point cloud. This resulting latent representation then serves as input to subsequent CAD specific decoder networks.

In the following section, we review two point encoder network architectures that are used throughout the thesis.

PointNet++ Encoder

PointNet++ [53] extends the original PointNet architecture [33] by introducing a hierarchical feature learning framework that captures local structures at different scales. Unlike its predecessor, which processed all points independently before a global pooling operation, PointNet++ applies PointNet recursively on nested partitions of the input point set through a hierarchical process of set abstraction layers.

Formally, given an input point cloud $\mathcal{X} \in \mathbb{R}^{n_p \times 3}$, which we denote as $\mathbf{F}_p^0 = \mathcal{X}$ to represent the initial feature level, PointNet++ processes the data through multiple set abstraction levels. Each set abstraction layer consists of three key operations: (1) sampling points to identify centroids, (2) grouping neighboring points around these centroids to form local regions, and (3) applying point-wise feature extraction followed by pooling to generate new features. The superscript $l \in \{0, 1, ..., L\}$ throughout our notation denotes the abstraction level in this hierarchical structure.

At the first abstraction level, PointNet++ applies a sampling and grouping strategy to

create a set of local regions:

$$\{\mathcal{R}_1, \mathcal{R}_2, ..., \mathcal{R}_{n_c}\} = \text{Sampling}(\mathbf{F}_p^0)\,, \tag{2.15}$$

where each region $\mathcal{R}_i \subset \mathbf{F}_p^0$ contains a centroid point and its neighboring points within a radius (ball query), thus each region covers a fixed spatial scale.

For each local region $\mathcal{R}_i$, a mini-PointNet [33] ϕ_{pnet}, consisting of multi-layer perceptron (MLP) networks and a max-pooling operation, extracts local features:

$$\mathbf{f}_i^1 = \phi_{pnet}(\mathcal{R}_i)\,. \tag{2.16}$$

These local features are then aggregated to form a new set of points with feature vectors:

$$\mathbf{F}_p^1 = \{(\mathbf{c}_1, \mathbf{f}_1^1), (\mathbf{c}_2, \mathbf{f}_2^1), \ldots, (\mathbf{c}_{n_c}, \mathbf{f}_{n_c}^1)\}\,, \tag{2.17}$$

where $\mathbf{c}_i$ is the coordinates of the centroid of region $\mathcal{R}_i$.

The complete set abstraction operation can be summarized as:

$$\mathbf{F}_p^{l+1} = \text{SetAbstraction}(\mathbf{F}_p^l)\,. \tag{2.18}$$

This process is repeated through multiple set abstraction levels, with each level producing increasingly abstract feature representations while progressively reducing spatial resolution. The final output $\mathbf{F}_p^L$ corresponds to the feature representation after L abstraction levels. The resulting multi-scale features can then be propagated back to the original points using feature interpolation techniques, ultimately producing a feature matrix $\mathbf{F}_p \in \mathbb{R}^{n_f \times d_f}$. This hierarchical approach makes PointNet++ particularly effective for CAD reverse engineering tasks, as it can simultaneously capture global shape characteristics necessary for classifying geometric primitives while preserving the fine-grained details required for precise boundary detection and feature recognition.

RandLA-Net Encoder

Building upon the hierarchical design principles of PointNet++, RandLA-Net [54] introduces an alternative neural architecture specifically optimized for processing large-scale point clouds. Unlike PointNet++ which uses simple max pooling to aggregate local features, RandLA-Net employs a more sophisticated Local Feature Aggregation (LFA) module that explicitly preserves local geometric structures. Each LFA module consists of two key components: (1) a local spatial encoding unit that captures relative positions between neighboring points, and (2) an attentive pooling mechanism that learns to weigh important features based on their local context.

Given a point cloud $\mathcal{X} \in \mathbb{R}^{n_p \times 3}$, which we denote as $\mathbf{F}_p^0 = \mathcal{X}$ to represent the initial feature level, the RandLA-Net encoder begins by finding the K nearest neighbors for each feature $\mathbf{f}_i^0$. Given a $\mathbf{f}_i^0$ and its K neighbors $\{\mathbf{f}_{i,1}^0, \ldots, \mathbf{f}_{i,k}^0, \ldots, \mathbf{f}_{i,K}^0\}$, a set of local spatial features $\hat{\mathbf{F}}_i^0 = \{\hat{\mathbf{f}}_{i,1}^0, \ldots \hat{\mathbf{f}}_{i,k}^0, \ldots, \hat{\mathbf{f}}_{i,K}^0\}$ is obtained by applying a MLP to encode the relative spatial relationship between $\mathbf{f}_i^0$ and each neighbor $\mathbf{f}_{i,k}^0$. This encoding explicitly incorporates the raw coordinates, relative displacement vectors, and Euclidean distances, allowing the network to better capture local geometric structures.

RandLA-Net then incorporates an attention mechanism to adaptively weight the contribution of each neighboring point:

$$\mathbf{s}_{i,k} = \phi_a(\hat{\mathbf{f}}_{i,k}^0)\,, \tag{2.19}$$

$$\mathbf{f}_i^1 = \sum_{k=1}^{K}, (\hat{\mathbf{f}}_{i,k}^0 \odot \mathbf{s}_{i,k})\,, \tag{2.20}$$

where $\mathbf{f}_i^1$ represents the feature vector corresponding to the point $\mathbf{p}_i$ after 1 LFA layer, ϕ_a is an attention function implemented as a shared MLP with softmax, and $\odot$ denotes element-wise multiplication.

For the l-th encoding layer, the feature transformation can be expressed as:

$$\mathbf{F}_p^l = \text{LFA}(\mathbf{F}_p^{l-1})\,. \tag{2.21}$$

Like PointNet++, RandLA-Net employs multiple encoding layers with progressive downsam-

pling. After L encoding layers, the final output feature matrix $\mathbf{F}_p^L \in \mathbb{R}^{n_f \times d_f}$ captures rich local and global geometric information while being able to process point clouds orders of magnitude larger than what is possible with PointNet++, making it particularly well-suited for CAD reverse engineering tasks where dense point clouds are required to capture fine-grained details.

2.2.2 Graph Neural Network Architecture

A graph $\mathcal{G}$ is a mathematical structure consisting of a set of vertices (or nodes) $\mathcal{V}^G$ and a set of edges $\mathcal{E}^G$ that connect pairs of vertices. Formally, a graph can be defined as:

$$\mathcal{G} = (\mathcal{V}^G, \mathcal{E}^G)\,, \tag{2.22}$$

where $\mathcal{V}^G = \{v_1, v_2, \ldots, v_n\}$ represents the set of n vertices, and $\mathcal{E}^G \subseteq \mathcal{V}^G \times \mathcal{V}^G$ represents the set of edges, where each edge $e_{ij} = (v_i, v_j) \in \mathcal{E}^G$ connects vertices v_i and v_j. In the context of machine learning, graphs are often enriched with additional information. Vertices and edges can be associated with feature vectors, resulting in an attributed graph. For a vertex v_i, its feature vector is denoted as $\mathbf{f}_i \in \mathbb{R}^{d_v}$, where d_v is the dimension of vertex features. Similarly, for an edge e_{ij}, its feature vector is denoted as $\mathbf{e}_{ij} \in \mathbb{R}^{d_e}$, where d_e is the dimension of edge features. Additionally, the connectivity pattern of a graph can be represented by an adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, where:

$$\mathbf{A}_{ij} = \begin{cases} 1, & \text{if } (v_i, v_j) \in \mathcal{E}^G \\ 0, & \text{otherwise}\,. \end{cases} \tag{2.23}$$

For weighted graphs, the adjacency matrix entries can take values other than 0 and 1 to represent the strength or weight of the connections between vertices.

The graph representation is particularly well-suited for the CAD reverse engineering problem. For example, a B-Rep can be represented as a graph structure where the graph vertices represent B-Rep faces and the graph edges the B-Rep edges. Point clouds can also

Graph	Vertices	Vertex Feature, $\mathbf{f}_i$	Edges	Edge Feature, $\mathbf{e}_{i,j}$	Adjacency, $\mathbf{A}$	Related Works
Mesh	Mesh vertices	Vertex coordinates and normals	Mesh edges	Edge lengths and dihedral angles	Mesh connectivity	[55, 56]
Voxel Grid	Voxels	Voxel coordinates and size	-	Spatial relationship between voxels	K-Nearest Neighbors	[57, 58]
Point Cloud	Points	Point coordinates, normals	-	Spatial relationship between points	K-Nearest Neighbors	[59, 60, 61, 62]
B-Rep	B-Rep faces	Face type, area	B-Rep edges	Edge type, length	B-Rep Topology	[13]
B-Rep	B-Rep faces	2D UV grid point coordinates and normals	B-Rep edges	1D UV grid point coordinates and normals	B-Rep Topology	[14]

Table 2.1: Examples of how Graph Neural Networks (GNN) can be constructed in the context of 3D reverse engineering. Graph structure can model both discrete 3D shapes (mesh, voxel, point cloud) and CAD models (B-Rep).

be represented as a graph where the vertices correspond to the points and the adjacency matrix is obtained from K-Nearest Neighbor search. Table 2.1 shows different methods to model 3D shapes in graph structures.

We now provide a general mathematical formulation of Graph Neural Networks (GNNs) as message-passing neural networks. A GNN layer consists of three steps: (1) a message computation, (2) an aggregation and (3) an update of the nodes and edges features [63, 60]. This process allows for the passing of information between connected nodes. Let $\mathcal{G}^0 = (\mathcal{V}^0, \mathcal{E}^0)$ be the initial graph with node features $\mathbf{f}_i^0$ and edge features $\mathbf{e}_{i,j}^0$ as previously defined. For each node $v_i \in \mathcal{V}^0$ a message $\mathbf{m}_i^1$ for the first GNN layer is computed using:

$$\mathbf{m}_i^1 = \square_{v_j \in N(v_i)} \phi_m(\mathbf{f}_i^0, \mathbf{f}_j^0, \mathbf{e}_{i,j}) \,, \tag{2.24}$$

where $N(v_i)$ is the set of neighbor nodes to v_i as defined by the adjacency matrix $\mathbf{A}$, $\square$ is an aggregation function such as sum, max pooling or an attention-based function, and ϕ_m is a function with learnable parameters, typically an MLP. Then, the vertex features can be updated using:

$$\mathbf{f}_i^1 = \phi_u(\mathbf{f}_i^0, \mathbf{m}_i^1)\,, \tag{2.25}$$

where ϕ_u is an update function with learnable parameters, typically an MLP. Similarly, edge features can be updated following:

$$\mathbf{e}_{i,j}^1 = \phi_e(\mathbf{f}_i^1, \mathbf{f}_j^1)\,, \tag{2.26}$$

where ϕ_e is a function with learnable parameters, typically an MLP. As a result, a new graph $\mathcal{G}^1 = (\mathcal{V}^1, \mathcal{E}^1)$ can be constructed. Note that it is possible to compute a new adjacency matrix at each layer using the update vertex features, hence creating a dynamic graph [60]. This message passing mechanism can be applied iteratively, so that after l-th GNN layer the resulting graph, $\mathcal{G}^l$, can be expressed as:

$$\mathcal{G}^l = \text{GNN}(\mathcal{G}^{l-1})\,. \tag{2.27}$$

Graph Neural Network offer a modeling approach that is particularly well-suited for CAD reverse engineering tasks as they allow for the encoding of a variety of unstructured and structured shape representations such as point clouds and B-Reps.

2.2.3 Transformer Architecture

The transformer architecture, first introduced by Vaswani et al. [64] for natural language processing tasks, has emerged as a powerful paradigm for modeling sequential data with long-range dependencies. Transformers employ an attention mechanism that allows modeling of relationships between all elements in a sequence. In the context of CAD reverse engineering, transformers are particularly well-suited for processing construction histories that are part of the feature-based representation. In this representation, a CAD object is modeled as a sequence of parametric operations (sketch and operation). The core components of the transformer architecture include:

- **Tokenization**: Converting input data elements (whether point cloud features, or para-

metric CAD sequence) into fixed-dimensional vector representations;

- **Positional Encoding**: Incorporating sequential information;
- **Self-Attention Mechanism**: Computing attention weights to capture relationships between operations within the same modality;
- **Cross-Attention Mechanism**: Enabling information flow between different modalities, such as point clouds and CAD sequences, by allowing tokens from one domain to attend tokens from another domain;
- **Encoder-Decoder Structure**: Facilitating the translation from one domain (e.g., point clouds) to another (e.g., CAD sequences).

The cross-attention mechanism is particularly important for CAD reverse engineering tasks that involve reconstructing a CAD model from a point cloud, cross-attention enables each element in the generated CAD sequence to selectively focus on relevant regions of the input point cloud.

In the following subsections, we will examine each component of the transformer architecture and its application to CAD reverse engineering, starting with the tokenization process.

Tokenization

The transformer architecture processes information through sequences of tokens. In the context of CAD reverse engineering, we first define a set of input elements $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{N_s}\}$ consisting of N_s elements. Each input element $\mathbf{x}_i$ can have varying dimensions and types depending on the source modality. For point clouds, $\mathbf{x}_i$ might represent point coordinates or feature descriptors obtained from a point cloud encoder. For CAD sequences, $\mathbf{x}_i$ could represent sketch elements (*e.g.*, lines, arcs, circles), modeling operations (*e.g.*, extrusion), or their associated parameters (*e.g.*, coordinates, distances, angles).

Tokenization refers to the process of projecting these heterogeneous input elements into a fixed-dimensional embedding space. Formally, we define a token sequence $\mathcal{S} =$

$\{\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_{N_s}\}$ where each token $\mathbf{t}_i \in \mathbb{R}^{d_t}$ is a d_t-dimensional vector in a continuous embedding space, with all tokens sharing the same dimensionality d_t.

The token embedding process can be formalized as a mapping function ϕ_e that projects each input element $\mathbf{x}_i$ to the token space:

$$\mathbf{t}_i = \phi_e(\mathbf{x}_i) \in \mathbb{R}^{d_t}\,. \tag{2.28}$$

For point cloud features, ϕ_e may be implemented as a multi-layer perceptron that transforms high-dimensional feature vectors into the token dimension. For categorical data such as operation types, ϕ_e can be a learnable embedding lookup table. For continuous parameters, ϕ_e might involve a quantization step followed by embedding, or direct encoding through a neural network.

Positional Encoding

Unlike recurrent neural networks, transformers process all tokens in parallel, and hence they lack a mechanism to capture the sequential ordering of input elements. Positional encoding addresses this limitation by incorporating position information directly into the token embeddings. In the context of CAD reverse engineering, this is crucial as the sequence of operations in a CAD construction history can carry significant semantic meaning.

Positional encoding can be implemented through various approaches. The original transformer architecture [64] employed a deterministic sinusoidal encoding. Alternatively, learned positional embeddings can be employed, where each position index is mapped to a learnable vector. This approach can be more flexible and potentially capture more complex positional relationships, but may be limited to the maximum sequence length encountered during training. For CAD sequences where the number of operations is typically bounded, learned positional embeddings are often used.

Regardless of the specific encoding method, the positional information is incorporated by adding it to the token embeddings, resulting in the final sequence of feature vectors that serve as input to the transformer:

$$\mathbf{f}_i = \mathbf{t}_i + \mathbf{p}_i \,, \tag{2.29}$$

where $\mathbf{f}_i \in \mathbb{R}^{d_t}$ is the final feature vector for position i, $\mathbf{t}_i \in \mathbb{R}^{d_t}$ is the token embedding, and $\mathbf{p}_i \in \mathbb{R}^{d_t}$ is the positional encoding vector for position i.
The resulting sequence $\mathbf{F} \in \mathbb{R}^{N_s \times d_t}$ is then defined as:

$$\mathbf{F} = \{\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_{N_s}\} \,. \tag{2.30}$$

This sequence preserves both the content information from the token embeddings and the sequential information from the positional encodings, enabling the transformer to effectively process ordered data such as CAD construction sequences and spatially structured point sets.

Single Query Attention

At the core of transformer architectures lies the attention mechanism, which allows the model to focus on relevant parts (*e.g.*, previous tokens or whole sequence) of the input sequence when predicting an output token. Single query attention computes a weighted sum of value vectors, where the weights are determined by the compatibility between a query vector and a set of key vectors [65].

Given a single sequence feature vector, $\mathbf{f}$ and T context feature vectors $\mathbf{f}_t$, the attention is computed as follows:

$$
\begin{aligned}
&\mathbf{q} = \mathbf{W}_q \mathbf{f}\,, \\
&\forall t : \mathbf{k}_t = \mathbf{W}_k \mathbf{f}_t\,, \\
&\forall t : \mathbf{v}_t = \mathbf{W}_v \mathbf{f}_t\,, \\
&\forall t : \alpha_t = \frac{\exp\left(\frac{\mathbf{q}^T \cdot \mathbf{k}_t}{\sqrt{d_{attn}}}\right)}{\sum_u \exp\left(\frac{\mathbf{q}^T \cdot \mathbf{k}_u}{\sqrt{d_{attn}}}\right)}\,, \\
&\tilde{\mathbf{v}} = \sum_{t=1}^{T} \alpha_t \mathbf{v}_t\,.
\end{aligned}
\tag{2.31}
$$

where $\mathbf{W}_q \in \mathbb{R}^{d_{attn} \times d_t}$, $\mathbf{W}_k \in \mathbb{R}^{d_{attn} \times d_t}$, and $\mathbf{W}_v \in \mathbb{R}^{d_{out} \times d_t}$ are learnable parameter matrices that transform the feature vectors into query, key, and value spaces, respectively. $\tilde{\mathbf{v}} \in \mathbb{R}^{d_{out}}$ is the output vector containing information aggregated from the context that is relevant to the query.

In the context of CAD reverse engineering, single query attention enables each element in a sequence to gather information from all other elements. For instance, in a CAD construction sequence, the attention mechanism allows each operation to focus on previously defined geometric elements that are relevant to its definition. Similarly, when processing point cloud data, attention enables each point to aggregate information from geometrically or semantically related points, facilitating the recognition of higher-level structures and patterns such as planes, cylinders, or design features.

Self-Attention and Cross-Attention

Building upon the single query attention mechanism, transformer architectures employ two primary types of attention: self-attention (SA) and cross-attention (CA). These mechanisms differ in how they source their query, key, and value vectors.

In self-attention, all query, key, and value vectors are derived from the same sequence of feature vectors. Given a sequence $\mathbf{F} = \{\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_{N_s}\}$, each feature vector $\mathbf{f}_i$ generates its corresponding query, key, and value vectors using the same projection matrices described previously. This allows each position to attend to all positions within the same sequence,

enabling the model to capture intra-sequence dependencies.

Cross-attention, in contrast, computes query vectors from one sequence and key-value pairs from another. Formally, given a query sequence $\mathbf{F}^q = \{\mathbf{f}_1^q, \mathbf{f}_2^q, \ldots, \mathbf{f}_{N_q}^q\}$ and a context sequence $\mathbf{F}^c = \{\mathbf{f}_1^c, \mathbf{f}_2^c, \ldots, \mathbf{f}_{N_c}^c\}$, cross-attention allows each position in the query sequence to attend to all positions in the context sequence. It is important to note that the query and context sequences may have different lengths (N_q can be different from N_c) and their feature vectors may have different dimensions ($\mathbf{f}_i^q \in \mathbb{R}^{d_q}$ and $\mathbf{f}_j^c \in \mathbb{R}^{d_c}$ where potentially $d_q \neq d_c$). This flexibility is particularly valuable when connecting different modalities or representation spaces. The query, key and value vector calculations from eq. 2.31 are now:

$$\begin{aligned} \mathbf{q}_i &= \mathbf{W}_q \mathbf{f}_i^q \,, \\ \mathbf{k}_j &= \mathbf{W}_k \mathbf{f}_j^c \,, \\ \mathbf{v}_j &= \mathbf{W}_v \mathbf{f}_j^c \,. \end{aligned} \tag{2.32}$$

To control information flow and enable efficient parallel processing, attention mechanisms often employ masks. In autoregressive (unidirectional) settings, a causal mask ensures that each position can only attend to itself and previous positions, preventing information leakage from future tokens:

$$\alpha_{ij} = \begin{cases} \dfrac{\exp\left(\frac{\mathbf{q}_i^T \cdot \mathbf{k}_j}{\sqrt{d_{attn}}}\right)}{\sum_{l \leq i} \exp\left(\frac{\mathbf{q}_i^T \cdot \mathbf{k}_l}{\sqrt{d_{attn}}}\right)} & \text{if } j \leq i \,, \\ 0 & \text{otherwise} \,. \end{cases} \tag{2.33}$$

This causal masking is particularly important when predicting the next token in a sequence, as it enforces the constraint that prediction of token at position i can only depend on tokens at positions less than or equal to i.

In bidirectional (non-autoregressive) settings, no such mask is applied, allowing each position to attend to all other positions regardless of their relative ordering. This bidirectional attention is especially useful for tasks requiring full contextual understanding, such as feature extraction from point clouds or analysis of complete CAD sequences.

In the context of CAD reverse engineering, cross-attention is particularly valuable when

translating between different representations. For example, when reconstructing a CAD sequence from a point cloud, queries derived from partial CAD constructions can attend to keys and values derived from point features, enabling the model to focus on relevant geometric regions when predicting the next construction operation.

Multi-Head Attention

The attention mechanisms described thus far represent the operation of a single attention head. In practice, transformer architectures employ multi-head attention, which runs multiple attention operations in parallel and then combines their results. This approach allows the model to jointly attend to information from different representation subspaces.

In multi-head attention, the model creates h different sets of query, key, and value projections each with their own sets of learnable projection matrices. The dimension of the query, key and value vectors for each head is typically smaller than the original feature dimensions. The outputs from all heads are concatenated and then linearly transformed to produce the final result.

This parallel computation of attention over different projection spaces offers several advantages. First, it enables the model to capture different types of relationships simultaneously as some heads may focus on local geometric features while others capture global structural patterns. Second, by using lower-dimensional projections for each head, multi-head attention maintains computational efficiency while increasing model expressivity. Finally, the different attention patterns across heads provide a form of ensemble learning, making the model more robust.

Transformer Encoder and Decoder

The attention mechanisms described in previous sections serve as fundamental building blocks for the complete transformer architecture, which consists of an encoder and a decoder, each composed of multiple identical layers stacked on top of each other, as illustrated in Figure 2.8. This multi-layer approach allows the model to build increasingly abstract

representations as information flows through the network.

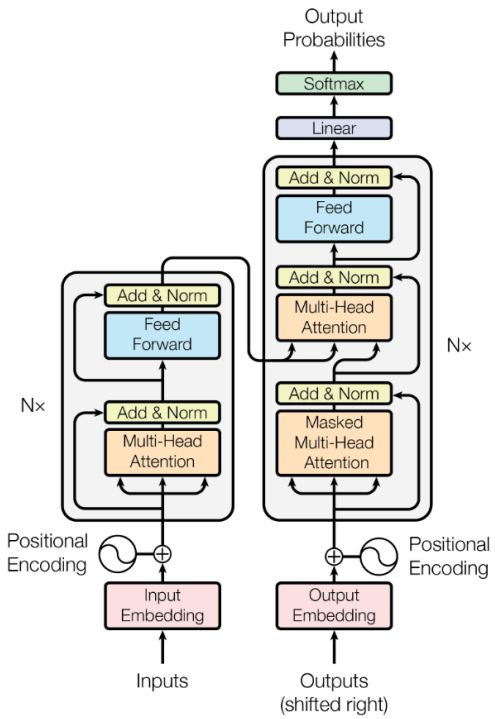


Figure 2.8: Transformer architecture [64]. Left: Transformer encoder. Right: Transformer decoder relying on the cross-attention mechanism.

The encoder transforms the input sequence into a continuous representation that captures its underlying structure. Each encoder layer contains two main components: a multi-head self-attention mechanism and a position-wise feed-forward network.

The decoder generates the output sequence element by element, using both the encoder's output and previously generated elements. Each decoder layer contains three main components: a masked multi-head self-attention mechanism, a multi-head cross-attention mechanism, and a position-wise feed-forward network.

Critically, both encoder and decoder employ residual connections around each sub-layer, followed by layer normalization. Layer normalization explicitly controls the mean and variance of individual neural network activations, stabilizing the learning process.

After processing through N layers of encoder and decoder stacks, the final decoder

output is projected to the vocabulary size and passed through a softmax function to produce a probability distribution over possible output tokens.

This probabilistic output enables various decoding strategies. During training, the model is typically trained to maximize the likelihood of the correct token. During inference, strategies such as greedy decoding (Top-1 sampling), beam search (maintaining multiple high-probability sequence candidates), or sampling-based approaches (drawing from the output distribution) can be employed to generate sequences.

In the context of CAD reverse engineering, the transformer encoder can process point cloud features or mesh representations, while the decoder can generate the sequence of CAD operations needed to reconstruct the 3D model. The probabilistic nature of the output can allow the model to propose multiple plausible CAD constructions for a given input.

2.3 Conclusion

This chapter has provided a comprehensive overview of the fundamental shape representations and deep learning methods that form the foundation of CAD reverse engineering. We began by examining the evolution of solid modeling paradigms, from early 2D projections to modern feature-based, history-based CAD. We then described the mathematical formulations of various 3D shape representations, contrasting the discrete representations commonly produced by scanning technologies (point clouds, meshes, and voxel grids) with the parametric CAD-specific representations (B-Rep and construction history) that capture design intent.

The inherent gap between these representation spaces—from unstructured geometric data to parametric, editable models—defines the central challenge of CAD reverse engineering. To bridge this gap, we discussed three key deep learning architectures: point cloud encoders, graph neural networks, and transformers, each offering complementary approaches to processing and transforming 3D data.

The remainder of this thesis will build upon these foundations to develop novel methods that bridge the representation gap in CAD reverse engineering.

Chapter 3

CADOps-Net: Jointly Learning CAD Operation Types and Steps from Boundary-Representations

In this chapter, we introduce our contribution named *CADOps-Net*, a neural network architecture for jointly learning CAD operation types and steps from Boundary Representations (B-Rep). This work addresses a sub-problem of the CAD reverse engineering task. In particular, we consider the B-Rep as an intermediate representation between the 3D scans and the feature-based representation. More specifically, we aim at recovering elements of the CAD construction history (CAD operation types and steps) by segmenting B-Rep faces, directly addressing the challenge of recovering design intent highlighted in Section 1.2.3. This is achieved by expanding the BRepNet [13] network to obtain face features and to predict the segmentation of the B-Rep faces according to the CAD operation type (*e.g.*, extrusion, revolution) that was used for the construction of this face. These labels are jointly learned with the grouping of the faces that were constructed at the same construction step through the CAD history sequence. Experimental results demonstrate that this joint learning strategy leads to an improvement in the accuracy of the prediction over existing methods. Moreover, to address the lack of realistic, complex, annotated dataset identified as the “CAD Data Chal-

lenge" in Section 1.2.4, we introduce the *CC3D-Ops* dataset that contains over 37 k B-Reps with corresponding per-face operation type and step annotations. This dataset contribution provides more complex models that better reflect real-world industrial designs compared to existing datasets. Finally, we show that the predictions from *CADOps-Net* can be used to deduce the sketches that were used to create the CAD model. As a result, this work is a first step towards addressing the complete CAD reverse engineering problem and lays the foundation for the more comprehensive approaches developed in subsequent chapters.

3.1 Introduction

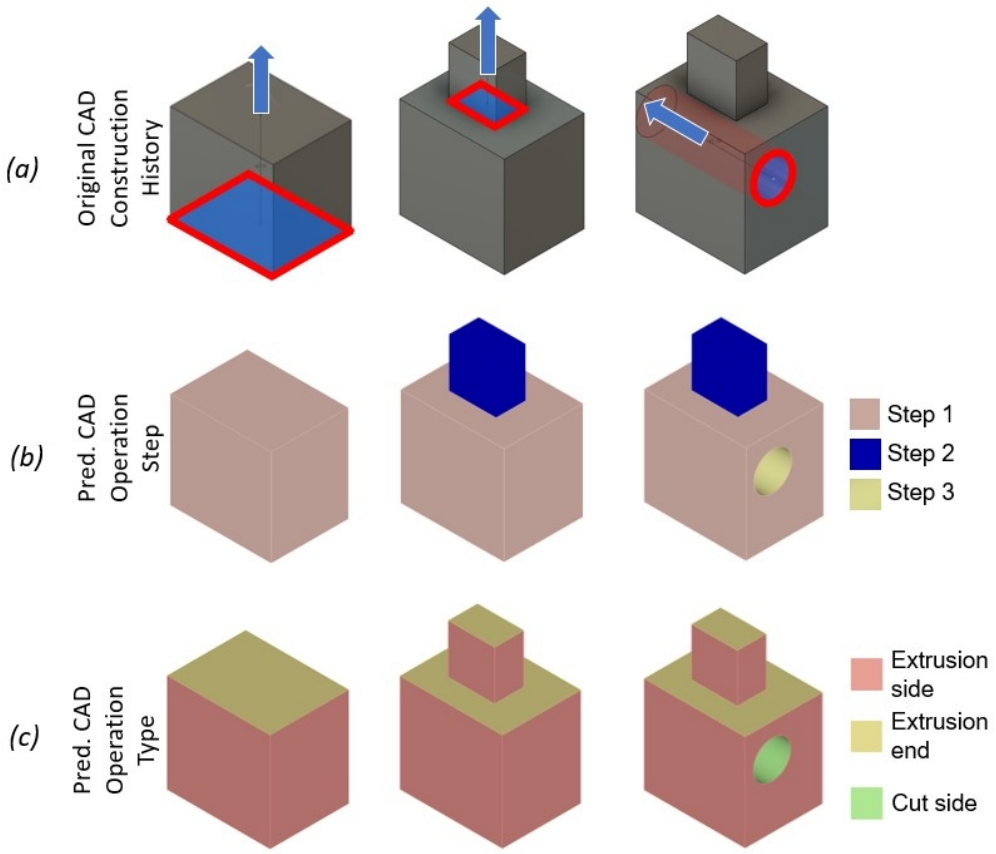


Figure 3.1: B-Rep segmentation into CAD operations types and steps.

As established in Chapter 1, Computer-Aided Design (CAD) has become the cornerstone of modern product development workflows across industries [66, 67, 31]. The power of CAD systems stems largely from their parametric nature, which enables engineers and designers to iterate efficiently by adjusting parameters of existing models to meet new requirements—whether customizing dental prostheses for individual patients [68] or adapting mechanical components for different applications [69]. However, this parametric editability hinges critically on access to the model's design history, which captures the sequence of operations used to create the final shape. In practice, this history is frequently unavailable, either because it was never recorded for generic 3D shapes [70] or because it was lost during translation between different CAD systems [71, 13]. This limitation has spurred significant research efforts aimed at inferring design history from geometric data alone [13, 72, 73, 70, 74, 26]—a process known as *3D reverse engineering*, which forms the central focus of this thesis.

Prior works attempted to recover the CAD design history, considering *Constructive Solid Geometry* (CSG) based models [70, 73] for simplicity. In CSG, a CAD model is represented by a set of rigidly transformed solid primitives (*e.g.* cube, sphere, cylinder) and combined using Boolean operations such as union, intersection, and difference [75]. However, modern CAD workflows use *feature-based* modeling, in which solids are created by iteratively adding features such as holes, slots, or bosses [74, 76]. These high-level features are sequentially created through drawing *sketches* and applying *CAD operations* such as '*extrusion*', '*revolution*', etc. Figure 3.1a illustrates an example of feature-based simple CAD model creation. Using this type of CAD modeling, the final model is stored in a data structure called *Boundary-Representation* (B-Rep). The B-Rep describes the geometry and the topology of the CAD model through faces, edges, loops, co-edges and vertices [13]. However, it does not include information about how these entities are designed. Accordingly, recent efforts in the state-of-the-art have focused on relating B-Reps to the design history [74, 13, 72]. In particular, two main directions have been followed: (1) segmenting the B-Rep faces into CAD operation types (*e.g.* '*extrusion*', '*revolution*') [72, 13] or higher-level machining features (*e.g.* '*holes*', '*slots*') [77] that allowed their creation; (2) inferring a sequence of

parametric sketches and extrusions that allowed the design of the B-Rep [74, 27, 23]. While the first group of works have the advantage of relating each face of the B-Rep to various types of CAD operations, they do not describe the relationship between the faces nor the steps of the construction. On the other hand, the works taking the second direction reconstruct the ordered sequence of the design history, including sketches, but they are usually limited to only one CAD operation type (*i.e.* '*extrusion*') as a simplification of the search space.

In this work, we combine both directions by segmenting the faces of the B-Reps into various CAD operation types and further decomposing them into steps of construction as shown in Figure 3.1. These two aspects are jointly learned using an end-to-end neural network, allowing the recovery of further information about the design history such as CAD sketches. The proposed method is evaluated on the publicly available Fusion360 dataset [27], and a newly introduced dataset that is closer to real-world challenges. The key **contributions** can be summarized as follows:

- A neural network, *CADOps-Net*, that operates on B-Reps is proposed to learn the segmentation of faces into CAD operation types and steps. We introduce a joint learning method within an end-to-end model.
- We create a novel dataset, *CC3D-Ops*, that builds on top of the existing CC3D dataset [41] by extending it with B-Reps and their corresponding per-face CAD operation type and step annotations. Compared to existing datasets [27, 15, 28], *CC3D-Ops* better reflects real-world industrial challenges thanks to the complexity of its CAD models.
- The proposed approach is evaluated on two datasets and compared to recent state-of-the-art methods. We further showcase some preliminary results on a possible downstream application consisting of CAD sketch recovery from B-Reps.

The rest of the chapter is organized as follows; In Section 3.2 related works are discussed followed by the problem formulation in Section 3.3. Section 3.4 describes the proposed *CADOps-Net*. The proposed *CC3D-Ops* dataset is introduced in Section 3.5. The experimental results are reported and analyzed in Section 3.6. Finally, Section 3.7 concludes this work and presents directions for future work.

3.2 Related Works

Learning representations for 3D shape modeling [78] is an important research topic that aims at finding the best deep feature encoding method. For instance, while a group of works leverages feature embedding for unordered and irregular point clouds [33, 53, 60, 79, 80] or regular grids of voxels [81, 82, 41, 83, 84], another group of works [85, 86, 87] defines convolution kernels and feature embedding techniques for meshes and manifolds. Other works [70, 13, 72, 73] focused on learning from high-level 3D shape representations such as CAD models. These methods either assume that the CAD models are obtained using CSG or feature-based modeling. In particular, the recovery of the CAD design history considering these two types of modeling has attracted a lot of attention [27, 72, 13, 73, 70].

CSG-based Approaches. Several approaches [73, 88, 75, 89] attempt to infer the design history of CAD models using CSG representation. For instance, when the input shape is a 3D point cloud, [70] and [89] convert it to the CSG tree (mainly binary-tree) of solid bodies which is a volumetric representation of simple geometrical primitives. Similarly, when the input is a B-Rep or a solid body, [90] and [88] describe unique CSG conversion steps (or vice-versa in [75]). The conversion reveals hierarchical steps involved in modeling solid bodies, whereas CAD models appear more as connected surface patches than volumetric solids [91]. Therefore, predicting CSG construction history may not reveal the actual CAD construction steps used in modern CAD workflows [74]. The latter mostly consider B-Reps instead of CSG and rely on feature-based modeling, which is addressed in our work.

Feature-based Approaches. The methods that either directly learn the B-Rep structure of a CAD model [14, 13, 92, 74, 77] or predict sketches and CAD operations [26, 93, 94, 95], are closely related to our work. The works in [93, 95] propose generative models for CAD sketches with a focus on the constraints of sketch entities. Therefore, they do not consider the connection between constrained CAD sketches and operations. On the other hand, methods like SolidGen [92], BRepNet [13], UV-Net [72], CADNet [77] put more emphasis on how to use the B-Rep data structure to obtain face embeddings followed by face segmentation, but obscuring the relation between the segmented faces and design steps. DeepCAD [26],

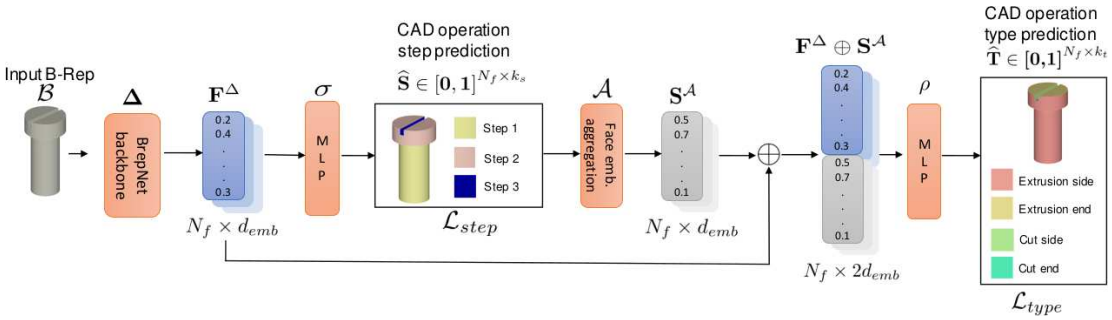


Figure 3.2: The *CADOps-Net* joint learning network architecture. The input B-Rep, $\mathcal{B}$, is first passed through a BrepNet backbone, $\mathbf{\Delta}$, to obtain face embeddings, $\mathbf{F}^{\Delta}$. These embeddings are then fed to an MLP layer, σ, to predict the face *op.step* segmentation, $\widehat{\mathbf{S}}$. Using these predictions, the face embeddings, $\mathbf{F}^{\Delta}$, are aggregated with a function $\mathcal{A}$ into step embeddings, $\mathbf{S}^{\mathcal{A}}$. Finally the concatenation, $\oplus$, of the face embeddings, $\mathbf{F}^{\Delta}$, and their corresponding step embeddings, $\mathbf{S}^{\mathcal{A}}$, are passed through an MLP layer, ρ to predict the *op.type* face labels.

Fusion360 [27] and Zone-graph [74] are the first set of methods, to the best of our knowledge, that relate parametric sketches and CAD operations proposing a generative model for CAD design. However, their models were restricted to only one type of CAD operations, namely extrusion. Finally, Point2Cyl [23] operates on point clouds to detect 2D sketches but is also limited to the CAD extrusion operation.

CAD Modeling Datasets. Besides Fusion360 [27], there are no datasets that provide both B-Reps and fully explicit construction history in standard format. For example, the ABC dataset [28] provides over 1 M CAD models with sparse construction history provided in Onshape proprietary format [27]. On the other hand, the SketchGraphs dataset [94] contains a large number of sketch construction sequences but not the B-Reps. Both MFCAD [15] and MFCAD++ [77] datasets contain B-Reps and machining feature labels. However, the samples are synthetic models and too simple to consider for industrial modeling tasks. CC3D dataset [41] offers over 50 k pairs of industrial CAD models as triangular meshes and their corresponding 3D scans, but without construction steps and B-Reps. *CC3D-Ops* supplements the CC3D dataset with these elements.

3.3 Problem Statement

A B-Rep $\mathcal{B}$ can be defined as a tuple of three sets of entities – *i.e.*, a set of N_f faces $\{f_1, f_2, \ldots, f_{N_f}\}$, a set of N_e edges $\{e_1, e_2, \ldots, e_{N_e}\}$, and a set of N_k co-edges (also known as directed half-edges) $\{j_1, j_2, \ldots, j_{N_j}\}$. Our main goal is to relate each face f in $\mathcal{B}$ with its construction history using three different types of features $\mathbf{F} \in \mathbb{R}^{N_f \times d_f}$, $\mathbf{E} \in \mathbb{R}^{N_e \times d_e}$, and $\mathbf{J} \in \mathbb{R}^{N_j \times d_j}$ extracted for the three entities, namely, faces, edges, and co-edges, respectively. The CAD construction history is defined as a sequential combination of sketches followed by some CAD operations. In this chapter, we are interested in learning (1) the type of CAD operations through the segmentation of each face that allowed for its creation, and (2) the CAD operation step to which the segmented face belongs.

3.3.1 CAD Operation Types

The choice of CAD operation types is crucial for constructing CAD models. For notation simplicity, let us denote them as ***op.types***. The geometry of the final CAD model, usually stored as a B-Rep, is obtained through these operations, which makes each face of the B-Rep directly related to a type of operation. In Figure 3.1c, we show some intermediate steps of CAD construction and how the faces of the corresponding B-Rep are obtained using different *op.types*. For example, the B-Rep of a cube that was obtained by sketching a 2D square and applying an extrusion operation, as in Figure 3.1a, would result in two faces with *'extrude end'* labels and four faces with *'extrude side'* labels. The ability to automatically infer the *op.type* that allowed for the creation of each face of the B-Rep constitutes a first, yet essential, step towards relating the geometry of the CAD model to its construction history. Recently introduced models [13, 72] proposed to learn the segmentation of B-Rep faces into *op.types*.

Formally, let us consider a B-Rep $\mathcal{B}$ labelled with the per-face *op.types*:

$$\mathbf{T} \;=\; [\mathbf{t}_1; \mathbf{t}_2; \ldots; \mathbf{t}_{N_f}] \;\in\; \{0,1\}^{N_f \times k_t}\,, \tag{3.1}$$

where k_t is the number of possible *op.types*. Here, $\mathbf{T} \in \{0,1\}^{N_f \times k_t}$ is an $N_f \times k_t$ matrix with binary entries, where each row $\mathbf{t}_j \in \{0,1\}^{k_t}$ can have only one element as 1 representing the *op.type* of the face f_j. The task of *op.type* segmentation consists of learning a mapping $\mathbf{\Phi}$, such that,

$$\mathbf{\Phi} : \mathbb{R}^{N_f \times d_f} \times \mathbb{R}^{N_e \times d_e} \times \mathbb{R}^{N_k \times d_j} \rightarrow \{0,1\}^{N_f \times k_t} , \tag{3.2}$$

$$\mathbf{\Phi}(\mathbf{F}, \mathbf{E}, \mathbf{J}) = \mathbf{T} . \tag{3.3}$$

It is important to highlight that the segmentation task of *op.types* uses the features of faces, edges and co-edges, but assigns a unique *op.type*, among a fixed number of possible types, to each face of the B-Rep. Despite its usefulness for reconstructing the CAD construction history of B-Reps, the segmentation into *op.types* is not sufficient as it does not describe the relationship between the faces nor the steps of the construction.

3.3.2 CAD Operation Steps

In addition to the operation types that are assigned to the faces of the B-Reps, our aim is to relate them further to the construction history. Accordingly, we propose a novel task consisting of segmenting the faces of B-Reps into CAD operation steps. For notation simplicity, they will be denoted as ***op.steps*** in what follows. While the segmentation into *op.types* aims at identifying the operation that was used to create each face, the purpose of the segmentation into *op.steps* is to group faces that were created at the same time step. An example is shown in Figure 3.1b.

Formally, let us consider a B-Rep $\mathcal{B}$ labelled with the per-face *op.steps* $\mathbf{S} \in \{0,1\}^{N_f \times k_s}$, where k_s denotes the number of *op.steps* in $\mathcal{B}$. Similarly to the *op.types* $\mathbf{T}$, the *op.steps* are represented by an $N_f \times k_s$ binary matrix $\mathbf{S} = [\mathbf{s}_1; \mathbf{s}_2; \ldots; \mathbf{s}_{N_f}] \in \{0,1\}^{N_f \times k_s}$. Each row of this matrix, $\mathbf{s}_j \in \{0,1\}^{k_s}$, can have only one element equal to 1 denoting the *op.step* for the face f_j. Segmenting the faces of B-Reps into *op.steps*, would require learning a mapping $\mathbf{\Psi}$,

$$\mathbf{\Psi} : \mathbb{R}^{N_f \times d_f} \times \mathbb{R}^{N_e \times d_e} \times \mathbb{R}^{N_k \times d_j} \rightarrow \{0,1\}^{N_f \times k_s} , \tag{3.4}$$

$$\mathbf{\Psi}(\mathbf{F}, \mathbf{E}, \mathbf{J}) = \mathbf{S} . \tag{3.5}$$

The proposed segmentation into *op.steps* is a challenging task for two main reasons: (1) unlike the *op.type* segmentation where the possible types are predefined, the labels of *op.steps* **S** are arbitrary and any combination of labels, in which faces belonging to the same step have identical labels, can be considered as correct; (2) predicting *op.steps* aims at grouping B-Rep faces according to the design history. Therefore, it requires learning the relationship between the different faces of the B-Rep in addition to its geometry and topology.

3.4 Proposed CADOps-Net

The proposed *CADOps-Net* jointly learns the *op.type* and *op.step* segmentation within the same model. In practice, the mappings $\mathbf{\Psi}$ and $\mathbf{\Phi}$, introduced in Section 3.3, are learnt using an end-to-end neural network. BRepNet [13] is used as the backbone of our model, as it has been shown to effectively operate on B-Reps. BRepNet uses the face, edge, and co-edge features $(\mathbf{F}, \mathbf{E}, \mathbf{J})$ of a B-Rep $\mathcal{B}$ to learn per-face embeddings using a succession of convolutions defined through specific topological walks and Multilayer Perceptron (MLP) layers. For more details about this backbone, readers are referred to [13]. In what follows, the BRepNet backbone will be denoted by $\mathbf{\Delta} : \mathbb{R}^{N_f \times d_f} \times \mathbb{R}^{N_e \times d_e} \times \mathbb{R}^{N_k \times d_j} \rightarrow \mathbb{R}^{N_f \times d_{emb}}$ and $\mathbf{f}^{\Delta}$ will be used as a notation for the embedding extracted using this backbone from a face f of a B-Rep $\mathcal{B}$. The proposed network is composed of two modules that are described below.

3.4.1 CAD Operation Step Segmentation

The CAD operation step module has two roles. Firstly, it predicts the per-face *op.step* labels. Secondly, it is used to aggregate the embeddings of faces belonging to the same step and produce embeddings for each group of faces obtained in a single *op.step*.

Learning CAD operation steps: The mapping $\mathbf{\Psi}$ introduced in Section 3.3.2 consists of

two components, *i.e.*, $\mathbf{\Psi} := \sigma \circ \mathbf{\Delta}$, where $\mathbf{\Delta}$ uses the features of the B-Rep $(\mathbf{F}, \mathbf{E}, \mathbf{J})$ and extracts per-face embeddings $\mathbf{F}^{\Delta} = [\mathbf{f}_1^{\Delta}; \mathbf{f}_2^{\Delta}; \ldots; \mathbf{f}_{N_f}^{\Delta}] \in \mathbb{R}^{N_f \times d_{emb}}$. σ is an MLP followed by softmax that maps the face embeddings $\mathbf{F}^{\Delta}$ into probabilities of predicted *op.steps* $\widehat{\mathbf{S}} = [\widehat{\mathbf{s}}_1; \widehat{\mathbf{s}}_2; \ldots; \widehat{\mathbf{s}}_{N_f}] \in [0,1]^{N_f \times k_s}$. Here, each face f_j would have a vector $\widehat{\mathbf{s}}_j \in [0,1]^{k_s}$ specifying its membership probabilities to the k_s *op.steps*. It is important to note that the number of *op.steps* in a CAD model is not known in advance. We assume the maximum number of steps, k_s, in a B-Rep $\mathcal{B}$ to be the largest number of possible steps per model computed on the training dataset.

As mentioned in Section 3.3.2, a particular challenge for predicting the *op.steps* is that the ground truth labels $\mathbf{S}$ are arbitrary. Therefore, the task consists of predicting the combination of steps that matches the ground truth labels. Inspired by [96, 23], we use a Hungarian matching [97] to find the best one-to-one correspondences between the predicted *op.steps* $\widehat{\mathbf{S}}$ and ground truth labels $\mathbf{S}$. Even though the Hungarian matching is not differentiable, it is only used to find the correspondences in the training phase, allowing for the computation of a *Relaxed Intersection over Union* (RIoU) [98] metric between pairs of predictions $\widehat{\mathbf{s}}$ and ground truth $\mathbf{s}$ as follows,

$$\text{RIoU}(\mathbf{s}, \widehat{\mathbf{s}}) = \frac{\mathbf{s}^{\mathsf{T}}\widehat{\mathbf{s}}}{||\mathbf{s}||_1 + ||\widehat{\mathbf{s}}||_1 - \mathbf{s}^{\mathsf{T}}\widehat{\mathbf{s}}} , \tag{3.6}$$

where $||.||_1$ denotes the ℓ_1 norm, and $^{\mathsf{T}}$ the vector transpose. The RIoU metric is further used to define the following *op.step* loss function,

$$\mathcal{L}_{step} = \frac{1}{N_f} \sum_{j=1}^{N_f} (1 - \text{RIoU}(\mathbf{s}_j, \widehat{\mathbf{s}}_j)) . \tag{3.7}$$

For inference, the Hungarian matching is not used and the predicted *op.steps* are given by taking the maximum probability over each $\widehat{\mathbf{s}}$.

CAD operation step embedding: In addition to predicting the per-face *op.steps* given a B-Rep, the same module is used to extract CAD step embeddings $\{\mathbf{s}_1^{\mathcal{A}}, \mathbf{s}_2^{\mathcal{A}}, \ldots, \mathbf{s}_{k_s}^{\mathcal{A}}\}$. This is achieved by aggregating the embeddings of faces predicted to belong to the same *op.step*.

Specifically, each *op.step* φ would have an embedding $\mathbf{s}_{\varphi}^{\mathcal{A}} \in \mathbb{R}^{d_{emb}}$, such that

$$\mathbf{s}_{\varphi}^{\mathcal{A}} = \underset{j=\arg\max \widehat{\mathbf{S}}_{:,\varphi}}{\mathcal{A}} \mathbf{f}_j^{\Delta} \,, \tag{3.8}$$

where $\widehat{\mathbf{S}}_{:,\varphi}$ denotes the per-face predicted *op.step* labels for φ, and $\mathcal{A}$ is an aggregation function that preserves the dimension of the input embeddings such as average or maximum. Finally, each face of the B-Rep will have the corresponding *op.step* embedding $\mathbf{s}^{\mathcal{A}}$ according to the predicted *op.step* label. These embeddings are finally stacked in a matrix $\mathbf{S}^{\mathcal{A}} \in \mathbb{R}^{N_f \times d_{emb}}$.

3.4.2 CAD Operation Type Segmentation

The introduced mapping $\mathbf{\Phi}$ to obtain the *op.type* segmentation from an input B-Rep shares the same BRepNet backbone $\mathbf{\Delta}$ used by the module of *op.type* segmentation. Moreover, it uses two other mappings, γ and ρ, where $\mathbf{\Phi} := \rho \circ \gamma \circ \mathbf{\Delta}$. The mapping

$$\gamma \,:\, \mathbb{R}^{N_f \times d_{emb}} \,\times\, \mathbb{R}^{N_f \times d_{emb}} \,\rightarrow\, \mathbb{R}^{N_f \times 2d_{emb}} \tag{3.9}$$

takes as input the face embeddings $\mathbf{F}^{\Delta}$ and outputs their concatenation with the corresponding step embeddings $\mathbf{S}^{\mathcal{A}}$. These concatenated embeddings are fed to an MLP with softmax which are represented by $\rho \,:\, \mathbb{R}^{N_f \times 2d_{emb}} \,\rightarrow\, \{0,1\}^{N_f \times k_t}$. The final *op.types* $\widehat{\mathbf{T}}$ can be obtained following,

$$\widehat{\mathbf{T}} = \rho(\mathbf{F}^{\Delta} \oplus \mathbf{S}^{\mathcal{A}}) \,, \tag{3.10}$$

where $\oplus$ is the column-wise concatenation operation. The loss function for the *op.type* segmentation is computed using the cross-entropy $\mathcal{H}$ between the predicted per-face *op.types* $\widehat{\mathbf{t}}$ and the ground truth labels $\mathbf{t}$,

$$\mathcal{L}_{type} = \frac{1}{N_f} \sum_{j=1}^{N_f} \mathcal{H}(\mathbf{t}_j, \widehat{\mathbf{t}}_j) \,. \tag{3.11}$$

The total loss function is the sum of the *op.step* and *op.type* losses,

$$\mathcal{L}_{total} = \mathcal{L}_{step} + \mathcal{L}_{type} \ . \tag{3.12}$$

The model jointly learns to predict the per-face *op.type* and *op.step* labels of a CAD model given its B-Rep, with the *op.type* being conditioned on the *op.step*.

3.5 CC3D-Ops dataset

In this section, we introduce the proposed *CC3D-Ops* dataset that contains over 37 k B-Reps with the corresponding per-face *op.type* and *op.step* annotations. The B-Reps and their corresponding annotations constitute an extension of the CC3D dataset [41]. First, the method used to extract the *op.step* and *op.type* is briefly discussed. Then, statistics demonstrating the complexity of the models and showing the distribution of the labels is presented.

3.5.1 CC3D-Ops Label Extraction

The proposed *CC3D-Ops* dataset contains B-Reps with per-face *op.type* and *op.step* annotations. An important aspect of segmenting faces into different construction steps of modeling operations is that these labels come from the real construction history of each CAD model in the dataset. In our case, this information is obtained from the native SolidWorks [99] Part File (.sldprt) format of a CAD model. A set of tools were developed based on the Solidworks API [99] to traverse a CAD model's construction history and to assign each face generated by respective modeling operation its *op.type* and *op.step* labels in B-Rep.

3.5.2 Dataset Complexity

The *CC3D-Ops* dataset contains a similar number of B-Reps (about 35 k) as the Fusion360 dataset [27]. However, the Fusion360 dataset [27] does not provide *op.step* labels and it includes relatively simple CAD models. The proposed *CC3D-Ops* dataset comes with more complex models that are closer to real-world industrial challenges. From the sample

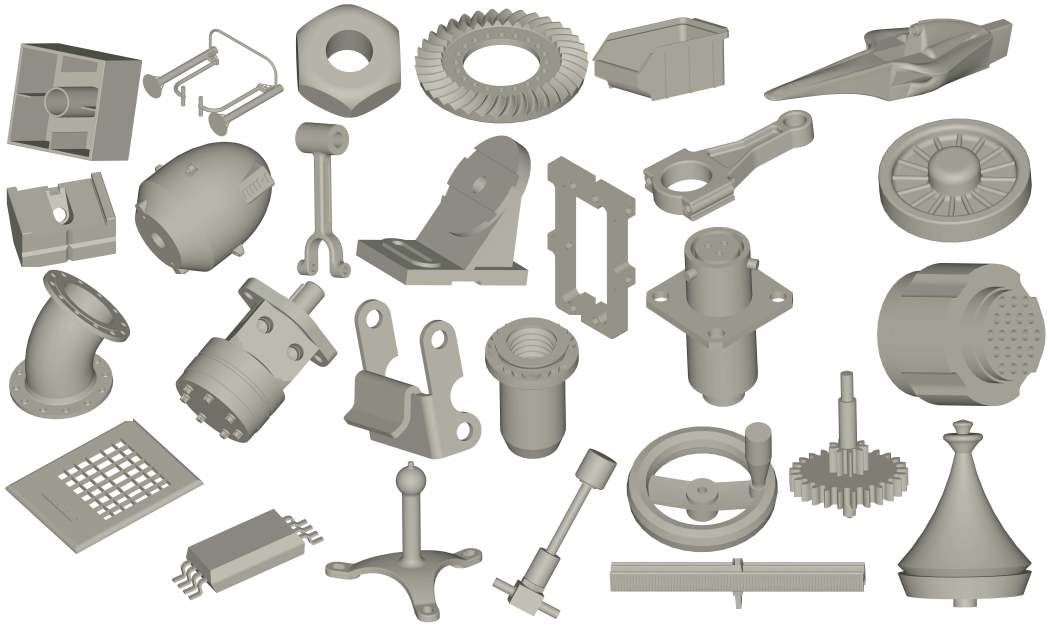


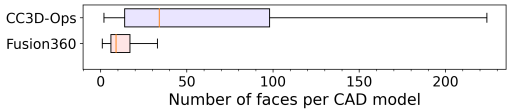
Figure 3.3: Sample CAD models from the *CC3D-Ops* dataset.

of *CC3D-Ops* CAD models (in B-Rep format) displayed in Figure 3.3, it can be noted that *CC3D-Ops* offers a wide variety of models both in terms of complexity and category.

In Figure 3.4a, we illustrate the distribution of *op.step* number per model as a box plot for both Fusion360 and *CC3D-Ops* datasets. It can be clearly observed that the distribution of *CC3D-Ops* is more skewed towards a higher number of *op.steps* than the one of Fusion360. Specifically, $\sim 48\%$ of the Fusion360 models are made of only one *op.step* and $\sim 80\%$ of them are constructed by 3 or less *op.steps*. On the other hand, only $\sim 20\%$ of the *CC3D-Ops* models are built with a single *op.step* and $\sim 44\%$ of them with 3 or less *op.steps*. Moreover, the maximum number of *op.steps* per model, k_s, is 59 for Fusion360 and 262 for *CC3D-Ops*.

Figure 3.4b shows the distribution of the number of faces per model for the *CC3D-Ops* and Fusion360 [27] datasets as box plots. This figure shows that the models in *CC3D-Ops* generally have more faces than in Fusion360. While 90% of the models of the Fusion360 dataset have 30 faces or less, such models represent only 50% of *CC3D-Ops*.

(a) Box plot showing the distribution of models with respect to the number of *op.steps* per model.



(b) Box plot showing the distribution of models with respect to the number of faces per model.

Figure 3.4: Distribution comparisons between the *CC3D-Ops* and Fusion360 [27] datasets.

3.5.3 CAD Operation Type Labels

The *op.type* face labels indicate the type of CAD operation used during the design process. While the most common CAD operation types (such as *extrusion*, *fillet* ...) are shared among most CAD software applications, some are software specific. The *CC3D-Ops* dataset introduces three new *op.types* to the eight present in Fusion360 which consists of, *cut revolve side*, *cut revolve end*, and *others*. Hence, the *CC3D-Ops* dataset contains 11 different *op.type* labels: *extrude side*, *extrude end*, *revolve side*, *revolve end*, *cut extrude side*, *cut extrude end*, *cut revolve side*, *cut revolve end*, *fillet*, *chamfer* and *other*. The *other op.type* represents less common types such as *helix*, *sweep*, *dome*, etc. The bar chart in Figure 3.5 displays the number of faces for each *op.type* label. The two least common *op.type* labels are *revolve end* and *cut revolve end* and the two most common operation types are *extrude side* and *other*. For a comparison with the Fusion360 dataset, we refer the reader to [13] where a similar bar chart can be found.

3.6 Experiments

3.6.1 Experimental Setup

Input Features: The input features of *CADOps-Net* are face, edge and co-edge features $(\mathbf{F}, \mathbf{E}, \mathbf{J})$ extracted from the B-Rep, $\mathcal{B}$. Following [13], the face type (*e.g.* plane, cylinder, sphere) and area are encoded in a single vector. 3D points are further sampled on each face using the UV-grid of the B-Rep and encoded as described in [72]. These two features are concatenated and used as face features. The features of the B-Rep faces are then

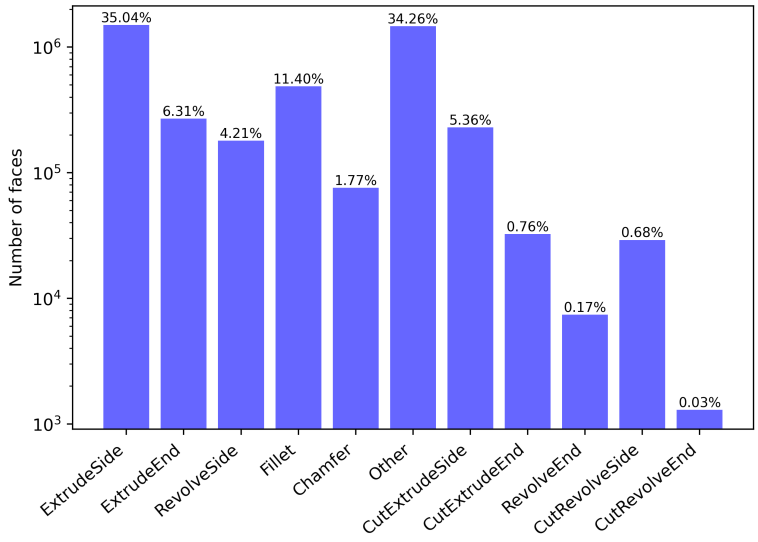


Figure 3.5: Bar graph of the number of faces for each *op.type* label over the *CC3D-Ops* dataset. The numbers above each bar represents the percentage of the number of faces with the corresponding type. Note: a *log* scale is used for the vertical axis.

concatenated in a row-wise fashion to form the matrix **F**. For edge features, a similar approach is taken by considering the type, convexity, closeness, length of the edge as in [13], and encoded sampled 3D points as done in [72]. The result is concatenated in an edge feature matrix $\mathbf{E}$. The co-edge features, **C**, are simple flags to represent the direction of the corresponding edges [13].

Network Architecture: The input features are passed through a BRepNet backbone, $\mathbf{\Delta}$, with the same parameters as in [13] using the wing-edge kernel. The dimension of the face embedding, $\mathbf{f}^{\Delta}$, is $d_{emb} = 64$. These embeddings are fed to an MLP followed by softmax, σ, to predict the *op.step*. The aggregation function used to compute the step embedding, $\mathbf{S}^{\mathcal{A}}$, is the average function. Each *op.step* embedding $\mathbf{s}^{\mathcal{A}}$ has the same dimension as $\mathbf{f}^{\Delta}$. The final face embedding, $\mathbf{f}^{\Delta} \oplus \mathbf{s}^{\mathcal{A}}$, are 128-dimensional. Lastly, the *op.type* is estimated by passing these embeddings through an MLP followed by softmax, ρ. In our experiments, the number of layers of the employed MLPs is 1.

Datasets: *CADOps-Net* is evaluated on the Fusion360 dataset [27] and the novel *CC3D-Ops*

dataset described in Section 3.5. Note that in Fusion360, the *op.step* annotations were derived from the *op.type* annotations as they were implicitly provided. The train, validation, and test sets for the Fusion360 dataset are the same as in [13]. For the *CC3D-Ops* dataset, the splitting ratios are approximately 65%, 15%, and 20% for the train, validation, and test sets.

Training details: The training was conducted for 200 epochs with a batch size of 100 using an NVIDIA RTX $A6000$ GPU. Adam optimizer is employed with a learning rate of 0.001 and beta parameters of 0.9 and 0.99.

Metrics: The performance of the network is evaluated on *op.type* and *op.step* segmentation tasks. To evaluate the *op.type* segmentation, we use the same metrics as in [13], namely, the mean accuracy (mAcc) and the mean Intersection over Union (mIoU). Note that we do not consider the mIoU for evaluating the *op.step* as the labels represent membership sets rather than predefined classes. Furthermore, the consistency between the *op.type* and *op.step* predictions is considered. For this purpose, we group the sub-*op.types*, such that *'extrude end'* and *'extrude side'*, into a single *'extrude' op.type*. Similar grouping is done for *'revolve'*, *'cut extrude'*, and *'cut revolve'*. We define an *op.step* prediction as consistent if all its faces have the same *op.type* prediction. To evaluate this consistency, two metrics are computed: (1) the first one, R_C, quantifies the overall consistency as the ratio of consistent predicted *op.steps*; (2) the second one quantifies the amount of consistency of a model as $S_C = \sum_i \frac{\max(n_{(t_1,s_i)},\ldots,n_{(t_{k_t},s_i)})}{n_{s_i}}$ where n_{s_i} is the number of faces with *op.step* label s_i and $n_{(t_j,s_i)}$ the number of faces with *op.type* label t_j and *op.step* label s_i. We then compute mS_C as the average over all the models.

3.6.2 Results and Discussions

Qualitative Evaluation: In Figure 3.6, we illustrate the predictions obtained by *CADOps-Net* on five models from the *CC3D-Ops* dataset. More predictions are provided in Section A of the Appendix. Despite the complexity of some models, it can be observed that most of the *op.type* predictions (left panel) were correct except for very few faces. On the other hand,

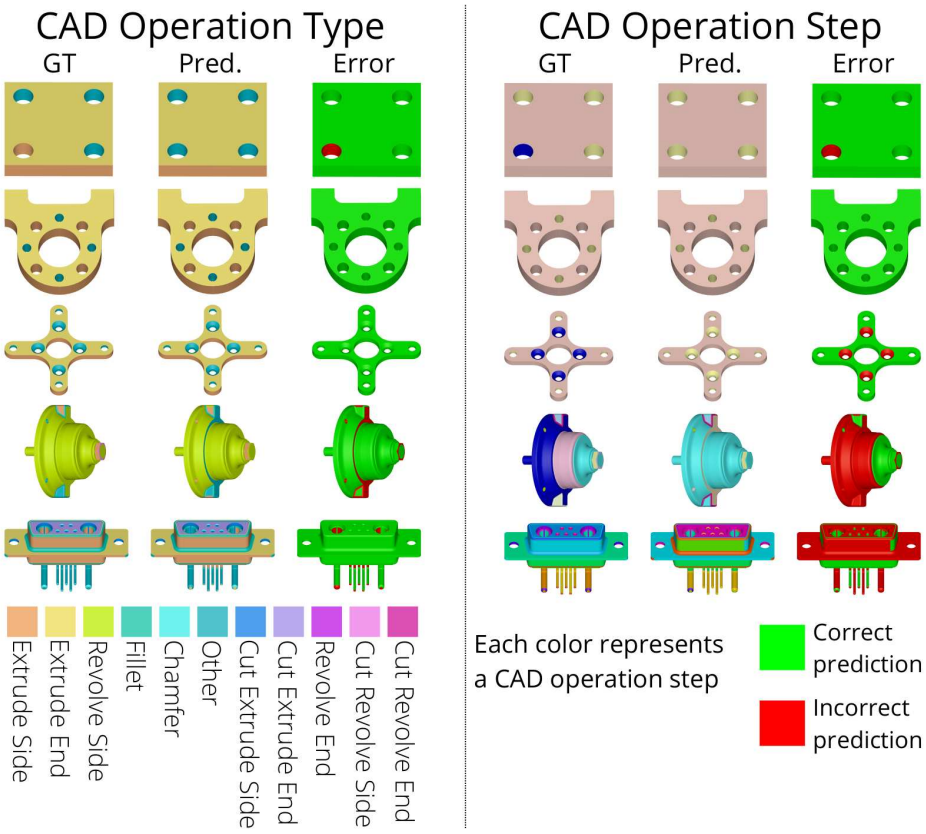
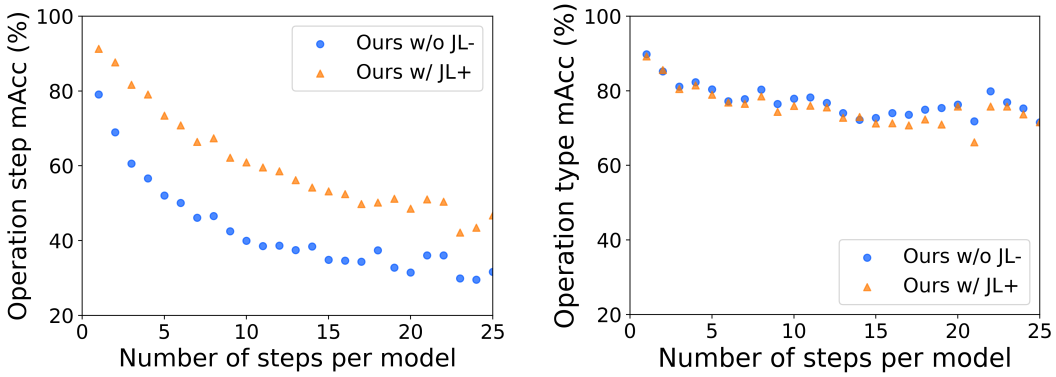


Figure 3.6: Sample predictions on five models from the *CC3D-Ops* dataset. (Left): The CAD operation type segmentation. (Right): The CAD operation step segmentation. For both tasks, the ground truth (GT) is shown in the left, the prediction (Pred.) in the middle, and the error (Error) in the right illustrating the correct/incorrect face predictions.

the segmentation into *op.steps* (right panel) was more challenging for complex models (two last rows) as the segmentation into *op.steps* requires the model to learn the relationship between the faces of the B-Rep according to the construction history. Such aspect is more challenging to capture for complex models than the *op.types* which could be hypothetically learned from the geometry and topology of the B-Reps. This hypothesis is further discussed in the quantitative evaluation.



(a) CAD operation step mAcc

(b) CAD operation type mAcc

Figure 3.7: Mean accuracy (mAcc) of CAD operation type and step segmentation *w.r.t* the number of steps per model on the *CC3D-Ops* dataset.

Quantitative Evaluation: In Table 3.1, we report the quantitative results of our approach compared to baselines. *CADOps-Net* (*Ours w/ JL*$^{+}$) is compared to the same model without the joint learning of *op.steps* and *op.types* (*Ours w/o JL*$^{-}$). In the latter, the *op.type* and *op.step* segmentation modules are trained independently. In the following, we first analyze the results for the segmentation into *op.steps* (column 5 of Table 3.1) and for the *op.type* segmentation (columns 3 and 4), then we discuss the consistency between the two types of predictions (columns 6 and 7).

As previously mentioned, predicting *op.steps* is a much more challenging task than *op.types* especially for models with a large number of *op.steps*. While the joint learning leads to small improvements on the *op.step* mAcc metric on the Fusion360 dataset, significant improvements can be observed on the *CC3D-Ops* dataset results with an increase of $\sim 14\%$.

	Model	*op.type*		*op.step*	*Consistency*	
		mAcc	mIoU	mAcc	R_C	mS_C
Fusion360	*CADNet* [77]	88.9	67.9	-	-	-
	UV-Net [72]	92.3	72.4	-	-	-
	BRepNet [13]	94.3	81.4	-	-	-
	Ours w/o JL$^-$	95.5	83.2	80.2	87.1	97.4
	Ours w/ JL$^+$	**95.9**	**84.2**	**82.5**	**93.3**	**98.7**
CC3D-Ops	*CADNet* [77]	57.5	26.9	-	-	-
	BRepNet [13]	71.4	35.9	-	-	-
	Ours w/o JL$^-$	**76.0**	43.0	48.4	40.7	82.7
	Ours w/ JL$^+$	75.0	**44.3**	**62.7**	**82.4**	**96.7**

Table 3.1: Results of the segmentation into CAD operation types and steps on the Fusion360 and *CC3D-Ops* datasets. All results are expressed as percentages. *Ours w/o JL*$^-$ denotes our method without joint learning. *Ours w/ JL*$^+$ refers to the proposed *CADOps-Net* with joint learning.

This difference of results can be explained by the higher complexity of *CC3D-Ops* models compared to those of Fusion360. Figure 3.7a shows the mAcc of *op.step* segmentation related to the number of *op.steps* per model on the *CC3D-Ops* dataset. It can be observed that for models with less than 25 *op.steps*, representing over 96% of the *CC3D-Ops* dataset, *CADOps-Net* scores consistently and significantly better than without joint learning. These observations demonstrate the importance of the joint learning for *op.step* segmentation. However, in both cases there is a major decrease in the *op.step* segmentation mAcc as the number of steps per model increases. This is expected since the task becomes increasingly challenging as the number of *op.steps* becomes larger. Note that we did not compare our results to state-of-the-art (BRepNet [13], UV-Net [72], and CADNet[77]) on the task of *op.step* segmentation as their methods are not designed to predict arbitrary face labels.

In order to evaluate the *op.type* segmentation of *CADOps-Net*, the results are compared to state-of-the-art results. On the Fusion360 dataset, we recorded slight improvements over [13], [72], and [77] in terms of mAcc. More significant improvements *w.r.t* [72] and [77] were obtained in terms of mIoU (more than 12% and 16%, respectively). On the *CC3D-Ops* dataset, our results clearly outperformed those of [13] and [77] on the two met-

rics. Furthermore, we compare *CADOps-Net* to the scenario where the joint learning is omitted. Table 3.2 and 3.3 show the IoU results of each *op.type* label for the Fusion360 and *CADOps-Net* datasets respectively. The results are shown for both *CADOps-Net* without joint learning (*Ours w/o JL*$^-$) and with joint learning (*Ours w/ JL*$^+$). The joint learning strategy does not have a significant impact on the *op.type* predictions. This is particularly the case on the Fusion360 dataset as shown in Table 3.2. *Ours w/ JL*$^+$ achieves slightly higher IoU for each class. On the other hand, the same trend cannot be found in the results obtained from the *CC3D-Ops* dataset. For 6 out of the 11 *op.type* classes, the difference between the IoUs obtained with joint learning and without is relatively small (less than 2%). For the *op.type* classes *cut revolve side* and *chamfer*, *Ours w/o JL*$^-$ scores higher than the *Ours w/ JL*$^+$ by 3.9% and 5.5% respectively. However, the joint learning method achieves higher results on 3 out of the 4 least common classes, namely *cut revolve end*, *revolve end* and *cut extrude end*. In particular, for the *revolve end op.type* that represents 0.17% of the dataset, the joint learning strategy results in an IoU that is about 17% higher than without joint learning. This demonstrates that even if the joint learning strategy achieves a comparable mIoU as without joint learning, *Ours w/ JL*$^+$ is able to learn more meaningful features for the underrepresented *op.types*.

In Figure 3.7b, we show the mAcc of *op.type* segmentation related to the number of *op.steps* per model. In contrast to the *op.step* segmentation, one can notice that the number of *op.steps* has a slight impact on the *op.type* mAcc. In other words, the *op.type* segmentation does not become more challenging when complex models with large number of construction steps are involved. Intuitively, it can be hypothesized that the *op.type* segmentation is more related to the geometry and topology of the B-Rep rather than its construction history.

The results on the consistency scores (R_C and mS_C) highlight the relevance of the joint learning approach. Despite relatively similar *op.type* and *op.step* mAcc scores on the Fusion360 dataset for *Ours w/ JL*$^+$ and *Ours w/o JL*$^-$, the joint learning approach produces more consistent results with an increase of $\sim 6\%$ in R_C score. Similarly on the *CC3D-Ops* dataset, the predictions from *CADOps-Net* are significantly more consistent with an increase of $\sim 41\%$ in R_C score and 14% in mS_C score. Therefore, the joint learning model

Fusion360	Per class IoU	
	Ours w/o JL$^-$	*Ours w/ JL*$^+$
Extrude side	94.0	**94.6**
Extrude end	91.7	**92.4**
Cut side	82.1	**83.9**
Cut end	75.2	**77.1**
Revolve side	85.1	**86.5**
Revolve end	48.7	**48.9**
Chamfer	91.2	**92.1**
Fillet	97.6	**97.8**

Table 3.2: *op.type* per class IoU for the Fusion360 dataset. All results are expressed as percentages.

CC3D-Ops	Per class IoU	
	Ours w/o JL$^-$	*Ours w/ JL*$^+$
Extrude side	**65.4**	64.9
Extrude end	59.7	**60.2**
Cut extrude side	17.8	**18.1**
Cut extrude end	10.0	**15.4**
Cut revolve side	**22.2**	18.3
Cut revolve end	1.1	**4.6**
Revolve side	**60.3**	59.8
Revolve end	23.8	**41.2**
Chamfer	**69.6**	64.4
Fillet	**84.1**	83.1
Other	**58.5**	57.2

Table 3.3: *op.type* per class IoU for the *CC3D-Ops* dataset. All results are expressed as percentages.

is able to extract face features that contain consistent information for both the *op.type* and *op.step* segmentation labels. The consistency property is essential for the process of reverse engineering.

3.6.3 Ablation Study

In order to provide a deeper insight into the joint learning approach, we conduct an ablation study on the aggregation function $\mathcal{A}$ of the face embeddings. Experiments are conducted with the following five scenarios: (1) the output face embeddings, $\mathbf{f}^{\Delta}$, from the BRepNet backbone

		op.type		*op.step*
	Agg. type	mAcc	mIoU	mAcc
CC3D-Ops	*No agg.*	73.0	40.2	61.5
	Soft labels	73.4	40.0	59.7
	Sum	70.4	34.4	62.6
	Max	74.3	42.0	62.2
	Avg	**75.0**	**44.3**	**62.7**

Table 3.4: Ablation study on the aggregation function used in the joint learning of *CADOps-Net*. All results are expressed as percentages.

are directly used to predict both the *op.type* and *op.step* without any aggregation (*No agg.*). (2) Another scenario concatenates the BRepNet face embeddings with the predicted soft labels of the *op.step* (*Soft labels*) again without any aggregation. (3) The last three scenarios focus on the type of aggregation function used to obtain the *op.step* embeddings, $\mathbf{S}^{\mathcal{A}}$, namely the maximum (*Max*), the average (*Avg*), and the sum of the embeddings combined with a softmax normalization (*Sum*). Table 3.4 shows the ablation results for both *op.type* and *op.step* segmentation tasks on the *CC3D-Ops* dataset. The results show that aggregating the face embeddings using an *Avg* pooling leads to slightly better overall performance.

3.6.4 CAD Sketch Recovery

Figure 3.8 illustrates preliminary results on how *CADOps-Net* predictions can be used to retrieve the CAD sketches. A sketch $\mathbb{Q}$ of a B-Rep $\mathcal{B}$ can be defined as a set of simple geometrical entities (*e.g.* straight lines, arcs). We consider a small subset of 20 models made of extrusions from the Fusion360 dataset. In the following, we describe the process for recovering the sketch corresponding to *op.step* 2 using the *CADOps-Net* predictions shown in Figure 3.8a. We first identify the faces for which the *op.type* was predicted as '*extrude side*'. Second, we cluster these faces according to their predicted *op.step*. Third, we store the face-normals $(\widehat{\mathbf{n}}_1^2, \ldots, \widehat{\mathbf{n}}_m^2)$ and sample UV-grid points on the faces. This allows to derive a common *axis of extrusion* $\widehat{a}$ and a *projection center* $\widehat{o}$. Finally, the predicted sketch $\widehat{\mathbb{Q}}^2$ is obtained by projecting the sampled points along $\widehat{a}$ (more details are in Section A of the Appendix). Figure 3.8a and 3.8b show qualitative results of successful and failed sketch

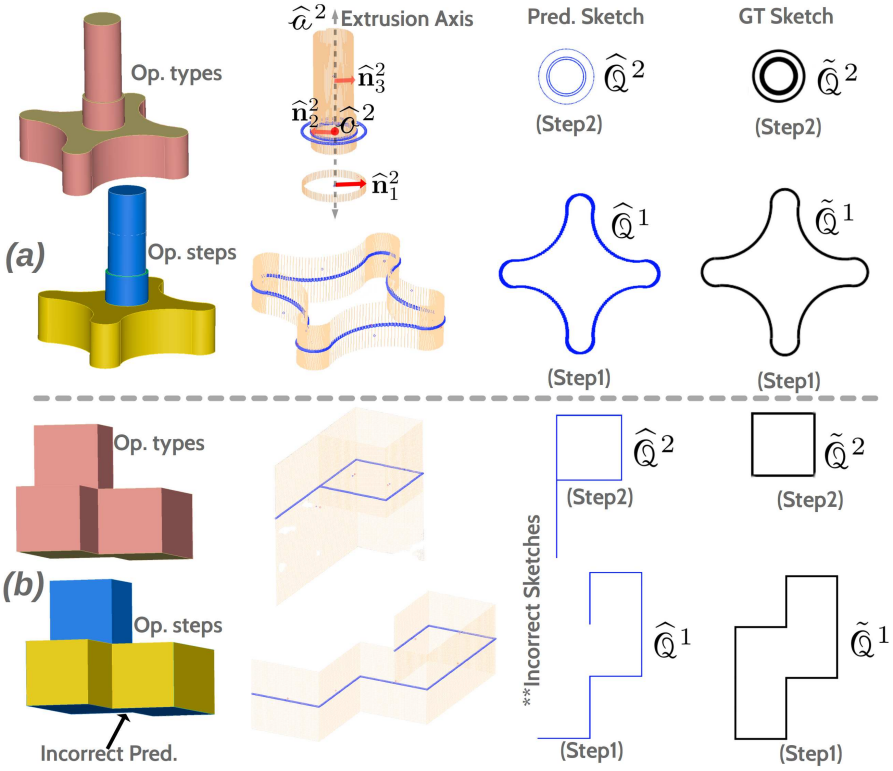


Figure 3.8: Sketch recovery from predicted CAD operation types (*op.types*) and steps (*op.steps*). *op.step* 1 and 2 are colored in yellow and blue, respectively. Figure 3.6 defines the color codes used for different *op.types*.

recoveries from correctly and incorrectly predicted *op.types*. These preliminary results on sketch recovery illustrate the relevance of *op.step* prediction in the context of 3D reverse engineering.

3.6.5 Limitations

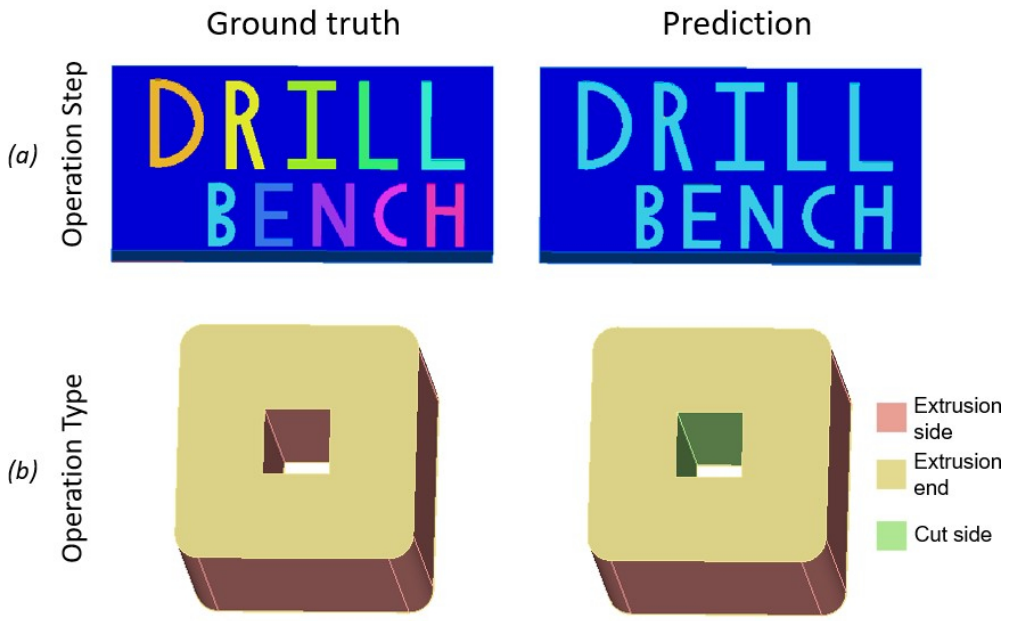


Figure 3.9: Failure cases of *CADOps-Net* for *op.step* segmentation in (a) and *op.type* segmentation in (b).

In CAD modeling, designers may opt for different design solutions. Consequently, the segmentation into *op.type* and *op.step* is not necessarily unique. An example for which the *op.step* prediction is valid despite not matching the ground truth can be found in Figure 3.9a. The letters were predicted as part of the same *op.step*, which could be a valid design approach. However, these letters were extruded with separate *op.steps* in the ground truth. In Figure 3.9b, an example with valid predictions of *op.types* not matching the ground truth is depicted. Here, the hole in the center of the shape was predicted as a '*cut*' type operation, while being an '*extrude*' in the ground truth. In general, CAD designers follow good practices

so that the final model reflects the design intent [34]. However, different designers might have their own set of good practices, making it difficult for a learning-based model to capture all the different design intents.

3.7 Conclusion

In this chapter, we presented *CADOps-Net*, a neural network architecture that jointly learns CAD operation type and step segmentation from B-Rep faces. This contribution addresses a fundamental aspect of the CAD reverse engineering challenge—recovering elements of design intent from geometric representations. The proposed joint learning strategy demonstrates significant improvements for the challenging task of CAD operation step segmentation, while simultaneously achieving state-of-the-art results on operation type segmentation. By successfully relating B-Rep faces to both their operation types and construction steps, our approach provides critical insights into the CAD construction history.

Furthermore, we demonstrated that these predicted segmentations can be leveraged to recover higher-level design information, such as the original 2D sketches—a crucial element for complete design intent recovery. To support this research and address the data limitations highlighted in Section 1.2, we introduced the *CC3D-Ops* dataset with its comprehensive operation type and step annotations across 37k B-Reps. This dataset offers more complex and industrially relevant CAD models compared to existing resources, providing a more realistic benchmark for evaluating reverse engineering methods.

While this work represents an important step toward CAD reverse engineering, it also revealed limitations in handling certain design ambiguities and complex models. As identified in our objectives, a more complete approach would require addressing the entire reverse engineering pipeline from point clouds to parametric CAD models. In the following chapters, we build upon the foundation established here by introducing transformer-based approaches that directly infer parametric CAD models from point clouds, thus addressing the end-to-end reverse engineering challenge.

Chapter 4

TransCAD: A Hierarchical Transformer for CAD Sequence Inference from Point Clouds

In this chapter, we introduce *TransCAD*, a hierarchical transformer architecture for CAD sequence inference from point clouds. This work represents a significant advancement beyond the B-Rep analysis presented in the previous chapter by addressing the end-to-end reverse engineering pipeline directly from point cloud data to parametric CAD models. While CADOps-Net focused on recovering construction history from an already available B-Rep, *TransCAD* tackles the more challenging task of inferring a fully parametric sketch-extrude sequence from raw scan data, thereby addressing both the scan artifacts and design intent recovery challenges identified in Section 1.2. Our approach employs a novel hierarchical learning strategy that mirrors the structure of CAD operations and sketches, enabling more effective feature extraction and parameter prediction. Furthermore, we introduce evaluation methodologies specifically designed for the CAD reverse engineering domain, addressing a critical gap in existing assessment frameworks. Through comprehensive experiments, we demonstrate that *TransCAD* not only achieves state-of-the-art performance on standard benchmarks but also exhibits robustness to input perturbations characteristic of real-world

scanning artifacts. This contribution marks a significant step toward practical, industrial application of automated CAD reverse engineering from scanned data.

4.1 Introduction

As established in Chapter 1, Computer-Aided Design (CAD) plays a pivotal role in contemporary manufacturing, with virtually every manufactured object originating from a CAD model. Building upon our discussion of CAD representations in previous chapters, we now focus on *feature-based modeling* [74], which has emerged as the dominant paradigm for modern CAD design. This approach enables engineers to create and manipulate 3D models through a series of *features*, individual elements or operations such as holes, slots, and fillets, that progressively modify the geometry of a CAD model. The process typically begins with the design of *planar sketches*, collections of *loops* composed of 2D curves, followed by CAD operations (extrusion, revolution, etc.) that transform these 2D sketches into 3D solid models. The resulting CAD model is thus represented by a sequence of these sketches and operations, capturing both the final geometry and the design intent. Feature-based modeling has gained widespread adoption due to its intuitive representation of design intent and seamless integration with CAD software, making it particularly well-suited for the iterative development of complex designs.

The recent availability of large CAD model datasets, such as ABC [28] and Fusion360 [27], has catalyzed significant research interest in developing learning-based approaches for feature-based modeling. Much of this research has focused on deep generative modeling [26, 100, 25, 36], where transformer-based networks are trained to create new CAD models or automatically complete partial designs via autoregressive inference. While this generative direction offers numerous potential applications for CAD software integration, comparatively less attention has been given to the problem of *reverse engineering*—the focus of this thesis. While the previous chapter addressed the recovery of construction history from B-Rep models, this chapter tackles the more challenging problem of direct reverse engineering from 3D scans to parametric CAD sequences. This process begins with the acquisition of point clouds

or triangular meshes captured from physical objects using commercial 3D sensors, presenting significant challenges due to scanning artifacts and the inherent ambiguity in recovering design intent from geometric data alone.

Some existing reverse engineering approaches investigate the recovery of alternative CAD model representations like Constructive Solid Geometry [101] (CSG) or Boundary-Representation (B-Rep) [13, 14, 38]. Other methods tackle feature-based reverse engineering and predict implicit representations of sketches and CAD operations from point clouds [23, 24]. Nevertheless, such approaches do not allow for seamless integration into CAD software and often require post-processing (*e.g.* parametric curve fitting). To address these limitations, models capable of learning explicit CAD sequence of parametric sketches and operations from point clouds are needed. This can be enabled within a generative learning framework as in [26]. In that work, an auto-encoder reconstructing CAD parametric sequences is proposed and the latent representation is used for generating novel CAD sequences. An extension for reverse engineering was proposed by replacing the CAD sequence encoder with a point cloud encoder trained to map point clouds to the latent representations. The main limitation of the above is the predefined latent space that cannot adapt to the variations present in real-world point clouds. This disconnection can cause the model to generalize poorly to unseen inputs, especially those with noise or irregularities that are present in 3D scans and that are not well-represented in the training data.

To that end, we propose *TransCAD*, a novel end-to-end trainable and single-stage hierarchical network for feature-based explicit CAD sequence reverse engineering from point clouds. Our network is hierarchical in the sense that it employs a two-tiered decoding process. Initially, a primary CAD sequence embedding is decoded, encapsulating high-level features of the design, that are then processed by secondary decoders, one dedicated to loop parameters and another to CAD operations. Each decoder specializes in a certain input, enabling a nuanced and precise recovery of CAD parameters. The decomposition of learned representations matches the decomposition inherent in the actual feature-based design process of conceptualizing a 3D model through distinct loop and operation steps. Moreover, *TransCAD* does not predict sketch primitive types explicitly as in [26]; instead, we

employ a unified primitive representation where types are determined solely by coordinates. Our formulation narrows the learning space by eliminating syntactically incorrect predictions and facilitates a seamless transition between different primitive types. Additionally, it allows for a cascaded parameter refining that further enhances model performance.

Another focus of this chapter is the evaluation of parametric CAD sequence. We identify several limitations of the existing evaluation framework used by [26, 102] and suggest a suitable metric for CAD sequence similarity based on mean average precision, computed in the unquantized parametric space.

Contributions: In summary our contributions are the following:

1. We propose *TransCAD*, a novel hierarchical architecture for feature-based reverse engineering. Our model is single-stage and end-to-end trainable. *TransCAD* allows for a compact CAD sequence representation that does not include categorical types and enables cascaded coordinate refinement.

2. We identify several limitations of the existing evaluation framework for feature-based reverse engineering and propose a new evaluation metric framework to ensure fair comparison among diverse network architectures.

3. Our model surpasses the performance of recent generative-based approaches while also bridging the gap to real-world applications by exhibiting robustness to perturbed point clouds.

The rest of the chapter is organized as follows. Section 4.2 reviews related works. Section 4.3 formulates the problem of feature-based CAD reverse-engineering. The proposed *TransCAD* is described in Section 4.4. Discussion on the current evaluation framework and suggested extension is introduced in Section 4.5. An experimental validation of the proposed network is provided in Section 4.6. Finally, conclusions are given in Section 4.7.

4.2 Related Works

Generative Models for CAD: The advent of large-scale 3D shape datasets [103, 28, 27], combined with the significant progress for generative models in vision [104, 21, 105, 106], has sparked interest in the generation of 3D shapes. Existing methods have been proposed for various 3D representations, including point clouds [107, 108, 109], 3D meshes [110, 111], voxel grids [112, 113], and signed distance functions [114, 115]. This work focuses on CAD model generation, which compared to the above is parametric and directly editable in CAD software. A line of work explores the generation of the Boundary-Representation (B-Rep), a collection of parametric surfaces connected via a structured topological graph. SolidGen [92] considers B-Rep synthesis based on transformers and two-level pointer networks. Brep-Gen [116] represents a B-Rep via a fixed tree of latent geometry representations that can be generated by a diffusion model. Feature-based CAD generation has also been recently explored. Most relevant to our work is DeepCAD [26], a non-autoregressive generative model capable of synthesizing novel CAD sequences based on a transformer auto-encoder architecture. In [100, 36] the authors also follow autoregressive strategies. HNC [100] uses a hierarchical model based on high-level concepts and a code tree for CAD model generation and auto-completion. Similarly, in SkexGen [36] a transformer architecture is used to generate CAD models in the sketch-extrude format by encoding the topology and geometry using different codebooks. All the aforementioned works are oriented around 3D shape generation and either do not address the reverse-engineering task or address it via adaptation of generative modelling leading to suboptimal performance.

CAD Reverse Engineering: Reverse engineering is a well-studied problem with a substantial research effort directed towards predicting geometric features of CAD models, by analyzing the corresponding point clouds. Parametric fitting techniques infer the parametarization of edges [16, 117, 18, 19, 17, 20] and surfaces [96, 22]. Various attributes of the B-Rep and CAD operations are recovered from 3D scans in [118]. CADOps-Net [38] recovers 2D sketches from faces segmented into their CAD operation steps. Reasoning about a CAD model via properties discovered by parametric fitting offers insights solely into its end-state,

without considering the sequential CAD design process intrinsic to feature-based modeling.

A step closer to CAD reverse engineering, a line of work explores the reconstruction of a point cloud into Constructive Solid Geometry (CSG) [119, 101, 120], a modelling technique that uses boolean operations to combine primitives into 3D models. Point2Cyl [23], on the other hand, predicts extrusion cylinders given a point cloud, but requires user input to combine cylinders. SECAD-Net [24] and ExtrudeNet [25] use a self-supervised learning strategy to recover CAD sequences in the form of implicit representations given voxels and point clouds, respectively. In contrast to feature-based modelling, 3D representations produced by these methods (CSG, extrusion cylinders, etc) have limited compatibility with modern CAD software workflows. The authors in [121] learn sketch-extrude sequences conditioned on a voxel input, however, the model relies on strong data priors and is limited to predefined extrusion combinations.

Closer to our work is DeepCAD [26] and subsequent MultiCAD [102]. Even though DeepCAD [26] proposes a non-autoregressive generative framework for feature-based CAD, authors explore further conditioning on input point clouds. Taking a similar direction, Multi-CAD [102] proposes a two-stage multimodal contrastive learning strategy of both point clouds and CAD sequences. The two aforementioned methods opt for separate stage learning for point clouds and CAD sequences. Concurrent to our work, the autoregressive strategy in [39] and the multimodal diffusion based approach in [122] attempt to solve the point cloud to CAD sequence problem. To our knowledge *TransCAD* is the first non-autoregressive single-stage architecture for feature-based reverse engineering.

4.3 Problem Statement

A CAD model $\mathbf{C} \in \mathcal{C}$ is constructed in a sequence of construction steps. Each step can be seen as a 2D parametric *sketch* $\mathbf{s} \in \mathcal{S}$ (*e.g.* set of lines, arcs, etc) followed by a CAD *operation* $\mathbf{o} \in \mathcal{O}$ (*e.g.* extrusion, revolution, etc) [36, 26]. Here, $\mathcal{C}$ is the set of all possible CAD models, $\mathcal{S}$ and $\mathcal{O}$ represent the sets of possible CAD sketches and operations, respectively. CAD models constructed exclusively from the *extrusion* operation type are considered in

this work. Extrusion $\mathbf{e} \in \mathcal{E}$, where $\mathcal{E}$ denotes the set of possible extrusions, is the most common operation and enables the description of a wide range of CAD models [26, 36, 100]. *TransCAD* aims at learning how to predict the sequence of CAD construction steps from an input point cloud. Formally, given a point cloud $\mathcal{X} = [\mathbf{p}_1, \ldots, \mathbf{p}_n] \in \mathbb{R}^{n \times 3}$, where $\mathbf{p}_i = [x_i, y_i, z_i]$ denotes the 3D coordinates of the point i and n the number of points, the objective of *TransCAD* is to learn the mapping

$$\begin{aligned} \mathbf{\Phi} &: \mathbb{R}^{n \times 3} \rightarrow \mathcal{C} \ , \\ \mathbf{\Phi}(\mathcal{X}) &= \{\mathbf{s}_l, \mathbf{e}_l\}_{l=1}^{L} \ , \end{aligned} \tag{4.1}$$

where L denotes the length of the CAD sequence. In what follows, the proposed formulations of sketches and extrusions are described.

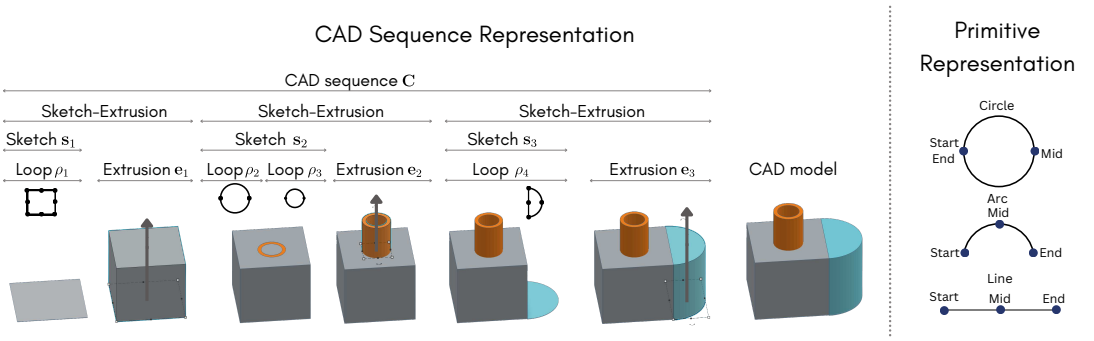


Figure 4.1: The sequential process of CAD modeling. A CAD sequence $\mathbf{C}$ can be decomposed into a hierarchical structure. The highest conceptual level is a sequence of sketch $\mathbf{s}$ and extrusion $\mathbf{e}$. A sketch can be made of one or more loops ρ. Each loop can be further decomposed into loop primitives, circle, arc and line. Each loop primitive can be described by a fixed number of parameters as shown on the right panel.

CAD Sketch and Extrusion Formulation: The proposed formulations for sketch and extrusion steps are inspired by [26, 36]. A sketch $\mathbf{s}$ is composed of one or more *loops* (see left panel of Figure 4.1). Each loop $\{\rho_j\}_{j=1}^{L_\rho}$, where L_ρ, denoting the number of loops, consists of one primitive (*i.e.* circle) or a combination of primitives (*i.e.* lines and arcs). In contrast to [26] which specifies the type of primitives in their representation, the pro-

posed primitive representation is type-agnostic (see right panel of Figure 4.1). In particular, each primitive δ is represented by three 2D coordinates of start, mid, and end points, *i.e.* $\delta = [(x_{start}, y_{start}), (x_{mid}, y_{mid}), (x_{end}, y_{end})] \in \mathbb{R}^6$. This representation has the advantage that the type of primitive can be deduced from the configurations of the points, hence reducing the search space and facilitating the transition between different types during training. In practice, the mid point of a line is replaced by a dummy value.

As in [26], we ensure that the loops are always closed by using the end point of a primitive as the start point of the next one. Further, a similar to [26] quantization is considered to reduce the parameters search space. As a result, a loop of n_p primitives $\rho_j \in \mathbb{R}^{6\times n_p}$ is considered as a quantized representation $\rho_j^\star \in [\![0..d_q]\!]^{6\times n_p}$, where d_q denotes the quantization interval. As for extrusion, similarly to [26], a quantized representation $\mathbf{e}_j^\star \in [\![0..d_q]\!]^{11}$ is considered to represent the sketch plane/scale and extrusion type/distances. Note that $\mathbf{e}^\star \in [\![0..d_q]\!]^{11\times L_e}$ and $\rho^\star \in [\![0..d_q]\!]^{6\times n_p\times L_\rho}$ will be used in the following to denote sequences of quantized extrusions and loops, respectively. Here, L_ρ and L_e denote the length of loop and extrusion sequences, respectively.

4.4 Hierarchical CAD Sequence Learning from Point Clouds

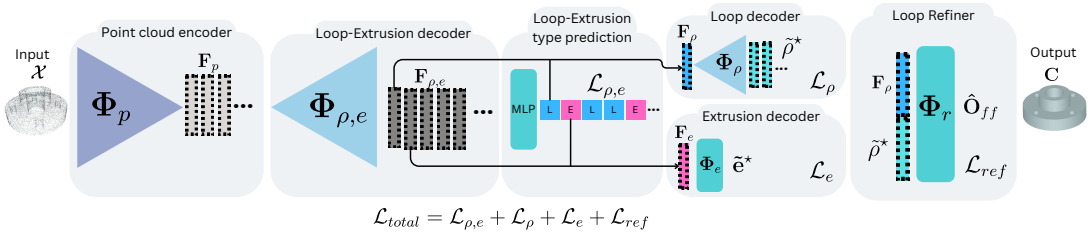


Figure 4.2: *TransCAD* model architecture. *TransCAD* is a hierarchical network composed of the following components: a point cloud encoder, a loop-extrusion decoder that predicts a high-level sequence which is then decoded by a loop decoder and an extrusion decoder. The predicted quantized loop parameters are then corrected by a loop refiner.

TransCAD non-autoregessively learns to predict a CAD sequence from an input point cloud in the format described in Section 4.3. First, the point cloud is encoded into point

features using a standard point cloud encoder. In order to facilitate the learning of CAD sequences, a hierarchical CAD sequence decoding is proposed. In particular, a high-level sequence of embedding corresponding to the loop and extrusion steps is learned. Those embedding are then fed to either a loop or extrusion decoder based on predicted type to learn the loop and extrusion parameters. Finally, the predicted loop parameters are further refined using the actual unquantized loop parameters (ground truth). The overall model architecture is depicted in Figure 4.2 and the different components are described below.

4.4.1 Point Cloud Encoder

The point cloud encoder $\mathbf{\Phi}_p$ consists of 4 layers of PointNet++ [53] and operates on an input point cloud $\mathcal{X}$. It outputs per-point features encoding local neighborhood information, $\mathbf{F}_p = [\mathbf{f}_p^1 \dots \mathbf{f}_p^n] \in \mathbb{R}^{n \times d_p}$, that encode local neighborhood information, d_p denotes the dimension of the features. Note that point normals of $\mathcal{X}$ are estimated using [123] and are provided as input to $\mathbf{\Phi}_p$ along with its 3D coordinates.

4.4.2 Loop-Extrusion Decoder

The main objective of the loop-extrusion decoder $\mathbf{\Phi}_{\rho,e}$ is to learn a high-level sequence of embedding $\mathbf{F}_{\rho,e} = [\mathbf{f}_{\rho,e}^1, \dots, \mathbf{f}_{\rho,e}^{L_{\rho,e}}] \in \mathbb{R}^{d_z \times L_{\rho,e}}$ corresponding to loops and extrusions from the point cloud features $\mathbf{F}_p$. Here, d_z and $L_{\rho,e}$ denote the embedding dimension and the length of the sequence, respectively. The decoder $\mathbf{\Phi}_{\rho,e}$ is composed of multi-head transformer-based blocks [64]. In the first block, learned constant embedding $\mathbf{F}_c \in \mathbb{R}^{d_z \times L_{\rho,e}}$ undergo a self-attention operation [64] and the resulting representation cross-attends to the point cloud features $\mathbf{F}_p$ to produce loop extrusion embedding for the first block $\mathbf{F}_{\rho,e}^1 \in \mathbb{R}^{d_z \times L_{\rho,e}}$ as follows,

$$\mathbf{F}_{\rho,e}^1 = \texttt{CA}(\texttt{SA}(\mathbf{F}_c), \mathbf{F}_p) \ , \tag{4.2}$$

where $\texttt{SA(.)}$ and $\texttt{CA}(.,.)$ denote the self and cross attention operators [64], respectively. The same self and cross attention operations are conducted in the subsequent blocks by feeding the output of each block as input to the next one,

$$\mathbf{F}^b_{\rho,e} = \texttt{CA}(\texttt{SA}(\mathbf{F}_{b-1}), \mathbf{F}_p) \ , \tag{4.3}$$

yielding the final sequence of embedding $\mathbf{F}_{\rho,e}$ after the last block. In order to ensure that each element $\mathbf{f}^i_{\rho,e}$ in the sequence embedding $\mathbf{F}_{\rho,e}$ corresponds to the right type (*i.e.* loop, extrusion, or end of sequence), a 3 layer `MLP` followed by `softmax` that operates on each $\mathbf{f}^i_{\rho,e}$ and predicts its type is introduced. A cross-entropy loss, $\mathcal{L}_{\rho,e}$, is computed between the predicted and ground truth types to supervise the learning of $\mathbf{F}_{\rho,e}$. Note that the loop-extrusion decoder is solely used to obtain a high-level sequence of loop and extrusion embedding. These embedding can then be decoded through either a loop decoder or an extrusion decoder to obtain their parameters. At training time, the ground truth type labels are used to identify which decoder should be used for each embedding, while at inference time the predicted types are used. The identification of loop and extrusion types results into separate loop embedding $\mathbf{F}_\rho \in \mathbb{R}^{d_z \times L_\rho}$ and extrusion embedding $\mathbf{F}_e \in \mathbb{R}^{d_z \times L_e}$ by splitting $\mathbf{F}_{\rho,e}$ according to loop and extrusion types.

4.4.3 Loop and Extrusion Parametrization

After obtaining the representation and the type of loop and extrusion steps, the parameters of both loops and extrusions are decoded from these representations using separate decoders.

Extrusion Decoder: As mentioned in Section 4.3, the extrusion sequence is described by a sequence of 11 quantized parameters $\mathbf{e}^\star \in [\![0..d_q]\!]^{11 \times L_e}$. In order to obtain these parameters from the extrusion sequence embedding $\mathbf{F}_e$, an extrusion decoder $\mathbf{\Phi}_e$ consisting of 3 `MLP` layers followed by `softmax` is used. The predicted probabilities of the extrusion sequence parameters $\tilde{\mathbf{e}}^\star \in [\mathbf{0}, \mathbf{1}]^{11 \times d_q \times L_e}$ are compared to the ground truth one-hot-encoded parameters in $\mathbf{e}^\star$ using a cross-entropy loss, $\mathcal{L}_e$.

Loop Decoder: Similarly to the extrusion decoder, the loop decoder $\mathbf{\Phi}_\rho$ predicts the quantized parameters of the loop sequence $\rho^\star \in [\![0..d_q]\!]^{6 \times n_p \times L_\rho}$ as explained in Section 4.3. Nevertheless, 4 layers of multi-head transformer blocks are employed instead of simple `MLP` layers. This is due to the sequential nature of loop decoding in contrast to extrusions. Note

that a similar strategy as loop-extrusion decoder is opted for the transformer block of loop decoder. The first block performs self-attention on learned constant embedding of loops $\mathbf{F}_c^{\rho} \in \mathbb{R}^{d_z \times n_p L_\rho}$ and the result cross-attends to loop embedding $\mathbf{F}_\rho$ as in Eq.(4.2). The same self and cross attention operations in Eq.(4.3) are conducted in subsequent blocks to yield a final representation at the last block $\mathbf{F}_\rho^b \in \mathbb{R}^{d_z \times n_p L_\rho}$. A linear layer followed by `softmax` is used to obtain predicted probabilities for the loop sequence parameters $\tilde{\rho}^\star \in [\mathbf{0}, \mathbf{1}]^{6 \times d_q \times n_p L_\rho}$ which are compared to ground truth one-hot-encoded loop parameters of $\rho^\star$ using a cross-entropy loss, $\mathcal{L}_\rho$.

Loop Refiner: As in many transformer-based architectures [26, 36, 124, 64], the quantization of loop parameters helps to reduce the search space and facilitates the learning. However, it has been observed in our case that it can lead to accumulation of quantization approximation errors. To overcome this issue, unquantized ground truth loop parameters are leveraged. In particular, a loop refiner $\mathbf{\Phi}_r$ composed of a 4 layer `MLP` is introduced. This refiner takes as input a concatenation of loop embedding $\mathbf{F}_\rho$ and their corresponding predicted parameter probabilities $\tilde{\rho}^\star$. It attempts to predict the offset $\hat{\mathbf{O}}_{ff} \in \mathbb{R}^{6 \times n_p L_\rho}$ between the predicted quantized loop parameters $\hat{\rho}^\star \in [\![0..d_q]\!]^{6 \times n_p L_\rho}$ and the unquantized ground truth loop parameters $\rho^\star \in \mathbb{R}^{6 \times n_p L_\rho}$. An MSE loss, $\mathcal{L}_r$, is computed between the predicted offset $\hat{\mathbf{O}}_{ff}$ and the one given by $\mathbf{O}_{ff} = \rho^\star - \rho$, to supervise the refiner and the rest of the network. Once the offset is predicted, it is added to the predicted quantized loop parameters yielding unquantized predicted loop parameters as follows $\hat{\rho} = \hat{\rho}^\star + \hat{\mathbf{O}}_{ff}$.

Total Loss: *TransCAD* is an end-to-end network with a training objective guided by the sum of the individual losses, $\mathcal{L}_{total} = \mathcal{L}_{\rho,e} + \mathcal{L}_\rho + \mathcal{L}_e + \mathcal{L}_r$.

4.5 Proposed Evaluation

In this section, we first outline the limitations of existing evaluation methods in CAD sequence. Then, our new proposed evaluation metric framework for assessing the performance of CAD sequence inference from point clouds is described.

DeepCAD[26] Evaluation for Feature-based Reverse Engineering: An evaluation frame-

work for CAD sequence was originally introduced in [26] and later used in [102]. This framework includes both accuracy for assessing the fidelity of the predicted sequence and Chamfer Distance (CD) to measure the quality of the recovered 3D geometry. Accuracy is assessed using two metrics, specifically *Command Type Accuracy* (ACC_{cmd}) and *Parameter Accuracy* (ACC_{param}) defined by

$$ACC_{cmd} = \frac{1}{N_c} \sum_{i=1}^{N_c} \mathbb{I}[t_i = \hat{t}_i] \ , \tag{4.4}$$

$$ACC_{param} = \frac{1}{K} \sum_{i=1}^{N_c} \sum_{j=1}^{|\hat{\boldsymbol{p}}_i|} \mathbb{I}\left[|\boldsymbol{p}_{i,j} - \hat{\boldsymbol{p}}_{i,j}| < \eta\right] \mathbb{I}[t_i = \hat{t}_i] \ , \tag{4.5}$$

where t_i and $\hat{t}_i$ are the ground truth and predicted command types (for commands representing primitives and extrusions), $\boldsymbol{p}_{i,j}$ and $\hat{\boldsymbol{p}}_{i,j}$ are ground truth and predicted command parameters, N_c denotes the total number of CAD commands and $\mathbb{I}[.]$ is the indicator function. $K = \sum_{i=1}^{N_c} \mathbb{I}[t_i = \hat{t}_i]|\boldsymbol{p}_i|$ is the total number of parameters of the correctly recovered commands and η is a tolerance threshold. The 3D geometry is evaluated with Chamfer Distance (CD) computed by sampling 2000 points on the ground truth and predicted shapes.

Limitations: We identify the following limitations of the aforementioned evaluation. **(1)** The proposed ACC_{cmd} overlooks the possibility of over-prediction in the CAD sequence. As indicated in Eq.(4.4), the computation of this metric sums across the set of ground truth CAD commands N_c. A predicted sequence could erroneously include extra loop-extrusion operations and still achieve a full score, as exemplified on the left panel of Figure 4.3. **(2)** The evaluation of ACC_{param} is conducted solely on the subset of K accurately identified commands, thus introducing a trade-off between ACC_{param} and ACC_{cmd}. This interdependence complicates the interpretation of results. **(3)** Assessment of parameter quality via ACC_{param} solely in terms of accuracy is failing to distinguish between the magnitudes of errors. A parameter inaccurately placed in an adjacent quantization bucket incurs the same penalty as one with a larger deviation, despite potentially minor implications on the CAD model's final geometry. These limitations cannot be entirely mitigated by complementing CAD command accuracies with the chamfer distance (CD) metric. While CD is a valuable assessment of

shape similarity, it does not address the core objective of reverse engineering: to accurately recover the designer's original CAD sequence. Two CAD models might be close in terms of CD yet possess vastly different CAD construction steps (see right panel of Figure 4.3).

Proposed Evaluation Framework: To overcome the identified challenges, we introduce the *mean Average Precision of CAD Sequence* (APCS), a novel evaluation metric tailored for feature-based reverse engineering. APCS adopts the concept of Average Precision (AP) commonly used in other tasks, to quantify the similarity between predicted and ground truth CAD sequences. We introduce the *CAD Sequence Similarity Score* (CSSS) that can be computed between predicted and ground truth CAD sequences as follows

$$CSSS(\hat{\mathbf{C}}, \mathbf{C}) = \frac{1}{2N_\delta} \sum_{j=1}^{N^\rho} \sum_{i=1}^{N_j^\rho} \left[S(\hat{\delta}_{j,i}, \delta_{j,i}) \cdot \mathbb{I}[t_{yp}(\hat{\delta}_{j,i}) = t_{yp}(\delta_{j,i})] \right] + \frac{1}{2N^e} \sum_{j=1}^{N^e} S(\hat{\boldsymbol{e}}_j, \boldsymbol{e}_j) \ , \tag{4.6}$$

where $t_{yp}(.)$ is a function that determines the type of each primitive δ (arc, line, etc), and $S(\hat{\boldsymbol{p}}, \boldsymbol{p}) = e^{-k||\hat{\boldsymbol{p}}-\boldsymbol{p}||}$ is a scoring function with $S(\hat{\boldsymbol{p}}, \boldsymbol{p}) \in [0, 1]$ with 1 assigned when predicted parameterization is identical to the ground truth. We define $N_j^\rho = \max(|\rho_j|, |\hat{\rho}_j|)$ where $|.|$ denotes set cardinality, $N^\rho = \max(L_\rho, \hat{L_\rho})$ where L_ρ is the number of predicted primitives and $\hat{L}_\rho$ is the number of ground truth primitives for loop ρ_j and $N_\delta = \sum_{j=1}^{N^\rho} \max(|\rho_j|, |\hat{\rho}_j|)$. Finally, $N^e = \max(L_e, \hat{L_e})$ where L_e is the number of predicted extrusions and $\hat{L}_e$ is the number of ground truth extrusions. The proposed CSSS metric evaluates both the operation type and parameter prediction. It assigns a score of 0 to loops with incorrectly predicted types, which gradually increases to 1 as parameter prediction improves. Assessment is conducted on the unquantized parameter space and calculates the score based on the maximum count of either predicted or ground truth primitives. This approach ensures that both over and under predicted sequences are penalized equally. We aggregate CSSS scores across various thresholds to derive the *mean Average Precision of CAD Sequences* (APCS). Furthermore, the median CD is used to measure shape similarity as in [26] with the difference that it is evaluated on 4096 points instead of 2000 in order to decrease the uncertainty in the CD measurement. All the reported CD measurements in this work are multiplied by 10^3.

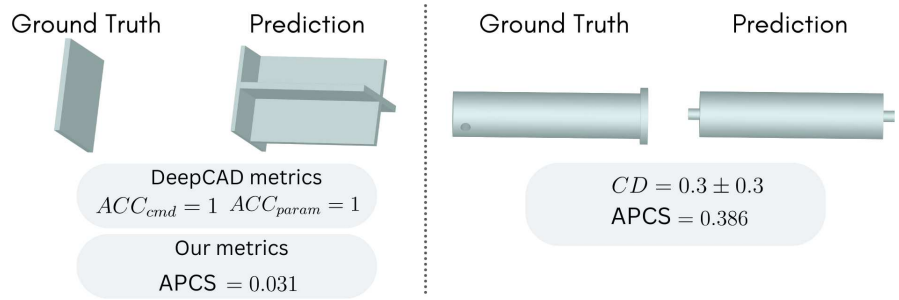


Figure 4.3: Two examples outlining the limitations of existing evaluation metrics. Left panel: the ground truth sequence (one sketch-extrusion) is a correct subset of the predicted sequence (three sketch-extrusions). The DeepCAD [26] metrics result in an accuracy of 1 for both commands and parameters. On the other hand, our proposed metric takes into account the over predicted sequence elements and the APCS is 0.031. The right panel showcases the limitations of the CD as a similarity measure. While the ground truth and predicted shapes are both composed of three extruded circle sketches, they are different in shape. However, the CD between the two shapes falls within the uncertainty range of ± 0.3. Note that the uncertainty in the CD measurement is estimated by taking the average CD between all the test samples and themselves.

Moreover, the ratio of predictions that cannot be reconstructed using [125] is reported as the invalidity ratio, IR.

4.6 Experiments

In this section, the experimental setup is first presented. Then, qualitative and quantitative results are analyzed. Afterwards, the components of *TransCAD* are ablated. Finally, the limitations of our model are outlined.

4.6.1 Experimental Setup

Dataset: For training and evaluation, the DeepCAD dataset [26] is used. The sketch extrusion sequences of the CAD models are processed in quantized $(8$ bits) and unquantized space. The size of the train, validation and test sets are $140\,294$, $7\,773$, and $7\,036$ CAD models, respectively. Moreover, cross-dataset evaluation is conducted on the Fusion360 dataset [27]

that contains $6\,794$ samples.

Training Details: The network is trained for 100 epochs with a batch size of 72 and an Adam optimizer is employed with a learning rate of 0.001 and a linear warm-up period of $2\,000$ steps as in [26]. The loop-extrusion decoder $\mathbf{\Phi}_{\rho,e}$ and the loop decoder $\mathbf{\Phi}_{\rho}$ are both transformer decoder with the same network architecture. They are both made of 4 layers, each made of 8 heads with a feed-forward dimension of 512. A dropout rate of 0.1 is used. The point encoder $\mathbf{\Phi}_{p}$ is PointNet++ [53]. The implementation provided in [126] was used. The input dimension $d_p = 6$ corresponds to the point and normal coordinates. The parameters for the 4 layers are as follows: number of points (512, 256, 128, 16) with radius (0.1, 0.2, 0.4, 0.8) and number of samples (64, 64, 64, 32). The training is conducted on an NVIDIA RTX $A6000$ GPU.

The input point clouds are extracted using [125] and are made of $n = 4096$ points. The dimension of the point features d_p is set to 16 and of loop-extrusion features d_z to 256.

Baselines: In order to evaluate the performance of *TransCAD*, two state-of-the-art methods, MultiCAD [102] and DeepCAD [26], and a retrieval baseline are used. As the code for MultiCAD [102] is not available, we report the results from the original paper. DeepCAD [26] is retrained with the same parameters and procedure as outlined in the original paper. One of the known limitations of the DeepCAD dataset is that it contains duplicate models [36]. While the works in [36] proposed a method to remove duplicate models that contain exactly the same CAD sequence (*i.e. sequence duplicates*), we find that this method does not remove all the duplicates as some models can have the same geometry but are constructed through a slightly different sequence of sketch extrusion operations (*i.e. geometrical duplicates*). Moreover some CAD models in the DeepCAD train set are almost identical to some models in the Fusion360 dataset [27], with often just the amount of extrusion varying slightly. Figure 4.4 shows examples of the different types of duplicates from the DeepCAD [26] and Fusion360 [27] datasets. We define a geometrical duplicate as a test set CAD model for which it exists a CAD model in the train set with a chamfer distance less than the uncertainty in the chamfer distance measurement ($\pm 3 \times 10^{-4}$ when 4096 points are sampled). From this definition and using the train set of the DeepCAD dataset [26], we observe that about 14% of the DeepCAD test set is made of geometrical duplicates and about 12% in the Fusion360

test set.

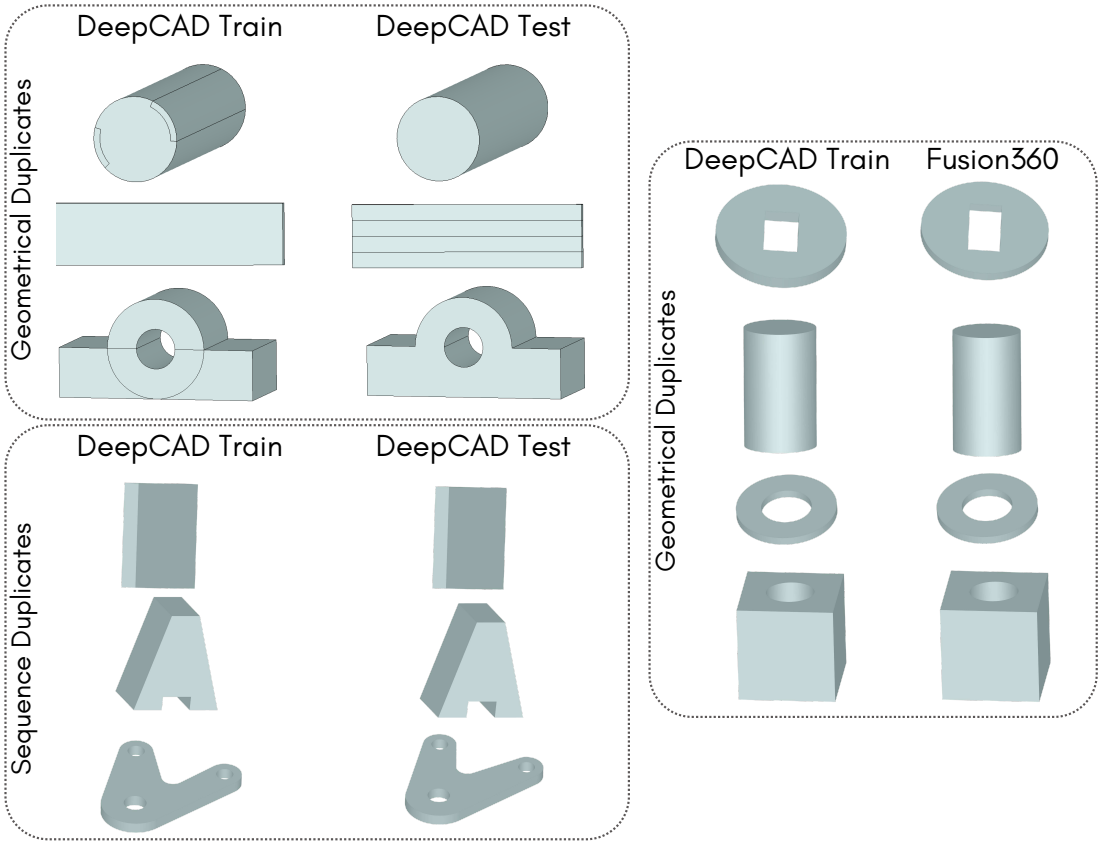


Figure 4.4: Examples of duplicate CAD models from the DeepCAD [26] and Fusion360 [27] datasets. On the top left panel, CAD models from the DeepCAD train set with geometrical duplicates in the test set are shown. Similarly, the right panel presents geometrical duplicates present in the Fusion360 [27] dataset. CAD models with identical CAD sequences, *i.e.* sequence duplicates, are displayed in the bottom left panel.

In order to address this issue, we propose a retrieval baseline. The retrieval baseline uses the point cloud encoder from a trained DeepCAD [26] to identify the closest latent vector from the train set for each test sample. As a result, the solution is always a train set CAD sequence.

4.6.2 Experimental Results

Qualitative Results: Figure 4.5 shows some qualitative results for the retrieval baseline, DeepCAD [26] and *TransCAD* (Ours) on both the DeepCAD [26] and Fusion360 [27] datasets. As mentioned in Section 4.6.1, the DeepCAD dataset contains many duplicates, not just in terms of CAD sequence but also in terms of geometrical shape. As a result, the retrieval baseline is able to identify accurately duplicates (most right column of the DeepCAD dataset panel) and in other cases the baseline manages to retrieve shapes with similar geometry as the ground truth CAD model. On the other hand, it can be noticed that DeepCAD [26] can even fail at retrieving duplicates. *TransCAD* is able to predict models that are similar in shape and also in terms of loop-extrusion sequence, even though it sometimes fails to predict the parameters accurately (second and fifth columns of DeepCAD dataset and fourth column of Fusion360 dataset). Further qualitative results can be found in the Appendix B.

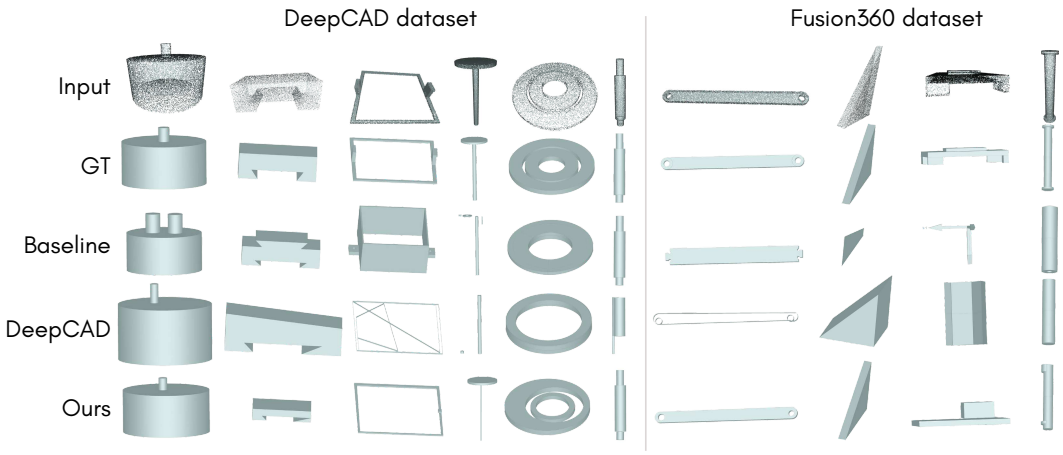


Figure 4.5: Qualitative results on the DeepCAD [26] dataset (left) and the cross-dataset Fusion360 [27] experiment (right).

Quantitative Results: The trends observed from the qualitative results are further supported by the quantitative results presented in Table 4.1. The APCS, on both DeepCAD [26] and Fusion360 [27] datasets, show that *TransCAD* is the most capable model at predicting correct CAD sequences. However, it can be noted that the retrieval baseline obtains the lowest

	Model	APCS↑	$CD\downarrow$	IR↓
DeepCAD	*Retrieval*	0.629	2.8	0
	MultiCAD [102]	-	8.1	0.115
	DeepCAD [26]	0.604	19.2	0.038
	Ours	0.732	4.5	0.011

	Model	APCS↑	$CD\downarrow$	IR↓
Fusion360	*Retrieval*	0.304	60.0	0
	MultiCAD [102]	-	42.2	0.165
	DeepCAD [26]	0.360	104.2	0.017
	Ours	0.365	33.3	0.024

Table 4.1: Quantitative results on the DeepCAD [26] dataset and cross-dataset experiment on Fusion360 [27]. The APCS results show that *TransCAD* (*Ours*) is able to recover CAD sequence most accurately.

Model	APCS↑						
	Line	Arc	Circle	Ext.	Origin	Orientation	Size
Retrieval	0.584	0.280	0.655	0.716	0.666	0.819	0.691
DeepCAD [26]	0.654	0.246	0.587	0.872	0.825	0.928	0.848
Ours	0.665	0.709	0.683	0.818	0.768	0.879	0.778

Table 4.2: Results of the different APCS components on the DeepCAD dataset [26].

CD by a small margin on the DeepCAD dataset and by a more significant margin on the Fusion360. One of the reasons is that this baseline always outputs a CAD model that is of roughly similar shape as the input even if the retrieved CAD sequence can vary from the ground truth.

Table 4.2 shows the APCS scores on the DeepCAD dataset [26] for each of the components averaged over the test set. It can be observed that *TransCAD* (*Ours*) obtains a significantly better score for the arc primitive and also to some extent for the circle primitive compared to the retrieval baseline and DeepCAD [26]. However, the scores corresponding to the placement of the 2D sketch in 3D (Origin, Orientation and Size) for *TransCAD* are slightly lower than for DeepCAD [26]. This trend can also be observed on the cross-dataset evaluation using the Fusion360 dataset [27] (see Table 4.3). Furthermore, the APCS score corresponding to the line primitive is significantly higher for *TransCAD* than for the other two baseline models.

To further analyse the results, the variations of the APCS and CD *w.r.t.* model complexity on the DeepCAD dataset are displayed in Figure 4.6. We define the model complexity as the lowest possible CD of a test point cloud sample with respect to the train samples. In

Model	APCS↑						
	Line	Arc	Circle	Ext.	Origin	Orientation	Size
Retrieval	0.214	0.045	0.546	0.453	0.294	0.509	0.451
DeepCAD [26]	0.368	0.144	0.639	0.844	0.774	0.866	0.859
Ours	0.515	0.383	0.622	0.749	0.693	0.836	0.782

Table 4.3: Results of the different APCS components on the Fusion dataset [27].

other words, the model complexity quantifies the amount by which a test sample is out of distribution from the train set in terms of shape. While *TransCAD* consistently outperforms on average all other baselines in terms of APCS for all model complexities, DeepCAD [26] can only perform better than the retrieval baseline for the more complex models. This shows that DeepCAD [26] often fails at retrieving the CAD sequence for simple models. In terms of CD, it can be observed that *TransCAD* and the retrieval baseline have similar performance. *TransCAD* can predict a CAD sequence that is closer to the ground truth one but the predicted overall shape can vary from the ground truth for more difficult samples.

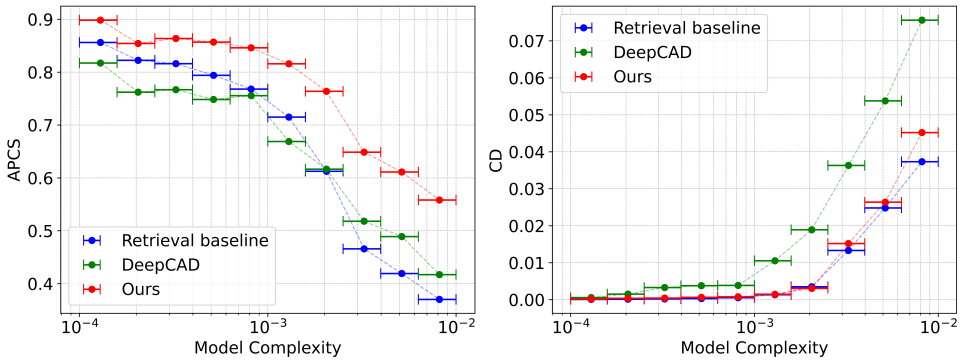


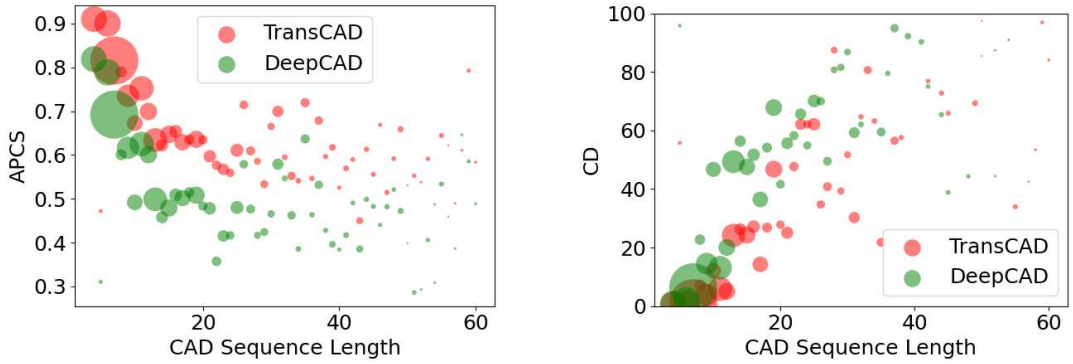
Figure 4.6: Left: Plot of the variation of the mean APCS *w.r.t.* model complexity. Right: Plot of the variation of the median CD *w.r.t.* model complexity. For both graphs the number of models in each bin, represented by the horizontal bounded lines, correspond to approximately the same number of test samples from the DeepCAD dataset [26].

Complex shape performance: Figure 4.7a show the APCS *w.r.t.* the sequence length for *TransCAD* and DeepCAD [26]. Similarly, Figure 4.7b presents the variation of CD *w.r.t.* the sequence length. The size of the data points is proportional to the number of models it represents. The performance decreases for both models as the length of the CAD sequence

Model	APCS↑	$CD\downarrow$	IR↓
Ours w/o hier.	0.687	7.0	0.016
Ours w/o refining	0.708	4.8	0.018
Ours	0.732	4.5	0.011

Table 4.4: Ablation results demonstrating the relevance of the hierarchical learning strategy and the loop refiner.

increases. However, *TransCAD* consistently outperforms DeepCAD.



(a) (b)

Figure 4.7: Comparison of (a) APCS and (b) CD metrics as functions of CAD sequence length expressed in the DeepCAD format on the DeepCAD dataset [26]. The size of the points is proportional to the number of CAD models with the corresponding CAD sequence length.

4.6.3 Ablation Study

In this section, the different components of the proposed network architecture are ablated. Table 4.4 shows the results for *Ours w/o hier.*, in which the learning is done without both the loop-extrusion decoder and the refining network, *Ours w/o refining* where the refining component is ablated and *Ours*. It can be noted that each component leads to an improvement in all the metrics.

The Loop-Extrusion module classifies the features $\mathbf{F}_{\rho,e}$ as *loop*, *extrusion* or *end of*

Model	APCS↑						
	Line	Arc	Circle	Ext.	Origin	Orientation	Size
Ours w/o hier.	0.569	0.473	0.529	0.749	0.640	0.843	0.670
Ours w/o refining	0.655	0.625	0.673	0.755	0.679	0.838	0.703
Ours	0.665	0.709	0.683	0.818	0.768	0.879	0.778

Table 4.5: Results of the different APCS components on the DeepCAD dataset [26] for the ablation study. *Ours w/o hier.* corresponds the proposed model without hierarchical learning, and Loop Refiner and *Ours w/o refining* to the proposed model without the Loop Refiner.

sequence type. These predictions are used to route the features $\mathbf{F}_{\rho,e}$ to either the loop decoder $\mathbf{\Phi}_\rho$ or extrusion decoder $\mathbf{\Phi}_e$ to obtain their parameters. It is worth noting that the $F1$ score on the loop-extrusion type prediction introduced for the hierarchical learning of *TransCAD* is 0.79. This implies that on most cases both loop and extrusion decoders receive embedding of the correct type. To demonstrate the impact of the Loop-Extrusion type classification, we conduct the following experiment: the ground truth Loop-Extrusion type labels are used instead of the predicted ones at testing time on the DeepCAD dataset [26]. In this scenario, the APCS metric evaluating the CAD sequence increases from 0.732 to 0.790. The *IR* also improves and decreases to nearly 0% (only 2 invalid models). Notably, there is no significant change in the *CD*. As a result, *TransCAD* is robust to moderate classification errors in the Loop-Extrusion prediction *w.r.t.* the final reconstruction. However, these errors might impact the performance of the predicted CAD sequence *w.r.t.* the ground truth. This suggests that the Loop-Extrusion classification errors might result in alternative yet plausible design paths.

Moreover, while the refining network is only applied on the loop parameters, we observe that the component of the APCS for the extrusion parameters are also higher when the network is trained with the refining component (see Table 4.5). The Loop Refiner is a component of the end to end pipeline of *TransCAD*. As a result, the backprogation of the Loop Refiner loss $\mathcal{L}_\rho$ acts on all the parameters of the network and can therefore impact the predictions at all levels.

4.6.4 Input point cloud perturbation

Reverse engineering is a real-world practical problem. The results in the previous section are obtained from sampling points from the B-Rep representations of the CAD models. While modern 3D sensors can reconstruct the mesh of models with high resolution, they still suffer from some artifacts such as noise and small missing parts. In order to evaluate the performance of our network in such realistic conditions, we run experiments in two scenarios, one in which noise is added and one in which small holes are created on the point cloud. In order to simulate realistic noise, Perlin noise [127] is added to the mesh from which the point coordinates and normals are extracted.

Perlin Noise Implementation: To simulate the noise created when an object is scanned using a 3D sensor, a Perlin noise [127] is added to the point cloud. The perlin noise is created using the following strategy. Starting from the mesh representation of the original CAD models, the faces are divided to ensure that the mesh contains a dense number of vertices. Then, a 3D Perlin noise is computed for each vertex using 64 octaves with a minimum and maximum magnitude of -0.001 and 0.001, respectively. Finally, the normals are recomputed from the mesh and points are sampled. A visual example of a perturbed mesh can be found in Figure 4.8.

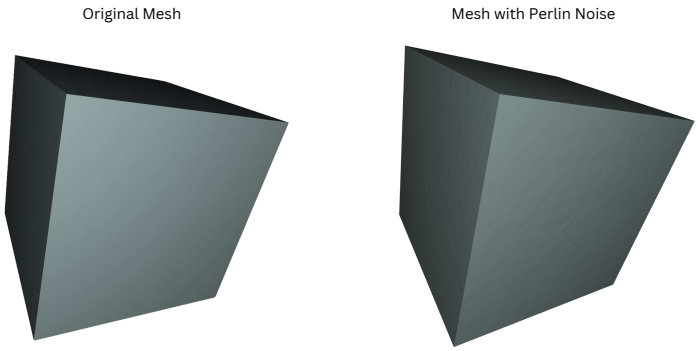


Figure 4.8: Example of a CAD model with its mesh representation (left) and perturbed representation with Perlin noise (right). Zooming on the figure might be required to best view the effect of the Perlin noise applied.

Holes Implementation: Holes in the input point clouds are created using the following

strategy. Firstly, for each point cloud the number holes is selected from a uniform distribution ranging from 1 to 10 included. Then, the ratio of points to be removed for each hole is chosen by sampling a normal distribution with mean 0.03 and standard deviation 0.015. Finally, for each hole a point is chosen at random and the corresponding number of nearest neighbors points are removed. The nearest neighbors are identified using a geodesic distance computed on the mesh surface. We ensure that the remaining number of points is at least $n = 4096$, which corresponds to the number of points used as input. Figure 4.9 shows some examples of point clouds on which holes have been created.

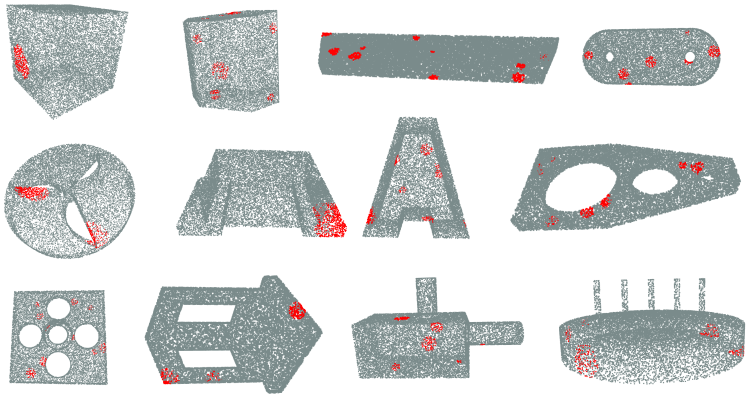


Figure 4.9: Examples of input point clouds in which holes have been created. The points highlighted in red represent the points that have been removed.

Qualitative & Quantitative Results: Qualitative results for both the hole and noise input perturbations can be found in Figure 4.10. Table 4.6 shows that *TransCAD* is more robust to such perturbations than other methods. As the noise also adds a disturbance to the direction of the input point normals, it leads to a larger drop in performance.

In order to further evaluate the sensitivity of our method to surface noise, we conduct an experiment in which the maximum magnitude of the Perlin noise on the input point cloud is increased compared to the one previously presented (now referred to as *Original Noise*). The results presented in Table 4.7 show that both *TransCAD* and DeepCAD are sensitive to the amount of noise. We also perform 2-epoch finetuning of both models on

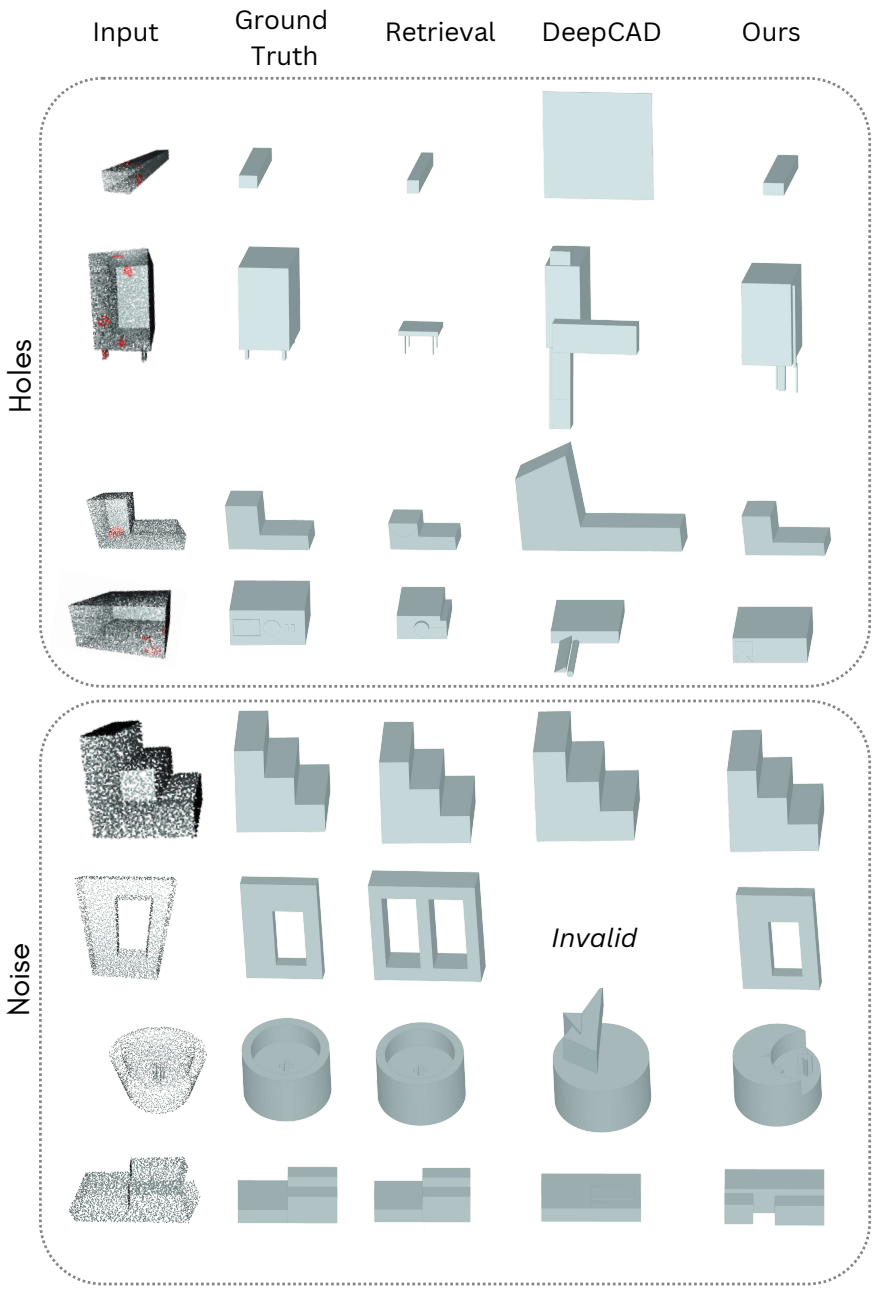


Figure 4.10: Qualitative results on DeepCAD dataset [26] using perturbed input point clouds. The top panel shows results for which holes are created on the input point cloud and the bottom panel shows results for which Perlin noise was applied.

	Model	APCS↑	$CD\downarrow$	IR↓
Noise	*Retrieval*	0.561	5.9	0
	DeepCAD [26]	0.550	31.1	0.047
	Ours	0.604	18.1	0.010

	Model	APCS↑	$CD\downarrow$	IR↓
Holes	*Retrieval*	0.607	4.1	0
	DeepCAD [26]	0.574	26.7	0.039
	Ours	0.732	4.4	0.012

Table 4.6: Results on the DeepCAD [26] dataset when the input point cloud is perturbated either with Perlin noise (left) or by creating holes (right).

Original Noise training data and report the results (*Finetune*, last row of Table 4.7). Notably, *TransCAD* almost recovers its performance with finetuning.

Noise increase	*TransCAD*		DeepCAD [26]	
	$APCS\uparrow$	$CD\downarrow$	$APCS\uparrow$	$CD\downarrow$
Original Noise	0.604	18.1	0.550	31.1
$+25\%$	0.570	21.1	0.541	31.6
$+50\%$	0.532	28.8	0.528	35.6
Finetune	0.724	5.1	0.572	27.7

Table 4.7: Results on the $APCS$ and CD metrics for different amount of noise added to the input point cloud on the DeepCAD dataset [26].

Furthermore, we conduct an experiment to evaluate the effect of the input cloud sparsity on the performance. Using *TransCAD* and DeepCAD [26] trained with input point clouds of size 4096 points, predictions for the test set with decreasing input point cloud sizes are generated. The results demonstrate that *TransCAD* is more robust than DeepCAD [26] *w.r.t.* input sparsity (see Table 4.8).

Input size (# Points)	*TransCAD*		DeepCAD[40]	
	$APCS\uparrow$	$CD\downarrow$	$APCS\uparrow$	$CD\downarrow$
4096	0.732	4.5	0.604	19.2
2048	0.729	4.9	0.577	24.8
512	0.705	8.1	0.526	38.5
256	0.673	12.7	0.443	70.3

Table 4.8: Results on the $APCS$ and CD metrics for different size of input clouds on the DeepCAD dataset [26].

4.6.5 Failure Cases and Limitations

In this section, we describe the reasons that lead *TransCAD* to predict invalid CAD models as measured by IR. Among the predictions of *TransCAD* on DeepCAD test set, only one contains a loop parametrization that results in an invalid CAD sequence. This shows that the proposed representation of the loop sequence leads to syntactically correct loops on practically all cases. However, in some cases the representation of the CAD sequence can be syntactically valid, yet it is not possible to reconstruct a B-Rep from it. For 75 test samples, the loop-extrusion decoder fails to predict an extrusion token, this implies that the predicted model is therefore an infinitely thin sketch and not a 3D model as expected. The rest of the invalid models are mostly due to a loop being made of a single line within a model, which cannot be extruded into a valid 3D shape within our context. Finally, Figure 4.11 shows examples for which the predicted sequences do not lead to a shape that is close to the ground truth one. In these examples, the input CAD models contain a large number of small features that *TransCAD* is unable to capture.

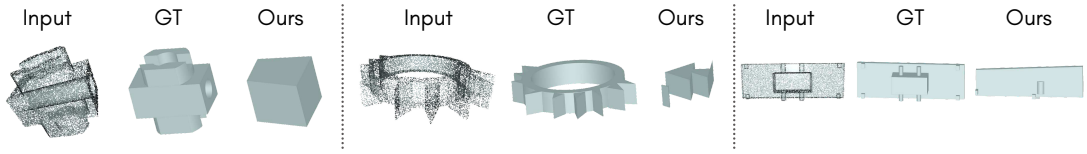


Figure 4.11: Examples for which *TransCAD* fails at recovering a shape close to the ground truth one.

4.7 Conclusion

In this chapter, we presented *TransCAD*, an end-to-end transformer-based neural network that addresses a critical challenge in CAD reverse engineering: the direct inference of parametric CAD sequences from point cloud data. Building upon the construction history recovery approach introduced in the previous chapter, *TransCAD* represents a significant advancement by eliminating the need for an intermediate B-Rep representation, thus addressing the

complete reverse engineering pipeline from scan to editable CAD model. Our approach introduces two key technical innovations: a hierarchical architecture that enables more effective learning of high-level loop-extrusion sequences, and a specialized loop refiner that significantly improves parameter prediction accuracy. Additionally, we proposed a unified primitive representation where all sketch elements are described by the same number of parameters.

Beyond architectural contributions, we identified fundamental limitations in existing evaluation methodologies for CAD sequence recovery and introduced a new metric, Average Precision of CAD Sequence (APCS), which enables more meaningful comparison of parametric CAD sequence predictions. This evaluation framework addresses the specific requirements of the reverse engineering domain, considering both geometric fidelity and design intent recovery.

Through comprehensive experimentation, we demonstrated that *TransCAD* achieves state-of-the-art performance on standard benchmarks while also exhibiting robustness to input perturbations characteristic of real-world scanning artifacts. These results validate our approach to hierarchical feature extraction and parameter prediction, while also highlighting the effectiveness of our proposed evaluation methodology. Despite these advances, our experiments revealed limitations in handling complex models with numerous small features, pointing toward directions for further research that we explore in subsequent chapters.

Chapter 5

abraCADabra: Inference Geometry Guided Search for Auto and Non Auto-Regressive Scan-to-CAD Networks

In this chapter, we introduce abraCADabra, a novel approach that extends our work on CAD sequence inference by implementing geometry-guided search strategies for both auto-regressive and non-auto-regressive architectures. While the previous chapter established the foundation of end-to-end point cloud to parametric CAD sequence translation through TransCAD, this work directly addresses two of the fundamental challenges identified in Section 1.2: the 3D scanning imperfections that corrupt geometric fidelity of input data, and the recovery of design intent when multiple valid interpretations are possible. By explicitly acknowledging the ambiguity inherent in scan-based reverse engineering, our approach mimics how expert CAD designers work—considering multiple design alternatives before selecting the most appropriate one. We develop inference-time search strategies that leverage geometric priors from input point clouds to more effectively explore the output probability space, significantly improving reconstruction quality while reducing invalid predictions. Additionally, to address

the data challenge highlighted in Section 1.2, we introduce the C3D-Recon dataset, the first collection of realistic 3D scans with authentic scanning artifacts paired with their corresponding parametric CAD sequences. This contribution represents a significant advancement toward practical reverse engineering applications by bridging the gap between the theoretical capabilities of neural networks and the real-world challenges of inferring CAD design from imperfect scan data.

5.1 Introduction

As established in Chapter 1, Computer-Aided Design (CAD) serves as the foundation of modern industrial manufacturing processes. While previous chapters have addressed various aspects of CAD reverse engineering—the process of creating 3D digital representations from existing physical objects—this chapter focuses on a fundamental characteristic of expert CAD design: the exploration of multiple design alternatives. In professional practice, reverse engineering plays a crucial role across diverse sectors, from recreating legacy components in the automotive industry to developing custom medical implants from anatomical scans and preserving cultural heritage artifacts through digital reconstruction [128]. However, this process traditionally demands significant expertise and time as designers must systematically explore numerous plausible design paths to achieve optimal results. The exploration phase is particularly challenging for complex geometries, requiring specialized technical knowledge to identify the most appropriate modeling approach. The methods presented in previous chapters have established foundations for automating parts of this process, but they have primarily focused on generating single solutions rather than emulating the exploratory nature of expert design practice. This chapter addresses this limitation by introducing search strategies that enable neural networks to consider multiple design alternatives, more closely mimicking human expertise while further advancing the automation of reverse engineering to reduce development cycles, minimize design costs, and enable innovation through improved understanding of existing designs [9].

The field of automated 3D reverse engineering traces its roots back several decades in

the computer vision and graphics communities [129, 130]. Prior to deep learning approaches, researchers developed a variety of computational techniques to translate physical objects into digital CAD models. These methodologies followed two primary strategies: first, the decomposition of point cloud data into distinct CAD components through classical segmentation algorithms [131], and second, the approximation of point cloud geometry using parametric surface fitting techniques [11]. While these traditional approaches established important foundations, they often required extensive manual intervention and struggled with complex geometries that exhibit intricate features and relationships.

Recent innovations in neural network architectures have enabled learning-based approaches for 3D reverse engineering from 3D scans of physical objects [39, 37]. Unlike their predecessors, these methods leverage datasets of design sequences created by human experts [26, 27] to learn the CAD modeling process in an end-to-end fashion. By observing how professional designers construct CAD models step by step, these neural networks can infer the underlying procedural logic and replicate similar construction sequences. While showing promising results, these approaches have a fundamental limitation: they generate a single solution without exploring the diverse design possibilities that human experts would consider. This is analogous to a designer who provides the first answer that comes to mind without careful consideration.

This limitation of existing deep learning approaches contrasts sharply with how expert CAD designers actually work—they mentally explore multiple design alternatives before selecting the most appropriate one. Interestingly, recent breakthroughs in artificial intelligence have demonstrated a potential solution to this problem. Systems like AlphaGo [132] and ChatGPT-o1 [133] have shown that neural networks' probability distributions can serve as effective guides through complex solution spaces. Their success stems from methodical exploration of these distributions, rather than accepting the first plausible output, which yields significant performance improvements. The underlying principle is that the model's uncertainty, encoded in its output probability distribution, contains valuable information about alternative solutions that can be systematically explored.

Drawing inspiration from these advances, we therefore pose the following question: *How*

can we leverage inference search strategies guided by geometric priors from input point clouds to improve CAD reverse engineering? To address this question, we propose extending existing auto-regressive (CAD-SIGNet [39]) and feedforward (TransCAD [37]) approaches by implementing advanced inference search strategies. Specifically, for the task of reconstructing CAD design sequences from point clouds, we leverage the geometric priors of the input to guide the inference search. This approach represents an important step toward emulating the expert reasoning process of CAD designers within deep neural networks, moving beyond single-solution methods toward systems that can explore and evaluate multiple design possibilities.

While our inference search strategies address the limitations of current approaches, we recognize that 3D reverse engineering is a real-world problem and therefore requires realistic data to be effective. To bridge this gap between theoretical advancements and practical implementation, we also propose the CC3D-Recon dataset; the first dataset to contain pairs of 3D scans and their corresponding CAD design sequences. This dataset serves as a critical complement to our methodological contribution, as it enables proper evaluation under conditions that more faithfully represent real-world reverse engineering challenges. Indeed, in practical scenarios, 3D scans contain numerous artifacts such as noise, occlusion, and smoothed edges [41]. These imperfections test not only a model's ability to reconstruct ideal geometry but also its robustness in inferring design intent from imperfect data. By developing both advanced search strategies and more realistic evaluation data, our work addresses two key limitations currently preventing automated 3D reverse engineering from achieving practical utility in industrial applications.

In summary, our contributions are as follows:

- A comprehensive analysis and comparison of auto-regressive and non-auto-regressive approaches for CAD reverse engineering, revealing their complementary strengths and limitations across different evaluation metrics.
- Development of novel geometry-guided inference search strategies that leverage input point cloud characteristics to explore the output probability space more effectively,

significantly improving reconstruction quality and reducing the rate of invalid CAD predictions.

- Introduction of the CC3D-Recon dataset, the first dataset containing pairs of realistic 3D scans with authentic scanning artifacts and their corresponding parametric CAD design sequences, enabling more robust evaluation of reverse engineering methods under real-world conditions.
- Extensive experimental validation demonstrating that inference search strategies can achieve substantial improvements in reconstruction accuracy while eliminating invalid predictions.

The rest of this chapter is organized as follows. Section 5.2 reviews related work in CAD reverse engineering and datasets. In Section 5.3, we formally define the problem of CAD sequence inference from point clouds. Section 5.4 details our methodology, presenting both the autoregressive and non-autoregressive baseline architectures and our proposed geometry-guided inference search strategies. In Section 5.5, we introduce the CC3D-Recon dataset, describing its creation process, properties, and unique characteristics that support realistic evaluation. Section 5.6 presents comprehensive experimental results across multiple datasets, comparing different search strategies and analyzing their impact on reconstruction quality. Finally, Section 5.7 summarizes our contributions and discusses directions for future research.

5.2 Related Works

5.2.1 CAD Reverse Engineering

The field of CAD reverse engineering has evolved along several distinct approaches, each addressing the fundamental challenge of non-unique representation in different ways. These approaches can be broadly categorized based on their underlying representation and methodology.

Geometry-Based Methods: Early approaches focus on uncovering the underlying geometry through parametric fitting techniques. This includes the segmentation and parameterization of geometric primitives such as edges [16, 117, 18, 19, 17, 20] and surfaces [96, 22, 29]. While these methods effectively capture geometric features, they often struggle to recover the connectivity between primitives (CAD topology), resulting in potentially invalid CAD models.

B-Rep-Based Methods: To address the limitations of purely geometric approaches, researchers have developed learning methods for Boundary Representation (B-Rep), which encodes both geometry and topology. Graph-based approaches [13, 14] have been proposed to segment B-Rep entities, while other works [27, 38, 74, 134, 135] have focused on recovering different elements of the CAD construction history from B-Rep inputs.

CSG-Based Methods: Another line of research represents CAD construction history as a sequence of 3D primitives with corresponding Boolean operations, known as Constructive Solid Geometry (CSG). This representation has enabled several unsupervised methods [136, 137, 101, 120] to be developed. Although CSG allows for modeling complex shapes, it is no longer the standard representation in modern CAD software.

Sketch-Extrude Sequence Methods: More aligned with contemporary CAD practices, DeepCAD [26] proposed representing CAD construction history as sequences of 2D sketches and extrusions. This sequential approach has spawned numerous transformer-based generative models [26, 36, 100, 138]. For reverse engineering specifically, several works [23, 24] have used the sketch-extrude representation to infer CAD models from point clouds, though these methods often require post-processing to ensure valid CAD sketches. Prismatic CAD [121] addressed this by predicting CAD sketch-extrude sequences from rounded voxels using pre-defined templates. Recent advancements include MultiCAD [102], which introduced a two-stage model based on contrastive learning, and CAD-Diffuser [122], which leverages diffusion-based architecture.

In this work, we extend two state-of-the-art transformer-based approaches in the sketch-extrude paradigm: TransCAD [37] (referred to as NAR-CADNet from now on), a non-autoregressive model that employs a hierarchical learning strategy, and CAD-SIGNet [39] (referred to as AR-CADNet from now on), an autoregressive model that proposes a sketch

instance guided attention mechanism. Our contribution lies in augmenting these architectures with inference search strategies that more effectively explore the output space by taking into consideration the geometry of the input point cloud.

5.2.2 CAD Datasets

Despite the ubiquity of CAD models in manufacturing industries, large-scale, high-quality labeled datasets for CAD research remain limited. The evolution of these datasets reflects the field's progression toward more complete representations and realistic applications.

Geometric and B-Rep Datasets: The ABC dataset [28] is a large-scale CAD data collection with one million models, though it only provides geometric features for surface patches and curves. Several datasets subsequently focused on B-Rep level annotations: MFCAD [76] and MFCAD++ [77] offered synthetically generated B-Reps with machining feature labels; Joinable [139] contained 8,251 CAD models with assembly information represented as graphs; BrepGen [116] provided 6,171 B-Rep furniture models with categorical labels; and BrepNet [13] contributed 35,680 B-Reps with face segmentation labels corresponding to their constructive CAD operations.

Construction History Datasets: While the above datasets offer valuable resources for developing learning-based methods on B-Reps, they lack the sequential construction information critical for true CAD reverse engineering. Addressing this gap, DeepCAD [26] (178,238 models) and Fusion360 [27] (8,625 models) introduced datasets containing explicit construction histories in the form of sketch-extrude sequences. These datasets enabled research into learning the sequential design processes followed by human CAD designers. However, they share a significant limitation: their point clouds are generated by sampling CAD model surfaces rather than capturing real-world objects with 3D sensors, resulting in idealized data lacking the artifacts and imperfections present in real scans.

Realistic Scan Datasets: To advance the field toward more realistic scenarios, the CC3D dataset [41] was introduced with 50,000 pairs of realistic 3D scans and corresponding CAD models as B-Reps. This dataset has been progressively enhanced with additional annotations, including sharp edge features [117], B-Rep CAD operation type and step labels [38], and

LAR point-to-BRep adjacency labels [29]. Despite these valuable additions, a critical gap remained: the lack of explicit construction histories paired with realistic scans.

In this chapter, we address this gap by introducing the CC3D-Recon dataset—the first dataset containing pairs of realistic 3D scans with authentic scanning artifacts and their corresponding parametric construction histories as sketch-extrude sequences. This dataset enables evaluation and training of CAD reverse engineering methods under conditions that more faithfully represent real-world applications, where input data is noisy, incomplete, and contains various scanning artifacts.

5.3 Problem Formulation

Having reviewed the existing approaches to CAD reverse engineering, we now provide a precise mathematical formulation of the problem and introduce our notation framework. This formulation serves as the foundation for both the auto-regressive and non-auto-regressive approaches we extend in subsequent sections.

Given a point cloud $\mathcal{X} = \{\mathbf{p}_i\}_{i=1}^{N} \in \mathbb{R}^{N\times 3}$, our goal is to reconstruct the underlying 3D model through a sequence of CAD modeling operations. We formulate this as a mapping from point clouds to CAD sequences and present two complementary learning strategies.

5.3.1 CAD Sequence Representation

Let $\mathcal{C}$ represent the set of all possible CAD sequences. A CAD sequence $C \in \mathcal{C}$ can be modeled as a sequence of L design sets: $C = \{c_l\}_{l=1}^{L}$, where each step $c_l = (\mathcal{S}_l, \mathcal{O}_l)$ pairs a sketch $\mathcal{S}_l$ with a modeling operation $\mathcal{O}_l$. The sketch follows a hierarchical structure:

$$\mathcal{S}_l = \{\mathcal{F}_j\}_{j=1}^{N_f^l} \quad \text{(faces)}, \tag{5.1}$$

$$\mathcal{F}_j = \{\rho_k\}_{k=1}^{N_\rho^j} \quad \text{(loops)}, \tag{5.2}$$

$$\rho_k = \{\delta_m\}_{m=1}^{N_\delta^k} \quad \text{(primitives)}. \tag{5.3}$$

At the lowest level, primitives δ_m are represented using 2D coordinates $\mathbf{p} = [p_x, p_y]^\top$:

$$\delta_m = \begin{cases} [\mathbf{p}_{\text{start}}, \mathbf{p}_{\text{end}}], & \text{for lines;} \\ [\mathbf{p}_{\text{start}}, \mathbf{p}_{\text{mid}}, \mathbf{p}_{\text{end}}], & \text{for arcs;} \\ [\mathbf{p}_{\text{center}}, \mathbf{p}_{\text{top}}], & \text{for circles.} \end{cases} \tag{5.4}$$

Each modeling operation $\mathcal{O}_l$ consists of common parameters Θ_l and operation-specific parameters Φ_l:

$$\mathcal{O}_l = \{\Theta_l, \Phi_l\}\,, \tag{5.5}$$

$$\Theta_l = \{\theta, \phi, \gamma, \tau_x, \tau_y, \tau_z, \sigma, \beta\}\,, \tag{5.6}$$

where (θ, ϕ, γ) defines the sketch plane orientation, (τ_x, τ_y, τ_z) specifies translation, σ controls scaling, and β denotes the boolean operation type. The operation-specific parameters Φ_l depend on the type of operation. In this work, only the extrusion operation is considered and $\Phi_l = \{d_+, d_-\}$, where d_+, d_- are extrusion distances in the direction of the sketch-plane normal and the opposite direction.

5.3.2 Learning Objective

The 3D reverse engineering objective can be modelled as the learning of a mapping

$$\Psi : \mathbb{R}^{n\times 3} \rightarrow \mathcal{C}\,, \tag{5.7}$$

$$C = \Psi(\mathcal{X})\,, \tag{5.8}$$

that maps the input point cloud to a CAD sequence $C \in \mathcal{C}$ such that the sequence produces a CAD model geometrically approximating the input point cloud $\mathcal{X}$ and adheres to the design patterns that would be followed by an actual CAD designer.

5.3.3 Learning Paradigms

We explore two complementary approaches for learning the mapping from point clouds to CAD sequences:

Auto-regressive Learning

Models the conditional probability of a token given the input point cloud $\mathcal{X}$ and the previous sequence tokens:

$$p_{\text{AR}}(\mathcal{C}|\mathcal{X}) = \prod_{i=1}^{n_{ts}} p(t_i|t_{j\,j<i}, \mathcal{X})\,. \tag{5.9}$$

Non-autoregressive Learning

Predicts all the tokens simultaneously:

$$p_{\text{NAR}}(\mathcal{C}|\mathcal{X}) = \prod_{i=1}^{n_{ts}} p(t_i|\mathcal{X})\,. \tag{5.10}$$

Each paradigm offers distinct advantages: auto-regressive learning can better model sequential dependencies but inference is slower, while non-autoregressive learning enables parallel prediction but may struggle with complex dependencies. We leverage both approaches in our methodology to provide complementary solutions to the CAD sequence inference problem.

5.4 Methodology

Having established the problem formulation for CAD reverse engineering from point clouds, we now present our methodology for enhancing sequence prediction through inference search strategies. Our approach builds upon two complementary architectures—auto-regressive (AR-CADNet) and non-auto-regressive (NAR-CADNet)—and extends them with geometry-guided search techniques. Both neural architectures produce probability distributions over possible CAD tokens, but rather than simply selecting the highest probability tokens, we explore how

the inherent uncertainty in these distributions can be leveraged to explore multiple plausible design solutions.

5.4.1 Classic Inference Strategies

The output of both AR-CADNet and NAR-CADNet is a probability distribution over the token values. The most common approach, to select the predicted token value, is to choose the one that has the highest probability (Top-1 sampling). This approach is motivated by the fact that the output probability can be interpreted as the prediction confidence. Upon examining the output probability distribution per token (see Figure 5.1 for AR-CADNet and Figure 5.2 for AR-CADNet), it can be observed that there is a high variance in the output probabilities, which means that the network does not always display a high degree of confidence in the predicted token value. This observation provides a strong motivation to explore other sampling strategies, in particular to consider sampling strategies that considers the geometry of the input point cloud.

Auto-regressive Sampling Given the predicted token probabilities $p(t_i|\{t_j\}_{j<i}, \mathcal{X})$, several sampling strategies can be employed:

- **Top-k sampling** selects tokens from the k most likely candidates according to the model's probability distribution. For each prediction step, the model identifies the k tokens with highest probabilities. The most common search strategy is to use Top-1 sampling, that is choosing the token value with the highest probability at each step.
- **Nucleus sampling** [140], also known as top-p sampling, dynamically adjusts the sampling pool by selecting the smallest set of tokens whose cumulative probability exceeds a threshold p. Rather than using a fixed number of candidates, nucleus sampling adapts to the model's confidence level—sampling from more options when the distribution is flat (low confidence) and fewer when the distribution is peaked (high confidence). This approach is particularly effective for CAD sequence generation because confidence levels vary significantly across different token positions (see Fig. 5.1).

- **Beam search** maintains the b most probable sequence candidates throughout the generation process. At each step, the model expands all current beam candidates with possible next tokens, scores the resulting sequences based on their cumulative log-probabilities, and keeps only the top b candidates for the next iteration. This approach considers the joint probability of the entire sequence rather than making greedy decisions at each step.
- **Hybrid sampling** combines different strategies at different stages of the generation process, specifically using Top-k sampling ($k = 5$) for the first token and reverting to Top-1 (greedy) sampling for all subsequent tokens. This approach addresses the observed pattern (see Fig. 5.1) where the model displays particularly low confidence for the initial token that have no preceding context, but higher confidence for later tokens that benefit from established context. The hybrid approach allows exploration of different starting points for the CAD sequence (*i.e.*extrusion distance), while maintaining deterministic continuation once a path is chosen. This reflects the design process where the first decision often has the greatest impact on the overall structure.

Non-auto-regressive Sampling The baseline approach for sampling in feed-forward network is to use Top-1 prediction for each token:

$$t_i = \text{argmax}_j \, p(t_i = j|\mathcal{X}) \, . \tag{5.11}$$

As in the auto-regressive strategy, it is also possible to apply nucleus sampling on the individual predicted tokens. However, it is important to note that beam search and hybrid sampling strategies are not applicable in the non-auto-regressive context because these methods rely on sequential token interdependencies that do not exist in this paradigm. Since the non-auto-regressive model predicts all tokens simultaneously with probabilities that are conditionally independent given the input point cloud, there is no notion of a sequence of decisions that could be maintained in multiple beams or treated differently at different positions. This fundamental limitation of non-auto-regressive models restricts the sampling

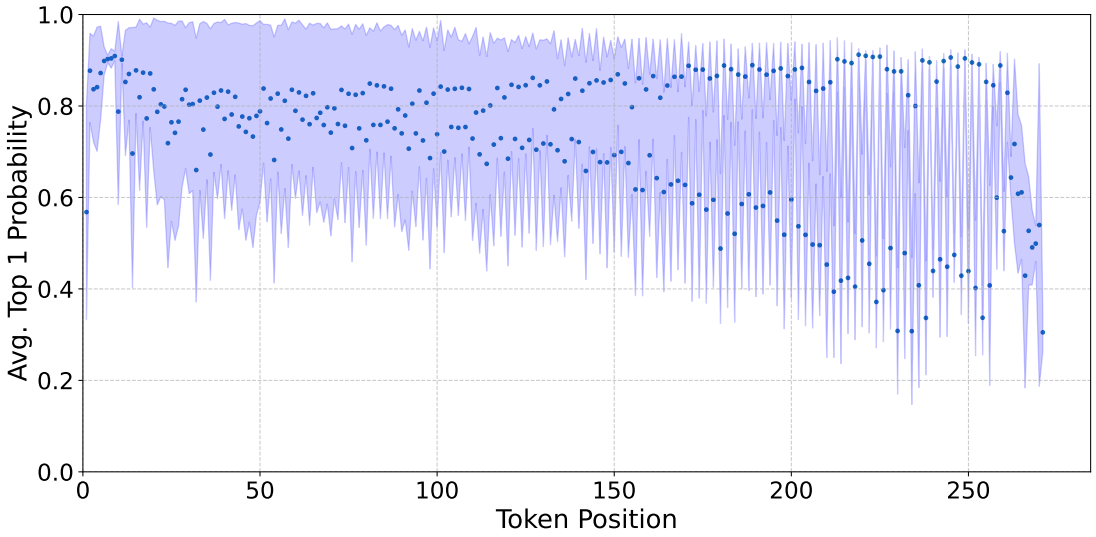


Figure 5.1: Variation of the average Top-1 probabilities with respect to token position for AR-CADNet. The shaded area shows the standard deviation. Overall, there is a high variance in the confidence level displayed by the network, suggesting that the Top-1 sampling strategy might not always be appropriate. The network appears to have a low confidence for the first token due to the lack of sequence context. The high confidence and low variance for tokens at position 5 to 8 corresponds to orientation and scale of the first extrusion plane.

strategies to those that operate on individual token probabilities in isolation.

5.4.2 Geometry-Guided Search

While the sampling strategies described above effectively leverage the model's inherent uncertainty, they operate solely on the probability distributions without considering the specific geometric context of the reverse engineering task. This is a significant limitation, as the input point cloud contains rich geometric information that can guide the search process toward more plausible CAD reconstructions. For instance, the spatial distribution of points provides clear constraints on possible extrusion distances, sketch plane orientations, and feature dimensions that should be respected in any valid solution.

Building on this insight, we propose to augment traditional sampling strategies with explicit geometric priors derived from the input data. These priors, which we denote as $\mathcal{G}$, serve to

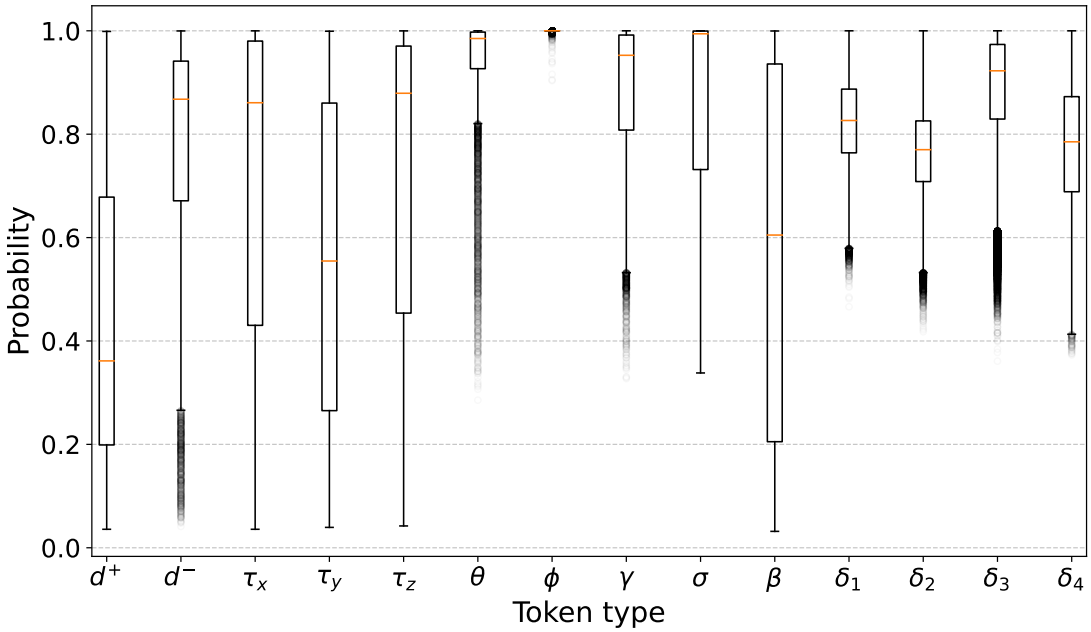


Figure 5.2: Box-plot graph showing the distribution of the Top-1 probabilities per token types for NAR-CADNet. As for AR-CADNet, tokens corresponding to the orientation (θ,ϕ, and γ) and the scale of the extrusion plane display the lowest variance, reflecting the biases from the training data.

constrain and guide the search process in ways that purely probabilistic approaches cannot. In other words, we modify token probabilities using geometric priors:

$$p(t_i|\cdot, \mathcal{G}) \propto p(t_i|\cdot) \cdot w(t_i|\mathcal{G}) \,, \tag{5.12}$$

where $p(t_i|\cdot)$ is the output probability given by the network and $w(t_i|\mathcal{G})$ is a function constraining the values of t_i given some geometric prior. This approach enables us to develop search strategies that are specifically tailored to the CAD reverse engineering problem, leveraging domain-specific knowledge that neural networks may not fully capture through training alone.

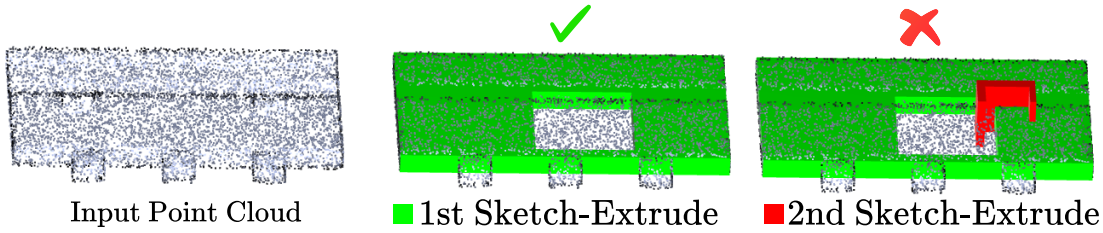


Figure 5.3: Bounding box guided search for AR-CADNet. The first predicted sketch-extrude is within the bounding box of the input point cloud, it is therefore kept as a potential candidate. The second predicted sketch-extrude lies outside the bounding box of the input, it is therefore rejected and backtracking is operated.

Geometry Guided Auto-Regressive Search

In the case of AR-CADNet, a backtracking strategy is employed given the bounding-box of the input point cloud as a geometrical prior $\mathcal{G}$ and a function that checks the validity of the partial CAD sequence (*i.e.* whether the sequence can be reconstructed in a valid CAD model). We combine this geometric validation with nucleus sampling on each token. More specifically, after the prediction of a sketch-extrusion step $c_l = (\mathcal{S}_l, \mathcal{O}_l)$, we check whether the partial predicted sequence can be reconstructed and whether it is within the bounding-box of the input point cloud. If the validation fails, rather than terminating the search, we backtrack to the start of the sketch-extrusion resample alternative tokens from the nucleus sampling distribution and explore this new path. This resampling and validation process continues until either a valid sketch-extrusion step is found or a maximum number of sampling has been reached. This strategy relies on the assumption that at each construction step the partial model has a bounding box smaller than or equal to the bounding box of the final model. By integrating backtracking with nucleus sampling, our approach efficiently identifies valid design paths while exploring diverse solutions that satisfy the geometric constraints of the input point cloud. Fig. 5.3 shows an example of how this is applied in practice.

Geometry Guided Non-Auto-Regressive Search

The non-auto-regressive nature of NAR-CADNet provides significant advantages when applying geometric priors to guide the search process. Unlike auto-regressive models,

NAR-CADNet can generate probability distributions for all tokens in a single forward pass, making sampling computationally efficient. This efficiency enables us to perform extensive validity checks and apply geometric constraints to both extrusion and loop parameters without prohibitive computational overhead.

Our approach implements a multi-stage validation and constraint process. For each candidate solution, we first validate the extrusion plane parameters Θ_l by verifying that they define a plane that intersects with the input cloud. When this validation fails, we resample those specific tokens using nucleus sampling while keeping other valid tokens fixed. This selective resampling is particularly efficient in the non-auto-regressive framework since tokens are predicted independently.

After establishing valid extrusion planes, we apply spatial constraints derived from the input geometry. For loop parameters (sketch primitives δ), we constrain the token probabilities by analyzing the bounding box of points that lie on the extrusion plane (see Fig. 5.4). This effectively masks out token values that would create sketch elements outside the observed point distribution, setting their probabilities to zero before sampling. Similarly, we constrain extrusion distance parameters (d_+, d_-) by identifying the farthest points from the extrusion plane in both positive and negative directions, ensuring that extrusions remain within the boundaries of the input point cloud.

This geometric masking and resampling approach allows NAR-CADNet to efficiently explore the constrained solution space while maintaining geometric validity, resulting in diverse yet plausible CAD reconstructions that respect the spatial characteristics of the input data.

5.5 CC3D-Recon Dataset

In this section, we introduce the proposed CC3D-Recon dataset. The CC3D-Recon dataset aims at offering realistic evaluation data for 3D CAD reverse engineering. It is the first dataset that contains realistic 3D scans of CAD models along with their construction history.

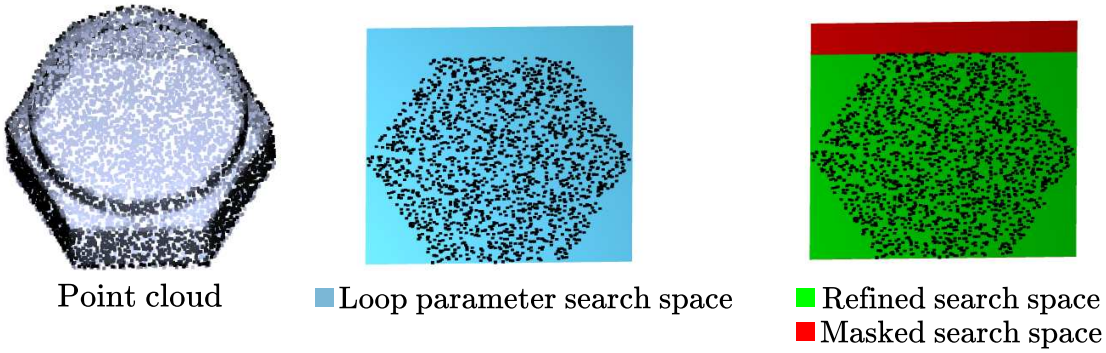


Figure 5.4: Masking of loop parameters for NAR-CADNet. Left: Input point cloud. Middle: Sketch plane as defined by the extrusion plane parameters Θ_l and the points from the input point clouds that lie on this plane. The range of possible loop primitive parameter values is shown in blue. Right: Parameter values outside the bounding-box of the points are masked (shown in red) effectively reducing the search space (shown in green).

5.5.1 Data Preprocessing

The CAD models from the CC3D-Recon dataset were collected from open repositories such as OnShape [141] and 3D Content Central [142]. The CAD models were then virtually scanned using a proprietary software developped by Artec3D [143] as in [41]. This process enables the creation of realistic 3D scans that contain artifacts such as missing parts, surface noise, noisy normals and smoothed edges. The CAD models containing only the extrusion operation were selected and were further parsed using the OnShape API to obtain the construction history in the form of parametric sketch-extrude sequences in the same format as the DeepCAD dataset.

5.5.2 Dataset Description

The CC3D-Recon is composed of 500 models for evaluation. Examples of the CAD models can be found in Fig. 5.5. All the models contain a CAD representation in the form of a STEP file, a sketch-extrude sequence describing the construction history, and a high resolution mesh of the corresponding scan. Fig. 5.6 displays examples of the magnitude of the surface noise (third column) and the angle difference (most right column) between the surface normals

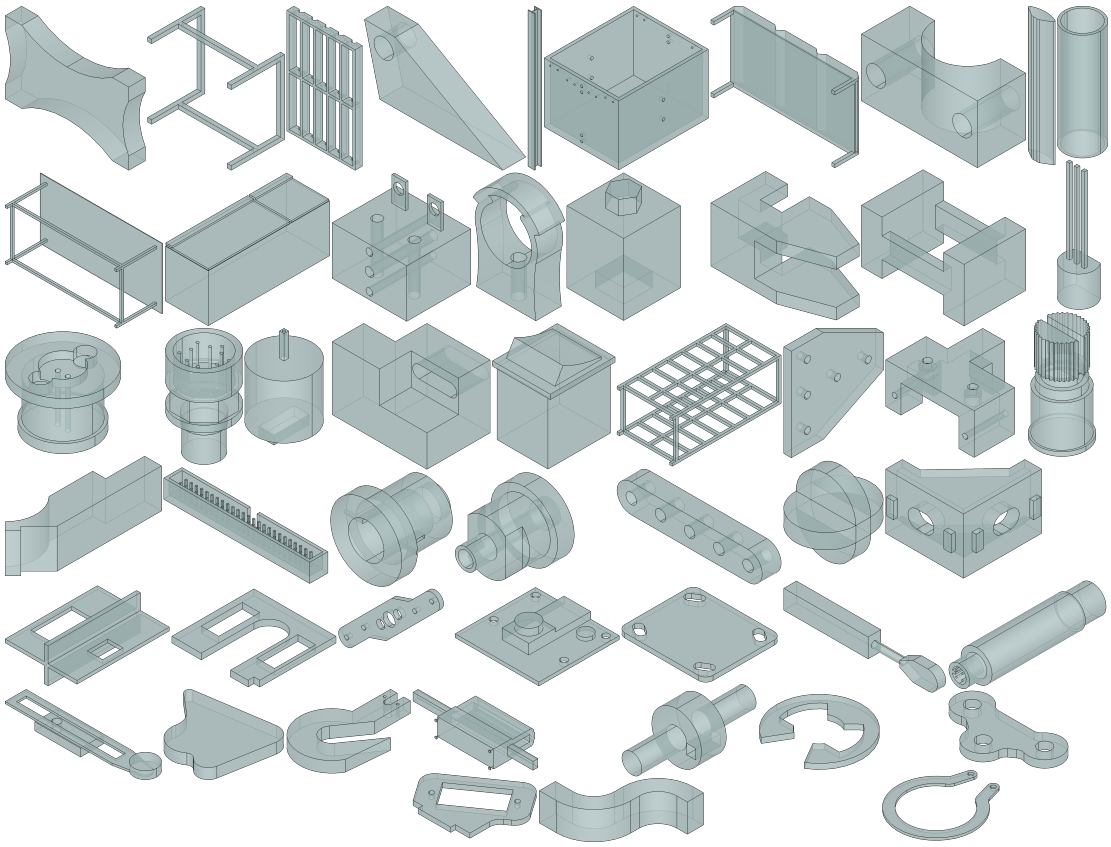


Figure 5.5: Examples of ground truth CAD models found in the CC3D-Recon dataset.

of the scan and the actual CAD model. It can be observed that near edges the surface and normal noises are particularly prominent, this is caused by smoothed features during the scanning process. Smoothed edges present a real challenge in the reverse engineering setting as the edges often define the sketches that are used to create the CAD model. Fig. 5.7 shows frequency density graphs for the distribution of the magnitude of surface noise and normal angle disturbance. It can be observed that most scan mesh vertex have a noise in magnitude that is less than 1% the size of the bounding box of the model and the a normal angle distrubance is less than 20%. However, there is large variance in the distribution of those quantities resulting in some mesh vertices with much larger disturbance. These realistic scanning artifacts, particularly the edge smoothing and normal distortion, make

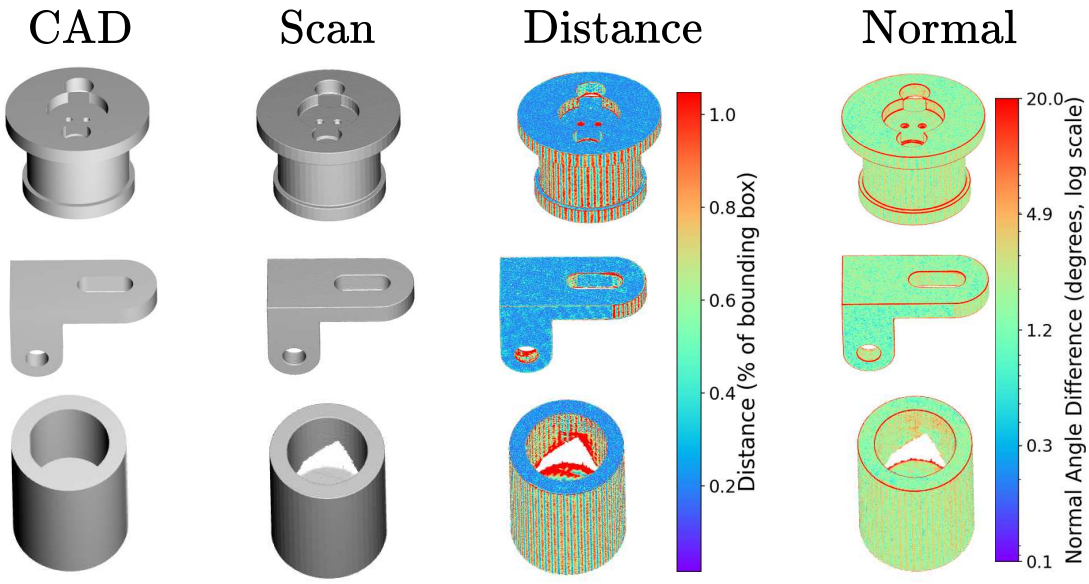


Figure 5.6: CC3D-Recon scan artifacts. The surface noise on the scans is depicted in the third column and the surface normal angle disturbance in the most right column. The noise is most prominent around edges and parts of the model that a sensor can difficultly reach. The ridges that can be observed around curved surfaces arises from the tessalation of the CAD model during the virtual scanning process.

CC3D-Recon a substantially more challenging benchmark for scan-to-CAD algorithms than previous datasets with synthetic point clouds, effectively testing their robustness in conditions that more closely resemble real-world reverse engineering scenarios.

5.6 Experimental Results

5.6.1 Training

In order to obtain a fair comparison between AR-CADNet and NAR-CADNet, both models are trained using the same point cloud encoder (LFA [54]) and the same curve primitive and extrusion formulation as defined in [39].

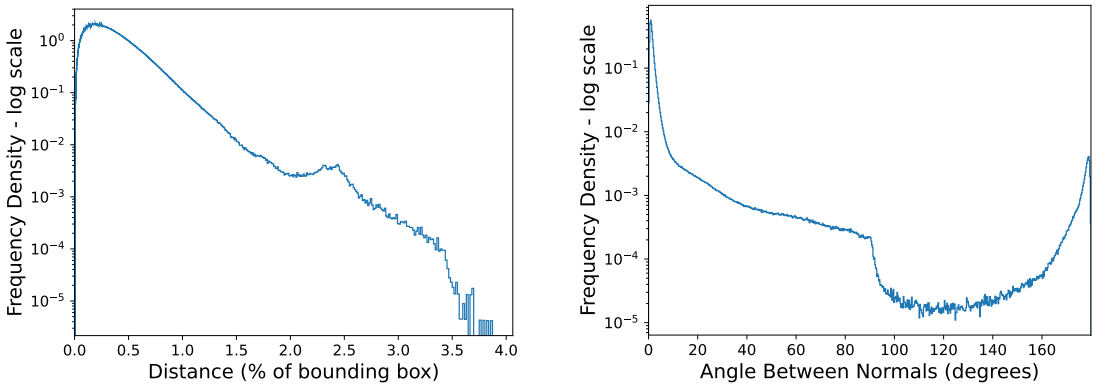


Figure 5.7: Frequency density graphs of the surface noise distance (left) and normal disturbance (right) on the scans of the CC3D-Recon dataset. While about 75% of the points from the scan have a noise displacement of 0.5% or less the size of the object bounding box, some points can have a large displacement (up to 4%). Similarly, the scan surface normals tend to be slightly disturbed with a median value of $6°$. However, due to smoothing of sharp features and missing parts caused by the scanning process a small portion of the normals can have much greater disturbance.

5.6.2 Baselines

The main focus of the experiments is to evaluate how the performance of autoregressive and non-autoregressive networks can be improved using different search strategies at inference time. Nevertheless, baseline results for DeepCAD [26], MultiCAD [102], HNC [100] and CAD-Diffuser [122] have been included for reference. The results presented are the ones reported in [122].

5.6.3 Metrics

In order to evaluate the results of the different models, three metrics are used. Firstly, to evaluate how close the predicted sequence is to the ground-truth one the APCS metric [37] is used. While this metric measures how close the predicted CAD sequences are to the ones that were produced by an actual CAD designer, it is necessary to also evaluate how close the geometry of the reconstructed CAD model is to the ground-truth one. To this end, the median chamfer distance (CD) is used. Note that all the CD values have been multiplied by

	Model	Search Strategy	DeepCAD Dataset			Fusion360 Dataset		
			APCS↑	Median CD↓	IR↓	APCS↑	Median CD↓	IR↓
Non-AR	DeepCAD [26]	Top-1	-	9.64	7.1	-	89.2	25.2
	MultiCAD [102]	Top-1	-	8.09	11.5	-	42.2	16.5
	NAR-CADNet	Top-1	0.708	6.59	6.2	0.512	13.12	12.1
	NAR-CADNet	Nucleus	0.722	1.15	0.4	0.515	3.02	0.0
	NAR-CADNet	Geometry Guided	0.725	0.62	0.0	0.527	1.67	0.1
AR	HNC [100]	Top-1	-	8.64	5.6	-	36.8	7.8
	CAD-Diffuser [122]	Top-1	-	3.02	1.5	-	3.85	1.7
	AR-CADNet	Top-1	0.742	0.26	5.1	0.548	0.56	9.3
	AR-CADNet	Hybrid	0.588	0.23	0.9	0.437	0.41	1.9
	AR-CADNet	Nucleus	0.707	0.24	0.0	0.518	0.41	0.0
	AR-CADNet	Beam	0.661	0.21	1.5	0.494	0.39	4.4
	AR-CADNet	Geometry Guided	0.718	0.20	0.0	0.519	0.38	0.0

Table 5.1: Quantitative comparison of different search strategies for CAD sequence inference from point clouds on DeepCAD and Fusion360 datasets. Multi-sampling search strategies (hybrid, nucleus and geometry guided) leads to a drastic reduction in invalidity ratio (IR). For both NAR-CADNetand AR-CADNetthe best performing strategy is the geometry guided search.

10^3. Finally, the ratio of predicted CAD sequence that cannot be reconstructed into a valid CAD model is measured using the invalidity ratio (IR).

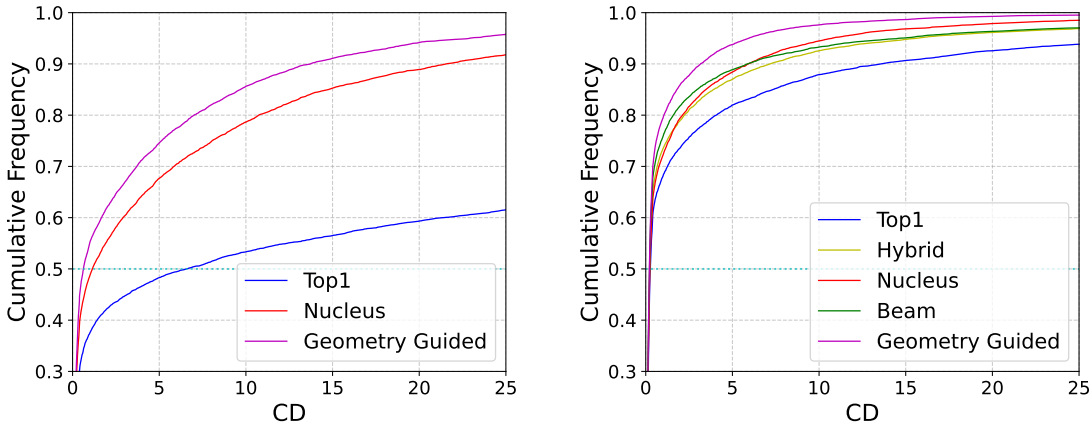
5.6.4 Results on DeepCAD and Fusion360 Datasets

The results of all methods on the DeepCAD [26] and Fusion360 [27] datasets are presented in Table 5.1. Note that all methods were trained on the DeepCAD dataset and that the Fusion360 dataset constitutes a cross-dataset evaluation setting. It can be observed that for both AR-CADNetand NAR-CADNet the ratio of invalid generated CAD sequences (IR) decreases drastically when multiple sampling is applied on the output token probabilities.

For AR-CADNet, the APCS metric that measures how close the predicted sketch-extrude sequence is to the ground truth one is lowest when using hybrid sampling and highest when using Top-1 sampling. This is clearly demonstrated in Fig 5.9(b), where the distribution of APCS per model is more skewed towards lower values when hybrid sampling is used. This is caused by the fact that hybrid sampling applies a Top-5 sampling only on the first token, which corresponds to the extrusion distance of the first sketch-extrude. This may lead to different to alternative valid sequences than the ground truth. On the other hand, using Top-1 sampling

strategy results in sequences that are closer to the ones in the training dataset, leading to the highest APCS result. However, an error in one token can have a drastic impact on the output geometry. The hybrid sampling strategy therefore favors geometrically consistent solutions over predicted sequences that follow the design patterns present in the training data. Using the beam search strategy with a beam size of 10, in which the combined probabilities of all the token is maximized, leads to improvements in the median CD demonstrating that the geometry of the predicted CAD models is closer to the ground truth ones. However, this method leads to a decrease in APCS metric as this approach leads to a deviation from the ground truth design patterns. From Table 5.1, it appears that AR-CADNet provides the best overall performance on all metrics on both DeepCAD and Fusion360 datasets when nucleus and geometry guided searches are used with a sampling count of 10. Nevertheless, Fig 5.9(b) clearly shows that the distribution of CD per model is more skewed towards lower CD values when the geometry guided search is used, as this search takes into account the input shape of the point cloud that needs to be reconstructed. Furthermore, as shown in Fig. 5.10(b), the geometry guided search results in a faster convergence and therefore may require a smaller amount of sampling to provide an appropriate prediction.

The non auto-regressive model, NAR-CADNet appears to have a much a lower performance than its auto-regressive counter-part when Top-1 sampling is used. Nevertheless, the non auto-regressive strategy contains several advantages that can be leveraged at inference time. First of all, as it is a feed-forward model, the probability distributions for all the tokens can be predicted through one forward pass. This results in a significantly faster inference and allows for a larger sampling count (50 for nucleus and geometry guided searches) and the possibility of applying more validity checks and geometrical constraints. As a result, from Table 5.1 it can be observed that even using a naive sampling technique such as nucleus sampling leads to a drastic improvement in geometric fidelity of the prediction as seen by the six fold decrease in median chamfer distance. Furthermore, guiding the search using the geometric prior of the input point cloud results in further improvements with a median chamfer distance of the same order of magnitude as AR-CADNet on the DeepCAD dataset. This trend can be clearly observed in Fig. 5.8(a). Unlike AR-CADNet, the search strategies applied



(a) NAR-CADNet

(b) AR-CADNet

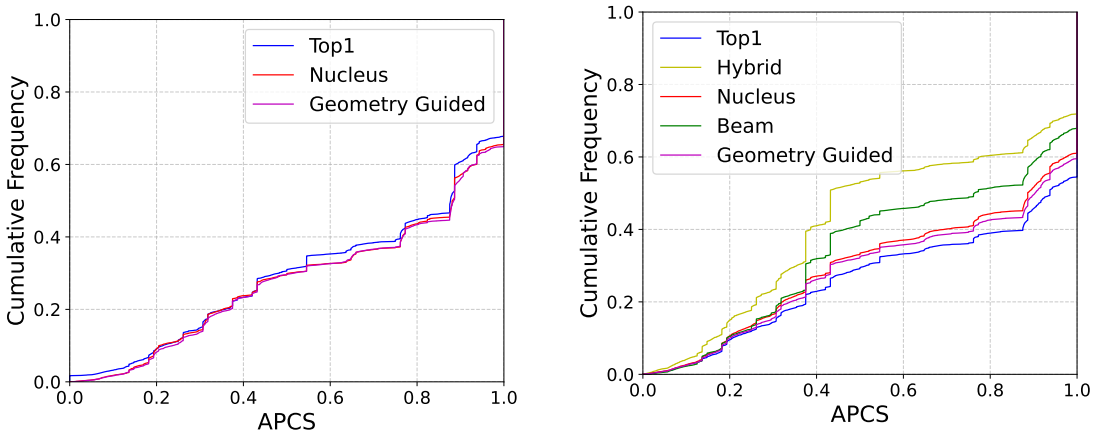
Figure 5.8: Cumulative frequency graphs of the Chamfer distance per model on the DeepCAD dataset. The geometry guided search strategy that takes into account the geometry of the input point cloud results in distributions that are skewed towards lower CD values.

on NAR-CADNet appear to provide a small improvement on the APCS metric. However, as shown in Fig. 5.9, the distributions of the APCS per model between the three different sampling strategies are quite similar and therefore the noted improvement in Table 5.1(a) is relatively negligible.

In summary, our experiments demonstrate that advanced sampling strategies significantly improve both models, with geometry-guided search providing the most substantial gains. While AR-CADNet achieves higher geometric accuracy with appropriate search techniques, NAR-CADNet offers advantages in inference speed and sampling capacity.

5.6.5 Results on the CC3D-Recondataset

The results on the CC3D-Recon are shown in Table 5.2. It can be observed that the dataset offers a more challenging scenario. In particular, the invalidity ratio for both AR-CADNet and NAR-CADNet using the top-1 sampling strategy becomes significantly larger (17% and 10% respectively). This increase in invalidity ratio can be attributed to the noisy input point cloud created by the scanning artifacts. Moreover, these high invalidity ratio makes the interpretation



(a) NAR-CADNet

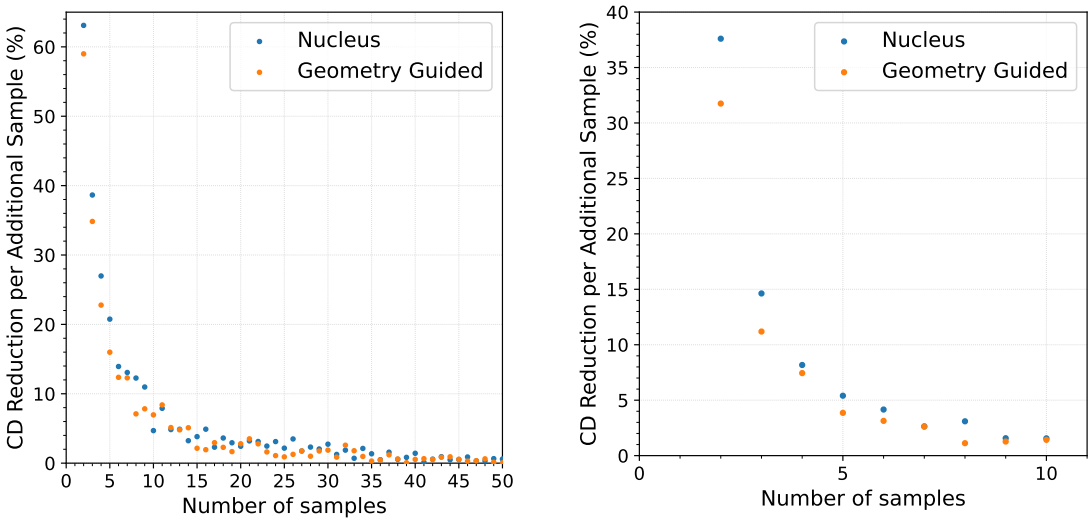
(b) AR-CADNet

Figure 5.9: Cumulative frequency graphs of the APCS per model on the DeepCAD dataset. The APCS metric measures how close the predicted CAD sequence is to the ground truth one in terms of token type and value.

of the corresponding median chamfer distances less reliable as it can only be computed on valid predictions. However, it can be observed that as in the DeepCAD and Fusion360 results, these shortcoming can be overcome using different sampling strategies on the output probabilities distribution. Notably, it can be noted that the geometry guided search provides the most substantial improvements for both the auto and non-auto-regressive methods.

5.6.6 Search Diversity

The effectiveness of the search strategies that generate multiple solutions (namely, nucleus and geometry guided) can also be evaluated by quantifying the diversity in the predictions made per input. The search strategy is considered effective if all the generated CAD sequences are unique, that is, if the search space is widely explored. The results for diversity (average ratio of unique predicted sequence per model) can be found in Table 5.3. While both NAR-CADNet and AR-CADNet display a high level of diversity in the generated samples across datasets and search methods, it is worth noting that the lowest diversity is achieved on the DeepCAD dataset by NAR-CADNet. This observation can be attributed to the high



(a) NAR-CADNet

(b) AR-CADNet

Figure 5.10: Scatter plots showing the median CD reduction per for each extra sampled CAD sequence.

confidence that the network displays when the testing data display patterns similar to the training dataset.

5.6.7 Qualitative Analysis

Qualitative results presented in Fig. 5.11 provide some further insights in the predictions.

Both AR-CADNet and NAR-CADNet are able to predict correctly simple shapes such as cuboids and cylinders. However, there might be different ways to construct a simple shape. For example, the ground-truth CAD model from the top row of the DeepCAD dataset results in Fig. 5.11 shows a shape that was constructed from a double-sided extrusion, resulting in additional edges around the center of the shape. However, it can be observed that the predictions from the different methods all recover the shape from a one-sided extrusion. The ambiguity caused by different plausible sketch-extrude sequences resulting in the same 3D shape cannot be resolved from the input point cloud.

Another limitation arising from the resolution provided by the input point clouds is the

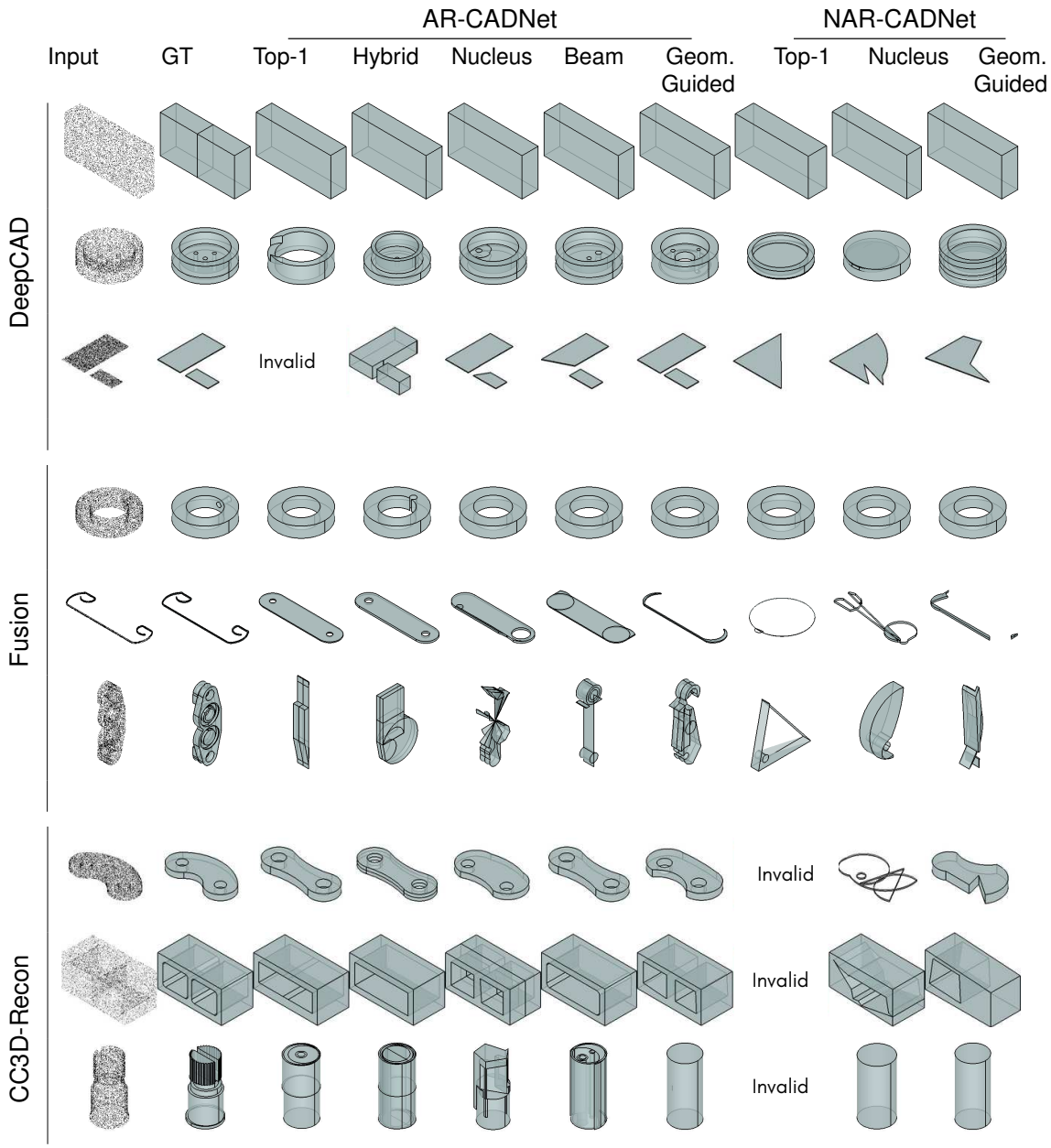


Figure 5.11: Qualitative results for the different search strategies for AR-CADNet and NAR-CADNeton the DeepCAD, Fusion360 and CC3D-Recon datasets.

Model	Search Strategy	CC3D-ReconDataset		
		APCS↑	Median CD↓	IR↓
NAR-CADNet	Top-1	0.605	6.03	17.2
	Nucleus	0.614	1.95	0.4
	Geometry Guided	0.612	0.88	0.6
AR-CADNet	Top-1	0.608	1.48	9.6
	Hybrid	0.474	0.37	1.0
	Nucleus	0.549	0.38	0.0
	Beam	0.568	0.36	3.6
	Geometry Guided	0.559	0.35	0.0

Table 5.2: Quantitative comparison of different search strategies for CAD sequence inference from point clouds on the CC3D-Recon dataset. The high median CDs from the Top-1 sampling strategy for AR-CADNet and NAR-CADNet demonstrate the complexity of inferring CAD sequences from 3D scan data. However, using a geometry guided search can significantly improve results.

Model	Search Strategy	Diversity↑		
		DeepCAD	Fusion360	CC3D
NAR-CADNet	Nucleus	0.961	0.983	0.985
	Geometry Guided	0.933	0.972	0.972
AR-CADNet	Nucleus	0.871	0.936	0.950
	Geometry Guided	0.829	0.997	0.993

Table 5.3: Comparison of the average ratio of unique sampled solution per input point cloud (Diversity). A diversity of 1 implies that all sampled CAD sequences are unique for a given input.

ability of the network to capture fine-grained details. This is examplified by the results on the first row from the Fusion360 dataset results as none of the models are able to predict the small sideway cylindrical hole.

The results on the CC3D-Recon dataset (bottom 3 rows of Fig. 5.11) reflect the high invalidity ratio of AR-CADNet using the top-1 sampling strategy. From the first row of the CC3D-Recon results, through AR-CADNet top-1, hybrid and beam searches the predictions are identical in shapes. It is interesting to note that this shape corresponds to one that can be found in the training dataset. Hence, AR-CADNet has a tendency to retrieve sequences that it was exposed to during training. This observation can also be made from the top-1 and hybrid results in the second row of the Fusion360. However, tendency of AR-CADNet to act as a retrieval model can be mitigated through the geometry guided search.

Finally, when the ground-truth CAD model is composed of a large amount of small details, both AR-CADNet and NAR-CADNet tend to predict shapes that are unreasonably complex (last row of Fusion360 dataset results) or overly simplistic (last row of CC3D-Recondataset results).

As a conclusion, it can be noted that using the geometry guided search can help overcome some of the shortcomings of the networks and improve the quality of the predicted shape. Nevertheless, both models can struggle when the ground-truth shape contains small details.

5.7 Conclusion and Future Work

In this chapter, we advanced our approach to CAD reverse engineering by exploring how inference search strategies can significantly improve the performance of both auto-regressive and non-auto-regressive models for CAD sequence prediction from point clouds. Building upon the single-solution methods presented in previous chapters, we demonstrated that strategic inference approaches can substantially enhance reconstruction quality beyond what architecture improvements alone can provide, directly addressing the challenges of recovering design intent and handling scanning artifacts outlined in Section 1.2.

Our comprehensive evaluation across DeepCAD, Fusion360, and our newly introduced CC3D-Recon dataset yielded three key findings that advance the state of CAD reverse engineering: (1) Geometric priors derived from input point clouds can effectively guide the search process to generate more accurate and valid CAD models, thereby addressing the scanning imperfections challenge; (2) Auto-regressive and non-auto-regressive approaches offer complementary strengths that can be leveraged through appropriate sampling strategies—AR-CADNet provides superior geometric accuracy when coupled with geometry-guided search, while NAR-CADNet enables faster inference with comparable results; (3) Search diversity is crucial for overcoming the inherent ambiguity in reverse engineering, enabling the exploration of multiple plausible design paths in a manner that more closely resembles expert human designers.

The CC3D-Recon dataset introduced in this work addresses the data challenge identified

in Section 1.2 by providing realistic scan data paired with ground truth CAD sequences, enabling more robust evaluation under real-world conditions. Our results on this dataset highlight both the increased difficulties posed by scan artifacts and the effectiveness of our proposed search strategies in mitigating these challenges.

While this chapter has demonstrated significant improvements in CAD sequence inference through geometry-guided search, several limitations remain. The current approach still struggles with very complex geometries containing numerous small features. Addressing this limitation, the next chapter will explore a fundamentally different approach that directly generates executable CAD code from point clouds, further advancing our goal of practical, industry-applicable reverse engineering solutions.

Future research directions could include incorporating user feedback into the search process to enable interactive refinement; developing adaptive sampling strategies that adjust based on model confidence and input complexity; and exploring the integration of these search strategies with more advanced neural architectures. As we have shown throughout this thesis, advancing CAD reverse engineering requires progress not only in model architectures but also in the inference strategies and evaluation methodologies that enable practical application in real-world industrial contexts.

Chapter 6

CAD-Recode: Reverse Engineering CAD Code from Point Clouds

In this chapter, we introduce CAD-Recode, a novel Large Language Model (LLM) based approach for CAD reverse engineering that directly generates executable Python code from point clouds. While previous chapters have progressively tackled different aspects of the reverse engineering pipeline—from recovering construction history in B-Reps to inferring parametric sketch-extrude sequences from point clouds with geometry-guided search—this work represents a paradigm shift that addresses all four fundamental challenges outlined in Section 1.2. By leveraging the code understanding capabilities of pre-trained LLMs combined with a lightweight point cloud projector, CAD-Recode generates code that can be immediately executed in standard CAD environments, eliminating the representation gap identified in Section 1.2 that has hindered practical industrial adoption. Furthermore, we address the data challenge through a procedurally generated dataset of one million diverse CAD sequences, providing precise control over both the quantity and quality of training examples. This approach not only overcomes the scanning artifacts challenge through improved reconstruction accuracy but also enhances the recovery of design intent by generating code that captures the parametric nature of CAD models in a human-readable format. The resulting system marks a significant advancement toward practical, industrial-

grade CAD reverse engineering, bringing us closer to the objective of fully automated translation from 3D scans to editable CAD models.

6.1 Introduction

As established throughout this thesis, Computer-Aided Design (CAD) modeling serves as the foundational methodology for designing manufactured objects across industries, from furniture to complex mechanical components [144, 145]. The previous chapters have addressed various aspects of CAD reverse engineering, progressively tackling the challenges identified in Chapter 1. Despite these advances, a fundamental limitation remains: creating a comprehensive 3D CAD model continues to be a time-consuming process requiring specialized expertise, as the model must not only capture the object's geometric shape but also preserve its functional requirements—the *design intent* we discussed in Section 1.2 [146, 147]. While our previous approaches have made significant progress in recovering construction history and generating parametric CAD sequences from point clouds, they still require post-processing to integrate with standard CAD environments. This chapter presents a solution that directly addresses this integration challenge by generating executable Python code from 3D scanned objects, thereby offering a more complete pathway to automated CAD creation that can be immediately utilized in industrial workflows [10].

Automated 3D CAD reverse engineering has a long history in the fields of computer vision and graphics [129, 130], with goals evolving alongside advancements in the field. These objectives have progressed from identifying CAD parts in 3D point clouds [129] to predicting the sequence of steps a designer may take to recreate a 3D scanned object in CAD software [122, 39]. This latter goal is particularly appealing, as it aims not only to produce a final CAD parametric model but also to capture the design steps behind it, enabling further editing within CAD software [23, 39]. In CAD software, designers typically construct their CAD model as feature-based design sequences, where a sequence of 2D sketches is transformed into 3D objects via operations such as extrusion and revolution [74, 27]. Following the release of large CAD datasets [28, 27, 41], recent works have focused on learning feature-based CAD

sequences from input point clouds, specifically as *sketch-extrude* sequences [26, 23, 25, 24, 102, 39, 37]. As depicted in Figure 6.1(a), although varying in methodology, these approaches share a common pipeline: (1) crafting a CAD sketch-extrude sequence representation, (2) converting raw CAD data [28, 27] into this format, and (3) training dedicated neural networks to output these representations from input point clouds.

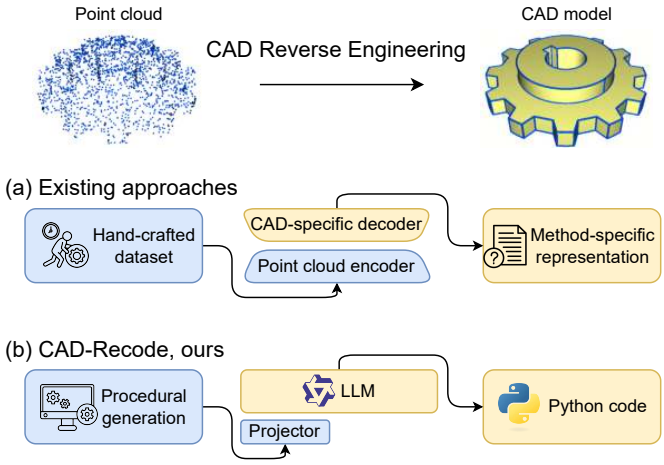


Figure 6.1: 3D CAD reverse engineering is a process of converting a point cloud into a CAD model (top). Existing methods are constrained by the use of method-specific CAD representations and limited hand-crafted training datasets (a). On the contrary, `CAD-Recode` employs a pre-trained LLM with a lightweight projector that translates point clouds into executable Python code and is trained on a procedurally generated dataset (b).

Despite recent advances in feature-based CAD reverse engineering, key limitations constrain the broader applicability of existing approaches. Firstly, existing methods often use customized CAD representations, such as custom CAD languages [26, 39, 37, 122] or simplified extrusion shapes [25, 24, 23], to facilitate model training. These representations are difficult to interpret, require post-processing to be compatible with CAD tools, and restrict design capabilities to basic operations. Secondly, these approaches typically rely on designing networks that output language-like CAD representations [39, 37] and training them from scratch. This requires the networks to learn not only the geometry of the point clouds, but also the syntax of the CAD sequence representation.

In this paper, we pose the following question: *In view of the recent breakthrough performance of Large Language Models (LLMs), how can their advanced language understanding be leveraged for CAD reverse engineering?*

To address this question, we base our approach on three key observations: (1) LLMs can generate valid Python code [148, 149], (2) modern CAD software increasingly supports modeling through Python scripting [150], and (3) recent efforts have shown that LLMs can be fine-tuned to process point clouds [151, 152]. As shown in Figure 6.1(b), *we propose* `CAD-Recode`*, a solution for CAD reverse engineering by fine-tuning an LLM to map input point clouds into CAD sketch-extrude sequences represented as Python code.* In particular, instead of crafting a CAD representation, we base our representation on the existing Python CadQuery library [150]. This code-based representation is not only interpretable but also inherently allows for incorporating modular CAD features and design practices such as reusing design elements and abstracting low-level design steps (*e.g.* 3D box to represent a four-line sketch of a square and its extrusion). To learn the mapping between point clouds and CAD Python code, we fine-tune a pre-trained LLM, Qwen2-1.5B [148], augmented with a lightweight, trainable point cloud projector. To train `CAD-Recode`, a potential approach could be using existing sketch-extrude datasets [26, 27] and converting them to Python CadQuery code. However, these datasets are limited in size and design features included due to the efforts required to convert their original proprietary representation into one that is suitable for learning. As a solution, we propose a *procedurally* generated training dataset composed of one million CAD sketch-extrude sequences as Python CadQuery code. This dataset consists of CadQuery Python scripts generated following predefined heuristics with randomized parameter selection. The execution of each generated script results in a parametric CAD model. Unlike existing CAD datasets, this procedurally generated dataset provides an alternative for learning the mapping between point clouds and CAD sketch-extrude sequences in Python code, with full control over the design features, patterns and dataset size included during training. Our contributions can be summarized to:

- A CAD sketch-extrude sequence representation in Python code using CadQuery [150] for CAD reverse engineering.

- `CAD-Recode`, the first LLM-based CAD reverse engineering model designed to predict CAD Python code from point clouds. The model, consisting of a pre-trained LLM and a point cloud projector is trained end-to-end to generate code that reconstructs the input geometry.
- A one million procedurally generated training dataset of CAD sketch-extrude sequences as CadQuery Python code. This provides precise control over dataset size, features, and design patterns included during training, resulting in significant performance improvement. We will make this dataset publicly accessible.
- Extensive experiments on three publicly available datasets show that `CAD-Recode` achieves substantial improvements over state-of-the-art methods in CAD reverse engineering. Moreover, we show that `CAD-Recode`, when operating on point clouds and generating CAD code, can be integrated with an off-the-shelf LLM to perform CAD Question Answering (CAD-QA) and CAD editing from point clouds.

The rest of this chapter is organized as follows. Section 6.2 reviews related works. Section 6.3 introduces the CAD code representation and the proposed dataset. `CAD-Recode` is formulated and described in Section 6.4. Experiments are presented in Section 6.5. Finally, conclusions and future works are given in Section 6.6.

6.2 Related Works

LLM, Point Cloud and CAD: Recent works have explored integrating point clouds with LLMs for various tasks, including 3D generation [153, 154], captioning [155, 151, 156], and question answering [157, 158]. These approaches typically employ complex point cloud encoders, either aligning with CLIP embeddings [152, 159, 160, 161, 162] or directly with LLM feature spaces [151]. Such methods require two-stage training: first pre-training the point cloud encoder, then fine-tuning with the LLM through instruction-based prompts.

In parallel, recent works have started exploring LLMs' capabilities in a range of CAD-related tasks. ReparamCAD [163] infers shape variations from parametric models and

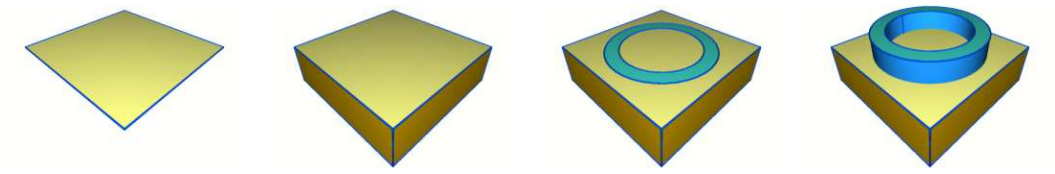
text descriptions, while CADTalk [164] generates semantic descriptions of CAD parts. The works in [165, 166] focus on the generation of CAD models from text using LLMs, and SGP-Benchmark [167] evaluates LLMs' understanding of CAD sketch-extrude sequences using CAD-specific question answering. While Img2CAD [168] attempts CAD reverse engineering from images using GPT-4V [149], it still requires a separate transformer for parameter inference. In contrast, `CAD-Recode` introduces the first approach for point cloud to CAD reconstruction combining point clouds with the sequence modeling capabilities of pre-trained LLMs.

CAD Reverse Engineering from Point Cloud: CAD reverse engineering aims to reconstruct parametric CAD models from 3D shapes (*e.g.*, meshes or point clouds) in a compatible representation with CAD software. A key challenge lies in the choice of this representation. A line of works attempts to address sub-problems of the CAD reverse engineering task by focusing on parameter estimation for edges and surface primitives [16, 117, 18, 17, 20, 96, 22, 29] or reconstructing B-Rep construction history [27, 13, 38, 74]. In order to obtain a representation that is closer to CAD modelling, several methods [136, 137, 119, 101, 120] use Constructive Solid Geometry (CSG), representing models as compositions of 3D primitives and Boolean operations. While this enables reconstruction, the CSG representation diverges from modern CAD workflows [74].

Recent works have adopted the more CAD-aligned sketch-extrude representation, introduced by DeepCAD [26] for CAD generation [100, 36] or predicting extrusion cylinder [23, 25]. Considering the sequential nature of sketch-extrude operations, methods have explored a template-based approach [121] given a rounded voxel input representation. Furthermore, transformer architectures have been investigated for both autoregressive [39] and non-autoregressive [26, 37] prediction of sketch-extrude sequences from point clouds. The work in [122] combines a lightweight pre-trained language model [169] with a point cloud encoder using a diffusion-based approach. Alternative methods using self-supervised [24] or unsupervised [136] learning still face integration challenges due to their non-standard sketch representations (*e.g.*, signed distance functions). In contrast to these approaches that require full parameter learning of specialized networks for both CAD geometry and representation

syntax, we leverage pre-trained LLMs that have been exposed to programming patterns through large-scale training on code repositories. Our method outputs Python code using the CadQuery library [150] that is directly executable and can easily be interpreted. Additionally, we address the data limitation through automated generation of a large-scale training dataset, enabling full control over design features included during training.

6.3 CAD Representation as Code



(a) Sketch-Extrude sequence

```
SOL Line -50 -50 Line 50 50 Line 50 -50 Line -25
-50 Ext. 0 0 1 0 0 0 14 0 NewBody SOL Circle 0 0 40
SOL Circle 0 0 30 Ext. 0 0 1 0 0 0 -10 0 Union
```

(b) DeepCAD representation

```
import cadquery as cq
w = cq.Workplane('XY')
w.box(100,100,14).union(
  w.sketch().circle(40).circle(30,mode='s')
  .finalize().extrude(-10))
```

(c) Our CadQuery representation

Figure 6.2: Sketch-extrude sequence (top) in DeepCAD representation (middle) and our CadQuery code (bottom).

Modern feature-based CAD modeling relies on sequences of 2D sketches and operations to create 3D models. Designers first draw geometric primitives (lines, arcs, circles) on a selected plane, then apply operations like extrusion or revolution to generate 3D geometry [74]. As depicted in Figure 6.2(a), we focus on sketch-extrusion sequences, a fundamental CAD modeling pattern widely adopted in previous works [26, 36, 39]. Below, we present our CAD representation, highlighting its advantages over existing language-like encodings, and describe our procedurally generated training data.

6.3.1 CadQuery Code

Recent approaches in CAD language modeling [26, 36, 102, 39, 37] encode sketch-extrude sequences as numerical vectors representing features and their parameters as shown in Figure 6.2(b). However, this representation constrains the modeling to specific CAD practices, lacks interpretability, and requires post-processing for CAD kernel compatibility. We propose using CadQuery [150] Python code to represent sketch-extrude sequences for CAD reverse engineering, offering the following advantages:

Modularity of CAD Features and Design Practices: Existing language-based CAD reverse engineering methods rely on custom representations of low-level geometric features (lines, arcs, circles) for sketch construction [26, 52]. This approach inherently limits the range of implementable features and design practices. In contrast, CadQuery provides comprehensive built-in CAD functionality, encompassing both low-level features and higher-level geometries like cylinders and boxes as shown in Figure 6.2(c). Furthermore, its programmatic nature enables variable reuse and code modularity. This allows reusing common design features or practices across models, as illustrated by the shared center coordinates across two circles in Figure 6.2 (c). This representation naturally accommodates diverse CAD features and design practices without requiring complex custom encodings or post-processing steps.

Interpretability and LLM Compatibility: The proposed representation, based on Python and CadQuery syntax, presents an alternative to abstract numerical encodings with improved interpretability. Its code-based format facilitates model editing both programmatically and through CAD software. Importantly, this representation aligns with pre-trained LLMs' demonstrated proficiency in Python code generation and manipulation. Indeed, state-of-the-art proprietary LLMs like GPT-4 [149] achieve over 90% accuracy on the Python code HumanEval benchmark [170], while even lightweight open-source models such as Qwen2-1.5B [148] show promising code generation capabilities. Hence, this code representation facilitates fine-tuning of pre-trained LLMs for the specific task of reverse engineering point clouds into CAD Python code and opens the doors for new capabilities with off-the-shelf LLMs.

6.3.2 Procedurally Generated Training Dataset

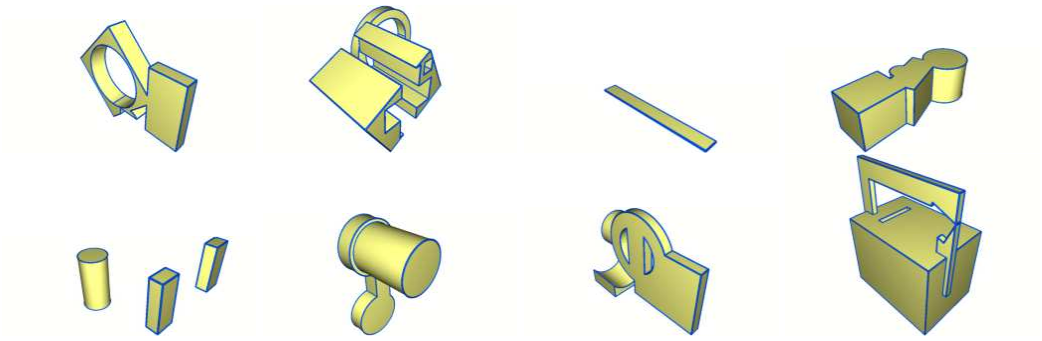


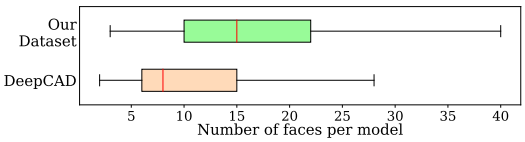
Figure 6.3: Examples of procedurally generated CAD models.

The training of current CAD sketch-extrude reverse engineering methods [26, 102, 39, 37, 122] predominantly rely on datasets collected from CAD model repositories [28, 41, 27]. Considerable efforts are required to parse the CAD models from their original proprietary representations to a suitable one for deep learning [26, 27]. As a result, existing datasets are restricted not only in scale, but also in control over the design features and patterns included in training.

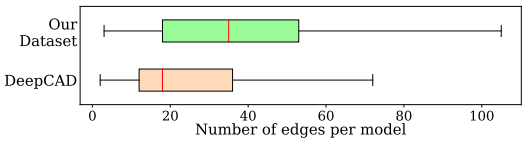
To address these limitations, we propose to procedurally generate a training dataset of one million CAD models in the form of sketch-extrude sequences written in Python CadQuery [150] code. Our proposed pipeline randomly generates sketch and CAD operation parameters guided by topological and geometrical heuristics to ensure control over the amount of generated models and the features in the generated codes. The algorithm outlining the steps of this generation pipeline is provided in the Appendix C.

In Figure 6.4, it can be observed that the distribution of CAD models in our training dataset is skewed towards models with larger face and edge count per model with wider interquartile ranges compared to the DeepCAD dataset [26]. As a result, our procedurally generated dataset provides a larger variety of models.

The modularity of CAD features is incorporated by utilizing both low-level primitives (*i.e.* circles, lines, and arcs) and their abstractions (*i.e.* boxes, cylinders, and rectangles) as well as reusing design elements within the generated sequences. In this work, we focus on some



(a) Box-plot graph of the distribution of the number of faces per model.



(b) Box-plot graph of the distribution of the number of edges per model.

Figure 6.4: Our 1 M procedurally generated training dataset displays distributions CAD models that are skewed towards models with larger edge and face count per model than the DeepCAD dataset (160 k models).

aspects of modularity (*i.e.*, reusing point coordinates, extrusion planes, and abstracting basic shapes such as boxes and cylinders). Further modularity features (*e.g.*, reusing functions corresponding to arbitrary CAD parts, additional CAD operations) can also be integrated in the future. Note that although our generated dataset does not include sequences from human-designed CAD models, it offers significant control over the features and design patterns to which the network is exposed during training. Examples of generated CAD models are shown in Figure 6.3.

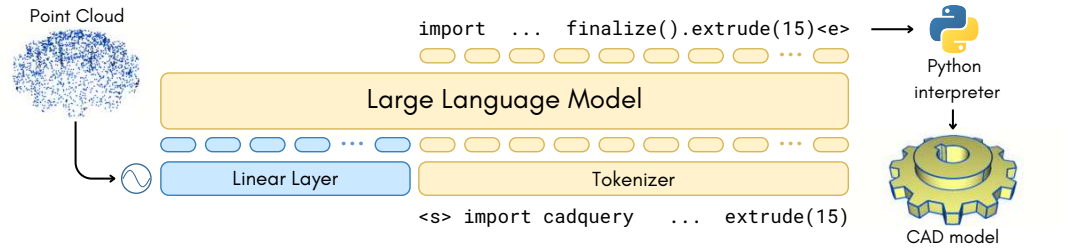


Figure 6.5: Overview of `CAD-Recode`. The pipeline comprises two parts: (1) a point cloud projector (marked blue) (2) a fine-tuned pre-trained LLM (yellow). An input point cloud is processed using (1), and outputs are then passed to an LLM (2), which predicts a CAD sketch-extrude sequence in the form of executable Python code.

6.4 CAD-Recode

Building on the introduced CAD code representation and generated training dataset outlined in the previous section, this section introduces `CAD-Recode`, our proposed model for predicting CAD sketch-extrude sequences as code from input 3D point clouds. We formalize the problem

of CAD code prediction, describe the architecture of `CAD-Recode`, and detail its training and inference processes.

6.4.1 Problem Statement

Let us denote the set of all possible code strings as Σ^*, where each code string is composed of elements from the finite set of alphanumeric characters and operators in the programming language Σ. Let $\phi_{\text{syn}} : \Sigma^* \to \{\texttt{True}, \texttt{False}\}$ represent the syntactic validation function for Python programming rules (*e.g.*, variable declarations, expression syntax, and statement structure), and $\phi_{\text{cad}} : \Sigma^* \to \{\texttt{True}, \texttt{False}\}$ denote the validation function for CAD-specific rules. The latter includes the syntactic validity of the code *w.r.t.* to the CAD library, *i.e.* CadQuery [150], and the geometric validity of the reconstructed model from the code (*e.g.*, an extrusion can only be applied on a closed loop of 2D primitives, a circle radius cannot be negative). An executable valid CAD code can be formally described by a code string $C \in \mathcal{C}$, where

$$\mathcal{C} = \{w \in \Sigma^* \mid \phi_{\text{syn}}(w) \wedge \phi_{\text{cad}}(w)\}\,, \tag{6.1}$$

represents the set of all valid CAD codes. This formulation ensures that any code string w in $\mathcal{C}$ satisfies both the syntactic requirements of Python (ϕ_{syn}) and the CAD code validation rules (ϕ_{cad}). Let $\mathcal{X} = \{\mathbf{p}_i\}_{i=1}^{n} \in \mathbb{R}^{n\times 3}$ denote an input point cloud, where each point $\mathbf{p}_i \in \mathbb{R}^3$ represents 3D Cartesian coordinates. The objective of `CAD-Recode` is to learn a mapping

$$\begin{aligned} \Psi &: \mathbb{R}^{n\times 3} \to \mathcal{C}\,, \\ C &= \Psi(\mathcal{X})\,, \end{aligned} \tag{6.2}$$

that maps the input point cloud to a valid CAD code $C \in \mathcal{C}$ such that the code, when executed, produces a CAD model geometrically approximating the input point cloud $\mathcal{X}$. Note that the CAD code execution results in a Boundary-Representation (B-Rep) [13]. Unlike meshes or point clouds, B-Rep is a parametric representation of the CAD model's geometry, enabling seamless integration into modern CAD software and allowing for further modifications. The goal of `CAD-Recode` is to infer the CAD code describing the design steps of the CAD model,

Method	Train Dataset		DeepCAD Test Set				Fusion360 Test Set			
	Name	Size	Mean CD↓	Med. CD↓	IoU↑	IR ↓	Mean CD↓	Med. CD↓	IoU↑	IR↓
DeepCAD [26]	DeepCAD	160k	42.5	9.64	46.7	7.1	330	89.2	39.9	25.2
PrismCAD [121]	DeepCAD	127k	–	4.28	72.1	16.2	–	4.75	65.3	18.0
Point2Cyl [23]	DeepCAD	35k	–	4.27	73.8	3.9	–	4.18	67.5	3.2
HNC-CAD [100]	DeepCAD	125k	–	8.64	65.3	5.6	–	36.8	63.5	7.3
MultiCAD [102]	DeepCAD	160k	–	8.09	–	11.5	–	42.2	–	16.5
TransCAD [37]	DeepCAD	140k	32.3	4.51	65.5	1.1	78.6	33.4	60.2	2.4
CAD-Diffuser [122]	DeepCAD	160k	–	3.02	74.3	1.5	–	3.85	63.2	1.7
CAD-SIGNet [39]	DeepCAD	160k	3.43	0.28	77.6	0.9	7.37	0.48	65.6	1.6
CAD-Recode	DeepCAD	160k	1.98	0.27	80.7	**0.0**	3.37	0.52	67.6	**0.1**
CAD-Recode	Ours	1M	**0.30**	**0.16**	**92.0**	0.4	**0.35**	**0.15**	**87.8**	0.5

Table 6.1: Comparison of CAD reverse engineering methods on DeepCAD and Fusion360 datasets. Our `CAD-Recode` trained on the 160 k DeepCAD dataset demonstrates an improvement over existing state-of-the-art methods both in terms of geometric fidelity and validity of the generated sketch-extrude sequences. Our procedurally generated dataset provides a significant boost in the prediction quality.

that when executed results in a B-Rep.

6.4.2 Proposed Model Architecture

`CAD-Recode` builds on pre-trained LLMs and their prior exposure to Python code, augmenting these with point cloud processing capabilities and CAD-specific Python code knowledge. As shown in Figure 6.5, its architecture consists of two components: (1) a point cloud projector mapping the 3D point cloud into learnable tokens, and (2) a pre-trained LLM-based auto-regressive CAD code decoder.

Point Cloud Projection Module: `CAD-Recode` introduces a lightweight projection module Ψ_p that directly maps a dense point cloud $\mathcal{X} \in \mathbb{R}^{n \times d_p}$, where $d_p = 3$ corresponds to the dimension of point coordinates, into a sequence of $n_p \ll n$ query tokens $\mathbf{Q}_p = [\mathbf{q}_p^1, \ldots, \mathbf{q}_p^{n_p}] \in \mathbb{R}^{n_p \times d_q}$, of embedding dimension d_q. The point cloud projector, trained in an end-to-end manner with the CAD code decoder module, consists of three simple components: (1) furthest point sampling to downsample the input point clouds to n_p points, (2) Fourier positional encoding [153] of coordinates, and (3) a linear layer projecting the encoded coordinatesinto $\mathbf{Q}_p$.

LLM as CAD Code Decoder: Our CAD code decoder, denoted as Ψ_{LLM}, adapts a pre-

trained LLM for the specific task of CAD code generation. We leverage the Qwen2-1.5B model [148] as our LLM backbone, chosen for its balanced trade-off between model capacity and computational requirements. The decoder's input consists of point query tokens $\mathbf{Q}_p$ from the point cloud projector, augmented with n_t code tokens $\mathbf{Q}_t \in \mathbb{R}^{n_t \times d_q}$ obtained by tokenizing the input code as in [148]. The complete input sequence is denoted as $[\mathbf{Q}_p; \mathbf{Q}_t] \in \mathbb{R}^{(n_p+n_t)\times d_q}$, where semicolon indicates concatenation along the sequence dimension. The LLM decoder generates the CAD code sequence through next-token prediction. As in [148], each predicted token is mapped to a symbol from the vocabulary Σ, which includes alphanumeric characters and operators.

Overall, `CAD-Recode` repurposes the LLM's sequence modeling capabilities for the specialized task of translating geometric point clouds into executable CAD code.

6.4.3 Training and Inference Details

Training Strategy: Our training process consists of a single stage. The model operates on query tokens of dimension $d_q = 1536$ and processes input point clouds downsampled to $n_p = 256$ points. Gaussian noise with mean zero and standard deviation of 0.01 is added to the coordinates of the input points with a probability of 0.5 per model. The network is trained on the procedurally generated CAD codes, hence exposed to the CAD features and design practices that were included in the algorithm. The training objective minimizes the Negative Log-Likelihood (NLL) of the target CAD code sequence, using special tokens ($<s>$ and $<e>$) to demarcate sequence boundaries. The point cloud projector Ψ_p learns geometric features from scratch, while the pre-trained decoder Ψ_{LLM} is fine-tuned for CAD code generation.

Inference Strategy: At inference time, the point cloud projector Ψ_p processes the input point cloud $\mathcal{X}$ to generate query tokens $\mathbf{Q}_p$, which are then fed to the decoder along with the start token $<s>$. The model autoregressively generates CAD code tokens until producing a complete code sequence C ending with token $<e>$. Following [39], we employ a test-time sampling approach where we generate ten distinct CAD code candidates, each from a different sampling of the input point cloud. For each candidate, we sample points from the

predicted CAD model and compute the Chamfer distance *w.r.t.* the input point cloud. The candidate with the minimum Chamfer distance is selected as the final output. This verification step effectively favors executable CAD code solutions that are geometrically consistent *w.r.t.* the input point cloud.

6.5 Experiments

In order to validate the effectiveness of `CAD-Recode`, we conduct a series of experiments across two different scenarios. The first scenario focuses on the reverse engineering task, where the goal is to reconstruct a CAD sketch-extrude sequence in Python code from a given input point cloud. The second assesses the interpretability and editibility of the generated CAD code with a proprietary LLM [149].

6.5.1 Reverse Engineering

Experimental Setup: The `CAD-Recode` implementation uses Qwen2-1.5B as the LLM decoder. The training configuration employs the AdamW optimizer with a learning rate of 0.0002 and weight decay of 0.01, while maintaining other parameters at their default values from the HuggingFace implementation [171], including the cosine learning rate scheduler. The training process is conducted for 100 k iterations, incorporating an initial warmup period of 1 k iterations. Using a single NVIDIA H100 GPU with a batch size of 18, the complete training process takes approximately 12 hours.

`CAD-Recode` is evaluated on three test datasets: DeepCAD [26] (8046 models), Fusion360 [27] (1725 models), and the real-world CC3D [118] (2973 models). The point clouds are obtained by sampling points on the meshes for DeepCAD and Fusion360. The CC3D dataset provides a real-world scenario with input point clouds sampled from actual 3D scans of CAD models containing surface noise, smoothed edges and missing parts.

Metrics: To evaluate the quality of the predicted CAD sketch-extrude sequences, we use three metrics: Chamfer Distance (CD) [39], Intersection over Union (IoU) [122], and Invalidity

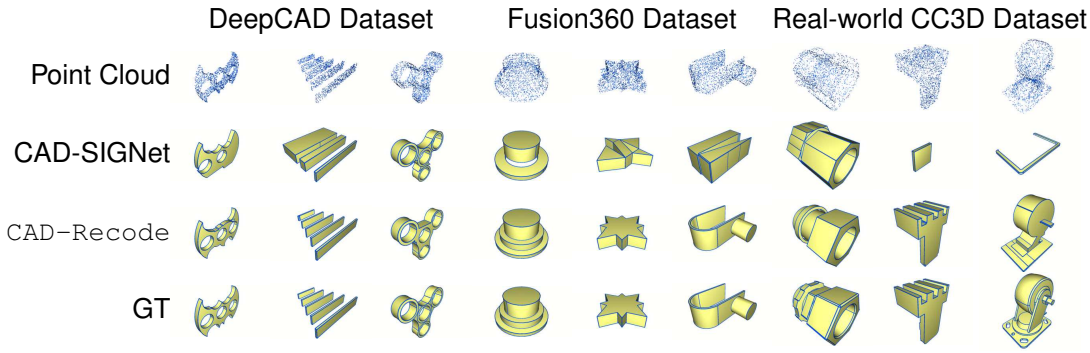


Figure 6.6: Qualitative results on the DeepCAD, Fusion360, and CC3D datasets. For each input point cloud (first row), we compare CAD models produced by CAD-SIGNet (second) and our `CAD-Recode` trained on our dataset (third) with a ground truth CAD model (bottom row). While CAD-SIGNet often fails to restore the general shape, CAD-Recode outputs only slightly deviate from ground truth in most cases.

Ratio (IR) [26]. We report both mean and median CD values computed using 8192 points to assess geometric accuracy. Reported CD values have been multiplied by 10^3. The IoU is computed from the resulting CAD model meshes and expressed as a percentage. The IR indicates the percentage of generated sequences that fail to produce a valid CAD model.

Results & Analysis: Table 6.1 presents results on the test sets of DeepCAD and Fusion360 datasets, where `CAD-Recode` establishes new state-of-the-art performance across all metrics. Note that the results of state-of-the-art methods in Table 6.1 are borrowed from [122], except for CAD-SIGNet [39], MultiCAD [102], TransCAD [37], and DeepCAD [26] which were taken from [39] and [37]. First, we convert the DeepCAD dataset (160k models) to CadQuery Python code and use it to train `CAD-Recode` (results are in row before last of Table 6.1). When trained on DeepCAD dataset as existing methods, `CAD-Recode` outperforms them in almost all metrics. These results showcase the effectiveness of `CAD-Recode` and the proposed CAD code representation.

Training on 1M generated samples results in substantial improvements in CD and IoU metrics while maintaining a negligibly low invalidity ratio (last row of Table 6.1), reflecting significantly better geometric fidelity in the predicted CAD models. `CAD-Recode` demonstrates a ten-fold improvement in mean CD and an increase of IoU by over 10% on both DeepCAD

Method	Mean CD↓	Med. CD↓	IoU↑	IR↓
DeepCAD [26]	–	263	–	12.7
CAD-SIGNet [39]	14.82	2.90	42.6	2.5
CAD-Recode	**0.76**	**0.31**	**74.2**	**0.3**

Table 6.2: Results on the CC3D dataset, where input point clouds are sampled from real 3D scans. CAD-Recode significantly outperforms DeepCAD, and CAD-SIGNet.

and Fusion360 datasets compared to the existing best methods. These results confirm that our large-scale procedurally generated training dataset provides substantial benefits.

As illustrated in Figure 6.6, this translates to consistent reconstruction quality, where CAD-Recode reliably produces CAD models that accurately capture the geometry from the input point cloud. In contrast, CAD-SIGNet [39] can generate shapes that deviate significantly from the target geometry, further highlighting the advantages of our approach.

Real-world Scenario: In Table 6.2, we evaluate CAD-Recode on the real-world CC3D dataset, where input point clouds are sampled from 3D scans and contain artifacts such as surface noise, smooth edges, and missing parts. Even under these challenging conditions, our method achieves significant improvements over CAD-SIGNet [39], with a 89% lower median CD and a 30% higher IoU, while maintaining a low IR. From the CC3D qualitative results in Figure 6.6, CAD-Recode is able to recover geometries that are much closer to the ground truth than current state-of-the-art. However, it can be observed that CAD-Recode still lacks the expressiveness to model complex shapes that contain operations beyond the extrusion operation such as revolution and fillet. This can be attributed to the choice of features and design practices in the procedurally generated training dataset. Nevertheless, we believe that this can be addressed in future works by incorporating further features in the dataset generation procedure. Our results on CC3D are compared with methods previously reported for this dataset [39], namely CAD-SIGNet and DeepCAD.

Invalid Predictions: The invalidity rate of CAD-Recode predictions is very low, below 1% on the DeepCAD [26], Fusion360 [27] and real-world CC3D [118] dataset. Some examples of invalid code predictions are presented in Figure 6.7. Invalid predictions happen when the CAD model contains features of dimension smaller than the resolution induced by quantization

Method	Train Dataset		Test-time	DeepCAD			Fusion360			Real-World CC3D		
	Name	Size	Sampling	CD↓	IoU↑	IR↓	CD↓	IoU↑	IR↓	CD↓	IoU↑	IR↓
Previous best [39]	DeepCAD	160 k	✓	3.43	77.6	0.9	7.37	65.6	1.6	14.80	42.6	4.4
CAD-Recode	DeepCAD	160 k	✓	1.98	80.7	0.0	3.37	67.6	0.1	3.79	56.4	0.0
CAD-Recode	Ours	160 k	✓	0.54	88.3	0.3	0.66	82.0	0.1	1.27	69.0	0.2
CAD-Recode	Ours	1 M	✓	**0.30**	**92.0**	0.4	**0.35**	**87.8**	0.5	**0.76**	**74.2**	0.3
Previous best [39]	DeepCAD	160 k	✗	6.81	77.3	4.4	14.5	58.4	9.3	32.59	39.1	15.5
CAD-Recode	Ours	1 M	✗	**0.75**	**89.3**	4.9	**0.89**	**84.2**	8.7	**3.05**	**65.6**	16.8

Table 6.3: Ablation of training data and test-time sampling. The results demonstrate the advantage of training on our procedurally generated data, while the test-time sampling helps reducing the invalidity ratio. CD stands for mean Chamfer distance.

(Figure 6.7(a) and (b)) or when the ground truth CAD model contains features, such as revolution or B-spline, that are not present in the training dataset (Figure 6.7(c) and (d))).

Ablation Study: To evaluate the different components of CAD-Recode, we conduct a comprehensive ablation study on the amount of training data, test-time sampling, and the number of input points and model parameters.

Training CAD-Recode on 160 k procedurally generated samples using the method described in Section 6.3.2 leads to significant improvements in geometric fidelity of the predicted samples over training on the DeepCAD dataset with the same amount of data (see row 2 and 3 of Table 6.3). Furthermore, scaling our training dataset to 1 M samples provides further improvements across all datasets (row 4 of Table 6.3). As compared to DeepCAD training dataset, our procedural dataset generation provides a better way of learning the mapping between point clouds and CAD codes which can be further improved by scaling up the dataset size.

We investigate the effectiveness of the test-time sampling approach that generates multiple CAD code predictions through different point cloud samplings, as described in Section 6.4.3. As shown in the third and last row of Table 6.3, the test-time sampling approach mainly helps reducing the ratio of invalid predicted CAD codes (IR). For comparison, CAD-SIGNet [39] employs a probability-based sampling. Yet, even without test-time sampling our method still performs better on the reconstruction metrics than CAD-SIGNet [39].

Results in Table 6.4 show the architecture ablation results on all metrics. Results show

```
import cadquery as cq
w0 = cq.Workplane('XY', origin=(0, 0, 0))
w1 = cq.Workplane('YZ', origin=(0, 0, 0))
r = w0.workplane(offset=0 / 2).cylinder( 0 , 98)
  .union(w1.workplane(offset=0 / 2).cylinder( 0 , 100))
```

(a) The ground truth model contains three very thin cylinders with height smaller than 1. As a result, CAD-Recode is not able to predict heights with sufficient precision due to quantization and predicts cylinders with height 0, producing an invalid model.

```
import cadquery as cq
w0 = cq.Workplane('XY', origin=(0, 0, 0))
r = w0.sketch().rect(200, 124).push([(-63.5, 25)]).rect(51, 60, mode='s')
    .push([(55, -25)]).rect(50, 60, mode='s').finalize() .extrude(0)
```

(b) As the ground-truth model has thickness less than 1, CAD-Recode predicts an extrusion distance of 0 as a quantized approximation (highlighted in yellow), resulting in an invalid CAD model.

```
import cadquery as cq
w0 = cq.Workplane('YZ', origin=(34, 0, 0))
w1 = cq.Workplane('XY', origin=(0, 0, 44))
r = w0.sketch().segment((-7, -35), (11, -36)).segment((11, -24)).arc((1,
↪  -14),
    (6, -2)).segment((-1, 19)).segment((11, 23)).segment((11, 28))
    .segment((11, 29)).segment((12, 29)).segment((12, 35))
    .segment((-4, 36)).close().assemble().finalize().extrude(-133)
  .union(w0.sketch().segment((5, -7), (14, -2)).segment((8, 8)).arc((7, 0),
    (5, -7)).assemble().finalize().extrude(63))
  .union(w1.sketch().arc((-100, 12), (-85, 10), (-70, 5)).arc((-68, 6),
    (-66, 5)).arc((-59, 4), (-52, 2)). arc((-51, 3), (-50, 4)) .arc((-72, 7),
    (-90, 12)).close().assemble().finalize().extrude(-88))
```

(c) The ground-truth CAD model is created with B-spline primitives. Since CAD-Recode supports only arc, circle and line primitives, it tries to approximate the solution with multiple arcs, but fails to provide a valid CAD model. In particular, the prediction contains an arc constructed from three co-linear points (highlighted in yellow), which raises an error in CadQuery.

```
import cadquery as cq
w0 = cq.Workplane('XY', origin=(0, 0, -79))
r = w0.sketch().segment((-100, -1), (-91, -1)).arc((0, -93),(91, -1))
    .segment((100, -1)).segment((100, 1)).segment((91, 1)).arc((0, 99),
    (-91, 1)).segment((-100, 1)).close().assemble().push([(0, -2)])
    .circle(90, mode='s').finalize().extrude(-2)
  .union(w0.workplane(offset=140 / 2).cylinder(140, 72))
  .union( w0.sketch().segment((-51, 15), (-50, 15)).arc((0, -53),(50, 15))
          .segment((51, 15)).segment((51, 27)).segment((48, 27)).arc((0,
      -53),
          (-48, 27)).segment((-51,
  ↪  27)).close().assemble().finalize().extrude(159))
```

(d) The ground-truth CAD model is created with a revolution operation. Since CAD-Recode supports only extrusion operation, it tries to approximate the solution with multiple arcs. However, one of the sketch (highlighted in yellow) results in a self-intersecting loop, which is not a valid face.

Figure 6.7: Examples of invalid predictions. Each row contains the ground-truth CAD model (left) and an invalid predicted CadQuery Python code (right). The CAD models in (a) and (b) are taken from the DeepCAD dataset and the CC3D dataset for (c) and (d). Invalid predictions mostly take place when the ground-truth contains features of very small dimension with respect to the size of the CAD model as in (a) and (b), or when the ground-truth model contains operations other than the ones supported as in (c) and (d).

Points	Model Size	DeepCAD				Fusion360				CC3D			
		Mean CD↓	Med. CD↓	IoU↑	IR↓	Mean CD↓	Med. CD↓	IoU↑	IR↓	Mean CD↓	Med. CD↓	IoU↑	IR↓
64	0.5 B	0.42	0.20	88.5	0.1	0.58	0.22	82.1	0.1	0.87	0.45	70.1	0.1
	1.5 B	0.36	0.19	89.3	0.0	0.43	0.20	83.7	0.1	0.83	0.42	71.2	0.0
128	0.5 B	0.36	0.18	89.9	0.1	0.43	0.18	84.3	0.1	0.87	0.38	71.9	0.1
	1.5 B	**0.27**	0.17	91.0	0.1	0.36	0.17	86.1	0.1	0.79	0.34	73.1	0.1
256	0.5 B	0.36	0.17	90.6	0.2	0.40	0.17	85.4	0.4	0.87	0.36	72.6	0.1
	1.5 B	0.30	**0.16**	**92.0**	0.4	**0.35**	**0.15**	**87.8**	0.5	**0.76**	**0.31**	**74.2**	0.3

Table 6.4: Ablation of architecture details.

that for the same size of input point clouds Qwen1.5b always produces better geometric performance (median CD and IoU) than Qwen0.5b. This can be attributed to the higher number of parameters as well as to the better ability of the model to produce valid python code before fine-tuning. Furthermore, increasing the size of the input point cloud demonstrates a similar pattern, with Qwen1.5b with an 256 input points appears to be the set of architecture parameters leading to the best performance. Note that the mean CD is a metric that is very sensitive to outlier predictions. While Qwen1.5b with 256 input points appears to result in the highest IR, it is negligibly low on all datasets (less 0.5%). This can also be explained by the fact that this setting produces more complex CAD sketch-extrude sequences, making them more susceptible to errors. Note that a key idea of our method is to leverage pre-trained LLMs as decoder of Python code. In the absence of LLM-based CAD reverse engineering methods, we compare our approach to SOTA methods despite the difference in model sizes. For reference, CAD-SIGNet contains 6 M parameters.

Test-time Sampling: The ablation study demonstrates the effectiveness of our test-time sampling strategy. This approach generates multiple plausible solutions by sampling different input point clouds. Figure 6.8 illustrates the qualitative results from different sampling instances. While `CAD-Recode` successfully captures the overall geometry across different samplings, fine-grained details may vary in reconstruction quality due to the relatively sparse point cloud input. However, this limitation can be effectively addressed by leveraging multiple sampling iterations to capture different aspects of the input geometry.

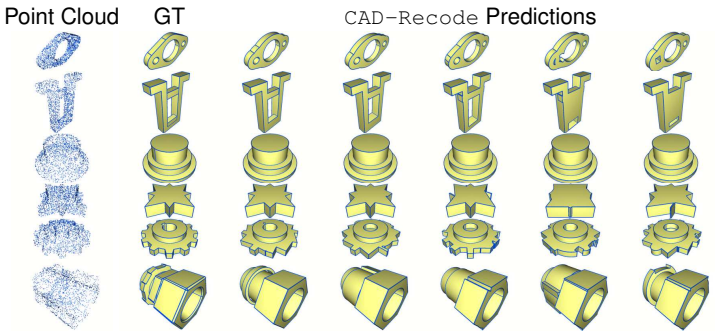


Figure 6.8: CAD-Recode predictions from different point cloud sampling on DeepCAD, Fusion360, and real-world CC3D datasets. For each prediction, 256 points are sampled randomly from the input point cloud.

6.5.2 CAD-QA and Editability

CAD-QA and LLM Interpretability: CAD SGP-Bench [167] is a benchmark of 1000 CAD-specific Question Answering (CAD-QA) tasks that test LLMs' understanding of CAD model geometry from sketch-extrude sequences encoded as in DeepCAD [26]. These questions require analyzing various geometric aspects, such as relative sizes and 3D primitive types. We extend this benchmark to evaluate CAD reverse engineering methods by using point clouds as input instead of CAD sequences. Our evaluation follows a two-stage process: first predicting sketch-extrude sequences from point clouds as CadQuery code with CAD-Recode, then using GPT-4o [149] to answer CAD-specific questions. Without requiring additional interpretation hints, our approach achieves 76.5% accuracy on this CAD-QA task (Table 6.5). For comparison, we evaluate two baseline approaches: CAD-SIGNet [39] and PointLLM [151]. When using CAD-SIGNet's output with GPT-4o, even with provided interpretation hints explaining the sequence format, the accuracy reaches only 63.2%. PointLLM, which directly processes point clouds for language tasks, achieves 42.3% accuracy when prompted with the CAD-specific questions. These results demonstrate that CAD-Recode effectively captures CAD geometric information while generating an output in a format that proprietary LLMs can naturally interpret and process.

Figure 6.9(a) depicts an example of point cloud and question that was used to obtain these results. In this particular question, the task is to deduce the number of holes present

Question: How many holes does the CAD object in the image have?
A) Two, B) Three, C) Four, D) Five
Correct Answer: B

(a) Point cloud and corresponding CAD-specific question

The CAD object in the image has five holes.

(b) PointLLM output from point cloud

To determine the number of holes in the CAD object, let's break down the CAD code step by step:
1. **Extrusion Command:**
Start;Ext:(128,130,128,128,130,128,128,128,NewBody,96);
This command sets up an extrusion with specific parameters and indicates that a new body will be created.
2. **First Loop:**
Line:(43,0);Arc:(213,0,255,43);Line:(213,85);Arc:(43,85,0,43);EOL;
- Line:(43,0) - A line from the starting point to (43,0).
- Arc:(213,0,255,43) - An arc with midpoint (213,0) and endpoint (255,43).
- Line:(213,85) - A line from the previous endpoint to (213,85).
- Arc:(43,85,0,43) - An arc with midpoint (43,85) and endpoint (0,43).
- EOL - End of the loop.

This forms a closed loop, which is a profile for extrusion.
3. **Second Loop:**
Circle:(43,43,43,51);EOL;
- Circle:(43,43,43,51) - A circle with the most left point at (43,43) and the most top point at (43,51).
- EOL - End of the loop.

This forms another closed loop, which is another profile for extrusion.
4. **End of Sketch and Extrusion:**
EOF;EOS;End;
- EOF - End of face.
- EOS - End of sketch.
- End - End of the code.

From the above analysis, we can see that the CAD object consists of:
- One main profile formed by the first loop.
- One circular profile formed by the second loop.

The circular profile represents a hole within the main profile. Since there is only one circular profile, the CAD object has one hole.
Answer: A

(c) GPT-4o output from CAD-SIGNet prediction

Let's break down the CAD code step by step to understand the 3D object it generates and determine the number of holes.
1. **Workplane Creation:**
w0 = cq.Workplane('XY', origin=(0,0,1))
This line creates a workplane on the XY plane with an origin at (0,0,1).
2. **Sketching the Shape:**
r = w0.sketch()
This initializes a sketch on the workplane.
3. **Defining the Shape:**
.segment((-66,-34),(50,-34)) .segment((50,-32)) .arc((100,0),(50,32)) .segment((50,34)) .segment((-66,34)) .segment((-66,32)) .arc((-100,0),(-66,-32)) .close()
These lines define a closed shape with segments and arcs. The shape appears to be symmetrical along the Y-axis.
4. **Adding Holes:**
.assemble() .push([(-66,0)]).circle(10, mode='s') .push([(0,0)]).circle(10, mode='s') .push([(66,0)]).circle(10, mode='s') .finalize()
These lines add three circles (holes) to the shape at positions (-66,0), (0,0), and (66,0).
5. **Extruding the Shape:**
.extrude(-4)
This line extrudes the shape by -4 units, creating a 3D object with the defined shape and holes.
From the code, we can see that three circles (holes) are added to the shape. Therefore, the CAD object has three holes.
Answer: B

(d) GPT-4o output from `CAD-Recode` prediction

Figure 6.9: Example of Point cloud CAD-QA (a) and answers provided by PointLLM (b), CADSIGNet and GPT-4o (c) and `CAD-Recode` and GPT-4o (d).

Method	CAD-QA Accuracy↑
PointLLM [151]	42.3
CAD-SIGNet [39] → GPT-4o	63.2
CAD-Recode → GPT-4o	**76.5**

Table 6.5: Point cloud CAD-specific question answering (CAD-QA) on the SGP-Bench benchmark. Our CAD-Recode supplied with a GPT-4o significantly outperforms baseline methods.

in the CAD model given the point cloud as input. Figure 6.9(b), the answer provided by PointLLM is shown and it can be observed that PointLLM is unable to retrieve the correct answer. It is worth noting that PointLLM is a network trained to answer semantic questions about object given its point cloud representation, as result in most cases the network is unable to describe geometric CAD-specific questions. For both CAD-SIGNet and CAD-Recode, the point cloud CAD-QA is done in a two step process. First the sketch-extrude is sequence is predicted from each network, then the sequence along with the question is passed through GPT-4o. Note that for CAD-SIGNet an interpretative hint is provided to provide context on the structure of the sequence. A sample output for CAD-SIGNet and GPT-4o can be found in Figure 6.9(c), and in Figure 6.9(d) for CAD-Recode and GPT4-o. As the sequence was incorrectly predicted by CAD-SIGNet the answer to the question is wrong (1 hole), whereas the prediction from CAD-Recode captured better the geometry of the input point cloud leading to a correct answer. It is worth noting, that despite not being provided any information about CadQuery Python code in the prompt, GPT-4o is able to breakdown the predicted sequence into its primitive components and provide correct and accurate geometric descriptions. This can be explained by the fact that LLMs are exposed to large amounts of code data during training. As a result, the CadQuery Python representation of CAD models is appropriate for LLM-based downstream tasks.

Editing Pipeline: Leveraging the interpretable nature of our code-based output, we present an automated editing pipeline using GPT-4o [149]. The goal of this pipeline is to integrate automated editability capabilities to CAD-Recode. To this end, we present a simple process using an off-the-shelf LLM, GPT-4o [149]. Starting from an output CAD Python code from

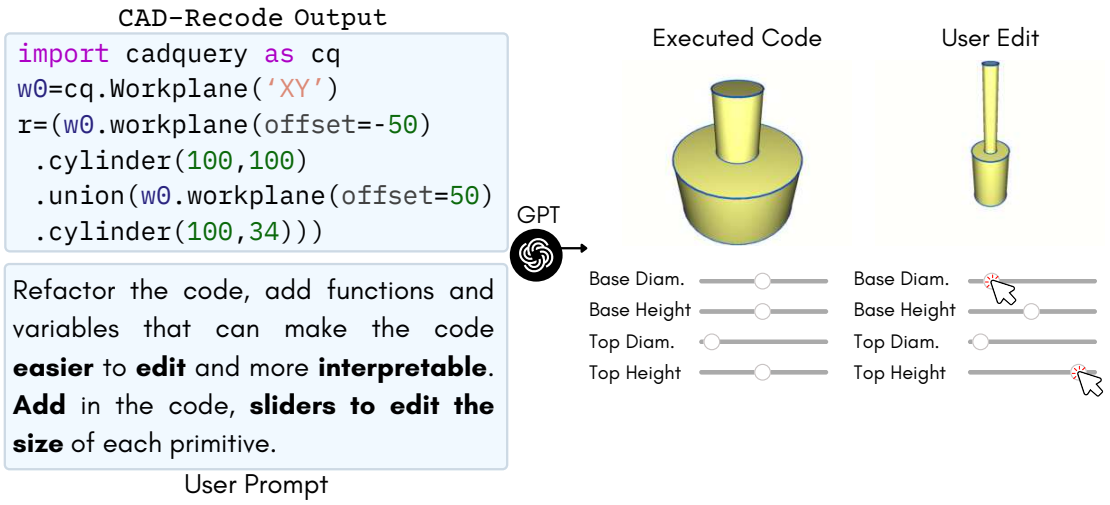


Figure 6.10: Interactive editing of a CAD model. Given the code output from CAD-Recode and a generic prompt, GPT-4o allows automated and interactive editing of the CAD model.

CAD-Recode, we prepare a simple and generic prompt for the LLM to generate a refactored version of the code such that when executed the user can change the dimensions of each primitive. The LLM is able to generate a code with comments that describe the different primitives semanticallly and include appropriate variables for the dimensions of each of the primitive, such as the height and the diameter of each cylinder. The code generated by the LLM, can be directly executed in a Jupyter notebook with the CadQuery and ipywidgets libraries. Figure 6.10 shows an example of the pipeline, the generated sliders and how can the shape be then edited. This demonstrates that the CAD representation as Python code within a reverse engineering scenario opens the door to new applications when combined with LLMs.

6.6 Conclusion

In this chapter, we have presented a novel approach to feature-based CAD reverse engineering by leveraging pre-trained Large Language Models and a Python-based CAD representation, effectively addressing the fundamental challenges identified in Section 1.2.

Our approach tackles the lack of unified CAD representation standards by introducing a code-based representation that is both interpretable by humans and directly executable in standard CAD environments. The scanning imperfections challenge is mitigated through improved reconstruction accuracy that significantly outperforms existing methods on three datasets, including the real-world CC3D dataset containing authentic scanning artifacts. Moreover, by generating code that explicitly captures design steps and parametric relationships, we enhance the recovery of design intent in a format that naturally supports subsequent editing.

The procedurally generated dataset of one million CAD programs directly addresses the CAD data challenge outlined in Section 1.2, providing unprecedented control over both the quantity and quality of training examples. This contribution not only improves model performance but also establishes a methodology for generating high-quality synthetic data for future CAD-related deep learning research.

Beyond these technical contributions, we have demonstrated that combining pre-trained LLMs with geometric understanding enables powerful new workflows where designers can reconstruct CAD models from point clouds and modify them through natural language. This capability represents a significant step toward more intuitive and accessible CAD tools, potentially reducing the expertise barrier that has traditionally constrained CAD usage.

We identify several promising directions for future research: (1) further exploiting the modularity of the proposed CAD code representation to support more complex modeling operations beyond extrusions; (2) scaling up both the LLM architecture and the procedurally generated dataset to enable reverse engineering of increasingly complex CAD models; and (3) exploring interactive workflows that combine the strengths of automated reconstruction with human design expertise. These avenues of investigation could further advance the field toward the goal of fully automated yet flexible CAD reverse engineering that seamlessly integrates into industrial design workflows.

Chapter 7

Conclusion

This concluding chapter presents a summary of the principal contributions to the CAD reverse engineering field. Additionally, we examine promising directions for future research that could extend our methodologies and further advance the field of automated CAD reconstruction.

7.1 Summary

This thesis has addressed the problem of 3D CAD reverse engineering, the process of automatically reconstructing parametric CAD models from 3D scanned data. This task faces several fundamental challenges that have hindered its practical implementation in industrial workflows. The first challenge stems from the 3D scanning processes, which introduce artifacts such as surface noise, missing data, and smoothed edges that corrupt the geometric fidelity of the input data. Second, the lack of unified CAD representation standards creates significant obstacles for translation between different formats and systems. Third, recovering design intent, the underlying rationale governing how a shape can be modified while respecting its functionality, presents a complex problem. Finally, the limited availability of high-quality CAD datasets with construction history annotations restricts the development of effective learning-based approaches.

Our first contribution, CADOps-Net, addressed these challenges from the perspective of Boundary Representation (B-Rep) analysis. By jointly learning CAD operation types

and steps from B-Rep faces, we demonstrated a novel approach to recovering elements of construction history directly from the final geometry. This network successfully segmented B-Rep faces according to their corresponding CAD operations while simultaneously grouping faces constructed at the same step. The joint learning strategy significantly improved accuracy compared to existing methods, particularly for the challenging task of step segmentation. To support this research, we introduced the CC3D-Ops dataset containing over 37 k B-Reps with operation type and step annotations, addressing the data challenge by providing more complex models that better reflect real-world industrial designs. Furthermore, we showed that predictions from CADOps-Net could be leveraged to recover original 2D sketches, representing an important step toward complete design recovery from B-Reps.

Building upon this foundation, TransCAD expanded our approach to address the end-to-end reverse engineering pipeline from point clouds to parametric CAD models. This hierarchical transformer architecture eliminated the need for an intermediate B-Rep representation. TransCAD leveraged a novel hierarchical learning strategy that mirrored the structure of CAD operations and sketches, enabling more effective feature extraction and parameter prediction. Additionally, we introduced a specialized loop refiner module to overcome limitations in sketch primitive parameter regression. From a methodological perspective, we contributed the mean Average Precision of CAD Sequence (APCS) metric, addressing critical limitations in existing evaluation frameworks. Our experimental results demonstrated that TransCAD achieved state-of-the-art performance while exhibiting robustness to input perturbations characteristic of real-world scanning artifacts.

While TransCAD established the feasibility of direct point cloud to parametric CAD reconstruction, our abraCADabra approach recognized a critical limitation: the reliance on single-solution predictions that fail to explore diverse design possibilities. Drawing inspiration from how expert CAD designers work, considering multiple alternatives before selecting the most appropriate solution, we developed inference-time search strategies that leverage geometric priors from input point clouds to more effectively explore the output probability space. Through a comprehensive analysis of both auto-regressive (CAD-SIGNet) and non-auto-regressive (TransCAD) architectures, we revealed their complementary strengths

across different evaluation contexts. The geometry-guided search strategies we introduced significantly improved reconstruction quality while eliminating invalid CAD predictions. To support evaluation under realistic conditions, we contributed the C3D-Recon dataset, the first collection of realistic 3D scans paired with corresponding parametric CAD design sequences.

Our final contribution, CAD-Recode, represented a paradigm shift that directly addressed all four fundamental challenges through a novel LLM-based approach. Rather than outputting intermediate representations requiring further processing, CAD-Recode translated point cloud data directly into executable Python code that, when executed, reconstructs the complete CAD model within existing CAD systems. This innovation operated at three complementary levels: at the representation level, we reconceptualized CAD sketch-extrude sequences as Python code; for network design, we leveraged the code-understanding capabilities of pre-trained Large Language Models, combining a relatively small LLM decoder with a lightweight point cloud projector; and to overcome data limitations, we created a procedurally generated dataset of one million diverse CAD sequences. The resulting system not only significantly outperformed existing methods across all datasets but also offered two additional benefits: direct integration with commercial CAD software through standard Python interfaces, and interpretability by off-the-shelf LLMs, enabling natural language-based CAD editing and CAD-specific question answering.

7.2 Future Directions

Building upon the foundations established in this thesis, we identify two promising research directions that could further advance CAD reverse engineering toward broader industrial adoption: interactive multimodal reconstruction systems that combine multiple input sources with user guidance, and assembly-based approaches that reconstruct complex objects as hierarchies of functional components.

7.2.1 Interactive Multimodal Reconstruction

A promising direction for future research is the development of interactive multimodal CAD reconstruction systems that combine point cloud data with additional input modalities such as reference photographs, text descriptions, or partial technical drawings. Such systems would allow users to guide the reconstruction process by providing complementary information when scans are incomplete or contain ambiguities—for example, specifying through natural language that a partially visible feature ”should be a threaded hole.” Multimodal Large Language Models (MLLMs) offer a natural framework for this integration, as they can reason across different input types and generate coherent CAD code that incorporates information from multiple sources. However, developing these systems presents substantial challenges, including training models to deduce accurate geometric information from different modalities, and establishing evaluation metrics that account for human-in-the-loop processes. Despite these challenges, interactive multimodal reconstruction holds significant potential for real-world applications, particularly in reverse engineering complex industrial components with critical features that may not be fully captured in scans alone, or in scenarios where domain-specific knowledge from experts needs to be incorporated into the automated reconstruction process.

7.2.2 Reconstruction as Assembly Parts

Another compelling avenue for future work involves approaching CAD reverse engineering as the reconstruction of assemblies rather than monolithic models, more closely reflecting how complex objects are actually designed and manufactured. This approach would decompose the input scan into meaningful components, reconstruct each part separately, and then establish the appropriate geometric relationships and constraints between them. Such a decomposition would not only produce more editable and reusable CAD models but would also better preserve the hierarchical design intent of complex manufactured objects. Large Language Models with their ability to reason about part relationships could play a crucial role in this process, potentially inferring functional relationships between components based on their

geometry and relative positions. Key challenges in this direction include developing robust methods for segmenting point clouds into functionally meaningful parts, inferring appropriate assembly constraints, and handling moving or articulated components where a single scan may not capture the full range of possible configurations. The industrial implications of this approach are particularly significant for sectors like automotive and aerospace, where complex assemblies are the norm and where the ability to reverse engineer not just geometry but also assembly structure could dramatically reduce redesign time and enable more efficient part replacement and maintenance workflows.

References

[1] Christof Ebert and Carlos Henrique C Duarte. “Digital transformation.” In: *IEEE Softw.* 35.4 (2018), pp. 16–21.

[2] David E Weisberg. “History of CAD”. In: *History* (2022).

[3] Aleksander Asanowicz. “Evolution of Computer Aided Design: three generations of CAD”. In: *Architectural Computing from Turing to 2000-eCAADe Conference Proceedings*. Vol. 17. 9. 1999, pp. 4–10.

[4] Kuang-Hua Chang. *e-Design: computer-aided engineering design*. Academic Press, 2015.

[5] Zhuming Bi and Xiaoqin Wang. *Computer aided design and manufacturing*. John Wiley & Sons, 2020.

[6] Mario Hirz et al. “Integrated computer-aided design in automotive development”. In: *Springer-Verl. Berl.-Heidelb. DOI* 10 (2013), pp. 978–3.

[7] Tom Verstraete. “CADO: a computer aided design and optimization tool for turbomachinery applications”. In: *2nd Int. Conf. on Engineering Optimization, Lisbon, Portugal*. 2010, pp. 6–9.

[8] Anh Hoang Truong. *Development of a Computer-Aided-Design-Based Geometry and Mesh Movement Algorithm for Three-Dimensional Aerodynamic Shape Optimization*. University of Toronto (Canada), 2014.

[9] Vinesh Raja and Kiran J Fernandes. *Reverse engineering: an industrial perspective*. Springer Science & Business Media, 2007.

[10] Robin H Helle and Hirpa G Lemu. “A case study on use of 3D scanning for reverse engineering and quality control”. In: *Materials Today: Proceedings* 45 (2021), pp. 5255–5262.

[11] Tamas Varady, Ralph R Martin, and Jordan Cox. “Reverse engineering of geometric models—an introduction”. In: *Computer-aided design* 29.4 (1997), pp. 255–268.

[12] Vinesh Raja. “Introduction to reverse engineering”. In: *Reverse engineering: an industrial perspective*. Springer, 2008, pp. 1–9.

[13] Joseph G. Lambourne et al. “BRepNet: A Topological Message Passing System for Solid Models”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021, pp. 12773–12782.

[14] Pradeep Kumar Jayaraman et al. “Uv-net: Learning from boundary representations”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 11703–11712.

[15] MFCAD. *A dataset of 3D CAD models with machining feature labels*. `https://github.com/hducg/MFCAD`. Online: accessed 02-June-2022.

[16] Yujia Liu et al. “{PC}2WF: 3D Wireframe Reconstruction from Raw Point Clouds”. In: *International Conference on Learning Representations*. 2021. URL: `https://openreview.net/forum?id=8X2eaSZxTP`.

[17] Gopal Sharma et al. “ParSeNet: A Parametric Surface Fitting Network for 3D Point Clouds”. In: *European Conference of Computer Vision (ECCV)*. Springer. 2020, pp. 261–276.

[18] Xiaogang Wang et al. “Pie-net: Parametric inference of point cloud edges”. In: *Advances in neural information processing systems* 33 (2020), pp. 20167–20178.

[19] Lequan Yu et al. “Ec-net: an edge-aware point set consolidation network”. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 386–402.

[20] Xiangyu Zhu et al. “Nerve: Neural volumetric edges for parametric curve extraction from point cloud”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 13601–13610.

[21] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising diffusion probabilistic models”. In: *Advances in neural information processing systems* 33 (2020), pp. 6840–6851.

[22] Haoxiang Guo et al. “Complexgen: Cad reconstruction by b-rep chain complex generation”. In: *ACM Transactions on Graphics (TOG)* 41.4 (2022), pp. 1–18.

[23] Mikaela Angelina Uy et al. “Point2Cyl: Reverse engineering 3D objects from point clouds to extrusion cylinders”. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 11850–11860.

[24] Pu Li et al. “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.

[25] Daxuan Ren et al. “Extrudenet: Unsupervised inverse sketch-and-extrude for shape parsing”. In: *European Conference on Computer Vision*. Springer. 2022, pp. 482–498.

[26] Rundi Wu, Chang Xiao, and Changxi Zheng. “DeepCAD: A Deep Generative Network for Computer-Aided Design Models”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2021, pp. 6772–6782.

[27] Karl D. D. Willis et al. “Fusion 360 Gallery: A Dataset and Environment for Programmatic CAD Construction from Human Design Sequences”. In: *ACM Trans. Graph.* 40.4 (July 2021), pp. 1–24. ISSN: 0730-0301. DOI: 10.1145/3450626.3459818. URL: https://doi.org/10.1145/3450626.3459818.

[28] Sebastian Koch et al. “ABC: A Big CAD Model Dataset For Geometric Deep Learning”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019, pp. 9601–9611.

[29] Kseniya Cherenkova et al. “Spelsnet: Surface primitive elements segmentation by b-rep graph structure supervision”. In: *Advances in Neural Information Processing Systems* 37 (2024), pp. 1251–1269.

[30] Abid Haleem et al. “Exploring the potential of 3D scanning in Industry 4.0: An overview”. In: *International Journal of Cognitive Computing in Engineering* 3 (2022), pp. 161–171.

[31] Stefano Tornincasa, Francesco Di Monaco, et al. “The future and the evolution of CAD”. In: *Proceedings of the 14th international research/expert conference: trends in the development of machinery and associated technology*. Vol. 1. Citeseer. 2010, pp. 11–18.

[32] Rana Hanocka et al. “Meshcnn: a network with an edge”. In: *ACM Transactions on Graphics (ToG)* 38.4 (2019), pp. 1–12.

[33] Charles R Qi et al. “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.

[34] Jeffrey Otey et al. “Revisiting the design intent concept in the context of mechanical CAD education”. In: *Computer-aided design and applications* 15.1 (2018), pp. 47–60. ISSN: 1686-4360. DOI: 10.1080/16864360.2017.1353733. URL: http://dx.doi.org/10.1080/16864360.2017.1353733.

[35] Basilio Ramos Barbero, Carlos Melgosa Pedrosa, and Raúl Zamora Samperio. “Learning CAD at university through summaries of the rules of design intent”. In: *International Journal of Technology and Design Education* 27 (2017), pp. 481–498.

[36] Xiang Xu et al. “SkexGen: Autoregressive Generation of CAD Construction Sequences with Disentangled Codebooks”. In: *International Conference on Machine Learning*. PMLR. 2022, pp. 24698–24724.

[37] Elona Dupont et al. “Transcad: A hierarchical transformer for cad sequence inference from point clouds”. In: *European Conference on Computer Vision*. Springer. 2024, pp. 19–36.

[38] Elona Dupont et al. “Cadops-net: Jointly learning cad operation types and steps from boundary-representations”. In: *2022 International Conference on 3D Vision (3DV)*. IEEE. 2022, pp. 114–123.

[39] Mohammad Sadil Khan et al. “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.

[40] Elona Dupont et al. “abraCADabra:Inference Geometry Guided Search for Auto and Non Auto-Regressive Scan-to-CAD Networks”. In: *In preparation for submission to IEEE Transactions on Pattern Analysis and Machine Intelligence* (2025).

[41] Kseniya Cherenkova, Djamila Aouada, and Gleb Gusev. “Pvdeconv: Point-Voxel Deconvolution for Autoencoding CAD Construction in 3D”. In: *2020 IEEE International Conference on Image Processing (ICIP)*. 2020, pp. 2741–2745. DOI: 10.1109/ICIP40778.2020.9191095.

[42] Danila Rukhovich et al. “CAD-Recode: Reverse Engineering CAD Code from Point Clouds”. In: *IEEE/CVF International Conference on Computer Vision (ICCV)*. 2025.

[43] Polly Brown. “CAD: Do computers aid the design process after all?” In: *Intersect: The Stanford Journal of Science, Technology, and Society* 2.1 (2009), pp. 52–66.

[44] Jami J Shah. “Designing with parametric cad: Classification and comparison of construction techniques”. In: *International workshop on geometric modelling*. Springer. 1998, pp. 53–68.

[45] Jon Peddie. *The history of visual magic in computers: How beautiful images are made in CAD, 3D, VR and AR*. Springer, 2013.

[46] M Pourazady and X Xu. “Direct manipulations of B-spline and NURBS curves”. In: *Advances in Engineering Software* 31.2 (2000), pp. 107–118.

[47] David F Rogers. *An introduction to NURBS: with historical perspective*. Elsevier, 2000.

[48] Foley James D and Dam A Van. *Computer graphics principles & practice second edition in c*. 1996.

[49] Marshall W Bern and Paul E Plassmann. “Mesh generation”. In: *Handbook of computational geometry* 38 (2000).

[50] Morteza Daneshmand et al. “3d scanning: A comprehensive survey”. In: *arXiv preprint arXiv:1801.08863* (2018).

[51] Ian Stroud. *Boundary representation modelling techniques*. Springer Science & Business Media, 2006.

[52] Ari Seff et al. “Vitruvion: A Generative Model of Parametric CAD Sketches”. In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=Ow1C7s3UcY.

[53] Charles Ruizhongtai Qi et al. “Pointnet++: Deep hierarchical feature learning on point sets in a metric space”. In: *Advances in neural information processing systems* 30 (2017).

[54] Qingyong Hu et al. “Learning semantic segmentation of large-scale point clouds with random sampling”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.11 (2021), pp. 8338–8354.

[55] Bahareh Shakibajahromi et al. “HyNet: 3D segmentation using hybrid graph networks”. In: *2021 International Conference on 3D Vision (3DV)*. IEEE. 2021, pp. 805–814.

[56] Bahareh Shakibajahromi, Edward Kim, and David E Breen. “Rimeshgnn: A rotation-invariant graph neural network for mesh classification”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2024, pp. 3150–3160.

[57] Yongjian Deng et al. “A voxel graph cnn for object classification with event cameras”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 1172–1181.

[58] Wonjoon Lee et al. “MSV-RGNN: Multiscale Voxel Graph Neural Network for 3D Object Detection”. In: *2023 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2023, pp. 3449–3453.

[59] Yingxue Zhang and Michael Rabbat. “A graph-cnn for 3d point cloud classification”. In: *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE. 2018, pp. 6279–6283.

[60] Yue Wang et al. “Dynamic Graph CNN for Learning on Point Clouds”. In: *ACM Trans. Graph.* 38.5 (Oct. 2019), pp. 1–12. ISSN: 0730-0301. DOI: 10.1145/3326362. URL: https://doi.org/10.1145/3326362.

[61] Weijing Shi and Raj Rajkumar. “Point-gnn: Graph neural network for 3d object detection in a point cloud”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 1711–1719.

[62] Jintai Chen et al. “A hierarchical graph network for 3d object detection on point clouds”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 392–401.

[63] Justin Gilmer et al. “Neural message passing for quantum chemistry”. In: *International conference on machine learning*. PMLR. 2017, pp. 1263–1272.

[64] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[65] Mary Phuong and Marcus Hutter. “Formal algorithms for transformers”. In: *arXiv preprint arXiv:2207.09238* (2022).

[66] Michael Tovey. “Drawing and CAD in industrial design”. In: *Design Studies* 10.1 (1989), pp. 24–39.

[67] Fatmir Azemi, Xhemajl Mehmeti, and Bekim Maloku. “The Importance of CAD/CAE systems in development of Product Design and Process of Optimization”. In: *2018 UBT International Conference*. Pristina, Kosovo: University for Business and Technology, 2018.

[68] E Solaberrieta et al. “Computer-aided dental prostheses construction using reverse engineering”. In: *Computer methods in biomechanics and biomedical engineering* 17.12 (2014), pp. 1335–1346.

[69] William B Thompson et al. “Feature-based reverse engineering of mechanical parts”. In: *IEEE Transactions on robotics and automation* 15.1 (1999), pp. 57–66.

[70] Tao Du et al. “Inversecsg: Automatic conversion of 3D models to CSG trees”. In: *ACM Transactions on Graphics (TOG)* 37.6 (2018), pp. 1–16.

[71] Byungchul Kim and Soonhung Han. “Integration of history-based parametric translators using the automation APIs”. In: *International Journal of Product Lifecycle Management* 2.1 (2007), pp. 18–29.

[72] Pradeep Kumar Jayaraman et al. “UV-Net: Learning from Boundary Representations”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021. arXiv: 2006.10211 [cs.CV].

[73] Gopal Sharma et al. “Csgnet: Neural shape parser for constructive solid geometry”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 5515–5523.

[74] Xianghao Xu et al. “Inferring cad modeling sequences using zone graphs”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 6062–6070.

[75] Murilo Antonio Salomgo Garcia, Nelson Vogel, and Marcos de Sales Guerra Tsuzuki. “New Algorithm for Converting a CSG Representation Into a B-Rep Representation”. In: *Proceedings of the COBEM 2005: 18 th International Congress of Mechanical Engineering*. 2005.

[76] Zhibo Zhang, Prakhar Jaiswal, and Rahul Rai. “FeatureNet: Machining feature recognition based on 3D Convolution Neural Network”. In: *Computer-Aided Design* 101 (2018), pp. 12–22. ISSN: 0010-4485. DOI: https://doi.org/10.1016/j.cad.2018.03.006. URL: https://www.sciencedirect.com/science/article/pii/S0010448518301349.

[77] Andrew R Colligan et al. “Hierarchical CADNet: Learning from B-Reps for Machining Feature Recognition”. In: *Computer-Aided Design* 147 (2022), p. 103226.

[78] Eman Ahmed et al. “A survey on deep learning advances on different 3D data representations”. In: *arXiv preprint arXiv:1808.01462* (2018).

[79] Wenxuan Wu, Zhongang Qi, and Li Fuxin. “PointConv: Deep Convolutional Networks on 3D Point Clouds”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[80] Yongcheng Liu et al. “DensePoint: Learning Densely Contextual Representation for Efficient Point Cloud Processing”. In: *International Conference on Computer Vision (ICCV)*. 2019.

[81] Daniel Maturana and Sebastian Scherer. “Voxnet: A 3d convolutional neural network for real-time object recognition”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, pp. 922–928.

[82] Zhijian Liu et al. “Point-voxel cnn for efficient 3d deep learning”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[83] Sk Aziz Ali et al. “RPSRNet: End-to-End Trainable Rigid Point Set Registration Network using Barnes-Hut 2D-Tree Representation”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021.

[84] Peng-Shuai Wang et al. “O-CNN: Octree-Based Convolutional Neural Networks for 3D Shape Analysis”. In: *ACM Trans. Graph.* 36.4 (July 2017). ISSN: 0730-0301. DOI: 10.1145/3072959.3073608. URL: https://doi.org/10.1145/3072959.3073608.

[85] Rana Hanocka et al. “MeshCNN: A Network with an Edge”. In: *ACM Transactions on Graphics* (2019).

[86] Shunwang Gong et al. “Spiralnet++: A fast and highly efficient mesh convolution operator”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*. 2019.

[87] Jonathan Masci et al. “Geodesic Convolutional Neural Networks on Riemannian Manifolds”. In: *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*. 2015.

[88] Aristides AG Requicha and Jarek R Rossignac. “Solid modeling and beyond”. In: *IEEE computer graphics and applications* 12.5 (1992), pp. 31–44.

[89] Qiaoyun Wu, Kai Xu, and Jun Wang. “Constructing 3D CSG models from 3D raw point clouds”. In: *Computer Graphics Forum*. Vol. 37-5. Wiley Online Library. 2018, pp. 221–232.

[90] Vadim Shapiro and Donald L. Vossler. “Construction and optimization of CSG representations”. In: *Computer-Aided Design* 23.1 (1991), pp. 4–20.

[91] Joachim Schöberl. “NETGEN An advancing front 2D/3D-mesh generator based on abstract rules”. In: *Computing and Visualization in Science*. 1997.

[92] Pradeep Kumar Jayaraman et al. “SolidGen: An Autoregressive Model for Direct B-rep Synthesis”. In: *Transactions on Machine Learning Research* (2023). Featured Certification. ISSN: 2835-8856. URL: https://openreview.net/forum?id=ZR2CDgADRo.

[93] Wamiq Para et al. “Sketchgen: Generating constrained cad sketches”. In: *Advances in Neural Information Processing Systems* 34 (2021).

[94] Ari Seff et al. “SketchGraphs: A Large-Scale Dataset for Modeling Relational Geometry in Computer-Aided Design”. In: *ICML 2020 Workshop on Object-Oriented Learning*. 2020.

[95] Yaroslav Ganin et al. “Computer-aided design as language”. In: *Advances in Neural Information Processing Systems* 34 (2021).

[96] Lingxiao Li et al. “Supervised fitting of geometric primitives to 3D point clouds”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 2652–2660. ISBN: 9781728132938.

[97] Harold W Kuhn. “The Hungarian method for the assignment problem”. In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97.

[98] Philipp Krähenbühl and Vladlen Koltun. “Parameter Learning and Convergent Inference for Dense Random Fields”. In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML’13. Atlanta, GA, USA: JMLR.org, 2013, III–513–III–521.

[99] Solidworks. *3D CAD Design Software*. `https://www.solidworks.com/`. Online: accessed 02-June-2022.

[100] Xiang Xu et al. “Hierarchical neural coding for controllable CAD model generation”. In: *Proceedings of the 40th International Conference on Machine Learning*. ICML’23. Honolulu, Hawaii, USA: JMLR.org, 2023.

[101] Kacper Kania, Maciej Zieba, and Tomasz Kajdanowicz. “UCSG-NET-unsupervised discovering of constructive solid geometry tree”. In: *Advances in neural information processing systems* 33 (2020), pp. 8776–8786.

[102] Weijian Ma et al. “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*. CIKM ’23. Birmingham, United Kingdom: Association for Computing Machinery, 2023, pp. 1766–1776. ISBN: 9798400701245.

[103] Joseph J Lim, Hamed Pirsiavash, and Antonio Torralba. “Parsing ikea objects: Fine pose estimation”. In: *Proceedings of the IEEE international conference on computer vision*. 2013, pp. 2992–2999.

[104] Ian Goodfellow et al. “Generative adversarial networks”. In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[105] Tero Karras et al. “Analyzing and improving the image quality of stylegan”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 8110–8119.

[106] Yingshan Chang et al. “WebQA: Multihop and Multimodal QA”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022, pp. 16495–16504.

[107] Panos Achlioptas et al. “Learning representations and generative models for 3d point clouds”. In: *International conference on machine learning*. PMLR. 2018, pp. 40–49.

[108] Guandao Yang et al. “Pointflow: 3d point cloud generation with continuous normalizing flows”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 4541–4550.

[109] Ruojin Cai et al. “Learning gradient fields for shape generation”. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*. Springer. 2020, pp. 364–381.

[110] Nanyang Wang et al. “Pixel2mesh: Generating 3d mesh models from single rgb images”. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 52–67.

[111] Charlie Nash et al. “Polygen: An autoregressive generative model of 3d meshes”. In: *International conference on machine learning*. PMLR. 2020, pp. 7220–7229.

[112] Jiajun Wu et al. “Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling”. In: *Advances in neural information processing systems* 29 (2016).

[113] Jun Li et al. “Grass: Generative recursive autoencoders for shape structures”. In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), pp. 1–14.

[114] Zhiqin Chen and Hao 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.

[115] Rundi Wu et al. “Pq-net: A generative part seq2seq network for 3d shapes”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 829–838.

[116] Xiang Xu et al. “Brepgen: A b-rep generative diffusion model with structured latent geometry”. In: *ACM Transactions on Graphics (TOG)* 43.4 (2024), pp. 1–14.

[117] Kseniya Cherenkova et al. “Sepicnet: Sharp edges recovery by parametric inference of curves in 3d shapes”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 2727–2735.

[118] Dimitrios Mallis et al. “SHARP Challenge 2023: Solving CAD History and pArameters Recovery from Point clouds and 3D scans. Overview, Datasets, Metrics, and Baselines.” In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 1786–1795.

[119] Markus Friedrich et al. “Optimizing evolutionary csg tree extraction”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2019, pp. 1183–1191.

[120] Fenggen Yu et al. “D2CSG: Unsupervised Learning of Compact CSG Trees with Dual Complements and Dropouts”. In: *Advances in Neural Information Processing Systems* 36 (2024), pp. 22807–22819.

[121] Joseph George Lambourne et al. “Reconstructing editable prismatic cad from rounded voxel models”. In: *SIGGRAPH Asia 2022 Conference Papers*. 2022, pp. 1–9.

[122] Weijian Ma et al. “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. 27154–27163.

[123] Open3D Development Team. *Open3D Documentation: open3d.geometry.estimate_normals*. Accessed: March 6, 2024. 2022. URL: https://www.open3d.org/docs/0.7.0/python%5C_api/open3d.geometry.estimate%5C_normals.html.

[124] Alexandre Carlier et al. “Deepsvg: A hierarchical generative network for vector graphics animation”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 16351–16361.

[125] Thomas Paviot. *PythonOCC*. Version 7.7.0. Accessed: March 7, 2024. Dec. 2017. DOI: 10.5281/zenodo.7471333. URL: https://dev.opencascade.org/project/pythonocc.

[126] Erik Wijmans. *Pointnet++ Pytorch*. Accessed: March 7, 2024. 2018. URL: https://github.com/erikwijmans/Pointnet2_PyTorch.

[127] Ken Perlin. "An image synthesizer". In: *ACM Siggraph Computer Graphics* 19.3 (1985), pp. 287–296.

[128] Wego Wang. *Reverse engineering: Technology of reinvention*. Crc Press, 2010.

[129] Tarek M Sobh et al. "Industrial inspection and reverse engineering". In: *Proceedings of 1994 IEEE 2nd CAD-Based Vision Workshop*. IEEE. 1994, pp. 228–235.

[130] David W Eggert, Andrew W Fitzgibbon, and Robert B Fisher. "Simultaneous registration of multiple range views for use in reverse engineering of CAD models". In: *Computer Vision and Image Understanding* 69.3 (1998), pp. 253–272.

[131] Alexander Agathos et al. "3D mesh segmentation methodologies for CAD applications". In: *Computer-Aided Design and Applications* 4.6 (2007), pp. 827–841.

[132] Scott R Granter, Andrew H Beck, and David J Papke Jr. "AlphaGo, deep learning, and the future of the human microscopist". In: *Archives of pathology & laboratory medicine* 141.5 (2017), pp. 619–621.

[133] OpenAI. *OpenAI o1 System Card*. Available at: https://cdn.openai.com/o1-system-card.pdf.

[134] Chao Zhang et al. "eCAD-Net: Editable Parametric CAD Models Reconstruction from Dumb B-Rep Models Using Deep Neural Networks". In: *Computer-Aided Design* 178 (2025), p. 103806. ISSN: 0010-4485. DOI: https://doi.org/10.1016/j.cad.2024.103806. URL: https://www.sciencedirect.com/science/article/pii/S0010448524001337.

[135] Shuming Zhang et al. “Brep2Seq: a dataset and hierarchical deep learning network for reconstruction and generation of computer-aided design models”. In: *Journal of Computational Design and Engineering* 11.1 (2024), pp. 110–134.

[136] Pu Li et al. “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.

[137] Fenggen Yu et al. “D2CSG: Unsupervised learning of compact CSG trees with dual complements and dropouts”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 22807–22819.

[138] Shengdi Zhou et al. “CADGen: Computer-aided design sequence construction with a guided codebook learning”. In: *Digital Twins and Applications* 1.1 (2024), pp. 75–87.

[139] Karl DD Willis et al. “Joinable: Learning bottom-up assembly of parametric cad joints”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 15849–15860.

[140] Ari Holtzman et al. “The Curious Case of Neural Text Degeneration”. In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=rygGQyrFvH.

[141] *OnShape*. Available at: http://onshape.com.

[142] *3D Content Central*. Available at: https://www.3dcontentcentral.com.

[143] *Artec3D*. Available at: https://www.artec3d.com.

[144] Antoine Brière-Côté, Louis Rivest, and Roland Maranzana. “Comparing 3D CAD models: uses, methods, tools and perspectives”. In: *Computer-Aided Design and Applications* 9.6 (2012), pp. 771–794.

[145] Fei Liu. “Fast industrial product design method and its application based on 3D CAD system”. In: *Computer-Aided Design and Applications* 18.S3 (2020), pp. 118–128.

[146] Ming Li, Frank C Langbein, and Ralph R Martin. “Detecting design intent in approximate CAD models using symmetry”. In: *Computer-Aided Design* 42.3 (2010), pp. 183–201.

[147] Jorge D Camba, Manuel Contero, and Pedro Company. “Parametric CAD modeling: An analysis of strategies for design reusability”. In: *Computer-Aided Design* 74 (2016), pp. 18–31.

[148] An Yang et al. *Qwen2 Technical Report*. 2024. arXiv: 2407.10671 [cs.CL]. URL: https://arxiv.org/abs/2407.10671.

[149] OpenAI. *GPT-4 Technical Report*. 2024. arXiv: 2303.08774 [cs.CL]. URL: https://arxiv.org/abs/2303.08774.

[150] CADQuery Developers. *CADQuery: A Python parametric CAD scripting framework*. https://cadquery.readthedocs.io/. Accessed: 2024-10-22. 2024.

[151] Runsen Xu et al. “Pointllm: Empowering large language models to understand point clouds”. In: *European Conference on Computer Vision*. Springer. 2024, pp. 131–147.

[152] Renrui Zhang et al. “Pointclip: Point cloud understanding by clip”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 8552–8562.

[153] Zibo Zhao et al. “Michelangelo: Conditional 3D Shape Generation based on Shape-Image-Text Aligned Latent Representation”. In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023. URL: https://openreview.net/forum?id=xmxgMij3LY.

[154] Fukun Yin et al. “Shapegpt: 3d shape generation with a unified multi-modal language model”. In: *IEEE Transactions on Multimedia* (2025).

[155] Jiaming Han et al. “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.

[156] Yining Hong et al. “3D-LLM: Injecting the 3D World into Large Language Models”. In: *NeurIPS* (2023).

[157] Yining Hong et al. “3d-llm: Injecting the 3d world into large language models”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 20482–20494.

[158] Sijin Chen et al. “LL3DA: Visual Interactive Instruction Tuning for Omni-3D Understanding Reasoning and Planning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 26428–26438.

[159] Xiangyang Zhu et al. “PointCLIP V2: Prompting CLIP and GPT for Powerful 3D Open-world Learning”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2023, pp. 2639–2650.

[160] Le Xue et al. “Ulip: Learning a unified representation of language, images, and point clouds for 3d understanding”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2023, pp. 1179–1189.

[161] Minghua Liu et al. “Openshape: Scaling up 3d shape representation towards open-world understanding”. In: *Advances in neural information processing systems* 36 (2024).

[162] Le Xue et al. “ULIP-2: Towards Scalable Multimodal Pre-training for 3D Understanding”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2024, pp. 27091–27101.

[163] Milin Kodnongbua et al. “ReparamCAD: Zero-shot CAD Re-Parameterization for Interactive Manipulation”. In: *SIGGRAPH Asia 2023 Conference Papers*. SA ’23. Sydney, NSW, Australia: Association for Computing Machinery, 2023. ISBN: 9798400703157. DOI: 10.1145/3610548.3618219. URL: https://doi.org/10.1145/3610548.3618219.

[164] Haocheng Yuan et al. “CADTalk: An Algorithm and Benchmark for Semantic Commenting of CAD Programs”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 3753–3762.

[165] Akshay Badagabettu, Sai Sravan Yarlagadda, and Amir Barati Farimani. “Query2CAD: Generating CAD models using natural language queries”. In: *arXiv preprint arXiv:2406.00144* (2024).

[166] Kamel Alrashedy et al. “Generating CAD Code with Vision-Language Models for 3D Designs”. In: *ICLR*. 2025. URL: https://openreview.net/forum?id=BLWaTeucYX.

[167] Zeju Qiu et al. “Can Large Language Models Understand Symbolic Graphics Programs?” In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=Yk87CwhBDx.

[168] Yang You et al. “Img2CAD: Reverse Engineering 3D CAD Models from Images through VLM-Assisted Conditional Factorization”. In: *arXiv preprint arXiv:2408.01437* (2024).

[169] V Sanh. “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”. In: *arXiv preprint arXiv:1910.01108* (2019).

[170] Mark Chen et al. *Evaluating Large Language Models Trained on Code*. 2021. arXiv: 2107.03374 [cs.LG].

[171] Qwen Team. *Qwen2-1.5B*. https://huggingface.co/Qwen/Qwen2-1.5B. Accessed: Nov. 2024. 2024.

Appendices

A CADOps-Net Additional Results

A.1 CAD Operation Types Qualitative Results

Figure A.1 shows sample results from *CADOps-Net* for the *op.type* segmentation task on both the Fusion360 and *CC3D-Ops* datasets. As discussed in Section 3.6.2 of the thesis, the *op.type* accuracy is not correlated to the complexity of the models. In particular, this can be observed from the results presented on the Fusion360 dataset in which *CADOps-Net* sometimes fails to predict the correct *op.types* for some simple models. Despite not matching the ground truth, some predictions are still valid within the context of CAD modelling, as outlined in Section 3.6.5.

A.2 CAD Operation Step Qualitative Results

Figure A.2 displays qualitative results for the *op.step* segmentation task on the *CC3D-Ops* and Fusion360 datasets. These qualitative results illustrate that while *CADOps-Net* is able to make accurate predictions for models with a small number of *op.steps*, the *op.step* accuracy decreases as the number of *op.steps* increases, as explained in Section 3.6.2.

A.3 Sketch Recovery Results

In CAD modeling, sketches are considered as starting points to build a kernel structure (in our case B-Rep) of the desired solid model. Nevertheless, the sketches are only part of the

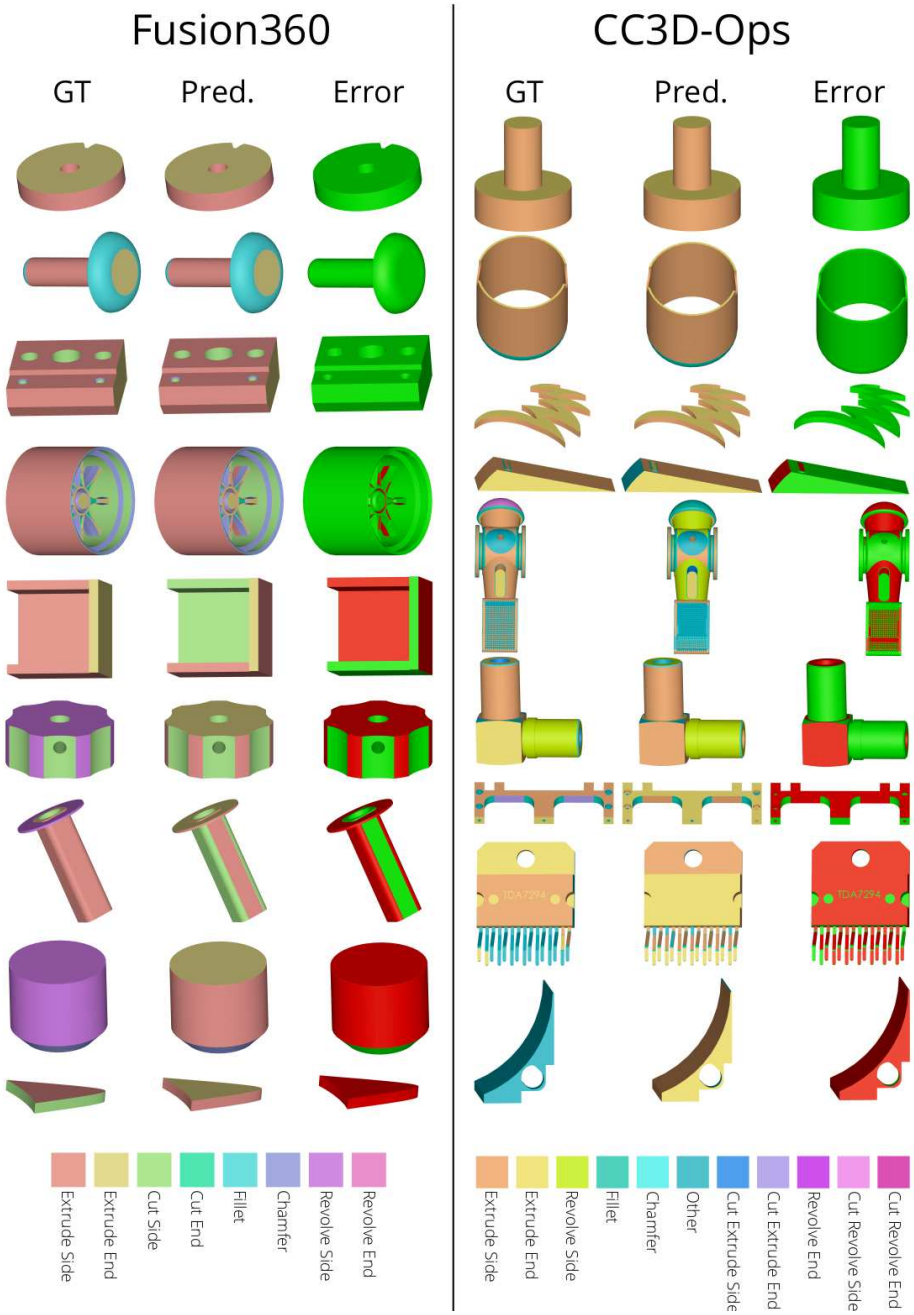


Figure A.1: CAD operation step qualitative results on the Fusion360 (left) and *CC3D-Ops* (right) datasets. For the *CADOps-Net* ground truth (GT) and predictions (Pred.). Correctly segmented faces are shown in green and incorrect in red in the Error columns.

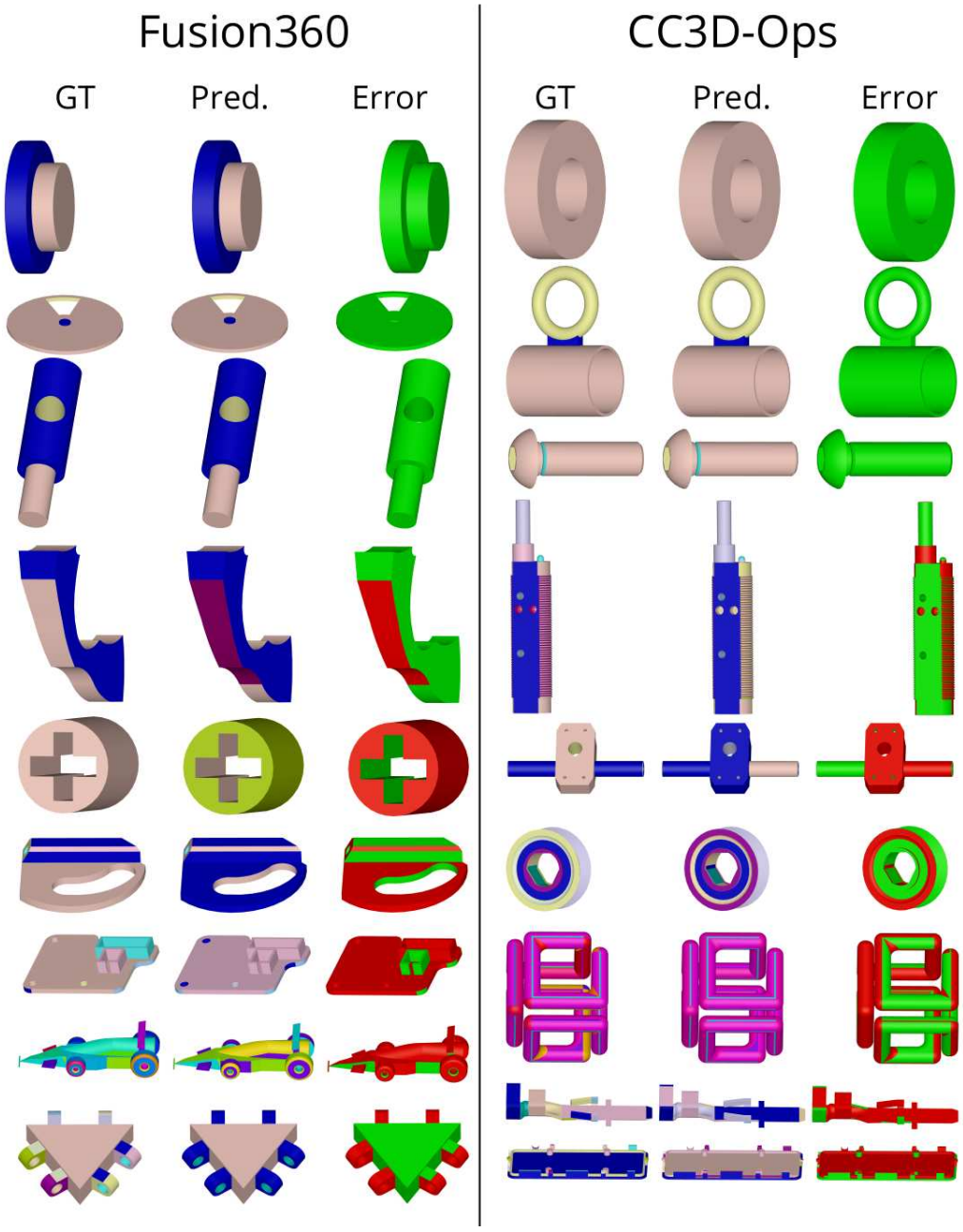


Figure A.2: CAD operation step qualitative results on the Fusion360 (left) and *CC3D-Ops* (right) datasets.For the *CADOps-Net* ground truth (GT) and predictions (Pred.), each color represents a CAD operation step. Correctly segmented faces are shown in green and incorrect in red in the Error columns.

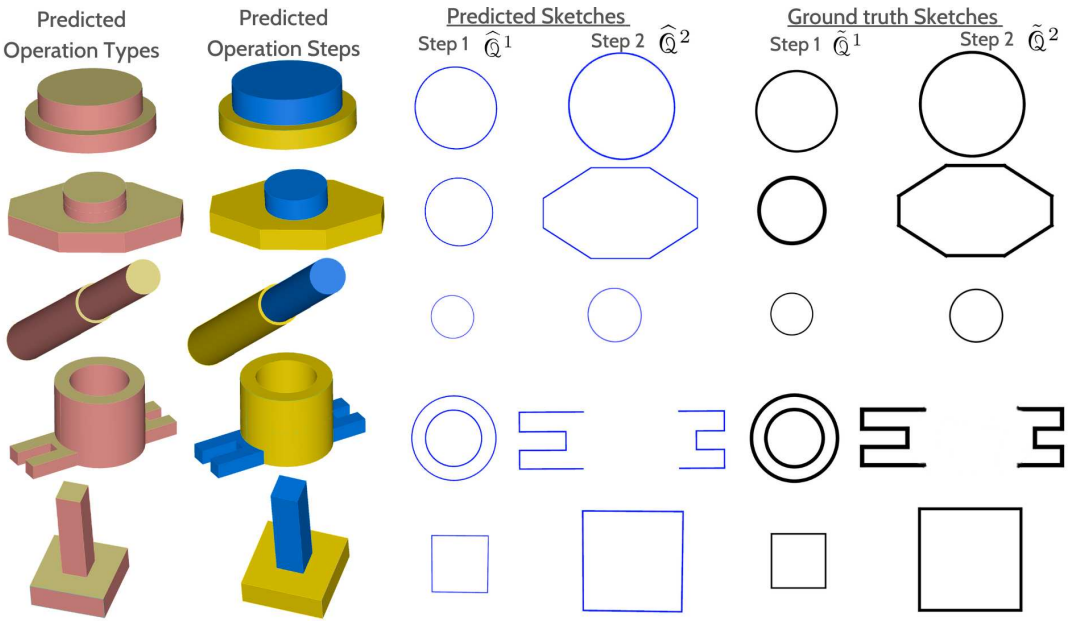


Figure A.3: Qualitative results on sketch recovery from correctly predicted *op.types* and *op.steps* by CADOps-Net. The models shown above include exactly two operation steps.

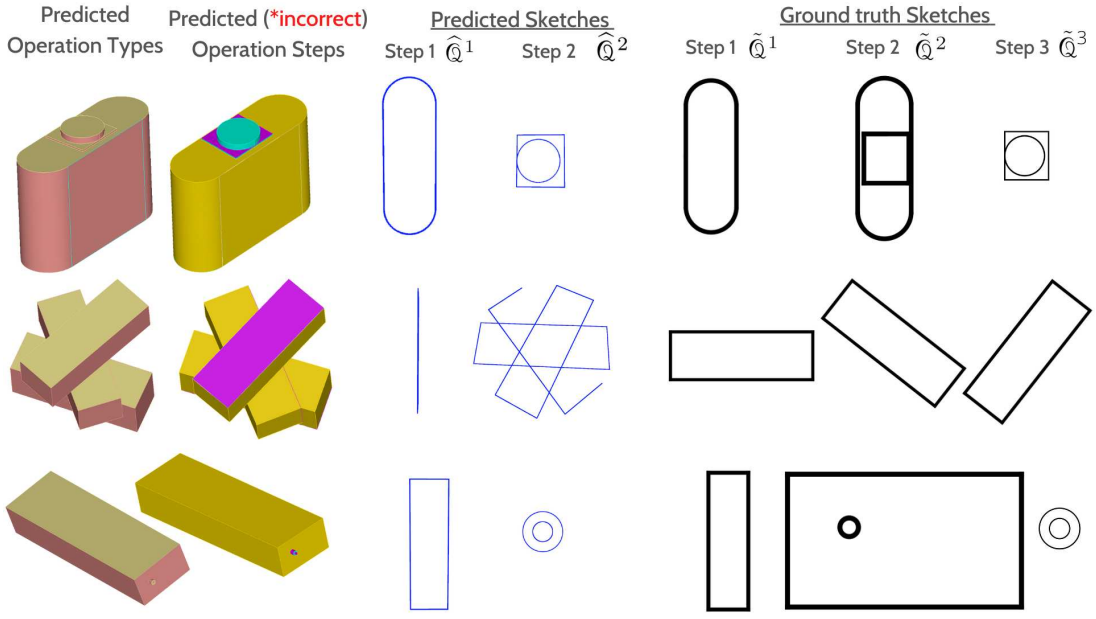


Figure A.4: Qualitative results on sketch recovery from incorrectly predicted *op.types* and *op.steps* by CADOps-Net. The models shown above include exactly three operation steps.

forward design process, and tracing them back from the final B-Rep is not straightforward. In this section, we present more qualitative results.

Figure A.3 and A.4 illustrate the qualitative results of sketches recovered using *CADOps-Net* predictions on randomly selected samples made by extrusions from Fusion360 [27] dataset. We dissect the sketch results based on correctly and incorrectly predicted *op.steps* in the two figures. We can observe that the sketch recovery is successful when the predictions of *op.steps* are correct (Figure A.3). In Figure A.4, the B-Reps were segmented into two *op.steps*, while the ground truth annotations indicated that they were designed through three *op.steps*. Such incorrect *op.step* prediction impacted the sketch recovery and resulted in erroneous sketches.

B TransCAD Additional Results

B.1 Qualitative Results

In this section, further qualitative results are presented by first comparing *TransCAD* to the retrieval baseline and then to DeepCAD [26].

Retrieval Baseline Comparison: Qualitative results comparing *TransCAD* to the the retrieval baseline can be found in Figure B.1. It can be observed that *TransCAD* can perform well on duplicate models and more importantly that it performs better than the retrieval baseline on non-duplicate models. *TransCAD* can identify most of the components of an unseen CAD model but sometimes fails to place those parts in the right location as suggested by the APCS scores presented in the previous section.

DeepCAD Comparison: Qualitative results showing the comparison between *TransCAD* and DeepCAD [26] can be found in Figure B.2. The results on *simple* and *difficult* models show that *TransCAD* is able to outperform DeepCAD [26] on most occasions.

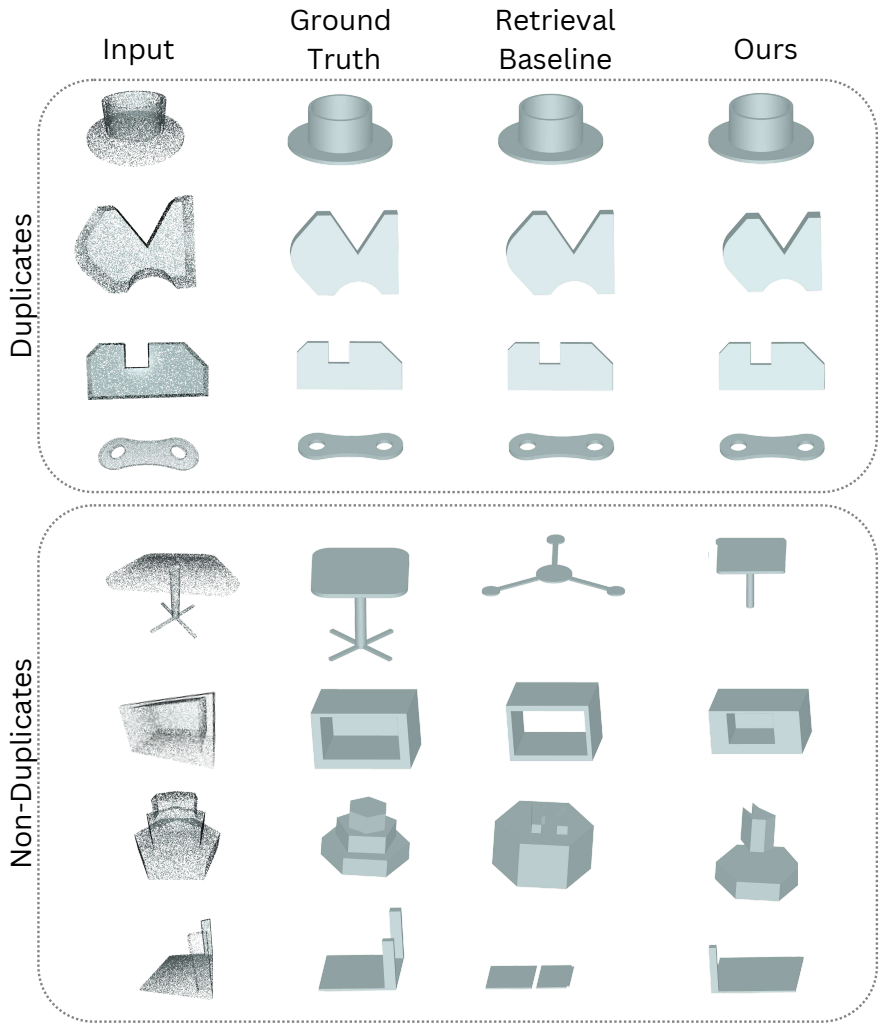


Figure B.1: Qualitative results showing the performance of *TransCAD* against the retrieval baseline on both duplicate CAD models (top panel) and non-duplicate models (bottom panel).

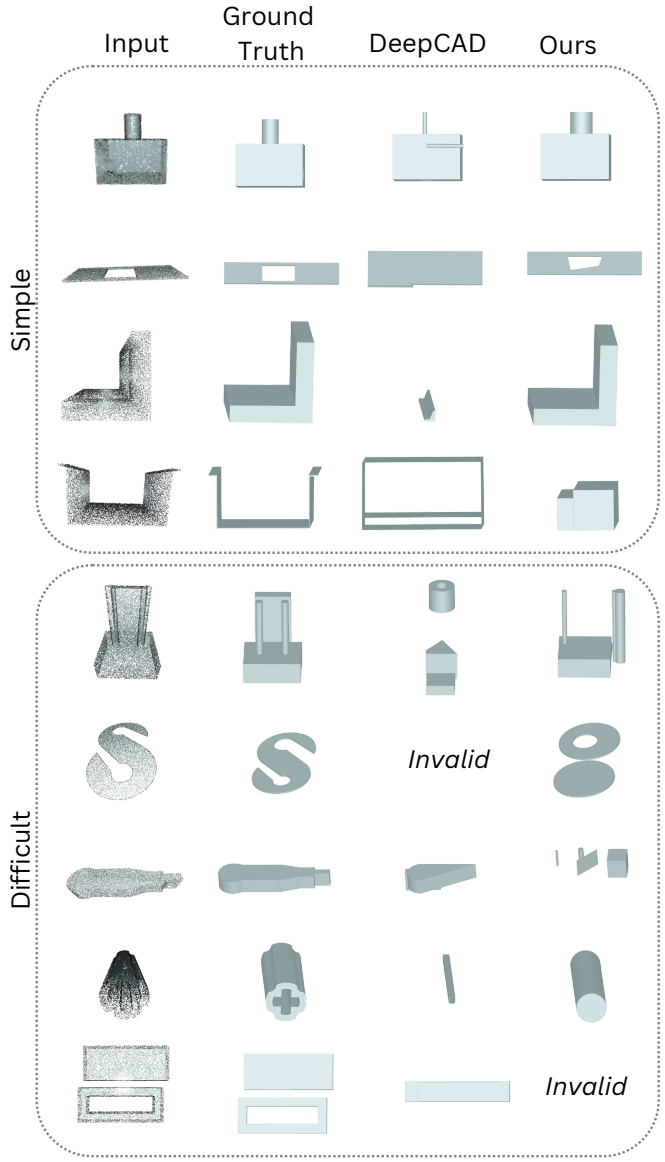


Figure B.2: Qualitative results showing the performance of *TransCAD* against the DeepCAD [26] on both simpler CAD models (top panel) and more complex models (bottom panel).

C CAD-Recode Data Generation and Additional Results

C.1 Training Dataset Generation Algorithm

The procedurally generated training dataset is presented. The main advantage of generating data over using the existing DeepCAD dataset for training is that the algorithm allows full control over the amount of data as well as the features and design patterns that the network is exposed to during training. We generate one million valid Python CadQuery code snippets, through an automated pipeline leveraging PythonOCC [125] and CadQuery [150]. The generation process consists of two primary components: (1) a sketch profile generator (Algorithm 1) that creates valid 2D sketches, and (2) a CAD model generator (Algorithm 2) that produces 3D CAD models from these sketches.

The sketch generation process combines primitive shapes (circles and rectangles) through boolean operations (union and cut). From each generated sketch, we extract the primitives (lines, arcs, and circles) from both inner and outer loops. The validity of the generated sketch is ensured through multiple verification steps, including verifying that loops do not intersect, and each primitive has a length greater than zero. Finally, we ensure that the randomly generated CAD code has not previously been generated using the duplicate detection protocol outlined in [36]. This ensures that each sample in the dataset is unique.

The CAD model generation procedure extrudes the validated sketches and combines them through union operations. The planes on which the sketches lie are randomly generated by choosing one of the three canonical planes translated by a random amount. Each resulting model undergoes normalization to fit within a unit bounding box centered at the origin. The parameters are quantized so that the coordinates of any point on the CAD surface are within the range -100 to 100 with a minimum resolution of 1 unit. We then simplify the sequence using higher level abstractions (rectangle, box, and cylinder) by considering the sequence parameters. Our validation framework verifies that a generated code w executes without errors (ϕ_{syn}). Furthermore, we check that the executed code produces a geometric valid CAD model (ϕ_{cad}) using the `BRepCheck_Analyzer` function from PythonOCC as in [26]. Invalid models are excluded from the dataset.

Algorithm 1 Generate2DSketch

function GENERATE2DSKETCH
    $numPrimitives \leftarrow RandInt(3, 8)$ ▷ Choose random number of shape primitives
    $compositeShape \leftarrow \emptyset$ ▷ Initialize empty shape
    for $i \leftarrow 1$ to $numPrimitives$ **do** ▷ Build shape by combining primitives
        $primitive \leftarrow$ random from {Circle, RotatedRectangle}
        $booleanOperation \leftarrow$ random from {Union, Cut} ▷ Union adds, Cut subtracts
        $compositeShape \leftarrow ApplyOperation(compositeShape, primitive, booleanOperation)$
    end for
    $boundaryLoops \leftarrow ExtractBoundaryLoops(compositeShape)$ ▷ Extract shape boundaries
    $boundaryComponents \leftarrow \emptyset$
    for $loop \in boundaryLoops$ **do** ▷ Process each boundary loop
        $(edgeSequence, isOuter) \leftarrow AnalyzeBoundary(loop)$ ▷ List of parametric curves (lines, arcs, circles)
        $boundaryComponents.Append((edgeSequence, isOuter))$
    end for
    $boundaryComponents \leftarrow ValidateShapeTopology(boundaryComponents)$ ▷ Ensure valid shape topology
    return $boundaryComponents$ ▷ Returns list of (edges, boolean) tuples
end function

Algorithm 2 GenerateCAD

function GENERATECAD
    $cadModel \leftarrow \emptyset$ ▷ Initialize empty CAD model
    $planes \leftarrow GenerateRandomPlanes()$ ▷ Create set of reference planes
    $sketches \leftarrow Generate2DSketch()$ ▷ Get sketches from Algorithm 1
    for $sketch \in sketches$ **do** ▷ Create 3D volumes from sketches
        $plane \leftarrow RandomSelect(planes)$ ▷ Select random reference plane
        $volume \leftarrow ExtrudeSketch(sketch, plane)$ ▷ Create 3D volume by extrusion
        $cadModel \leftarrow BooleanUnion(cadModel, volume)$ ▷ Add volume to model
    end for
    $cadModel \leftarrow NormalizeModel(cadModel)$ ▷ Ensure the model fits within a unit box
    $cadModel \leftarrow QuantizeParameters(cadModel)$ ▷ Discretize model parameters
    $cadModel \leftarrow SimplifyCADModel(cadModel)$ ▷ Identify high-level abstractions (rectangle, box, and cylinder)
    $cadModel \leftarrow ValidateCADModel(cadModel)$ ▷ Ensure validity of CadQuery code and CAD model geometry
    $cadModel \leftarrow CheckDuplicate(cadModel)$ ▷ Ensure that the sequence has not previously been generated.
    return $cadModel$
end function

```
import cadquery as cq
w0 = cq.Workplane('ZX', origin=(0, -13, 0))
r = w0.workplane(offset=-87 / 2).moveTo(52.5, 10.5).box(57, 83, 87)
  .union(w0.workplane(offset=23 / 2).moveTo(-29, 0).cylinder(23, 30))
  .union(w0.workplane(offset=113 / 2).moveTo(-29, 0).cylinder(113, 52))
```

```
import cadquery as cq
w0 = cq.Workplane('ZX', origin=(0, -30, 0))
r = w0.sketch().segment((-30, -27),(-18, -31)).segment((-19,
↪  -31)).segment((-19, -100))
    .segment((38, -100)).segment((38, -31)).segment((10, -31)).segment((13,
    ↪  -23))
    .arc((30, -13), (23, 5)).segment((33, 33)).segment((16, 39)).arc((-12,
    ↪  99),(-9, 33))
    .close().assemble().finalize().extrude(60)
```

```
import cadquery as cq
w0 = cq.Workplane('YZ', origin=(-14, 0, 0))
r = w0.workplane(offset=17 / 2).moveTo(4, -73.5).box(104, 53, 17)
  .union(w0.sketch().segment((-78, 23), (2, -55)).segment((40, -17))
    .arc((42, -24),(48, -30)).segment((48, 5)).segment((61, 5))
    .segment((78, 22)).segment((-2, 100)).close().assemble()
    .push([(0, 22)]).circle(50, mode='s').finalize().extrude(29))
```

```
import cadquery as cq
w0 = cq.Workplane('XY', origin=(0, 0, 42))
w1 = cq.Workplane('YZ', origin=(-17, 0, 0))
r = w0.sketch().arc((-12, 6), (34, -29),(-1, 16)).segment((5,
↪  4)).segment((-8, -2))
    .close().assemble().finalize().extrude(56)
  .union(w0.sketch().arc((-42, 54), (-12, 71), (19, 54)).segment((19, 78))
    .segment((-42, 78)).close().assemble().finalize().extrude(58))
  .union(w1.sketch().segment((-44, -100), (51, -100)).segment((51,
  ↪  5)).segment((27, 5))
    .arc((-58, 40),(-44, -51)).close().assemble().reset()
    .face(w1.sketch().arc((-54, -17), (-26, -34),(3, -17)).close()
    .assemble(), mode='s').reset().face(w1.sketch().segment((-54, 14), (3,
    ↪  14))
    .arc((-26, 31), (-54, 14)).assemble(),
    ↪  mode='s').finalize().extrude(-13))
```

```
import cadquery as cq
w0 = cq.Workplane('YZ', origin=(-22, 0, 0))
w1 = cq.Workplane('ZX', origin=(0, -19, 0))
r = w0.sketch().segment((-100, -83),(-67, -83)).segment((-80,
↪  -52)).segment((-75, -50))
    .segment((-75, 62)).segment((17, 62)).segment((17, -62)).segment((-40,
    ↪  -62))
    .segment((-37, -71)).segment((-65, -83)).segment((43, -83))
    .segment((43,83)).segment((-100,83)).close().assemble().finalize().extr
    ↪  ude(8)
  .union(w1.sketch().segment((-77, -53),(76, -53)).arc((76, -48),(77, -42))
  .segment((77, 53)).segment((-77, 53)).close().assemble()
  .push([(38.5, 2.5)]).rect(9, 57, mode='s').finalize().extrude(119))
```

Figure C.1: Examples from our procedurally generated training dataset. Each row contains CadQuery Python code and a corresponding CAD model. Examples contain not only basic *line*, *circle*, and *arc* primitives, but also higher-level abstractions such as *rect*, *box*, and *cylinder*.

Figure C.1 presents examples of CAD models alongside their corresponding CadQuery Python code from our procedurally generated dataset. It is worth noting that the generated codes are fairly compact, this was designed to facilitate training. All code examples are directly executable using a standard Python interpreter with the CadQuery library. The codes follow a consistent three-part structure: (1) necessary library import, (2) definition of sketch planes, and (3) sketch-extrude operations combined through union.

C.2 Further Experimental Results

Qualitative Results: Additional qualitative results for the reverse engineering of CAD models from point clouds are presented for DeepCAD (Figure C.2), Fusion360 (Figure C.3), and real-world CC3D (Figure C.4) datasets. `CAD-Recode` consistently generates shapes that closely approximate the input point cloud geometry, whereas CAD-SIGNet [39] can generate predictions that greatly differ from the input.

Code Outputs: Figure C.5 illustrates the predicted code sequences and their corresponding reconstructed shapes. The predicted codes have a syntax that is consistent with the procedurally generated training examples, showing that `CAD-Recode` successfully learns both the features and CAD design patterns established in the training set.

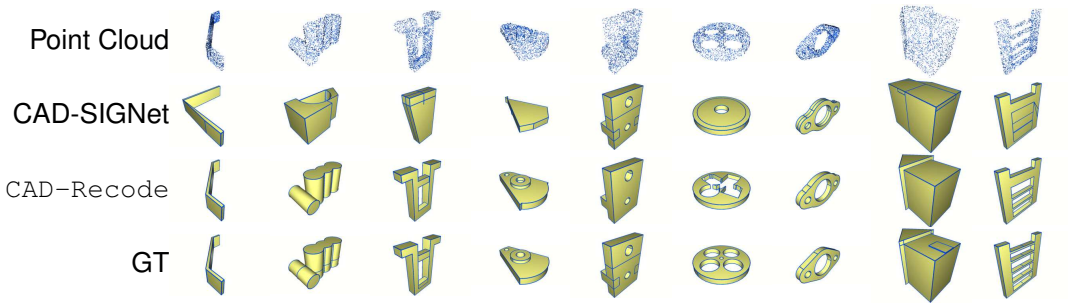


Figure C.2: Qualitative results on the DeepCAD dataset.

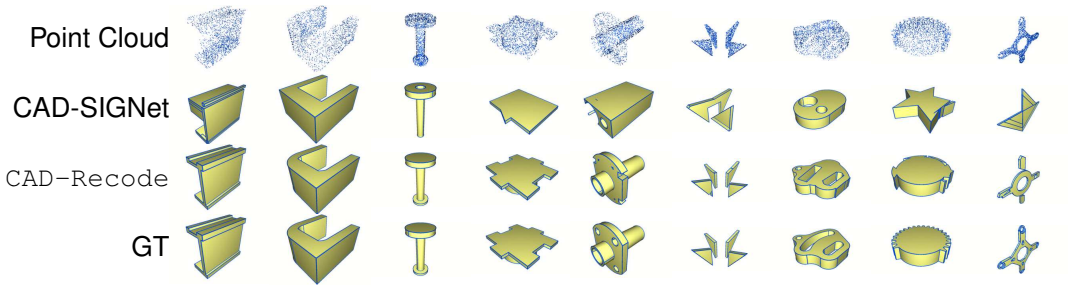


Figure C.3: Qualitative results on the Fusion360 dataset.

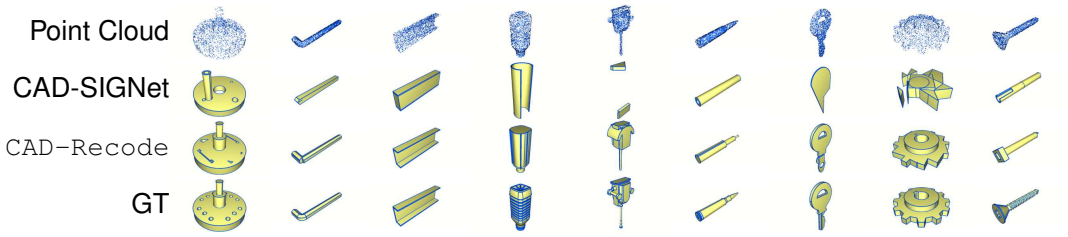


Figure C.4: Qualitative results on the real-world CC3D dataset.

```
import cadquery as cq
w0 = cq.Workplane('XY', origin=(0, 0, -17))
r = w0.sketch().segment((-48, -64), (24, -64)).segment((24,
↪  -43)).segment((-27, -43))
    .segment((-27, 60)).segment((27, 60)).segment((27,
    ↪  -39)).segment((48, -39))
    .segment((48, 100)).segment((-48,
    ↪  100)).close().assemble().finalize().extrude(20)
  .union(w0.sketch().segment((-82, -100),(-27, -100)).segment((-27,
  ↪  80)).segment((27, 80))
    .segment((27, -100)).segment((82, -100)).segment((82,
    ↪  -79)).segment((48, -79))
    .segment((48, 100)).segment((-48, 100)).segment((-48,
    ↪  -79)).segment((-82, -79))
    .close().assemble().finalize().extrude(34))
```

```
import cadquery as cq
w0 = cq.Workplane('ZX', origin=(0, 40, 0))
w1 = cq.Workplane('XY', origin=(0, 0, -19))
r = w0.sketch().arc((-24, -47), (41, -99),(87, -32)).segment((88,
↪  -32)).segment((88, 100))
    .segment((82, 100)).segment((82, -52)).arc((34, -94), (-18,
    ↪  -52)).segment((-18, 100))
    .segment((-24, 100)).close().assemble().finalize().extrude(-80)
  .union(w1.workplane(offset=-69 / 2).moveTo(52, 0).cylinder(69,
  ↪  32))
```

```
import cadquery as cq
w0 = cq.Workplane('ZX', origin=(0, 20, 0))
r = w0.sketch().circle(61).circle(25, mode='s').push([(34, 4)])
    .circle(4, mode='s').finalize().extrude(-41)
  .union(w0.sketch().segment((-100, 19), (-88, 11)).segment((-97,
  ↪  -34)).segment((-67, -41))
    .segment((-77, -66)).segment((-57, -74)).segment((-57,
    ↪  -72)).segment((-56, -72))
    .segment((-56, -75)).segment((-32, -80)).segment((-35,
    ↪  -95)).segment((-16, -100))
    .segment((-11, -83)).segment((33, -100)).segment((45,
    ↪  -70)).segment((68, -76))
    .segment((76, -61)).segment((66, -56)).segment((100,
    ↪  -30)).segment((88, -19))
    .segment((97, 34)).segment((67, 41)).segment((77,
    ↪  66)).segment((57, 74))
    .segment((51, 69)).segment((51, 70)).segment((32,
    ↪  76)).segment((35, 95))
    .segment((16, 100)).segment((11, 83)).segment((-33,
    ↪  100)).segment((-45, 70))
    .segment((-68, 77)).segment((-76, 62)).segment((-66,
    ↪  56)).close().assemble()
    .circle(26, mode='s').finalize().extrude(-20))
```

Figure C.5: `CAD-Recode` predictions on DeepCAD (top row), Fusion360 (middle row), and CC3D (bottom row) datasets. Each row contains predicted CadQuery Python code and its result after execution in Python interpreter.