algorithms

MDPI

Article

AI-Driven Optimization of Functional Feature Placement in Automotive CAD

Ardian Kelmendi [1] and George Pappas [2,*]

[1] Electrical and Computer Engineering Department, Lawrence Technological University, Southfield, MI 48075, USA; akelmendi@ltu.edu
[2] MSAI Grad Program, Lawrence Technological University, Southfield, MI 48075, USA
* Correspondence: gpappas@ltu.edu

Abstract

The automotive industry increasingly relies on 3D modeling technologies to design and manufacture vehicle components with high precision. One critical challenge is optimizing the placement of latches that secure the dashboard side paneling, as these placements vary between models and must maintain minimal tolerance for movement to ensure durability. While generative artificial intelligence (AI) has advanced rapidly in generating text, images, and video, its application to creating accurate 3D CAD models remains limited. This paper proposes a novel framework that integrates a PointNet deep learning model with Python-based CAD automation to predict optimal clip placements and surface thickness for dashboard side panels. Unlike prior studies that focus on general-purpose CAD generation, this work specifically targets automotive interior components and demonstrates a practical method for automating part design. The approach involves generating placement data—potentially via generative AI—and importing it into the CAD environment to produce fully parameterized 3D models. Experimental results show that the prototype achieved a 75% success rate across six of eight test surfaces, indicating strong potential despite the limited sample size. This research highlights a clear pathway for applying generative AI to part design automation in the automotive sector and offers a foundation for scaling to broader design applications.

Keywords: computer-aided design; artificial intelligence; python; algorithm

Academic Editor: Frank Werner

Received: 22 May 2025
Revised: 28 July 2025
Accepted: 14 August 2025
Published: 2 September 2025

Citation: Kelmendi, A.; Pappas, G. AI-Driven Optimization of Functional Feature Placement in Automotive CAD. *Algorithms* **2025**, *18*, 553. https://doi.org/10.3390/a18090553

Correction Statement: This article has been republished with a minor change. The change does not affect the scientific content of the article and further details are available within the backmatter of the website version of this article.

1. Introduction

Generative AI has demonstrated significant promise across various fields, enabling the creation of text, images, and videos from simple prompts. Computer-aided design (CAD) software such as Siemens NX ver. 2306 provides design engineers with the flexibility to develop complex parts in a 3D environment, complete with tools for lighting and texturing. However, a significant gap remains in automatically and intelligently placing discrete functional features on complex 3D surfaces based on physical requirements. While the long-term vision for this technology may be a general-purpose tool, the focus of this paper is specifically on solving this challenge within the demanding application scenario of automotive component design. To ground our research, we address a critical problem from our industrial partner, Marelli, an automotive manufacturer. In this context, the primary application proposed by Marelli was the development of a prototype to identify optimal clip placements that minimize flex and material thickness. These clips secure the side panel to the dashboard, and their placement is critical for effectively reducing flex. Since

determining the optimal locations is a time-consuming and costly process, leveraging deep learning with a PointNet model as a prototype offers valuable insights into scaling the approach to different AI architectures. Therefore, the primary objective of this research is to develop and evaluate a deep learning prototype capable of automating such feature placement tasks on complex automotive surfaces. Our innovative step is to treat this as a 3D spatial optimization problem where a PointNet deep learning model acts as a computationally efficient surrogate for traditional engineering analysis. Designing a part requires substantial time and resources, and identifying the most effective solution can be challenging. This paper will explore various techniques that researchers have applied and, based on the results, provide recommendations on how best to address this problem.

2. Related Work

Shortcomings of Existing Methods and Positioning of This Study

Despite these advancements, several critical shortcomings persist in existing generative AI methods for CAD, particularly when addressing the precise placement of functional features on complex surfaces under engineering constraints:

Limited B-rep Generation: Although BrepGen [1] is a promising step, the direct and robust generation of complex B-rep models with precise geometric and topological integrity from diverse inputs remains an active area of research. Many methods still rely on intermediate representations (e.g., voxels, meshes, point clouds) that require post-processing for conversion to manufacturing-ready CAD. Crucially, these methods often focus on creating entire models rather than optimally placing discrete features on pre-existing complex geometries.

Lack of Functional and Fabrication Constraint Integration: As highlighted by Faruqi et al. [2], many generative models prioritize aesthetic appeal over functional viability. Integrating engineering constraints, manufacturing processes, and material properties into the generative process is crucial for creating truly useful CAD models. For our specific problem, this translates to models that can predict optimal placements based on physical criteria like minimizing flex, which is often a downstream engineering analysis.

Scalability and Complexity: Generating highly complex assemblies or intricate parts with detailed features remains computationally intensive and often beyond the capabilities of current models without significant human intervention. Furthermore, applying generative methods to specific sub-problems, such as feature placement on arbitrary complex surfaces, requires efficient handling of large point cloud or mesh data from these surfaces.

Interpretability and Control: Many deep learning models operate as "black boxes", making it difficult for designers to understand why a specific design was generated or to precisely control its attributes beyond initial prompts. For critical automotive components, transparent and controllable feature placement is essential.

Data Scarcity: High-quality, diverse, and well-annotated datasets of CAD models, especially those with associated functional requirements or manufacturing data, are scarce, hindering the training of robust generative models for specialized tasks like optimal feature placement.

This study aims to address these limitations by focusing on the automated and intelligent placement of discrete functional features on complex 3D surfaces based on physical requirements. Specifically, we propose developing and evaluating a deep learning prototype for identifying optimal clip placements that minimize flex and material thickness on automotive side panels, a critical problem identified by our industrial partner, Marelli.

Our innovative step is to treat this as a 3D spatial optimization problem where a PointNet deep learning model acts as a computationally efficient surrogate for traditional engineering analysis. While existing work explores full CAD generation or material classification, our research specifically targets the challenge of intelligent feature placement on existing complex geometries, a bottleneck in automotive design. By leveraging a PointNet model, which is adept at processing unordered point cloud data directly from 3D surfaces, we aim to provide valuable insights into scaling this approach to different AI architectures and automating a time-consuming and costly engineering process. This allows us to provide a more robust and practical generative AI solution focused on optimizing discrete functional element placement, bridging the gap between conceptual design and fabrication-ready component refinement.

The motivation behind using point clouds for neural network research was inspired by Hua in the paper [3]. The goal of the Pointwise paper is to introduce a new convolution operator for point clouds to apply a Convolution Neural Network (CNN). CNN traditionally uses kernels for feature extraction on image data, but point clouds exist in 3D space, requiring a new approach to convolution. Hua et al. created a Pointwise convolution operator for computer vision for handling point clouds. The paper used Pointwise for scene understanding and compared other models with the Pointwise model, but the Pointwise model struggled to understand a large number of point clouds. The researchers noted in the conclusion that a robust solution for large-scale point clouds would be interesting future work [3]. The research shows the potential of point clouds in deep learning research for future generative AI models.

The Pointnet model was introduced by Qi and Su in the research paper [4] for classification, segmentation, and scene semantic parsing. The model uses max pooling to aggregate information from all points, ensuring invariance to input permutation. The paper shows the model's robustness to outliers and input corruption, and the model is shown to be computationally efficient according to Qi [4]. The Pointnet model is shown to be limited to capturing unordered data of various types, so the model struggles with relationships between neighboring points. Future research implies that other methods like graph neural networks (GNNs) or attention mechanisms would allow for a better understanding of the relationships of the point clouds when classifying. Attention mechanisms are used in transformer and diffusion models to better understand the embedding features [5,6].

3. Methodologies

While existing frameworks offer valuable approaches to predictive modeling, classification, and clustering [7,8], they often present significant limitations when addressing the specific challenge of automating optimal feature placement on complex 3D surfaces within CAD environments. Traditional machine learning algorithms, though adept at learning from diverse datasets and evaluated through metrics like accuracy, primarily focus on static predictions or classifications. This contrasts sharply with our problem's requirement for dynamically generating precise, functionally viable feature placements that adhere to strict engineering tolerances, such as minimizing flex in automotive components. The inherent complexity of 3D geometry and the need for parametric control within CAD software are often overlooked by these general-purpose approaches.

Furthermore, while multimodal machine learning [9] broadens the scope of data types beyond text or numerical inputs, its application to complex engineering design problems like optimal clip placement often struggles with the intricate representation and manipulation of geometric information. For example, frameworks such as UV-net [1,10], which integrate Graph Neural Networks (GNNs) and Convolutional Neural Networks (CNNs) with Boundary Representation (B-rep) for CAD model training, demonstrate progress in

understanding existing designs. However, these approaches are predominantly focused on analysis, reconstruction, or general shape generation, rather than the specific task of intelligently placing discrete functional features that directly impact a part's performance metrics like stiffness or tolerance. They typically lack the direct link to engineering analysis needed to validate optimal placements for specific physical requirements.

Similarly, advances in generative AI, such as OpenAI's ChatGPT3.5, DALL-E v2, and Stable Diffusion [11], excel at creating images or text based on prompts. While these tools showcase powerful generative capabilities, they are fundamentally limited in their ability to produce domain-specific, parametrically controlled outputs like functional STEP files directly usable within engineering software. They generate visual representations rather than precise, actionable 3D models with embedded engineering data. This pervasive gap highlights the critical need for a tailored approach that addresses the unique requirements of automating optimal feature placement for manufacturing, where precision and adherence to physical constraints are paramount.

Proposed Deep Learning Framework

To overcome the aforementioned limitations and specifically address the challenge of automating optimal clip placement for dashboard side panels, we propose a novel framework that integrates a PointNet deep learning model with Python-based CAD automation. Unlike conventional machine learning models that often treat 3D data as flattened projections or voxel grids, PointNet operates directly on unordered point clouds, making it particularly well-suited for processing the irregular and sparse data typical of 3D scanned or modeled surfaces. This direct processing eliminates the need for complex preprocessing steps that can introduce information loss, allowing the model to more effectively learn the intrinsic geometric features relevant to our task.

Our approach treats the identification of optimal clip placements as a 3D spatial optimization problem. The PointNet model acts as a computationally efficient surrogate for traditional engineering analysis, which would typically involve time-consuming finite element analysis (FEA) or iterative physical prototyping to evaluate flex and material thickness. To clarify the ground truth for "optimal" placements, the dataset used for training our PointNet model was derived from rigorous industrial processes. Specifically, original engineering data points for optimal clip placement were generated through comprehensive Finite Element Analysis (FEA) simulations and iterative physical prototyping. These methods were used to evaluate the flex of the panel and the deformation of the material under various load conditions, with "optimal" configurations defined by the lowest acceptable levels of deformation and movement while adhering to manufacturing constraints. By learning from this dataset of FEA-validated clip placements relative to surface geometry and flex properties, PointNet is designed to predict areas on the dashboard side panel that are optimal for clip securement to minimize movement and material deformation.

The core of our proposed framework involves the following steps:

- **Data Generation and Preprocessing**: We generate placement data, potentially leveraging generative AI techniques to simulate diverse panel geometries and corresponding optimal/suboptimal clip locations and surface thicknesses. This data is then converted into point cloud format, suitable for PointNet input.
- **PointNet Model Architecture**: Our PointNet model is designed with shared MLPs and max pooling. The network learns to extract global and local features from the input point cloud, enabling it to infer optimal placement coordinates and associated thickness values.

- **Python-based CAD Automation Integration**: The predicted optimal clip placements and surface thicknesses from the PointNet model are then seamlessly integrated into a CAD environment (e.g., Siemens NX) using Python-based automation scripts. These scripts leverage the CAD software's API to directly generate fully parameterized 3D models, incorporating the optimized features. This ensures that the generated designs are immediately usable for manufacturing and further engineering analysis.
- **Experimental Validation**: We evaluate the prototype's performance by testing its ability to identify optimal clip placements and surface thicknesses across various dashboard side panel geometries. The success rate is measured by the model's ability to achieve minimal flex and acceptable material usage, as validated through simulation, physical testing, or expert review.

This integrated deep learning and CAD automation framework provides a practical and scalable method for automating part design specifically tailored for automotive interior components, offering a clear pathway for applying generative AI to critical engineering challenges.

4. Results

The aim of this section is to detail the implementation of our prototype for automating functional feature placement and present its initial evaluation. While the long-term vision involves a comprehensive generative AI framework, this paper focuses on developing a proof-of-concept using a PointNet model to act as a computationally efficient surrogate for traditional engineering analysis in determining optimal clip placements.

Following extensive research, our identified solution for this prototype integrates a custom PointNet model with pythonocc-core7.8.1 as the core library for handling B-rep data and geometric operations, and CadQuery for manipulating CAD models programmatically. Our approach begins with a STEP file as input, representing the base 3D geometry of the automotive component (e.g., a side panel). It is critical to clarify that our system does not generate the entire CAD model from scratch. Instead, it processes an existing CAD model to precisely and optimally place discrete functional features (clips) onto its complex 3D surfaces. This addresses a significant bottleneck in automotive design, where manual determination of such attachment points is time-consuming and costly.

The overall process is visualized in Figure 1. The input STEP file is first processed to extract its surface geometry and convert it into a point cloud representation, which serves as the input to our PointNet model. The PointNet then outputs predicted coordinates for the clip placements. These coordinates are then used by PythonOCC and CadQuery to programmatically insert the detailed clip geometry into the original CAD model, producing a new STEP file with the integrated clips. The output STEP file thus represents the original design intent with the added, optimized functional features.

The prototype was developed within a Python environment, leveraging tools like CadQuery and CQ-Editor. CQ-Editor provides an interactive environment that allows real-time visualization of the CAD model as the Python code is developed and updated (Figure 2). This immediate feedback mechanism is invaluable for iterative development in CAD automation. The choice of open-source tools like CadQuery and its compatibility with environments such as WSL (Windows Subsystem for Linux) enhances accessibility and collaboration for programmers and design engineers Appendix A. The code also utilizes standard Python libraries such as regex for parsing data and os for file path management.

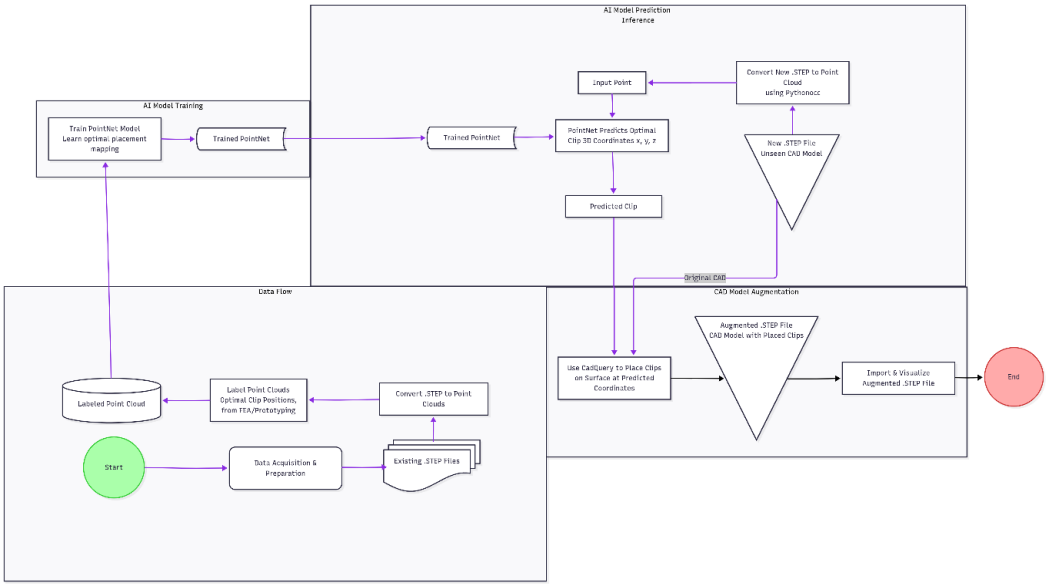
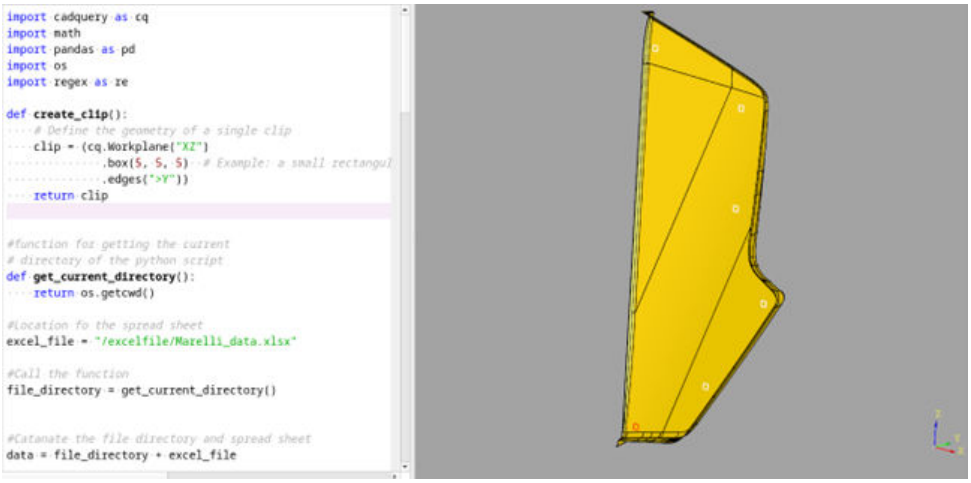


Figure 1. Flowchart illustrating the proposed Python program workflow for automated clip placement. The input is an existing STEP file of a CAD component, which is converted to point cloud data for the PointNet model. The model predicts optimal clip coordinates, which are then used to generate a new STEP file with the clips integrated.



```
import cadquery as cq
import math
import pandas as pd
import os
import regex as re

def create_clip():
    # Define the geometry of a single clip
    clip = (cq.Workplane("XZ")
            .box(5, 5, 5)  # Example: a small rectangul
            .edges(">Y"))
    return clip


#function for getting the current
# directory of the python script
def get_current_directory():
    return os.getcwd()

#Location fo the spread sheet
excel_file = "/excelfile/Marelli_data.xlsx"

#Call the function
file_directory = get_current_directory()


#Catanate the file directory and spread sheet
data = file_directory + excel_file
```

Figure 2. A screenshot displaying the Python code **(left)** in CQ-Editor and the resulting CAD model (**right**) with the placed clips. The white squares and the red square indicate the positions of the predicted clips on the side panel surface.

Prototype Implementation Details: The core implementation involves defining necessary libraries, functions for clip placement, and procedures for parsing input data and executing the placement.

4.1. Define Libraries

The foundational libraries for this project include cadquery for CAD model manipulation and importing STEP files, math, pandas for data handling, os, regex, numpy, and specific modules from OCC.Core for B-rep operations. Utility libraries like customtkinter and tkinter are used for GUI elements, and subprocess is used for system interactions (Listing 1).

Listing 1. The libraries is what allows the Pointnet model to interact with the CAD model.

```
import cadquery as cq
import math
import pandas as pd
import os
import regex as re
import numpy as np

from OCC.Core.STEPControl import STEPControl_Reader

from OCC.Core.TopExp import TopExp_Explorer
from OCC.Core.TopAbs import TopAbs_FACE
from OCC.Core.BRepAdaptor import BRepAdaptor_Surface

from customtkinter import *
import customtkinter as ctk
from tkinter import filedialog
import subprocess
```

4.2. Define Functions

The create_clipV3 function is central to placing the clips (Listing 2) Initially, for visualization and development, placeholder squares were used. In the final prototype, this function imports an actual 3D clip geometry from a specified STEP file and places it at given (x,y,z) coordinates on the input surface using cadquery.Assembly. This approach ensures that the output is a geometrically valid CAD assembly.

Listing 2. Takes in the output coordinates from the PointNet model to place the clips in 3D space on the side panel.

```
# Improved version to use assembly to combine the surface and
    the clips
def create_clipV3(result, clip_positions):
    """
    Adds clips to a surface at specified positions using an
        assembly.

    Args:
        result (cadquery.Shape): The surface to add clips to.
        clip_positions (list of tuples): A list of (x, y, z)
            tuples representing clip positions.

    Returns:
        cadquery.Assembly: The assembly containing the surface
            and clips.
    """

    # Check if the surface (result) is valid
    if result is None or result.val() is None:
        raise ValueError("The surface (result) is null or
            invalid.")
```

```
    # Import the clip from the STEP file
    base_clip = cq.importers.importStep("clips/METAL_CLIP-TOWER_XY_NO-DH.stp")

    # Check if the clip is successfully imported
    if base_clip is None or base_clip.val() is None:
        raise ValueError("Failed to import clip. The base_clip object is null.")

    # Rotate the base clip initially
    base_clip = base_clip.rotate((1, 0, 0), (0, 0, 0), -90)

    # Create an assembly
    assembly = cq.Assembly()

    # Add the surface to the assembly
    assembly.add(result, name="Surface")

    # Iterate through the clip positions and add each clip to the assembly
    for position in clip_positions:
        x, y, z = position

        # Create a new instance of the clip
        new_clip = base_clip

        # Check if the new clip is valid
        if new_clip.val() is None:
            raise ValueError(f"Invalid clip shape at position {position}. Null TopoDS_Shape object.")

        # Translate the clip
        translated_clip = new_clip.translate((x, y, z))

        # Add the translated clip to the assembly
        assembly.add(translated_clip, name=f"Clip_{position}")

    return assembly
```

4.3. Read the Spreadsheet and Step File

Input data is loaded from an Excel spreadsheet containing optimal clip positions (derived from prior engineering analysis) and the base STEP file of the component. The pandas library is used to read the spreadsheet, normalize column names, and extract coordinates. The cadquery.importers.importStep method reads the base 3D model (Listing 3).

Listing 3. The code visualizes the surface in 3D space.

```

#Location fo the spread sheet
excel_file = "/excelfile/Marelli_data.xlsx"

#Call the function
file_directory = get_current_directory()


#Catanate the file directory and spread sheet
data = file_directory + excel_file

#Read in the spread sheet
df = pd.read_excel(data)

# Remove spaces and convert to upper case for all column names
df.columns = [col.replace(' ', '_').upper() for col in df.
    columns]

#Use the data for creating the CAD modal

result = cq.importers.importStep("/home/pythonocc/autocade/
    sheetdata/LMC_INPUT.stp")
```

4.4. Parsing the Spreadsheet

The spreadsheet parsing logic extracts the (X,Y,Z) coordinates for each clip position (Listing 4). A dictionary clip_positions is used to group coordinates by clip number, leveraging regex to identify relevant columns. These are then converted into a list of tuples, clip_positions_list, which serves as the target output for our deep learning model.

Listing 4. Iterate through each column and group by clip number.

```
# Initialize an empty dictionary
clip_positions = {}

# Define the regex pattern to match column names
pattern = re.compile(r'CLIP(\d+)_(X|Y|Z)')

# Iterate through the columns and group them by clip number
columns_by_clip = {}
for col in df.columns:
    match = pattern.match(col)
    if match:
        clip_number = int(match.group(1))
        coordinate = match.group(2)
        if clip_number not in columns_by_clip:
            columns_by_clip[clip_number] = {}
            columns_by_clip[clip_number][coordinate] = col
```

The resulting clip_positions_list provides the ground truth (x,y,z) coordinates for optimal clip placement, reflecting the precise tolerances required for minimizing flex in the side panel (Listing 5).

Listing 5. Organize the data from the spreadsheet.

```
# Iterate through the grouped columns and extract the
    positions
for clip_number, coords in columns_by_clip.items():
  clip_positions[clip_number] = []
  for index, row in df.iterrows():
    if 'X' in coords and 'Y' in coords and 'Z' in coords:
    x_val, y_val, z_val = row[coords['X']], row[coords['Y']],
        row[coords['Z']]
    if pd.notna(x_val) and pd.notna(y_val) and pd.notna(z_val)
        :
      clip_positions[clip_number].append((x_val, y_val, z_val)
         )
```

After, the dictionary is converted into a list of tuples (Listing 6).

Listing 6. Here the code converts the dictionary into a list of tuples containing the coordinates of each clip.

```
# Convert dictionary to list of tuples
clip_positions_list = []
for clip_number in sorted(clip_positions.keys()):
  clip_positions_list.extend(clip_positions[clip_number])

# Print the resulting list of tuples
print(clip_positions_list)
```

The result is a list of tuples that provides the ability to place the clips in the required coordinates for every side panel. The placement of the clips is important to have as little flex of the side panel by millimeters of tolerance.

4.5. Placing the Clips

Finally, the create_clipV3 function is called to place the actual 3D clip geometries at the predicted (or ground truth, for training/testing) coordinates on the side panel. For visualization purposes during development, color coding was used to differentiate individual clips (Listing 7).

Listing 7. Placing the clips on the processed side panel.

```
# Define a color mapping for each clip number
color_mapping = {
        1: (1, 0, 0),  # Red
        2: (0, 1, 0),  # Green
        3: (0, 0, 1),  # Blue
        4: (1, 1, 0),  # Yellow
        5: (1, 0, 1),  # Magenta
        6: (0, 1, 1),  # Cyan
```

```
}

# Display each clip at each coordinate with color coding
assembly = cq.Assembly()
for clip_number, positions in clip_positions.items():
  color = color_mapping.get(clip_number, color_mapping[
      clip_number])
  # Default to white if no color is specified
  for pos in positions:
    clip = create_clip().translate(pos)
    assembly.add(clip, color=cq.Color(*color))
```

4.6. PointNet Module

The neural network central to this prototype is the PointNet model [4], a pioneering architecture for directly processing unordered point cloud data. While PointNet++ [12] offers hierarchical learning, the basic PointNet was chosen for this initial proof-of-concept due to its simplicity and effectiveness for global feature learning from point clouds.

CAD to Point Cloud Transformation: To enable the PointNet model to process the CAD geometry, the input STEP file representing the automotive side panel is first converted into a point cloud. This involves sampling points directly from the surface of the B-rep model using PythonOCC functionalities. For this prototype, a uniform sampling density of 2048 points was applied across the surface of the side panel to generate the input point cloud (shape: N times three, where N is the number of points, times three for the x,y,z coordinates). This point cloud serves as the raw geometric data input to the PointNet.

PointNet Model Details: The model is implemented using the PyTorch version 2.4 library. The architecture features a shared Multilayer Perceptron (MLP) for point-wise feature extraction, followed by a max-pooling layer to aggregate global features from the entire point cloud. This global feature vector is then branched into two separate fully connected pathways:

- **Clip Position Prediction Branch**: This branch outputs 18 values, representing the (x, y, z) coordinates for 6 distinct clips (6 clips $\times$ 3 coordinates = 18 outputs). This is the primary output used for automated placement.
- **Thickness Prediction Branch**: This branch is designed to predict a single thickness value. Although integrated into the model, this branch of thickness prediction was not fully utilized in the current experiment due to the focus on establishing the core clip placement functionality and limitations in integrating comprehensive thickness analysis into the data. The motivation for this branching architecture is to allow early layers to learn fundamental geometric features of the CAD model, with specialized branching paths, then refine weights for clip position and thickness predictions.

Training Details and Results: The model was trained using the Adam optimizer with a learning rate of 0.001 over 1500 epochs. These parameters were selected to serve as a proof-of-concept for the feasibility of using deep learning for automated clip placement. The dataset for training comprised a small set of 8 distinct side panel geometries, with their corresponding optimal clip positions derived from prior engineering analysis (e.g., spreadsheet data) (Listing 8).

Listing 8. The PointNet class using a point-wise MLP to predict coordinates and thickness.

```
class PointNetWithTwoOutputs(nn.Module):
    def __init__(self, num_points=1024):  # Can still keep num_points as a default
        super(PointNetWithTwoOutputs, self).__init__()

        # Point-wise MLP (shared across points)
        self.fc1 = nn.Linear(3, 64)   # From input (x, y, z)
        self.fc2 = nn.Linear(64, 128)
        self.fc3 = nn.Linear(128, 256)

        # Max pooling layer to aggregate global features
        # No longer fixed to 1024 points, we will adjust dynamically
        self.global_pool = nn.MaxPool1d(kernel_size=1)  # Placeholder, adjust this in forward

        # Thickness prediction branch
        self.fc_thickness_1 = nn.Linear(256, 128)
        self.fc_thickness_2 = nn.Linear(128, 64)
        self.fc_thickness_3 = nn.Linear(64, 1)  # Single output for thickness

        # Clip coordinates prediction branch
        self.fc_clips_1 = nn.Linear(256, 128)
        self.fc_clips_2 = nn.Linear(128, 64)
        self.fc_clips_3 = nn.Linear(64, 18)  # 18 outputs for 6 clips (x, y, z) coordinates, 6*3=18

    def forward(self, x):
        # Input x is of shape (B, N, 3) where B is batch size, N is the number of points

        # Point-wise MLP
        x = torch.relu(self.fc1(x))  # (B, N, 64)
        x = torch.relu(self.fc2(x))  # (B, N, 128)
        x = torch.relu(self.fc3(x))  # (B, N, 256)

        # Max pooling across N points to get global features
        N = x.size(1)  # Dynamically get the number of points (N) in this batch
        x = x.permute(0, 2, 1)  # Change to (B, 256, N) for pooling
        self.global_pool = nn.MaxPool1d(kernel_size=N)  # Adjust MaxPool1d to the current N
        x = self.global_pool(x)  # (B, 256, 1)
        x = x.squeeze(-1)        # (B, 256)

        # Thickness prediction branch
        thickness = torch.relu(self.fc_thickness_1(x))
```

```
        thickness = torch.relu(self.fc_thickness_2(thickness))
        thickness_output = self.fc_thickness_3(thickness)  # Scalar output (B, 1)

        # Clip position prediction branch
        clips = torch.relu(self.fc_clips_1(x))
        clips = torch.relu(self.fc_clips_2(clips))
        clips_output = self.fc_clips_3(clips)  # (B, 18)
        clips_output = clips_output.view(-1, 6, 3)  # Reshape to (B, 6, 3)

        return thickness_output, clips_output
```

Our initial experimental results demonstrate a 75% success rate (6 out of 8 samples) in accurately predicting the clip positions within a defined tolerance. This success rate indicates the foundational potential of using a PointNet-based approach as a surrogate for complex engineering analysis in design automation.

Lack of State-of-the-Art (SOTA) Comparison: It is important to note that direct quantitative comparison to existing state-of-the-art methods is challenging due to the highly specific nature of our problem: automated optimal discrete functional feature placement on complex existing CAD surfaces for specific engineering criteria (minimizing flex). Current SOTA methods in generative CAD primarily focus on generating entire CAD models from scratch (e.g., BrepGen, DeepCAD) or material classification, which are distinct from our focused task. Our work represents an initial exploration into this specific application domain.

Analysis of Failed Cases: The remaining 25% of cases (2 out of 8 samples) resulted in inaccurate clip placements. Preliminary analysis suggests that these failures may be attributed to a combination of factors:

- Limited Data Diversity: The small prototype dataset might not sufficiently capture the full range of geometric variations or complex optimal placement patterns required for robust generalization across all samples.
- Model Limitations: While PointNet is effective for global feature learning, its sensitivity to local geometries and fine-grained details pertinent to optimal placement might be a contributing factor without more advanced architectural considerations (e.g., hierarchical features like PointNet++ or local feature aggregation).
- Overfitting: As indicated by the high number of epochs for a small dataset, the model showed signs of overfitting to the limited training data, which could lead to poor performance on unseen variations.

Further research will focus on expanding the dataset, exploring more robust model architectures, and refining training strategies to improve generalization and accuracy across a wider range of automotive components. This implementation serves as a crucial initial validation that generative AI can indeed be applied to solve specific, complex challenges in CAD model refinement.

5. Discussion

The prototype implementation, which used a PointNet model for automated clip placement, yielded a success rate of 75% (6 of 8 samples). This initial performance, despite being based on a limited dataset, serves as a significant proof-of-concept for the feasibility of employing deep learning as a computationally efficient surrogate for traditional engi-

neering analysis in optimizing functional feature placement. The model demonstrated a fundamental understanding of geometric surfaces and the general task of putting clips.

However, a direct analysis of these results also highlights critical limitations inherent to the current prototype. The small sample size (eight distinct side panel geometries) coupled with the observed overfitting (as noted by the chosen training parameters and epochs in Section 5) significantly restricts the model's ability to generalize to unseen variations in geometry and optimal placement logic. This inherently limits its readiness for a production environment. Although the model grasped the existing surface characteristics, its capacity to infer and apply complex contextual rules governing optimal clip placement—rules often derived from intricate physical requirements such as minimizing flex—was constrained by the simplicity of the architecture and the narrow scope of the training data. For example, the two failed cases often exhibited incorrect spatial distribution or minor deviations from the expected optimal positions, suggesting that the model struggled with nuanced geometric and load-bearing considerations that are paramount in automotive design.

To overcome these limitations and evolve towards a production-ready solution, the next crucial step in this research involves exploring more advanced deep learning architectures that are capable of capturing richer contextual information and generalizing across a wider array of design scenarios. Attention mechanisms, commonly found in transformer models, which are widely used in natural language processing for sequence prediction, offer a promising avenue. These models could potentially learn the interdependencies between different points on the CAD surface and the relative placement of clips, thereby understanding the surrounding context and implicit design rules more effectively.

Furthermore, diffusion models present a compelling candidate for future development. These generative models, renowned for their success in image generation by progressively denoising a Gaussian noise input based on a prompt, hold significant potential for 3D CAD. Kalischek et al. [13] demonstrated the adaptability of diffusion models to 3D space, showing their ability to generate complex tetrahedral meshes. This "TetraDiffusion" approach highlights how modified diffusion models can operate in 3D, potentially allowing for the generation of not just coordinates, but perhaps even localized geometric features or variations. Diffusion models could also incorporate parameters or conditioning mechanisms to directly restrict output quality and ensure adherence to design constraints, mitigating the risk of generating physically non-viable solutions. The work by Thomas et al. [14], introducing Kernel Point Convolution (KPConv) for calculating point cloud convolutions in Euclidean space, offers a potential method for integrating detailed local feature learning into such advanced models, inspired by traditional image-based convolution but adapted for irregular point cloud data.

In conclusion, while our current implementation's six out of eight successful samples demonstrate the fundamental viability of applying deep learning to specialized CAD automation tasks, the insights gained from this prototype underscore the imperative for larger, more diverse datasets and sophisticated model architectures. Future research will focus on leveraging models like Transformers or diffusion models, possibly incorporating advanced point cloud convolution techniques, to achieve the robustness and precision required for real-world automotive component design and optimization.

6. Conclusions

This research successfully demonstrates the potential of AI-driven optimization for the placement of functional features within CAD models, specifically applied to automotive components. Our prototype, leveraging a PointNet model as a computationally efficient surrogate for traditional Finite Element Analysis (FEA), successfully predicts optimal clip placements to minimize flex and material deformation. The implementation effectively

used CadQuery for programmatic manipulation and integration of clips into the base sheet model.

Our experimental results, which achieved a success rate of 75% (six of eight samples), provide a compelling proof of concept. This initial validation underscores the viability of using deep learning to automate a typically time-consuming and manual design optimization task. While the current prototype, built on a limited dataset, is not yet production-ready, these findings offer critical insights into the challenges and opportunities for integrating AI into precise engineering design workflows.

Looking ahead, future research will focus on overcoming current limitations, particularly in model generalization and handling more complex contextual rules for optimal placement. The most promising avenues involve exploring advanced generative AI architectures, such as Transformer models combined with 3D autoencoders. These models could significantly enhance the AI's ability to capture intricate representations of CAD geometries and learn complex design correlations. Furthermore, investigating diffusion models adapted for 3D space presents an exciting direction for generating diverse and highly constrained design variations, moving beyond simple coordinate prediction to more comprehensive generative design assistance. Such advancements will be crucial for creating robust, production-ready AI solutions that truly revolutionize efficiency and innovation in CAD-based engineering design.

Author Contributions: Conceptualization, A.K. and G.P.; methodology, A.K. and G.P.; software, A.K. and G.P.; validation, A.K. and G.P.; formal analysis, A.K. and G.P.; investigation, A.K. and G.P.; resources, A.K. and G.P.; data curation, A.K. and G.P.; writing—original draft preparation, A.K. and G.P.; writing—review and editing, A.K. and G.P.; visualization, A.K. and G.P.; supervision, G.P.; project administration, G.P.; funding acquisition, G.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Data Availability Statement: No new data were created or analysed in this study. Data sharing is not applicable to this article.

Acknowledgments: The authors acknowledge the contributions of Ashley Tennant, Abraham Camacho, Shannon Combs, Chris Walters, Mike English, and Sanjeeva Palaka from Marelli to the research titled "Implementation of CAD Models in Programming for Utilization in Generative Artificial Intelligence".

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

The implementation is set up with Windows Subsystems for Linux (WSL) on Windows 11. The decision to use WSL allows for the use of Linux terminal commands, and documentation on GitHub uses Linux due to the open source nature. Installing WSL is simple using *wsl–install*; this provides the latest version of WSL and starts with Ubuntu. After rebooting the system for the changes to take effect, the implementation uses Ubuntu-22.04, so for reproducibility, we recommend using the command *wsl–install–dUbuntu*-22.04, then running *wsld Ubuntu*-22.04 going forward. We also recommend performing a system update by running *sudo apt update* to make sure the system is up to date [15].

Once WSL is set up and the user is in the Linux Command Line Interface (CLI), it is important to install Miniconda for setting up virtual environments for Python projects. From the Anaconda documentation, copy and paste the commands to install Miniconda [16]. Then, run the command */miniconda*3/*bin*/*condainitbash* to set up the command path to

use the *conda* command. After, it is best to run *source /.bashrc* to let the changes take effect. The result should be to the left of the username in the CLI, which should say "(base)", showing that conda is working.

Miniconda now installed. The ability to create virtual environments is a powerful tool in software development to isolate Python packages to avoid version dependency conflicts. Setting up the virtual environment follows the command conda create–name cadqueryenv python = 3.11, and this command creates a virtual environment with the name *cadqueryenv* with Python version 3.11 as the environment chosen for implementation [17].

The virtual environment should be running. The environment is activated by typing the command *condaactivatecadqueryenv*, then the user is in the virtual environment and can install the necessary packages. The first package to install is CadQuery itself using command *pipinstallcadquery*. The other dependency to install is OCP for the Open CASCADE Technology (OCCT) for CAD modeling using Python. The command to install the library is *conda install-c conda-forge-c cadquery ocp*. The command to install OCP uses conda to install the OCP package, which is helpful for installing all of the essential dependencies required for the library to work [18]. The final step is obtaining the CQ-Editor, which enables the use of Python scripting and CAD visualization. First, installing the GitHub version 0.5.0 would provide all of the important files. Then, after the command finishes running the last command *shCQ-editor-master-Linux-x*8664.*sh*, make sure it is in the right file directory before executing the command.

References

1. Xu, X.; Lambourne, J.G.; Jayaraman, P.K.; Wang, Z.; Willis, K.D.; Furukawa, Y. BrepGen: A B-rep Generative Diffusion Model with Structured Latent Geometry. *arXiv* **2024**, arXiv:2401.15563. [CrossRef]
2. Faruqi, F.; Tian, Y.; Phadnis, V.; Jampani, V.; Mueller, S. Shaping Realities: Enhancing 3D Generative AI with Fabrication Constraints. *arXiv* **2024**, arXiv:2404.10142. [CrossRef]
3. Hua, B.S.; Tran, M.K.; Yeung, S.K. Pointwise convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 23 June 2018; pp. 984–993.
4. Qi, C.R.; Su, H.; Mo, K.; Guibas, L.J. Pointnet: Deep learning on point sets for 3D classification and segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 652–660.
5. Vaswani, A. Attention is all you need. *arXiv* **2024**, arXiv:1706.03762. [PubMed]
6. Rombach, R.; Blattmann, A.; Lorenz, D.; Esser, P.; Ommer, B. High-resolution image synthesis with latent diffusion models. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, New Orleans, LA, USA, 18–24 June 2022; pp. 10684–10695.
7. Gao, Z.; Sossou, D.; Zhao, Y.F. Machine learning aided design and optimization of conformal porous structures. In Proceedings of the International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers, Virtual, 17–19 August 2020; Volume 83983, p. V009T09A041.
8. Purwar, A.; Desai, K.A.; Canfield, S.; Rai, R.; Nie, Z. Machine Learning and Representation Issues in CAD/CAM. *J. Comput. Inf. Sci. Eng.* **2024**, *24*, 010301. [CrossRef]
9. Song, B.; Zhou, R.; Ahmed, F. Multi-modal machine learning in engineering design: A review and future directions. *J. Comput. Inf. Sci. Eng.* **2024**, *24*, 010801. [CrossRef]
10. Hou, J.; Luo, C.; Qin, F.; Shao, Y.; Chen, X. FuS-GCN: Efficient B-rep based graph convolutional networks for 3D-CAD model classification and retrieval. *Adv. Eng. Inform.* **2023**, *56*, 102008. [CrossRef]
11. Yonder, V.; Dulgeroglu, O.; Dogan, F.; Cavka, H. The Role of the Computational Designer From Computer-Aided Design to Machine Learning-Aided Design A study on generative models and design prompts. In Proceedings of the International Conference on Education and Research in Computer Aided Architectural Design in Europe, Graz, Austria, 20–23 September 2023.
12. Qi, C.R.; Yi, L.; Su, H.; Guibas, L.J. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Adv. Neural Inf. Process. Syst.* **2017**, *30*.
13. Kalischek, N.; Peters, T.; Wegner, J.D.; Schindler, K. TetraDiffusion: Tetrahedral Diffusion Models for 3D Shape Generation. *arXiv* **2024**, arXiv:2211.13220.
14. Thomas, H.; Qi, C.R.; Deschaud, J.E.; Marcotegui, B.; Goulette, F.; Guibas, L.J. Kpconv: Flexible and deformable convolution for point clouds. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Seoul, Republic of Korea, 27 October–2 November 2019; pp. 6411–6420.

15. Mattwojo. Install WSL. Available online: https://learn.microsoft.com/en-us/windows/wsl/install (accessed on 7 August 2024).
16. Anaconda, Inc. *Miniconda—Miniconda Documentation*; Anaconda, Inc.: Austin, TX, USA, 2025.
17. CadQuery. CadQuery/cadquery: A Python Parametric CAD Scripting Framework Based on OCCT. Available online: https://www.anaconda.com/docs/getting-started/miniconda/main (accessed on 23 June 2024).
18. CadQuery. CadQuery/OCP. Github. Available online: https://github.com/CadQuery/OCP (accessed on 15 June 2024).