AI-Assisted Computer Aided Design of Mechanical Parts

By

Tala Ibraheem & Joud Bani-Issa & Tala Hashem

Supervised by

Dr. Osama Abu-Sharkh

Submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF SCIENCE

in

COMPUTER ENGINEERING

at

PRINCESS SUMAYA UNIVERSITY FOR TECHNOLOGY

Amman, Jordan

Second Semester 2023-2024

This is to certify that I have examined

this copy of an engineering documentation by

Tala Ibraheem, Joud Bani-Issa, and Tala Hashem

And have found that it is complete and satisfactory in all respect,
And that any and all revisions required by the final Examining Committee have been made

---

Osama Abu-Sharkh

Acknowledgment

We extend our heartfelt gratitude and appreciation to our family members and friends for their unwavering support and encouragement throughout this journey. Their love and understanding have been our source of strength and motivation. All our gratitude and appreciation to our university supervisor Dr.Osama Abu Sharkh and our Istari Digital Supervisors Bahaa Abu Nojaim and Kanaan Al-Manasrah for their time and effort.

We thank Princess Sumaya University for Technology (PSUT) for providing the resources and platform essential for this study. PSUT's commitment to innovation and research excellence has been invaluable. Without the collective support, patience, and encouragement from these remarkable individuals, this accomplishment would not have been possible. Thank you for believing in us and for helping us achieve our goals.

T. Ibraheem

J. Bani Issa

T. Hashem

Abstract

The importance of CAD modeling lies in its ability to streamline and enhance the design process, allowing for precise and efficient creation of complex mechanical structures. CAD modeling enables designers to visualize, test, and refine their ideas in a digital environment, reducing the need for physical prototypes and accelerating the development cycle.

This project aims to transform mechanical design processes by integrating generative models into CAD systems. Using the capabilities of pre-trained language models, we generate innovative CAD solutions through a two-phase methodology. The first phase, we create the dataset and fine-tune the pre-trained models on it and the final result is a JSON file containing instructions on how to construct 3D models.

In the second phase, we convert the JSON file into CadQuery code. CadQuery, renowned for its efficacy as a script-based CAD modeling framework, provides the perfect platform for this transformation. By translating the JSON description into CadQuery code, we ensure a seamless transition from abstract design concepts to practical CAD implementations.

The overarching goal of this project is to popularize the CAD modeling process, making it significantly more accessible and efficient. By automating the generation and conversion phases, we enable users of varying skill levels, regardless of their familiarity with CAD software or programming, to create sophisticated CAD models effortlessly. This approach not only accelerates the design process but also opens new possibilities for innovation and creativity in mechanical design.

Table of Contents

Acknowledgment 3

Abstract 4

List of Figures 7

List of Tables 8

Chapter 1 Introduction 9

1.1 Motivation 9

1.2 Objectives 10

1.3 Design Requirements 10

1.4 Realistic Constraints 11

1.5 Engineering Standards 11

1.6 Abridged Description of the Proposed System 11

1.7 Task Distribution 12

1.8 Organization 13

Chapter 2 Background and Literature Review 14

2.1 Background 14

2.1.1 Large Language Models 14

2.1.2 CadQuery 16

2.1.3 Fusion 360 17

2.1.4 Gradio 20

2.2 Literature Review 21

Chapter 3 System Design 26

3.1 Design Requirements 26

3.2 Analysis of Design Requirements 27

3.3 Realistic Constraints 28

3.4 Different Design Approaches 28

3.5 Project System Integration 29

3.6 Proposed System 30

3.6.1 Mechanical Components 30

3.6.2 Dataset Creation 35

3.6.3 JSON to CadQuery Conversion 41

3.6.4 Overall System 42

Chapter 4 Implementation 44
4.1 Large Language Model Architecture 44
4.2 Model Configuration 47
4.2.1 Quantization 47
4.2.2 Tokenization 49
4.2.3 LoRa Configuration 50
4.2.4 Training Preparation 51
4.3 Scenario 52
Chapter 5 Experimental Results and Performance Evaluation 53
5.1 Training parameters 53
5.2 Evaluation Metric 54
5.3 Training Outcome 57
5.3.1 Validation 57
5.3.2 Comparison Based on Loss Function 58
5.3.3 Comparison based on Bleu Score 59
5.3.4 Comparison based on Similarity Text 60
5.3.5 Comparison based on MAPE 62
5.4 Filling the JSON file 63
5.5 CadQuery Conversion 64
5.6 Website Creation 68
5.6.1 Platform Choice: Squarespace 68
5.6.2 Website Functionality 68
5.6.3 About Us page 69
5.6.4 Website design 69
5.7 Meeting Design Requirements within Realistic Constraints 71
Chapter 6 Conclusion and Future Work 74
References 77

List of Figures

FIGURE 1: PROPOSED SYSTEM. 12
FIGURE 2: ILLUSTRATION OF SKETCHING AND EXTRUSION. 17
FIGURE 3: MODEL SHOWCASE IN FUSION 360. 20
FIGURE 4: SYSTEM INTEGRATION. 30
FIGURE 5: RING DIMENSIONS. 31
FIGURE 6 : TUBE DIMENSIONS. 32
FIGURE 7: HEX NUT DIMENSIONS. 33
FIGURE 8: MILD SQUARE BAR DIMENSIONS. 34
FIGURE 9: LEAF SPRING SHACKLE LINK DIMENSIONS. 35
FIGURE 10: MODEL PREDICTION FORMAT. 35
FIGURE 11: DATASET DISTRIBUTION. 36
FIGURE 12: HEXAGONAL POINT DISTRIBUTION. 38
FIGURE 13: POINTS DISPLAY. 39
FIGURE 14: DATASET CREATION. 41
FIGURE 15: DETAILED DESCRIPTION OF THE PROJECT. 43
FIGURE 16: MODEL ARCHITECTURE. 45
FIGURE 17: MULTI-HEAD ATTENTION. 45
FIGURE 18: LINEAR PROJECTION PROCESSING. 46
FIGURE 19: IMPACT OF VARIOUS QUANTIZATION LEVELS ON THE OPT MODEL. 48
FIGURE 20: TOKENIZATION. 49
FIGURE 21: TRAINING PREPARATION. 51
FIGURE 22: SIMILARITY TEXT BY WORDS CODE. 56
FIGURE 23: MAPE CODE. 57
FIGURE 24: COMPARISON OF MODELS BASED ON R AND ALPHA. 58
FIGURE 25: LOSS CURVE AT R=64 AND A=64. 59
FIGURE 26: LOSS CURVE AT R=16 AND A=16. 59
FIGURE 27: COMPARISON OF THE MODELS BASED ON THE BLEU SCORE. 60
FIGURE 28: COMPARISON OF THE MODELS BASED ON TEXT SIMILARITY AGAINST LEARNING RATES. 61
FIGURE 29: FILLING THE JSON FILE. 64
FIGURE 30: GENERATED CODE [A, B, C, D]. 67
FIGURE 31: MODELS 3D VIEW. 68
FIGURE 32: HOME PAGE. 69
FIGURE 33: OPTIONS FOR THE USER. 70
FIGURE 34: ABOUT US PAGE. 70

List of Tables

TABLE 1: TASK DISTRIBUTION 12
TABLE 2: LEADERBOARD SCORE 15
TABLE 3: PROMPTING FORMAT FOR THE THREE MODELS 16
TABLE 4: DATA FORMAT DESCRIPTION 18
TABLE 5: OPERATIONS OF SKETCH AND EXTRUDE ENTITIES 19
TABLE 6: CHARACTERISTICS COMPARISON OF SOME CAD MODELERS 22
TABLE 7: EXAMPLE ON THE CONVERSION OF JSON INFORMATION TO CADQUERY 43
TABLE 8: MODELS SIZES AFTER QUANTIZATION 47
TABLE 9: AVERAGE BENCHMARK SCORE DROP AFTER QUANTIZATION 49
TABLE 10: EFFECT OF R ON THE NUMBER OF PARAMETERS 51
TABLE 11: HYPERPARAMETERS OVERVIEW 53
TABLE 12: TIME TAKEN BY EACH MODEL DURING EVALUATION 57
TABLE 13: ACCURACY OF THE DIMENSIONS BETWEEN THE FINAL MODELS 63
TABLE 14: ACHIEVEMENT OF DESIGN REQUIREMENTS 71
TABLE 15: FULFILMENT OF REALISTIC CONSTRAINTS 73

Chapter 1 Introduction

This chapter provides a brief introduction to the project. In sections 1.1 and 1.2, we provide a brief description of the proposed project and highlight the importance of addressing it, followed by the intended objectives of this project. In the following sections: 1.3, 1.4, and 1.5, a list of design requirements, realistic constraints as well as relevant engineering standards used in the project are provided, respectively. In section 1.6, we dig a little bit deeper into the technicalities and design of the proposed system. The task distribution among the team members is provided in section 1.7 and finally the organization of the rest of the documentation is provided in section 1.8.

1.1 Motivation

With time, Artificial Intelligence (AI) is constantly implemented more and more in all aspects of life from education and medicine to chat boxes. It naturally follows that AI will also be introduced to the manufacturing industry. The integration of Artificial Intelligence (AI) into Computer-Aided Design (CAD) gave new abilities in creating and optimizing designs.

AI is applied in CAD design through two approaches: AI-assisted design and AI-generated design. In AI-assisted design, the technology offers recommendations to designers based on design criteria, reducing time and effort. On the other hand, AI-generated design utilizes algorithms to produce design alternatives that align with predefined criteria.

This project focuses on utilizing large language model (LLM) concepts to develop an AI Assisted Computer Aided Design (CAD) of basic Mechanical Parts. The motivation of this project stems from the introduction of the novel technique of using CadQuery code to simulate 3D CAD models of basic mechanical parts.

CadQuery is a relatively new tool. It is an open-source python library for building parametric 3D CAD models. CadQuery serves a few purposes: Build models with scripts that are as close as possible to how you would describe the object to a human, using a standard, already established programming language. Create parametric models that can be very easily customized by end users.

This project is driven by the flexibility of letting the user define the dimension requirements of the wanted mechanical part.

1.2 Objectives

The objective of this project is to design and develop an AI-assisted tool that can facilitate and expedite the design of basic mechanical parts. The project aims to create a tool that utilizes LLM concepts to facilitate the design of mechanical parts using natural language commands and prompts leading to a more efficient, and user - friendly mechanical design process.

The aim is for the system to create a json file that contains parameters that the model accurately extracted from the user prompt and then convert the json file containing the extracted parameters into a CadQuery code that if desired can be rendered into a 3D model using a 3D viewer. By combining CadQuery's precision with JSON's abstraction and compatibility, designers can achieve a well-rounded approach that balances detail and flexibility in CAD modelling.

1.3 Design Requirements

The design of the system shall achieve the following requirements:

- The system shall include the development of a user-friendly interface for users to input their design requirements for a mechanical part using natural language commands and prompts.
- The system shall exploit LLM’s capabilities to analyze and understand the user's description and requirements.
- The work shall include the generation of a dataset of at least 3 mechanical parts.
- Each mechanical part within the dataset shall have at least 100 samples.
- The pre-trained LLM shall be fine-tuned using the generated dataset.
- The system shall generate a CadQuery code based on the user's input.
- The system shall be able to convert the CadQuery code to a file that is readable by CAD viewers.
- The system shall be able to display the mechanical part to the user.

- The system shall be able to determine the mechanical part according to the user's input with an accuracy of at least 70%.

1.4 Realistic Constraints

The realistic constraints that should be taken into consideration in the design process are:

Economic Constraints: The cost shall not exceed 400JDs.

Manufacturability & Sustainability Constraints: The unavailability of datasets that contain many samples of various mechanical parts to train an LLM. The computational complexity to fine-tune the pre-trained LLM. The updated versions of LLMs.

Ethical: Potential privacy concerns and signing Istari Digital NDA.

1.5 Engineering Standards

The following engineering standard is relevant to this projects work:

- STL: is a popular file format used in computer-aided design (CAD) and 3D printing to represent three-dimensional models. These files use a group of triangular facets to describe the surface geometry of three-dimensional objects. Because the format is easy to use and widely supported by CAD applications, it is used to save and exchange 3D models.

1.6 Abridged Description of the Proposed System

With the goal of achieving a final CAD model from user text descriptions, this approach ensures that users are not required to have proficiency in modelling. It is simply a matter of specifying the right parameters and dimensions for the model. As described in Figure 1, The process begins by taking the pre-trained LLM, which has been trained on large amounts of textual data and use it for the fine-tuning phase. In this phase, the pre-trained LLM is fine-tuned on a customized dataset for basic mechanical parts. This fine-tuning process enhances the LLM's adaptability to real word scenarios. Once the fine-tuned LLM is established, it is ready to address user requests, providing the expected output of the phase to be a JSON file, which after then will be converted to CadQuery code that will be useful for future use.

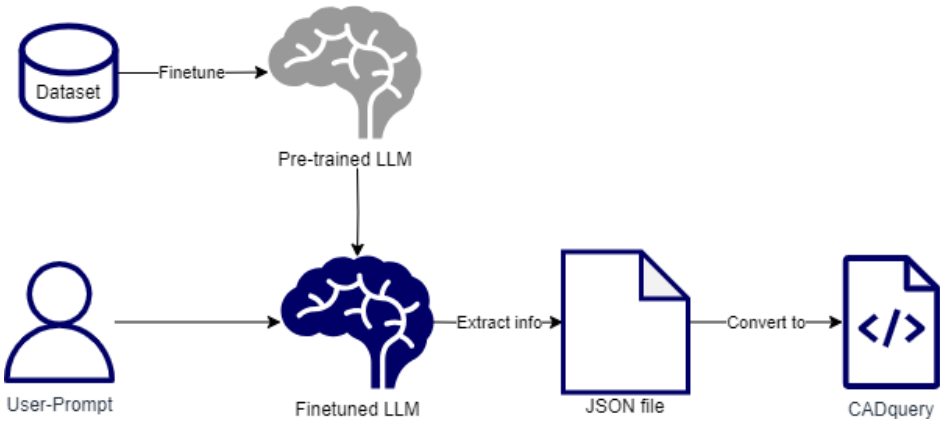

Figure 1: Proposed System.

1.7 Task Distribution

Table 1 shows the task distribution among the members of the group.

Table 1: Task Distribution

Task	**Accomplished By**		
	Tala .H	**Joud**	**Tala. I**
Dataset filtering and selecting shapes.	✔	✔	✔
Setting up and operating Fusion 360 GYM environment.	✔	✔	✔
Designing and implementing the process of dataset creation.	✔	✔	✔
Designing and implementing the fine-tuning process of the LLMs	✔	✔	✔
Training the LLMs	✔	✔	✔
Designing and implementing the tools that create the JSON files	✔	✔	✔
Designing and implementing the tool that converts	✔	✔	✔

the JSON files to CadQuery codes			
Creating the Website	✔	✔	✔
Documentation	✔	✔	✔

1.8 Organization

The document is organized as follows: Chapter 2 presents the needed theoretical background to fully comprehend and develop the designed system. Additionally, a literature review of different similar approaches and methods was conducted in the same chapter. Chapter 3 describes the design process, from the analysis of all requirements and constraints to the comparison of different design approaches that could be implemented. Then, ultimately developing a complete design to solve the proposed problem. Chapter 4 delves into the implementation requirements of the project. Also, the chapter explains how the models are configured as well as a dive into their architecture. Chapter 5 presents the results and findings. The chapter compares the whole process across the three LLMs. Then, the chapter describes the following steps after training in which is the filling of the JSON file and the conversion of the JSON to CadQuery. Finally, Chapter 6 represents the conclusion and future work.

Chapter 2 Background and Literature Review

This chapter introduces the key concepts employed in this project. Additionally, it presents a review of previous research and work related to the project. This chapter aims to provide a comprehensive understanding of the theoretical foundations and the existing body of knowledge relevant to the study.

2.1 Background

This chapter will present the background knowledge for this project. The following sections explain topics relevant to the project. Section 2.2 presents how the system is integrated using the components described in section 2.1. Section 2.3 presents previous literature like this project in certain aspects.

2.1.1 Large Language Models

LLMs are advanced deep learning models, typically based on the transformer architecture [1] which transforms one sequence into another. They consist of an encoder and a decoder with self-attention capabilities, allowing them to extract meaning from sequences of text and understand relationships between words and phrases. Unlike earlier models like recurrent neural networks (RNN), transformers process entire sequences in parallel, utilizing GPUs for faster training. LLMs, such as those based on transformers, undergo unsupervised training or self-learning. Through exposure to vast amounts of data, they grasp basic grammar, languages, and general knowledge. This is achieved through self-supervised or semi-supervised learning, where the model predicts the next word in a sequence. The ability to parallel process sequences enhances training efficiency compared to sequential methods.

LLMs that are already trained on lots of text are super helpful for understanding and generating language. Platforms like Hugging Face make these models easy to use and find. Hugging Face offers a bunch of these pre-trained models such as Falcon7B, LLAMA, and LLAMA2, that are good for different language tasks like understanding context or summarizing text. Performance across these LLMs can vary, as illustrated in Table 2. It's crucial for users to refer to performance benchmarks and comparisons to choose the model that aligns best with their specific task requirements. The Hugging Face Leaderboard [2], is a widely utilized platform among engineers and data scientists for evaluating LLMs. The leaderboard shown within Table 2 relies on Eleuther AI Language Model Evaluation Harness, a unified framework designed to test generative language models developed by EleutherAI. Hugging Face

doesn't stop at just giving you these pre-trained models. You can also "fine-tune" them, kind of like customizing a tool to fit your needs better. It's like taking a general-purpose tool and making it perfect for the particular task. This flexibility is handy for people who want the benefits of big pre-trained models but need them to be more specific to what they're working on [3] [4].

Table 2: Leaderboard Score

Model	*Leaderboard Score*
Falcon-7B	47.01
MPT-7B	48.7
Llama-7B	49.71
Llama-2-7B	54.32
Llama-2-13B	58.67
MPT-30B	55.7
Falcon-40B	61.5
Llama-65B	62.1

This project focuses on finetuning three main LLMs: Mistral 7B, Falcon 7B and, Llama 2 7B.

A. Mistral-7B

In the domain of NLP, attaining high model performance often entails increased computational costs and latency. The Mistral 7B model exemplifies this balance by outperforming previous top models across benchmarks while maintaining efficient inference. Leveraging grouped-query attention (GQA) and sliding window attention (SWA), Mistral 7B accelerates inference speed and reduces memory requirements, crucial for real-time applications. SWA also enhances Mistral 7B's ability to handle longer sequences effectively at reduced computational cost. Mistral-7B-v0.1 outperforms Llama 2 13B on all benchmarks tested. When training on Mistral the dataset follows a prompting format which is the system/instruct ‘<s>[INST] [/INST] </s>’ such as the example shown in Table 3 [5].

B. Falcon-7B

Falcon-7B, a 7B parameters causal decoder-only model developed by TII and trained on 1,500B tokens of RefinedWeb enhanced with curated corpora, is offered under the Apache 2.0 license. It stands out for its optimized architecture for inference, featuring FlashAttention and multi-query mechanisms. This model surpasses comparable open-source models and is suitable for various NLP tasks, including text generation and summarization. However, it's important to note that Falcon-7B requires further fine-tuning for specific use cases. While it supports multiple languages, its training data is primarily in English and French, and it may carry biases present in online content. Falcon models adopt a prompting format that is called triple hash '###' as shown in Table 3 [6] [7].

C. Llama 2 – 7B

Meta has recently unveiled the Llama 2 family of large language models (LLMs), which includes models ranging from 7 billion to 70 billion parameters. Specifically tailored for dialogue applications, Llama 2 employs supervised fine-tuning (SFT) and reinforcement learning with human feedback (RLHF) to align with human preferences for helpfulness and safety. It's important to note that Llama 2 is designed for commercial and research purposes in English. Tuned versions, labeled Llama-2-Chat, are particularly suited for assistant-like chat interactions. Additionally, pre-trained models can be adapted for various natural language generation tasks. Llama 2 prompting format as shown in Table 3 uses is the same as Mistral [8].

Table 3: Prompting Format for the Three Models

Model	*Special Tag*	*Prompting Format*
Falcon 7b	###	### {user_prompt:} ### {Assistant:}
LLama2 7b	<s>[INST] [/INST] </s>	<s>[INST]{{ user_prompt }} [/INST] {{ function_call }}</s>
Mistral	<s>[INST] [/INST] </s>	<s>[INST]{{ user_prompt }} [/INST] {{ function_call }}</s>

2.1.2 CadQuery

CadQuery is a user-friendly Python library designed for building parametric 3D CAD models. Its primary goals include creating models with scripts that closely resemble human descriptions, using a standard programming language. The library allows the creation of parametric models easily customizable by end users. It supports the output of high-quality CAD formats like STEP, AMF, and 3MF, in addition to traditional STL. In CadQuery, operations refer to the various actions or manipulations performed on 3D models to create, modify, or analyze geometric entities. CadQuery is a

Python library built on top of the Open CASCADE Technology (OCCT), providing a scriptable and parametric interface for 3D modeling within the Python programming language. The library allows users to carry out a range of operations on 3D shapes, including creating primitives, combining and subtracting shapes, and applying transformations. In CadQuery, the operations of sketching and extruding serve as fundamental building blocks for constructing 3D models within the Python scripting environment. Sketching involves defining a 2D profile by specifying geometric entities like lines, arcs, and circles. Following the sketching phase, the extrude operation is employed to give depth and volume to the 2D sketch, transforming it into a fully realized 3D object as shown in Figure 2.

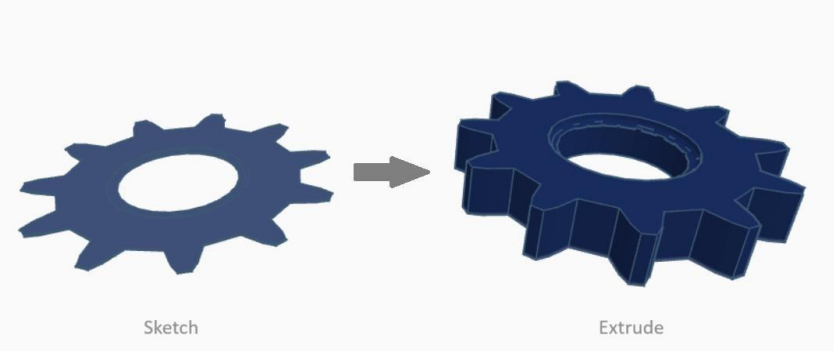

Figure 2: Illustration of Sketching and Extrusion.

2.1.3 Fusion 360

A. Fusion 360 gallery dataset:

In the world of digital design, a CAD model is like a detailed digital blueprint. It contains all the information needed to create and manipulate 3D models, making it a crucial asset for designers and engineers. Fusion 350 Gallery Reconstruction Dataset [9], offers a collection of 8,625 CAD designs created by humans. These designs are described in a straightforward language using basic 'sketch' [10] and 'extrude' [11] actions. The dataset focuses on mechanical components or subset of components, making it possible to rebuild the final 3D shapes. These CAD designs, totaling 8,625, are expressed using simple language, involving actions like sketching and extruding. By combining these actions with Boolean operations, a wide variety of expressive 3D designs can be created.

The dataset contains different formats for each component. Every CAD design in the dataset is thoughtfully expressed as: OBJ, JSON, STP, SMT, and PNG which add a layer of flexibility for integrating the model in different aspects. Table 4 below provides an overview of the diverse file formats present in the dataset.

Table 4: Data format Description

Data Format	*Description*	*Environments and Applications*
.json	A human-readable data interchange format derived from JavaScript object notation. It supports data types such as strings, numbers, objects, arrays, Booleans, and null. Keys and string values must be enclosed in double quotes.	• Text Editors • Command Line • Programming Environments
.smt	The .smt file essentially acts as a home language for Fusion 360's CAD kernel, making it a reliable and accurate source for preserving the ground truth geometry of B-Rep entities, including bodies and faces.	• AutoCAD • Autodesk Fusion 360 • QuickBooks
.step	In CAD and 3D printing, STEP files are crucial for sharing three-dimensional model data across different software, ensuring compatibility and precision in manufacturing. They retain detailed geometric and material information, aiding in design, production, and quality control, making them essential in modern engineering and manufacturing.	• FreeCAD: • Autodesk Fusion 360 • Online Viewers
.obj	It's like a digital recipe that describes how a 3D object should look. In this case, it's used to show the mesh data of B-Rep faces. Each face is turned into a bunch of triangles, and the .obj file neatly organizes these triangles.	• Windows 3D Viewer • Online 3D Viewers • Blender

B. Fusion 360 JSON Files:

Fusion 360 JSON files serve as a detailed documentation tool for mechanical components, outlining their construction through sketch and extrude operations. The metadata section of each file includes essential information, such as the 'component_name' specifying the component type (e.g., 'Ring,' 'Tube,' or 'Hex Nut'). The "parent_project" field, aligning with Fusion 360's assembly structure, but will be excluded as the focus is solely on individual components without assemblies.

Within the 'entities' structure, 'Sketch Entities' and 'Extrude Entities' are distinguished. For sketch entities, various top-level data structures detail aspects like points, curves, constraints,

and transformation properties, offering a glimpse into the sketch's composition as shown in Table 5. While this provides a high-level overview, developers can refer to the Fusion API documentation [12] for low-level insights into sketch entity operations.

Extrude entities introduce key data structures like "profiles" and "operation," defining the type of extrusion feature as given in Table 5. Additional structures such as "start_extent" and "extent_type" detail the starting extent and type of extrusion as shown in Table 5. Arrays like "extrude_bodies" and "extrude_faces" capture information about resulting bodies and faces from the extrusion process. Collectively, these structures provide a comprehensive understanding of the three-dimensional geometry resulting from the extrusion of sketches, empowering users to manipulate and comprehend the intricacies of mechanical components in Fusion 360.

Table 5: Operations of Sketch and Extrude Entities

Sketch	***Extrude***

"Sketch1": {
"name": "Sketch1",
"type": "Sketch",
"points": { },
"curves": { },
"constraints": { },
"profiles": { },
"transform": { },
"reference_plane": { }
}
	"Extrude1": {
	"name": "Extrude1",
	"type": "ExtrudeFeature",
	"profiles": [ ],
	"operation":
	"NewBodyFeatureOperation",
	"start_extent": { },
	"extent_type":
	"OneSideFeatureExtentType",
	"extent_one": { },
	"faces": { },
	"bodies": { },
	"extrude_bodies": [ ],
	"extrude_faces": [ ],
	"extrude_side_faces": [ ],
	"extrude_end_faces": [ ],
	"extrude_start_faces": [ ]
	}

C. Fusion 360 GYM:

The Fusion 360 Gym is an interactive and innovative environment associated with Autodesk's Fusion 360 CAD software. It acts as a simulation space where users can communicate with the software using language commands related to sketching and extruding. In the Fusion 360 Gym setup, there's a server component that plays a crucial role in processing commands and facilitating

interactions between the Fusion 360 software and an external client. This server operates locally within the Fusion 360 application, making it capable of handling commands without relying on external servers.

The server is loaded within the Fusion 360 environment, essentially acting as a behind-the- scenes operator. When connected to an external client, it becomes responsive to commands issued by the client. For instance, a command like 'reconstruct (json file)' can be sent from the client to the server. This command instructs the server to reconstruct a 3D model based on the information provided in a JSON file. The JSON file likely contains details about the design, and the server processes this information to generate the final 3D model within the Fusion 360 software. Figure 3 shows an example of a model displayed using the environment.

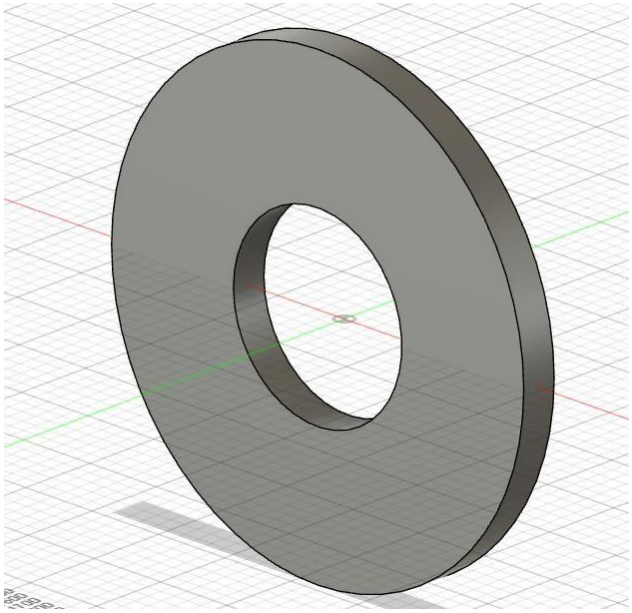

Figure 3: Model Showcase in Fusion 360.

2.1.4 Gradio

Gradio is an open-source Python library that allows developers to quickly create interactive user interfaces (UIs) for machine learning models and other computational functions. With Gradio, you can wrap functions (such as a machine learning model’s prediction function) with a web-based interface, which can then be shared and interacted with via a web browser. This is particularly useful for testing, demonstrating, and deploying models [13].

2.2 Literature Review

The produced dataset in [14] is the one referenced in this project. The paper addresses the need for real-world human-designed datasets of ground-truth Computer-Aided Design (CAD programs, essential for advancing learning-based approaches in CAD. It emphasizes the potential of neural networks in solving CAD-related problems, including 3D shape generation and the recovery of CAD programs. However, progress has been hindered by the absence of a dataset with human-designed CAD geometries and corresponding ground-truth CAD programs. To fill this gap, the authors introduce the Fusion 360 Gallery reconstruction dataset, comprising 8,625 CAD programs represented in a simple and expressive Domain Specific Language (DSL). The DSL involves sketch and extrude operations, along with Boolean operations, enabling the creation of a diverse range of 3D designs. The paper also introduces the Fusion 360 Gym, an interactive environment capable of interpreting the language of sketch and extrude, simulating the iterative construction process of a human designer. As a use case, the paper standardizes the problem of programmatic CAD reconstruction from a target geometry. It provides a benchmark with training and test sets and outlines evaluation criteria. The proposed approach involves training a policy, a message-passing network (MPN), through imitation learning on ground truth construction sequences. During inference, the algorithm utilizes a neural-guided search in the Fusion 360 Gym environment to discover correct CAD programs. Geometry data is presented in multiple formats, including Boundary Representation (.smt and .step files) and Mesh (.obj format). The SMT files represent ground truth geometry, and the STEP format serves as an alternate B-Rep file format. The files are organized in a single directory, following a specific naming convention to represent projects, files, components, and extrude indices. Design complexity is a key consideration, with the dataset scoped to provide a baseline for CAD reconstruction learning approaches. The modeling operations are restricted to sketching and extruding, narrowing the design space. The distribution of designs indicates that the majority have a single body, and designs commonly feature 5-10 faces. The construction sequence, defined as the series of sketch and extrude operations leading to the final geometry, is provided in JSON format. The dataset's parametric history allows the extraction of relationships, including the order of modeling operations, types of geometry created, and valuable topology information from the sequence of B-Rep models. The feature diversity in the dataset enables the exploration of various learning representations and architectures for comparison.

The work in [15] addresses the challenge of automating the generation of training data for sophisticated robotic tasks within the context of Agile Manufacturing or Production. It proposes the use

of parametric CAD models and simulation techniques to overcome the impracticalities of obtaining large amounts of real-world data. The primary objectives include the analysis of script-based parametric modelers, the generation of parametric CAD models for a gear part, the integration of these models into a data generation pipeline, and the evaluation of pose estimation accuracies for the generated parametric gear-set. [15] performs analysis of script-based parametric CAD modelers. The analysis was done on the modelers OpenSCAD, FreeCAD, CadQuery, PythonOCC, ImplicitCAD, and OpenJSCAD. Table 6 provided below presents a summary of various characteristics, including the capability to export standard parametric files, the type of 3D modeling interface, the programming language, and the learning curve for seven distinct parametric CAD modelers. The analysis led to the choice of FreeCAD for 3D modeling and automation of parametric gears due to its Python scripting and graphical interface. Literature studies in the referenced paper favored voting-based methods for pose estimation, leading to the implementation of the PVN3D method for accurate gear pose estimation. Implemented FreeCAD scripting to automate the creation of custom involute gear models. Utilized Python scripting in FreeCAD to autonomously generate 8 different involute gears by adjusting various parameters, achieving the goal of parts customization. Achieved by exporting parametric models as STEP and mesh files, addressing gazebo simulator limitations for STEP files by exporting as Collada. The data generation pipeline produced over 2000 samples, offering an efficient alternative to real object data generation. The final step involved evaluating 6-DoF pose-estimation accuracies of the 8-gear models using the PVN3D method. Results in Table 6 indicate that pose estimation with 8 key points outperforms higher key point estimations, aligning with the optimal choice suggested by the author for network learning efficiency.

Table 6: Characteristics comparison of some CAD modelers

Parametric CAD Tool	***STEP Export***	***3D Modelling Interface***	***Programming Language***	***Learning Curve***
OpenJSCAD	No	Script-based	JavaScript	High
Implicit CAD	No	Script-based	OpenSCAD language interpreter	High
FreeCAD	Yes	GUI + Script Based	Python	Medium
PythonOCC	Yes	Script-based	Python	High
OpenSCAD	No	Script-based	Functional language	Easy
BRL-CAD	Yes	Script-based	Embedded	Very High

CadQuery v1.2	Yes	Script-based	Python	High
CadQuery v2.0	Yes	Script-based	Python	High

The work in [16] introduces a solution for enhancing footrest design customization in the wheelchair industry. The authors emphasize the challenges of wheelchair design, particularly in accommodating various anthropometries and disabilities. The central focus is on leveraging 3D, AI, and computer vision to streamline and improve the customization process. The paper starts by highlighting the significance of 3D printing in Industry 4.0, with a specific focus on wheelchairs. It emphasizes the need for customization in footrest design, especially for patients lacking control over their lower limbs. The proposed methodology integrates computer vision and AI to define footrest customization parameters. The design, based on specifications from orthopedic technicians, includes a base fixed to standard footrests with vertical walls conforming to the shape of the shoe. The paper utilizes CadQuery, in conjunction with AI to streamline and enhance footrest design customization in the wheelchair industry to standard footrests with vertical walls conforming to the shape of the shoe. CadQuery is employed as a scripting modeling language. Specifically, it serves as the core for designing the footrest component. The scripting modeling language in CadQuery enables the generation of the footrest component as an output of a constraint-driven function. The script, running in the environment of the CadQuery library, takes anthropometric measurements as input and generates a watertight 3D model of the footrest component. The footrest design is algorithmically created, providing a personalized design according to the received input dimensions. The AI model is integrated into the workflow to support the footrest design process. Computer vision and machine learning techniques are employed to extract and rectify anthropometric measurements. Here's a breakdown of how AI is utilized:

Computer Vision Component: The first step involves drawing the contours of a patient's shoe on an A4 sheet, which is then photographed. Computer vision is used to extract dimensions, including shoe length, top shoe width, and bottom shoe width. OpenCV is employed for this purpose, and the dimensions extracted serve as input for the subsequent machine-learning model. Machine Learning Component: The dimensions obtained from the computer vision component are fed into a machine learning algorithm. The machine learning model, a Multilayer Perceptron with one hidden layer, is designed to rectify any biases or errors present in the computer vision measurements. Three separate models are trained for shoe length, top width, and bottom width. The goal is to improve the accuracy of anthropometric measurements and enhance the overall design customization process. In summary, the

workflow involves extracting measurements using a combination of computer vision and machine learning. The measurements obtained from the machine learning model, which rectifies biases and errors from the computer vision component, are then utilized as input parameters for the CadQuery scripting modeling language. CadQuery is utilized as the scripting modeling language for the parametric design of the footrest component, ensuring a constraint-driven and automated design process. The integration of AI, involving computer vision and machine learning, enhances the accuracy of anthropometric measurements, contributing to the creation of personalized and well-fitted footrest designs within the wheelchair customization workflow.

The authors of [17] presents a machine learning model designed to automate the generation of highly structured 2D sketches, a challenging aspect of Computer-Aided Design (CAD) models. The objective is to aid engineers in creating designs more efficiently, harnessing the flexibility and power of deep learning, particularly the Transformer architecture. The model combines general-purpose language modeling with a data serialization protocol, showcasing adaptability to the complex CAD domain. The method involves describing structured objects using Protocol Buffers and explores techniques inspired by recent advances in language modeling to capture distributions of serialized objects. The paper also highlights the initial step of converting JSON-format sketches from the Onshape platform into Protocol Buffers, maintaining original structures for domain agnosticism. A dataset of over 4.7 million preprocessed parametric CAD sketches is introduced, representing a significant scale-up in terms of training data and model capacity compared to existing literature. The proposed approach demonstrates promising results for both unconditional synthesis and image-to-sketch translation, making a substantial contribution to the field of machine-generated CAD sketches.

The model's core, a combination of a Transformer and Pointer Net, is guided by an interpreter that decides which field of the Protocol buffer message corresponds to the generated value at each step. The authors leverage the analogy between sketch construction and natural language modeling, where selecting the next constraint or entity in a sketch resembles generating the next word in a sentence. The contributions of the paper include the development of a method for describing structured objects using Protocol Buffers, techniques for capturing distributions of objects, and the collection of a large dataset of parametric CAD sketches for model validation.

The work in [18] provides a slightly similar work. The research focuses on extracting attribute information from household data and providing recommendations for missing attributes. The

researchers developed a sample framework for generating 3D models with substantial descriptive capacity and minimal erroneous information, focusing on six furniture classifications (Beds, Chairs, Dressers, Tables, Lamps, and Sofas). Leveraging interactive assistance based on LLM, the model generates illustrative sentences to help accurately describe objects. The image generation model is trained using a household dataset, eliminating background elements to ensure that generated objects align with descriptions. The model utilizes the Controlnet module to modify intermediate images during the 3D generation stage, allowing users to regenerate images through additional description text prompts. The methodology encompasses a text-to-3D procedure involving text processing, text-to-image, and image-to-3D. The text processing phase incorporates LLM-guided prompt recommendation and a user-centric semi-automated system with user intervention. It leverages Named Entity Recognition (NER) to detect missing tags in sentences, and the language model is fine-tuned through instruction fine-tuning techniques. The diffusion methods involve a two-stage coarse-to-fine framework called Chat3D, employing SDS Loss for Score Distillation in image generation. The 3D reconstruction phase utilizes InstantNGP with SDS loss for NeRF optimization.

The results showcase the development of a sample framework for 3D models with descriptive capacity and minimal erroneous information, specifically targeting furniture categories and styles. The model's objective can be altered based on the requirements of the developer. The interactive modification of generated images enhances the user experience. In conclusion, the paper proposes an integrated approach for user-friendly 3D model generation, demonstrating its effectiveness in furniture category synthesis.

Chapter 3 System Design

The overall structure of this chapter is described in the following. Design requirements are discussed in section 3.1 and analyzed in section 3.2. Realistic constraints are discussed in section 3.3. The different design approaches that were considered for the development of the system are discussed in section 3.4. Overall description of the system design, an in-depth description of the subsystem design, and the methodologies of implementing each subsystem to achieve its purpose are thoroughly described in the following sections.

3.1 Design Requirements

As aforementioned in Chapter 1, the design requirements are as follows:

- The system shall include the development of a user-friendly interface for users to input their design requirements for a mechanical part using natural language commands and prompts.
- The system shall exploit LLM’s capabilities to analyze and understand the user's description and requirements.
- The work shall include the generation of a dataset of at least 3 mechanical parts.
- Each mechanical part within the dataset shall have at least 100 samples.
- The pre-trained LLM shall be fine-tuned using the generated dataset.
- The system shall generate a CadQuery code based on the user's input.
- The system shall be able to convert the CadQuery code to a file that is readable by CAD viewers.
- The system shall be able to display the mechanical part to the user.
- The system shall be able to determine the mechanical part according to the user’s input with an accuracy of at least 70%.

3.2 Analysis of Design Requirements

The following is a detailed analysis of the design requirements:

- The system shall include the development of a user-friendly interface for users to input their design requirements for a mechanical part using natural language commands and prompts, the interface has to be easy to use.
- The system shall exploit LLM's capabilities to analyze and understand the user's description and requirements specifically focusing on extracting the necessary details such as the required mechanical shape and important parameters of the mechanical part.
- The work shall include the generation of a dataset of at least 3 mechanical parts, specifically tubes, rings, and hex nuts. This dataset is formed through a combination of user prompts explicitly requesting these mechanical parts, along with their specific parameters, and the corresponding generation of JSON files capturing the detailed information for each part.
- Each mechanical part within the dataset shall have at least 100 samples providing the model with enough varied samples to learn how to accurately extract and represent the important information.
- The pre-trained LLM shall be fine-tuned using the generated dataset to adapt and specialize its capabilities for this task, making it well-suited to this project' specific requirements in mechanical parts parameters extraction.
- The system shall generate a CadQuery code based on the user's input. This code is formed by converting information from what the user requests into a JSON file, which is then used to generate the CadQuery code.
- The system shall be able to convert the CadQuery code to a file, such as STL, that is readable by CAD viewers, ensuring seamless integration with CAD editors and viewers for comprehensive visualization and editing capabilities.
- The system shall be able to display the mechanical part to the user.
- The system shall be able to determine the mechanical part according to the user's input with an accuracy of at least 70%. The accuracy will be assessed by comparing the system's predictions with the actual outcomes, utilizing metrics such as precision, recall, and F1-score to comprehensively evaluate the model's performance.

3.3 Realistic Constraints

The realistic constraints that should be taken into consideration in the design process are:

- **Economic Constraints:** The cost shall not exceed 400JDs. This cost will be for using Colab pro which is a subscription plan for Google's Collaboratory platform. It offers advanced GPU and TPU resources for faster model training.
- **Manufacturability & Sustainability Constraints:** Challenges in training Large Language Models (LLMs) for mechanical parts recognition include limited access to diverse datasets featuring numerous samples of various components. Additionally, the computational complexity involved in fine-tuning pre-trained LLMs poses a significant obstacle. Moreover, keeping up with the latest versions of Large Language Models (LLMs) is a constant challenge. As these models evolve with new features and structures, it becomes necessary for practitioners to adjust their existing methods.
- **Ethical:** Potential privacy concerns and signing Istari Digital NDA. In the context of potential privacy concerns, ensuring the confidentiality of the project idea is essential. By signing Istari Digital's Non-Disclosure Agreement (NDA), all parties commit to preserving the integrity of the project concept.

3.4 Different Design Approaches

Various approaches were explored in this project, focusing on mechanical parts, dataset preparation, and the utilization of large language models (LLMs).

When it comes to 3D modeling datasets, there are several options to consider, like Fusion 360, ShapeNet, and ABC: A Big CAD Model Dataset for Geometric Deep Learning. These datasets are filled with digital models that people use for various purposes. Typically, they contain files in a format called STL, which are like blueprints for 3D shapes. However, Fusion 360 stands out because it's the only dataset that also provides JSON files. These JSON files contain extra information that could be useful for this project. Having this additional data from Fusion 360 gives an advantage and helps support the project in a unique way.

The Fusion360 dataset contains a wide collection of shapes. The shapes found in the dataset can be either assembled shapes, meaning more than one basic part joined together. The rest of the shapes in the dataset are simpler shapes. Due to the requirements of this project, the shapes selected

are basic shapes since assembled shapes can be too complex and require 3D specialists to decipher.

In the world of language models, there are many open source models available, each with different sizes and abilities. For example, Falcon comes in different versions, ranging from small (7 billion parameters) to very large (180 billion parameters). Llama also has variations with 7 billion and 70 billion parameters, and Mistral 7b. However, we're choosing to focus on models with 7 billion parameters because our current devices can handle these models well. Trying to use larger models would be difficult for our devices and might not work as smoothly. So, by concentrating on models with 7 billion parameters, we can make sure we use our resources effectively and avoid any problems with performance.

3.5 Project System Integration

The primary goal of the system is to generate a JSON file that represents the 3D model of a shape requested by the user. The structure of the JSON file which contains the necessary steps to structure the 3D model is fixed. However, the dimensions such as side length, area, perimeter, radius, and diameters, vary according to the user's request and they define the specific characteristics of the shape.

The process begins with computing essential dimensions based on the user prompt. The extracted parameters are then included in a function call “Generate_file”, designed to populate a predefined JSON template for the shape. This function call fills the relevant parameters into the JSON file.

The Language Model (LLM) plays a crucial role in this process by identifying and extracting the necessary dimensions and features from the user's prompt, understanding the relationships between these dimensions and other relevant features such as area and perimeter, and computing features needed to complete the JSON file, even if they are not explicitly provided by the user. By accurately extracting and predicting these required values, the LLM ensures that the JSON file is correctly generated, facilitating the creation of the 3D model.

The next step in the process depends on the user's requirements. If the user needs to modify the CAD model and add other components to build an assembly using CAD modeling, they can select the first option as give in Figure 4. This option allows them to open the JSON file in Fusion 360 using Fusion GYM and make the necessary modifications to the model. Alternatively, if the user prefers to model the design using code, they can choose the second option. This option provides a conversion of the JSON file to CadQuery code. By installing the CadQuery library, the user can use the converted code to modify

the model programmatically. Finally, if the user needs the model as an STL file for 3D printing, they can select the third option to get the exported STL file. This flexibility ensures that users can choose the most suitable method based on their specific needs and preferences.

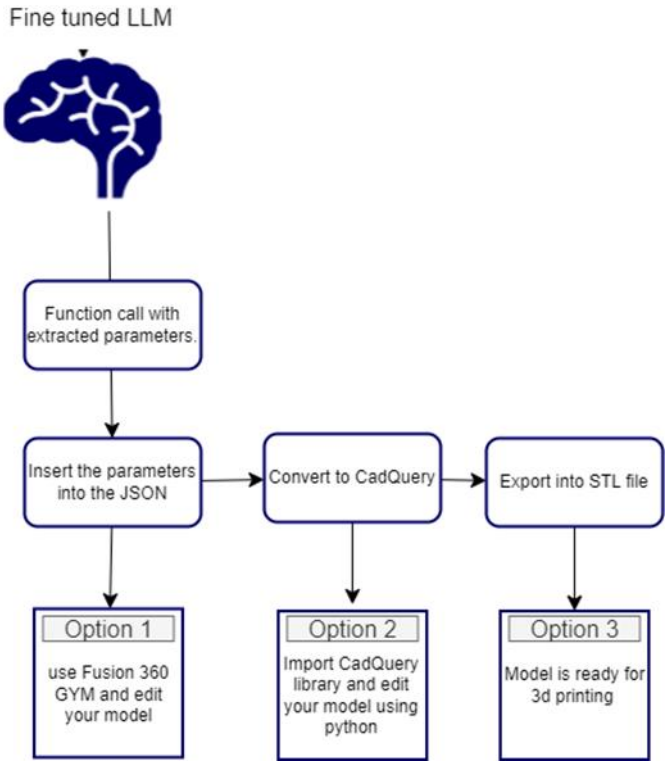

Figure 4: System Integration.

3.6 Proposed System

This section explored the factors to be considered when finetuning this project that uses LLM technology and gives an overall description of the proposed system. The project includes choosing Mechanical Components, Dataset Creation including Prompt generation, Pre-processing, Parameter Extraction, and Equation Computation. Additionally, we focused on the conversion of JSON files to CadQuery code.

3.6.1 Mechanical Components

We chose five simple parts as the primary components to demonstrate that LLMs can proficiently generate JSON files for these simple parts. This provides evidence that the model's capabilities extend to more complex shapes due to its familiarity with the basic components. Additionally, this decision is reinforced by the constraint of the referenced dataset [9], containing a finite number of labeled shapes that we can choose. The main five components that we will deal with in this project are flat O-rings,

tubes, hex jam nuts, Mild Square bars, and Leaf Spring Shackle Link.

A. Rings

Flat O-rings, also known as squared rings, are circular in shape with flat sides referred to as a toroid. These unique O-rings stand out due to their square or rectangular cross-sections, and their distinctive design is achieved through a specialized manufacturing process as can be seen in Figure 4. These O-rings are typically created by either cutting them from extruded tubes. In the manufacturing process, precision is key to shaping and sizing, ensuring that the O-rings meet specific dimensional requirements for their intended applications. Common materials used for crafting flat O-rings include elastomers such as nitrile rubber (NBR), silicone, and fluorocarbon (Viton), each offering unique properties like flexibility, heat resistance, and chemical compatibility, catering to diverse industrial needs. Flat O-rings are employed in a variety of industries including transportation. Flat O-rings are essential for sealing the various fluids that are a component of the systems in buses, trucks, and automobiles. Also, flat O-rings play a crucial role in the design of airplanes because they shield jet engines from dangerous circumstances and large temperature swings [19]. The main three parameters used for describing a flat O-ring are inner diameter (ID), outer diameter (OD), and the height(H) as shown in Figure 5. Based on the Standard USA Square-Rings Sizes [20], the range of values for ID in mm varies between (2.90-658.88). OD values are (6.26-672.34) and the height has values of (1.68-6.73).

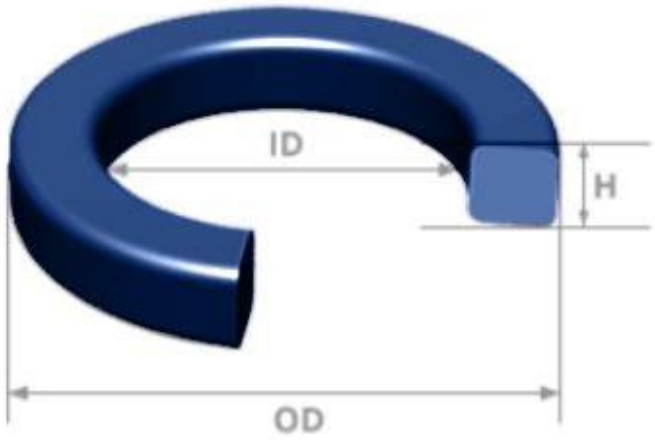

Figure 5: Ring Dimensions.

B. Tubes

A tube such as the one shown in Figure 6 below is a cylindrical hollow structure that plays a crucial role in various applications. It is characterized by its elongated shape, with a uniform hollow space inside. Tubes are commonly made from materials like steel, aluminum, or plastic, and they come in different sizes and thicknesses to suit diverse needs. Their simple yet effective design makes them essential components in a wide range of everyday applications, whether used in plumbing systems, structural frameworks, or industrial machinery. These tubes come with specifications that further enhance their utility. The outer diameter (OD) ranges from 10.2 mm to 139.7 mm, offering flexibility to cater to different applications. The wall thickness (WT) varies from 0.5 mm to 16.0 mm, providing options for strength and structural requirements. Tubes are typically provided in standard production lengths, ranging from 5 to 6 meters. However, a specific and precise length can be supplied upon request [21].

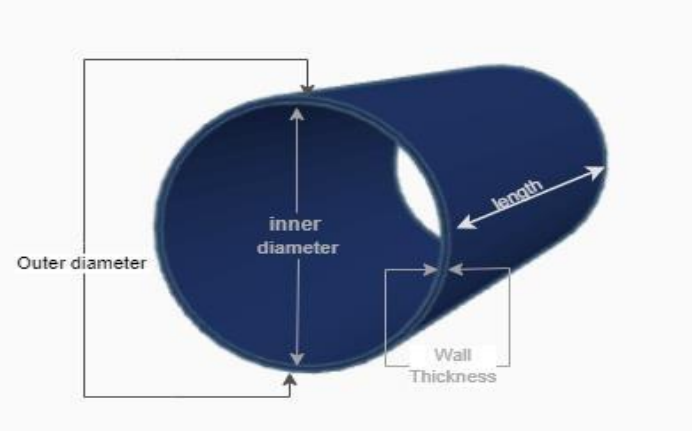

Figure 6 : Tube Dimensions.

C. Hex Jam Nuts

Hex jam nuts are a type of nut that has a hexagonal (six-sided) shape and is specifically designed to be thin in profile. Unlike regular hex nuts, hex jam nuts have a reduced thickness, making them suitable for applications where space is limited. The primary function of a hex nut jam is to prevent the loosening of a standard nut, they are found frequently used in situations where there is a need for locking arrangements, especially in scenarios where vibration or motion could lead to the loosening of nuts [22]. Hex jam nuts have three key dimensions that determine their compatibility and usage. The width across flats (F) is the distance between opposite flat surfaces, crucial for selecting the right-sized wrench or socket. The width across corners (G) is the distance from one corner to the opposite corner, often used in engineering specifications. The diameter represents the distance across the center of the hex nut, essential for ensuring a proper fit with bolts. Additionally, the height (H), often referred to as the thickness or the distance from the bottom to the top of the nut, is a critical dimension influencing the engagement of threads and the overall fit in an assembly. Based on ASME B18.2.2, which stands to a standard specification published by the American Society of Mechanical Engineers (ASME) that covers the dimensions and specifications for hex nuts [21],the dimension of a hexagonal shape should be carefully outlined as follows: The width between flats ranges from 10.87 mm to 114.30 mm, the width across corners, measured from one corner to the opposite, varies from 12.40 mm to 131.98 mm and the nut's height, extending from bottom to top, falls between 5.38 mm and 67.41 mm. Figure 7 shows a representation of the Hex Nut.

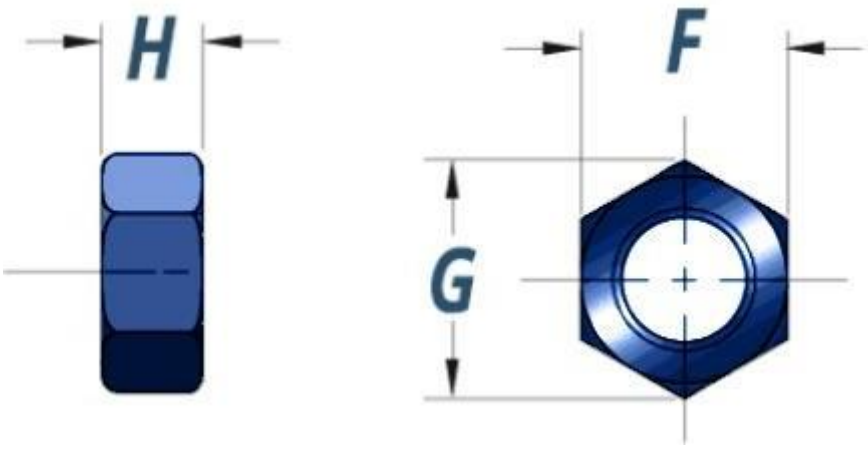

Figure 7: Hex Nut Dimensions.

D. Mild Square Bar

Mild square bars as shown in Figure 8, typically refer to square-shaped bars made from mild steel. Mild square bars, commonly made from mild steel, serve essential functions across various industries. In construction, they provide vital support to building frameworks, ensuring stability. Additionally, they are utilized in manufacturing for their durability and ease of shaping. Metalworkers use them to craft gates, fences, and other structures, while engineers rely on them to reinforce buildings and machinery. In the automotive sector, they contribute to the strength of chassis and suspension components. On farms, they aid in the construction of agricultural equipment and fencing. For DIY enthusiasts, they offer opportunities for creating furniture and sculptures. Crucially, mild square bars play a significant role in infrastructure projects such as bridges and railings, as well as in utility applications like power lines and telecommunications networks. Their strength, versatility, and affordability make them indispensable in various endeavors. Mild Square bars have 2 main attributes, the side length (S) and Height (H) as shown in Figure 8 [23].

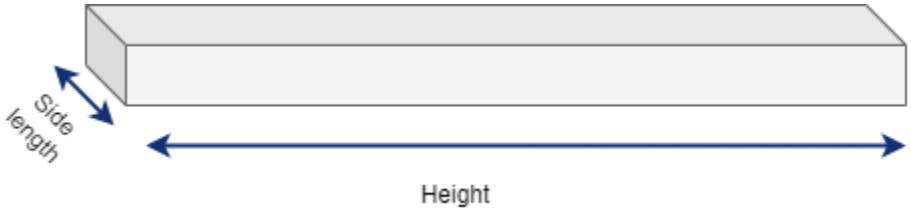

Figure 8: Mild Square Bar Dimensions.

E. Leaf Spring Shackle Link

Leaf spring shackle links as shown in Figure 9, are components of leaf spring suspension systems commonly used in vehicles such as trucks, trailers, and some SUVs. They are crucial for allowing the leaf springs to flex and absorb shocks while the vehicle is in motion. Shackle links connect the leaf springs to the chassis of the vehicle, providing a pivot point that allows the springs to move up and down as the vehicle encounters bumps and uneven terrain. Shackle links typically consist of a metal bracket or hanger attached to the chassis at one end, and a shackle at the other end that connects to the eye of the leaf spring. The shackle allows for movement as the leaf spring flexes, providing a degree of freedom that helps to improve riding comfort and stability. The three main attributes of the shackle link are the radius of the two circles, the length between the centers of the two circles and the height of the Link. Figure 9 clarifies where on the shape these attributes can be found [24].

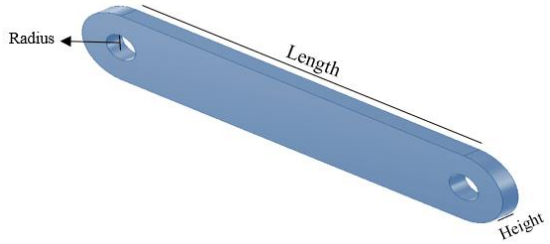

Figure 9: Leaf Spring Shackle Link Dimensions.

3.6.2 Dataset Creation

The dataset created for this project comprises two main components: a user prompt and its corresponding function call, 'Generate_file'. Each user prompt serves as an input query or instruction for creating a shape, while the associated function call includes parameters relevant to the user's input. Figure 10 illustrates an example of the dataset. Besides the dimensions specified in the user prompt, the function call includes additional parameters such as the area and perimeter of the main profile for the shape, along with the name of the requested shape. These additional parameters are not directly provided by the user prompt but are generated as part of the dataset output. The names of the areas and perimeters are concatenated with the shape's name to help models establish more precise relationships for each shape.

The dataset comprises five distinct shapes: Rings, Tubes, Hex Nuts, Mild Square Bars, and Shackle Linkages. The distribution of shapes is visually represented in Figure 11. This allocation across different shapes ensures that the model receives ample and diverse examples from each category. Although the dataset is not evenly distributed, it captures various combinations of parameters for each shape, with each combination assigned the same number of instances. This methodology guarantees a comprehensive representation of parameter variations, promoting the model's ability to learn and generalize effectively. We initially generated 6,000 instances and tested them on a dummy model. We then incrementally increased the number of instances until the model's loss stabilized. Ultimately, we finalized the dataset at 16,695 instances, achieving the desired stability in model performance.

Figure 10: Model Prediction Format.

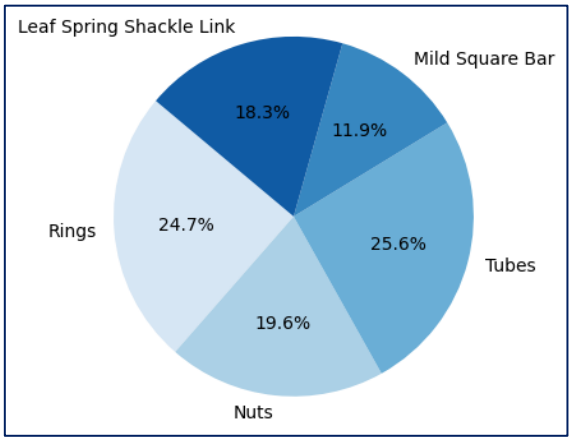

Figure 11: Dataset Distribution.

A. Prompt Generation

By using ChatGPT, we can easily create lots of different user prompts with different ways of saying things using different styles. This saves time because now it is not required to come up with all prompts manually. ChatGPT helps to generate diverse inputs for the model, making it better at understanding various ways people might ask for things. This makes the model more flexible and readier to handle different types of user questions.

B. Parameter Extraction and Equation Computation

To generate additional instances with diverse values for each file, a systematic analysis of the values provided within the reference files was conducted. During this analysis, equations corresponding to each value in the reference file were derived. These equations serve as functional representations of the relationships observed within the original dataset. By using these equations, new instances with different values can be generated, allowing for the expansion of the dataset.

- Rings and Tubes Equations

To construct a ring, we start by initiating a new sketch. Select the center point of (0,0,0). For the design and specify the outer circle's radius. Draw the outer circle accordingly. Subsequently, within the same sketch, create a concentric inner circle, defining its radius or diameter. In terms of dimensions, use the appropriate notation, such as D for diameter and R for radius, to specify the dimensions of both the inner and outer circles.

$$D_{outer} = R_{outer} * 2 \quad (1)$$

$$D_{inner} = R_{inner} * 2 \quad (2)$$

$$A1 = \pi * {R_{inner}}^2 \quad (3)$$

$$A2 = \pi * {R_{outer}}^2 - \pi * {R_{inner}}^2 \quad (4)$$

$$P1 = 2 * \pi * Rinner2 \quad (5)$$

$$P2 = 2 * \pi * Router2 \quad (6)$$

Where A1 refers to the area of the inner circle and A2 refers to the area of two concentric circles. Whereas P1 and P2 refer to the perimeter of the inner circle and perimeter of outer circle respectively. Following the sketch operation, the extrusion distance is defined to determine the height or thickness of the resulting sketch. The value of the extrude specifies if the generated components is a ring or a tube, where rings have relatively small thickness values compared to tubes.

- Hex Nuts Equations

Constructing a hexagonal nut involves several steps. Begin by defining the center point, often designated as (0,0,0) in the three-dimensional space. Next, establish the start and end points for each side of the hexagon. These points will form the vertices of the hexagon, creating the outline of the nut as shown in Figure 12. The starting point of the first line is designed to be as the equations below. These equations represent (x, y) values of each point P.

$$P_1: (X_1, Y_1, Z_1) = (side\ length, 0,0) \quad (7)$$

$$P_2: (X_2, Y_2, Z_2) = (side\ length * \cos(300), side\ length * \sin(300), 0) \quad (8)$$

$$P_3: (X_3, Y_3, Z_3) = (side\ length * \cos(240), side\ length * \sin(240), 0) \quad (9)$$

$$P_4: (X_4, Y_4, Z_4) = (side\ length * -1, 0, 0) \quad (10)$$

$$P_5: (X_5, Y_5, Z_5) = (side\ length * \cos(120), side\ length * \sin(120), 0) \quad (11)$$

$$P_6: (X_6, Y_6, Z_6) = (side\ length * \cos(60), side\ length * \sin(60), 0) \quad (12)$$

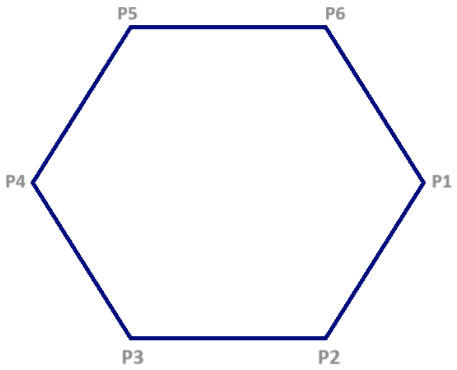

Figure 12: Hexagonal Point Distribution.

After sketching the lines to form the hexagon and the concentric circle with diameter D, the next step involves calculating properties such as area (A) and perimeter (P) as give in the following equations.

$$D_{circle} = R_{circle} * 2 \quad (13)$$

$$D_{hexagon} = side\ length * 2 \quad (14)$$

$$A_{hexagon} = \frac{\sqrt{3}}{2} * Side\ length^2 \quad (15)$$

$$A_{circle} = \pi * R_{circle}^{\ 2} \quad (16)$$

$$A_{difference} = A_{hexagon} - A_{circle} \quad (17)$$

$$P_{circle} = \pi * R_{circle}^{\ 2} \quad (18)$$

- Mild Square Bar Equation

To construct a mild square bar, we begin at the origin point (0,0,0) and proceed to create a square with a specified side length provided by the user. The square's profile is defined in the xy-plane, with vertices at (0,0,0), (side_length,0,0), (side_length, side_length,0), and (0, side_length,0). This forms a planar shape with an area denoted as A square and a perimeter denoted as P square. Next, this square profile is extruded along the z-axis by a height value also specified by the user. This extrusion results in a three-dimensional bar with a square cross-section.

$$A_{Square} = Side\_Length^2 \quad (19)$$

$$A_{Square} = 4 * Side\ Length \quad (20)$$

- Leaf Spring Shackle Link Equation

To construct a shackle link, begin by plotting the main points that define the model's geometry. Figure 13 below shows the display of the points that express the rectangular part of the shackle link. Using these points as an outline to construct the whole shape, the first step is identifying the center points P1 and P2 of the circles that will form the rounded ends of the shackle link. Next, plot the outer profile points P3, P4, P5, and P6 that outline the shape of the shackle link. With the main points identified, draw circles centered at P1 and P2 with the appropriate radius to represent the circular ends of the shackle link. Connect the plotted points using straight lines to define the sides of the shackle link. Draw a line segment from P3 to P4 and another from P5 to P6, establishing the straight edges that will connect the rounded ends.

Finally, construct circular arcs between specific points to complete the rounded ends of the shackle link. Draw an arc from P4 to P5 along the circle centered at P2, and another arc from P6 to P3 along the circle centered at P1.

$$P_1: (X_1, Y_1, Z_1) = (0\ , (\frac{Link\_Width}{2}), 0) \tag{21}$$

$$P_2: (X_2, Y_2, Z_2) = (\text{Link\_Length}, \frac{Link\_Width}{2}), 0) \tag{22}$$

$$P_3: (X_3, Y_3, Z_3) = (0,0,0) \tag{23}$$

$$P_4: (X_4, Y_4, Z_4) = (\text{Link\_Length}, 0, 0) \tag{24}$$

$$P_5: (X_5, Y_5, Z_5) = (\text{Link\_Length}, \text{Link\_Width}, 0) \tag{25}$$

$$P_6: (X_6, Y_6, Z_6) = (0, \text{Link\_Width}, 0) \tag{26}$$

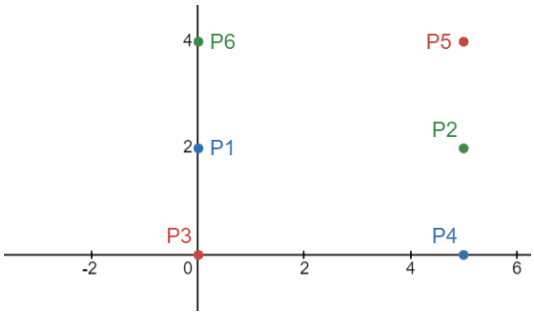

Figure 13: Points Display.

Once the outline of the shackle link's profile is constructed using the defined points, circles, lines, and arcs, the final profile is ready to be extruded to create the three-dimensional shape as Figure 9. The next step involves calculating the area A and premier P of the profiles.

$$A_{inner\_circle} = \pi * joint\_radius^2 \tag{27}$$

$$A_{left\_arc} = \frac{\pi * arc\_radius^2}{2} \tag{28}$$

$$A_{right\_arc} = \frac{\pi * arc\_radius^2}{2} \tag{29}$$

$$A_{two\_arc} = 2 * \pi * arc\_radius \tag{30}$$

$$A_{link} = \left(Link_{Length} * Link_{Width}\right) + A_{left\_arc} + A_{right\_arc} - 2 * A_{inner\_circle} \tag{31}$$

$$P_{link} = \left(Link_{Length} + Link_{Width}\right) + P_{two\_arcs} \tag{32}$$

C. Finalizing the Dataset

In the finalizing process, we initially relied on JSON files to anticipate the equations needed for computing each value. By using these equations, we could generate a bunch of instances with varying and diverse values. This step was crucial for ensuring that the dataset encompassed a wide range of parameters, enabling more comprehensive model training.

Subsequently, integrated the function calls of diverse parameters with their corresponding user prompts, a sample is illustrated in Figure 14. Then we merged all instances in one file. This integration is crucial because most large language models require the dataset to be in a single file format for efficient processing.

For validation purposes, we filled the values generated from the function calls into a JSON format and opened this JSON file in Fusion 360. This validation process allowed us to confirm that the models derived from the dataset functioned well within the Fusion 360 environment, ensuring their practical utility and reliability for design applications.

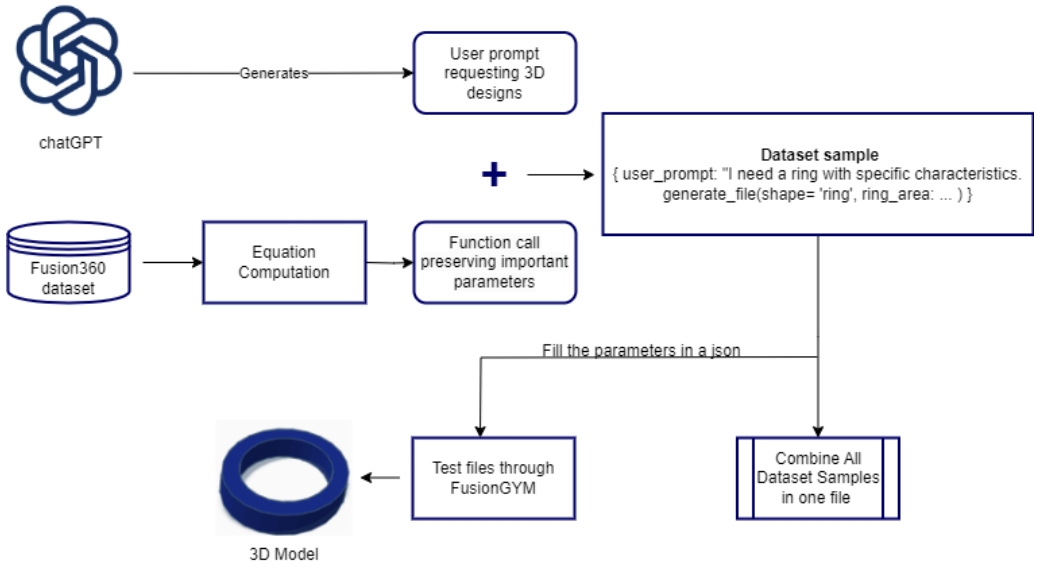

Figure 14: Dataset Creation.

3.6.3 JSON to CadQuery Conversion

After the training of the model, the output gives a function call that is then evaluated to a Function called ‘Generate_file’. This function constructs the JSON file using the extracted parameters from the user prompt. The subsequent step involves mapping the sketches and extrudes operations that are present inside the JSON file to functionalities within the CadQuery library. A code was implemented to map the operations using CadQuery capabilities. Table 7 shows some of the mappings associated with the code. Importantly, after this conversion process, the resulting CAD model code can be exported as an STL and opened in any CAD software.

Table 7: Example on the conversion of JSON information to CadQuery

Operation	*JSON information*	*CadQuery*

Sketch	"entities": {
	"entity1": {
	"points": {
	"origin": {
	"type": "Point3D",
	"x": 0.0,
	"y": 0.0,
	"z": 0.0
	},…}
	"outer_circle": {
	"type": "SketchCircle",
	… ,
	
	"center_point":
	"point1",
	"radius": 8.89
	},...
		# parameters
		origin = (0.0,0.0,0.0)
		radius =8.89
		
		# creating circle
		outer_circle = (cq.Workplane("XY").
		circle(radius). translate(center_point))

Extrude	"entity2": {
	"name": "Extrude1",
	"type":"ExtrudeFeature",
	"profiles": [
	{
	
	"profile":
	"profile1",
	"sketch":
	
	"entity1"
	}
	],…
	
	"extent_one": {
	"distance": {
	"type": "ModelParameter",
	"value": 3.0,
	"role": "AlongDistance"
	},…
		# extruding
		entity2 = entity1.extrude(3.0)

3.6.4 Overall System

Chapters 2 and 3 provide a comprehensive overview of all the components used in the system. The LLMs incorporated are Falcon 7B, Llama 2 7B, and Mistral 7B. Additionally, the usage of JSON files and CadQuery is explained, along with the selection of shapes from the Fusion360 dataset. With all relevant components introduced, the overall system becomes clear. Figure 15 below offers a detailed description of the entire system.

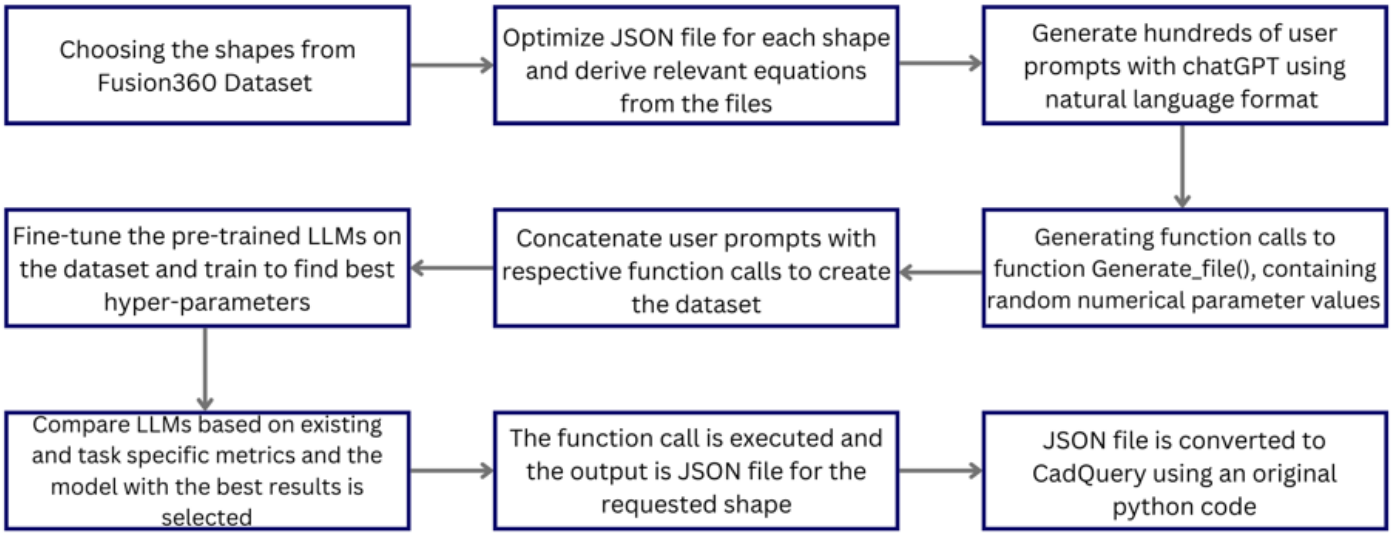

Figure 15: Detailed Description of the Project.

Chapter 4 Implementation

Chapter four, Implementation, focuses on the practical aspects of deploying the Large Language Model architecture, model configuration, and a potential scenario for the project. It includes an explanation of how the model architecture is set up and configured to suit the project's needs. Additionally, a specific scenario is outlined to demonstrate the model's capabilities and effectiveness in addressing real-world challenges.

4.1 Large Language Model Architecture

LLM has different designs designed for different natural language processing tasks, with different features and uses. The encoder-decoder architecture, which consists of independent encoder and decoder components, is one popular architecture. After processing input sequences, the encoder creates a context representation that includes the semantic information included in the input. The decoder receives this context vector after which it uses the encoded context to produce output sequences. Examples of encoder-decoder models are BART, mBART, T5, and Flan-T5. Training these models can be computationally expensive and time-consuming due to the large number of parameters involved [25] [26].

Encoder-only Large Language Models (LLMs), such as OpenAI's GPT, are built with a single component which is the encoder. These models specialize in autoregressive output, which refers to generating each output token sequentially, where each token is predicted based on the model's previous predictions. They're particularly effective for tasks like text generation and language modeling, where the ability to create meaningful and coherent sentences from limited starting information is crucial.

On the other hand, decoder only LLMs, like Falcon, Mistral and Llama2, focus on generating text outputs without needing an explicit input sequence [27]. Instead, they rely on prompts or instructions from users to produce coherent text. Therefore, generating one output at a time refers to predicting a complete sequence or output in a single step, without dependency on previously generated tokens. These models are used for applications requiring interactive text generation based on minimal user input. Given that the task relies on instructed user prompts, this project will work with decoder-only architectures for the LLMs.

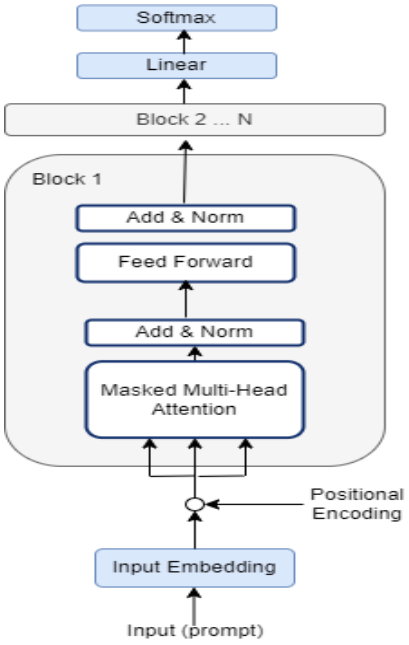

Figure 16: Model Architecture.

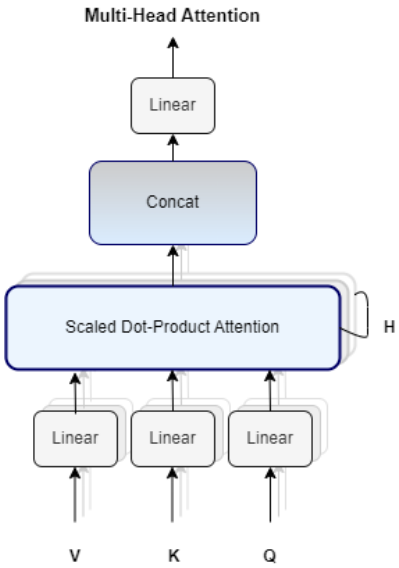

Figure 17: Multi-Head Attention.

The architecture of large language models (LLMs) based on transformers consists of stacked layers of attention and feedforward modules. These models utilize self-attention mechanisms to capture relationships and dependencies across input sequences effectively. Figure 16 above illustrates the architectural flow. To begin, the input text undergoes tokenization, breaking it down into smaller units such as words or sub-words. Each token is then embedded into a continuous vector representation, encoding semantic and syntactic information. Additionally, positional encoding is applied to add unique positional information into the word embeddings for understanding the sequence's structure.

In the Transformer's self-attention block, a key concept is multi-head attention, shown in Figure 17, which allows the model to process different aspects of the input simultaneously. This is done by using multiple sets of learned attention weights to perform several self-attention operations at once. Each token in the input sequence is associated with three vectors: query, key, and value. The query vector represents the token itself and is used to calculate attention scores by comparing it with the key vectors of other tokens. These attention scores determine how much each token should contribute to the final output.

The process involves matrix multiplication between the input embeddings and learned weight matrices (Wq, Wk, Wv) to compute the query, key, and value vectors. During training, these weight matrices are adjusted (or optimized) through backpropagation. By doing this, the model learns to assign

appropriate attention to different parts of the input sequence, capturing relationships and contextual information effectively.

After the self-attention step, each token in the sequence undergoes independent processing through a feed-forward neural network (FFNN). This FFNN consists of multiple fully connected layers with non-linear activation functions like ReLU [28], its role is crucial in enhancing the model's ability to learn complex patterns and relationships within the tokens.

Additionally, layer normalization is applied after each sub-component or layer within the transformer architecture. This normalization technique helps stabilize the learning process and improves the model's ability to generalize across different inputs. Finally, the output layers of the transformer model are linear projections followed by SoftMax activation that is commonly used to generate the probability distribution over the next token as displayed in Figure 18.

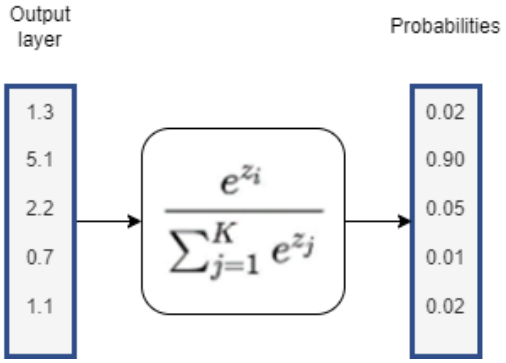

Figure 18: Linear Projection Processing.

4.2 Model Configuration

4.2.1 Quantization

LLMs, with their massive size and complexity, often face challenges related to memory usage, this particularly refers to the format in which weights and activations are stored. Typically, these models represent parameters using 32-bit floating-point numbers, a standard format that allows for precision in numerical calculations. However, storing parameters and weights in a 32-bit format requires significant memory resources, especially for large models with millions or billions of parameters. This reliance on 32-bit precision contributes to the high memory usage observed during both the training and inference stages of LLMs.

Researchers have been looking into ways to tackle the memory problem of LLMs. One approach they've explored is quantization. Essentially, quantization involves converting the model's parameters and weights from their original 32-bit format to lower precision alternatives, like 16-bit floating point or even integer formats such as 8-bit or 4-bit. This conversion helps shrink the model's memory usage, making it more manageable, especially for situations where resources are limited. So, by using quantization LLMs can maintain their performance while avoiding the problem of consuming too much memory.

Table 8 shows the difference between model sizes before and after applying quantization. Quantization is a procedure that can be done through a library called ‘bitsandbytes’ using a function called ‘BitsandBytesConfig’.

Table 8: Models Sizes After Quantization

Model	*Original size (FP 16)*	*Quantized Size (FP4)*
Falcon 7b	15GB	65MB
LLama2 7b	13.5GB	3.9GB
Mistral 7b	14GB	3.5GB

The work in [29] investigated how different levels of precision affect the accuracy of the models when they're set up for immediate use, without fine-tuning. The research found that using 4-bit quantization consistently gave better results across different models, like OPT, Pythia, GPT-2, and BLOOM, which are well-known LLMs. This shows that using 4-bit setups helps in making models smaller while still performing well, creating a balance between memory usage and model efficiency. In Figure 19 below, it is observed that the impact of various quantization levels on the OPT model. The figure demonstrates that 16-bit, 8-bit, and 4-bit quantization resulted in stable lines, while 3-bit quantization showed instability. Given the goal of maximizing mean zero-shot accuracy thus, determined that the 4-bit quantization format was the most suitable choice.

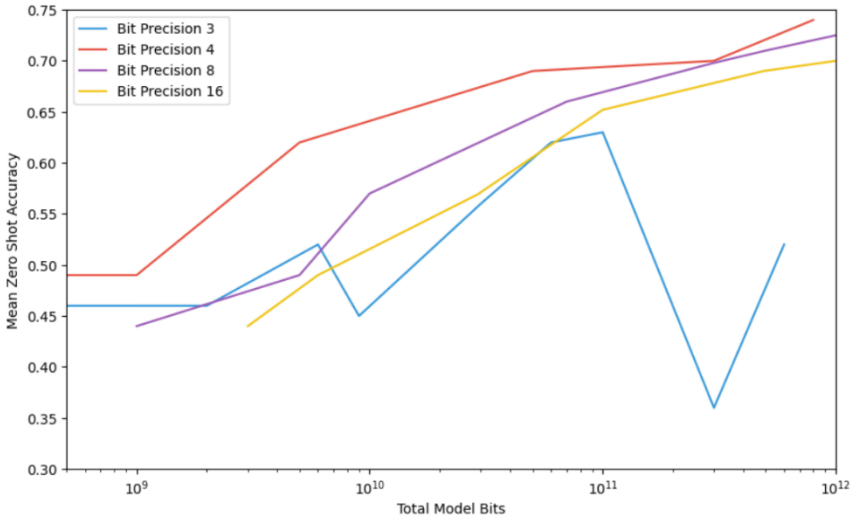

Figure 19: Impact of Various Quantization Levels on the OPT Model.

To study the effect of quantization on the performance of the models,1,200 models were selected [30] based on their HuggingFace score or their evaluation across seven established benchmarks. Next, a performance drop analysis was conducted to determine the impact of quantization on these models. This involved calculating the performance drop ratio for each model across all benchmarks and then computing the average drop across all models and benchmarks. Table 9 shows examples on the performance of a couple of models used during the study. The findings revealed an average performance decline of 12% in quantized models. However, the extent of this drop varied across benchmarks, ranging from as little as a 2% decrease to as much as a 28% reduction as illustrated in Table 9.

Table 9: Average Benchmark Score Drop After Quantization

Original Model	Quantized Model	HF Score	ARC	HellaSwag	MMLU
Mistral/Mixtral-8x7B-v0.1	TheBlock/Mixtral-8x7B-v0.1-GPTQ	0	-0.01	0.01	0.01
Itsliupeng/Mixtral-8x7B-v0.1-top3	TheBlock/Mixtral-8x7B-v0.1-GPTQ	0.05	0.03	0.02	0.04
Open-Orca/Mistral-7B-OpenOcra	Mncai/Mistral-7B-OpenOcra-1k	0.02	0.02	-0.01	0
Igaalvs/mistral-7b-platpus1k	Mncai/Mistral-7B-openplatypus-1k	0	0.02	-0.02	0.05
Mncal/Llama2-7B-gunanaco-1k	Mikael110/llama-2-7b-guanaco-fp16	0.02	0	0.01	0.03
Codellama/CodeLlama-34b-instruct-hf	TheBlock/CodeLlama-34B-instruct-fp16	0.03	0	0	0
	Avg score drop per bench	0.14	0.12	0.16	0.12
	Total average score drop	0.12			

4.2.2 Tokenization

Tokenization serves as a foundational step in the preprocessing pipeline of fine-tuning large language models by breaking down raw text into smaller units known as token. Tokenization enables the model to process text efficiently by converting it into a format that the model can understand and learn from. Tokenization in this scenario involves converting the input data into a format understandable by the LLM, where each token represents a meaningful unit of information. For the task, the project uses a class called AutoTokenizer from the Hugging Face Transformers library. This tool automatically picks the right tokenizer for the model, which we've specified with the variable model_name. With AutoTokenizer, it is ensured the tokenizer matches the relevant model and prepares input data for training different large language models. Figure 20 gives an example of how the AutoTokenizer performs the tokenization process.

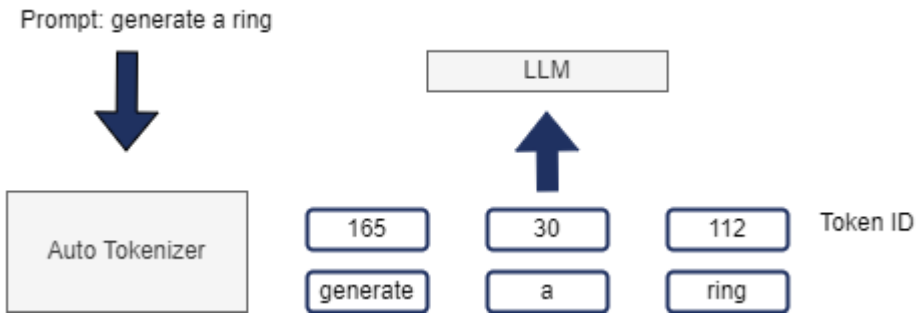

Figure 20: Tokenization.

4.2.3 LoRa Configuration

Low-rank adaptation (LoRA) is a method employed for efficiently fine-tuning Large Language Models (LLMs) with fewer parameters. Unlike full model fine-tuning, which can be resource-intensive in terms of time and computational costs due to the large model size, LoRA introduces a limited set of trainable parameters while keeping the original model parameters unchanged and frozen. This approach aims to achieve effective adaptation for specific tasks with reduced overhead. In the context of adapting pre-trained neural networks to new tasks, the weight matrix $W_0 \in R^{d \cdot k}$, is kept frozen and unchanged while the adaptation focuses on updating a low-rank approximation ΔW, as shown in equation 33, LoRa refer to the decomposition of the matrix ΔW into $B \in R^{d \cdot r}$ and $A \in R^{r \cdot k}$, which are matrices with reduced dimensions compared to W_0 designed to efficiently capture the necessary modifications for the new task. The rank r of this low-rank representation is determined as the minimum between the input dimension d and the output dimension k. R represents the rank of the low rank matrices learned during the finetuning process. As this value is increased, the number of parameters needed to be updated during the low-rank adaptation increases. Intuitively, a lower r may lead to a quicker, less computationally intensive training process, but may affect the quality of the model thus produced [31] [32]. By isolating the trainable parameters in A and B, the adaptation process is streamlined, enabling effective learning without the need to modify the entire weight matrix Wo directly.

$$W_0 + \Delta W = W_0 + BA \tag{33}$$

$$h = W_0 x + \Delta W x = W_0 x + BAx \tag{34}$$

During the modified forward pass of the neural network, the output is calculated as equation 34. Here, $W_0 x$ corresponds to the output using the pre-trained weights, while BAx introduces the contributions from the low-rank adaptation. Initially, BA is zero because B is initialized as zero, and then the update BAx is scaled by an amplification factor α / r, where α is a constant set to r to try and do not tune it. By focusing on this selective and controlled update mechanism, this project enhances stability and efficiency during the fine-tuning process, facilitating effective adaptation of pre-trained models to new tasks. Table 10 shows the effect of r on the number of parameters of the fine-tuned model compared to the original parameters of the model. By decreasing the rank, parameters can be reduced up to 0.292% of the total 7b parameters [33].

Table 10: Effect of r On the Number of Parameters

Rank	*Number of Parameters*
r =64	2.29 %
r =16	0.58 %
r = 8	0.292 %

4.2.4 Training Preparation

In the training phase, this project follows a structured approach to prepare the model. Firstly, it is to initialize the tokenizer associated with each model, facilitating the conversion of text data into a format compatible with the neural network. Subsequently, load the model from the Hugging Face library and employ 4-bit quantization, effectively reducing model precision.

To further refine the model, integrate the PEFT LoRa configuration. This configuration allows to fine-tune weights of A and B matrices, freezing the model's weight parameters W_0. After setting up the model and its configurations, load the training dataset and evaluation dataset. Finally, combine all components into the trainer object and it is ready for training. Figure 21 sums up the training preparation process.

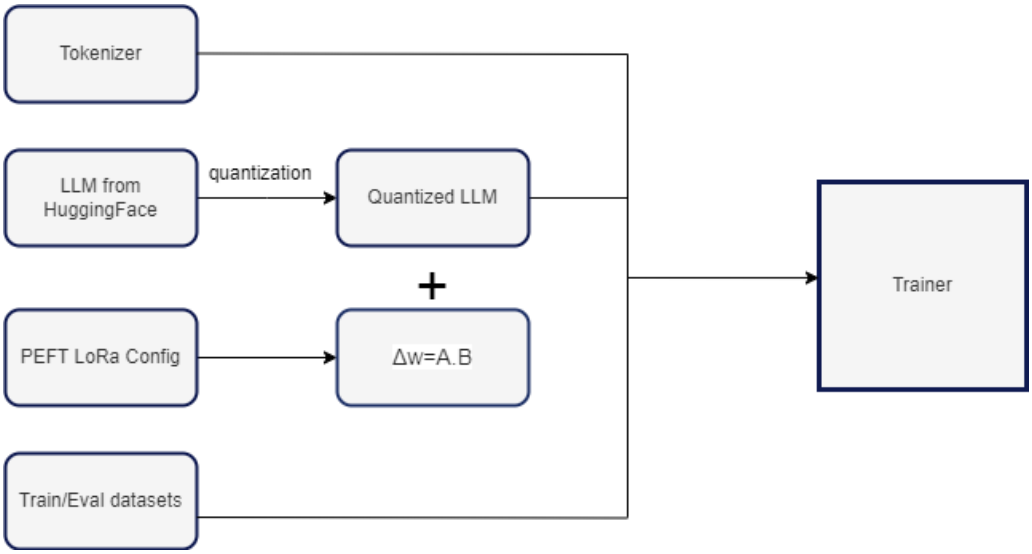

Figure 21: Training Preparation.

4.3 Scenario

A vehicle manufacturing company is in the process of designing a new product. The company is still in the initial stages of the design phase. In this phase, one of the objectives is to assemble all mechanical components for the new product and test their compatibility with the product. One of the mechanical components needed for the new product is Mild Square Bar. Before going into the next stage, the team is required to 3D print all components to examine them and decide on how to proceed. As such, the team makes use of the CADIFY software application. CADIFY is a software application where the user enters a request for a mechanical part with specific dimensions. CADIFY then gives 3 options to the user. To download either the JSON file, the CadQuery Code that sketches the component, or the STL file. So, the team enters the dimensions wanted for the Mild Square Bar and downloads its STL file. The team then 3D prints the Bar on a 3D printer using the STL file.

Chapter 5 Experimental Results and Performance Evaluation

Chapter five talks about the training parameters controlled as well as the evaluation metrics used. The chapter also presents the results after comparing the three LLMs.

5.1 Training parameters

As mentioned in previous sections, this project showcases the performance differences across 3 large language models. In this section, there will be comparison of three LLM models: Mistral 7B, Llama 2 7B, and Falcon 7B and then. The three models have been pre-trained on approximately 7 billion parameters and are trained for text generation purposes.

While training the models, there are shared parameters that can be controlled. These parameters are usually found in the training parameters and in the LoRa configuration where both are fed to the trainer function and can be changed according to the task for optimization purposes. Table 11 below mentions the description of the most prominent parameters to be considered for fine-tuning in this project in addition to the values to be considered while training. The optimal values for the rank, alpha, and learning rate parameters will be determined through experimentation. Therefore, careful selection and tuning of the learning rate is essential, as it significantly impacts the efficiency and effectiveness of the training process. By testing multiple configurations, values that minimize the loss function during training will be selected [34].

Table 11: Hyperparameters Overview

Hyperparameters	*Description*	*Value*
Rank	The rank of the low-rank matrices	Through experimentation.
Alpha	Constant used for amplification factor	Through experimentation.
Learning Rate	Determines how much the model's weights are adjusted in response to the estimated error during training.	Through experimentation.
Dropout	Randomly "dropping out" (setting to zero) a fraction of the neurons or units in the neural network during each training to prevent overfitting	0.05
Batch Size	Determines the number of individual training examples processed together before updating the model's weights.	4

Number of Epochs	Number of complete iterations over the entire training dataset.	1

Large Language Models (LLMs) employ cross-entropy as a loss function during their training process to assess the difference between the predicted probability distribution of words and the actual distribution observed in the training data. Cross-entropy loss calculates how well the predicted probabilities match the true probabilities by assigning a higher penalty to larger deviations. In the process of selecting the best model, the value of the cross-entropy loss function will be a key consideration.

5.2 Evaluation Metric

LLM models can be evaluated using some existing generic Token-Similarity metrics. Token similarity metrics measure the resemblance between texts generated by LLMs and reference texts. They are important for evaluating how well LLMs can generate a desired sequence of words based on contextual information. These metrics are commonly used by LLMs to evaluate the quality of generated texts and their impact on tasks such as machine translation and text summarization. Table x Defines Key metrics in this category including Perplexity, BLEU (Bilingual Evaluation Understudy), and ROUGE (Recall-Oriented Understudy for Gisting Evaluation).

This project deploys the BLEU Score since it fits the evaluation criteria for this project. Bleu Score is used to assess how well a machine-translated text compares to one or more reference translations. The method computes the degree of agreement between the created text and reference texts in terms of n-grams, or consecutive word sequences. A closer match to the source texts is indicated by a higher BLEU score, which suggests higher translation quality. The BLEU score typically ranges from 0 to 1, with 1 being a perfect match. When n = 1, this is called Unigram precision where the focus is solely on the presence of words, not their order or context. It measures how well the generated text captures the vocabulary of the reference text. Unigram Blue Score is chosen because the output of the model is a function call that follows a specific format. A unigram BLEU Score will inform us on how well the model was able to capture the format of the referenced output. Figure 10 shows the format of the referenced output. Equation 35 shows how BLEU score is calculated [35] [36].

$$BLEU = BP * \exp\left(\frac{1}{4} * \sum_{n=1}^{4} Log\ precision_n\right) \tag{35}$$

BP is the Brevity Penalty. BP penalizes cases where the generated text is too short and only partially matches the reference text, neglecting key information. The Brevity Penalty can be found in Equation 36.

$$Brevity\ Penalty(BP) = \min\left(1, \frac{Tokens\ in\ generated\ text}{Tokens\ in\ reference\ text}\right) \quad (36)$$

In the world of statistics, there is a similarity metric called Jaccard similarity that is also relevant to this project. Jaccard similarity is a statistical metric used to assess the similarity and variety of sample sets. It is also known as the Jaccard index or coefficient. It computes the intersection's size divided by the two sets' union's size. This metric is especially useful for comparing text outputs since it highlights the presence or lack of elements, which makes it ideal for assessing language models (LLMs).

As such, this project makes use of the Jaccard coefficient in a function that calculates the average similarity text by words between the actual output and the predicted output from the fine-tuned model. In this function, the actual output and the predicted output have been converted to lowercase to ensure that the comparison is case-insensitive. Then, it splits each text into a list of words using the split() method, which separates the text based on spaces. Next, the function converts these lists of words into sets to eliminate duplicates and facilitate set operations. It computes the intersection of the two sets, representing the common words in both texts and the union of the sets, representing all unique words appearing in either text.

The Jaccard similarity coefficient is then calculated, which is a measure of similarity between two sets defined as the size of the intersection divided by the size of the union. If the union is empty (indicating that both texts contain no words), the similarity score is set to 0.0 to avoid division by zero. Otherwise, the similarity score is computed as the ratio of the size of the intersection to the size of the union. Finally, the function returns this calculated Jaccard similarity coefficient. Figure 22 shows the code to calculate the Similarity text by words [37] [38].

```python
def calculate_word_similarity(text1, text2):
    # Tokenize the texts into words
    words1 = text1.lower().split()
    words2 = text2.lower().split()

    # Compute the intersection and union of words
    intersection = set(words1) & set(words2)
    union = set(words1) | set(words2)

    # Calculate the Jaccard similarity coefficient
    if len(union) == 0:
        similarity_score = 0.0
    else:
        similarity_score = len(intersection) / len(union)

    return similarity_score
```

Figure 22: Similarity Text by Words Code.

However, even with the existence of the above metrics, it is generally advised to design a task-specific human evaluation metric. This is because LLM models are used differently according to the task at hand, and in this project, a tailored evaluation metric is necessary considering that the goal of the evaluation is to measure how accurately the model can extract the dimensions from the user prompt and to find the area and perimeter from the dimensions. Accordingly, the third evaluation metric for this project is a metric tailored to the requirements of this project. This metric is the accuracy of the numerical values extracted from the predicted output using the Mean Absolute Error Percentage (MAPE).

The model is not only required to follow the reference format in Figure 10 but to also accurately extract the shape requested, the numerical values of the dimensions of the requested shape from the input (user prompt) and, give a numerical prediction on the shape area and perimeter. For this MAPE is used. MAPE expresses the average absolute error as a percentage of the predicted numerical values from the actual numerical values. The code in Figure 23 shows how the MAPE is calculated. Accuracy can be achieved from the MAPE simply by subtracting the MAPE from 100. For example, an accuracy of 90% means that the model was able to extract the numerical values at an error rate of 10% of the actual values. It is important to note that an accuracy of 90% does not mean that the model was able to predict 90% of the evaluation dataset accurately and 10% of the evaluation dataset inaccurately but it is a measure of each numerical value in respect to its actual value.

```
from re import M
# Calculate Mean Absolute Error (MAE) for each attribute
def calculate_mae(actual, predicted):
    n = len(actual)
    if n == 0:
        return 0
    return (sum(abs(a - p)/a for a, p in zip(actual, predicted)) / n)*100
```

Figure 23: MAPE Code.

5.3 Training Outcome

During the training process, the project utilized the A100 GPU available on Colab, known for its robust performance in deep learning tasks. This GPU, one of the premier options in Google Colab, enabled us to train the models efficiently. Conducted separate training sessions for each model, experimenting with different hyperparameters. Subsequently, Saved the trained models on Hugging Face and evaluated their performance based on the evaluation dataset. Additionally, recorded the loss curve using the Weights & Biases platform, which streamlines the model workflow from start to finish [39].

5.3.1 Validation

Regarding validation, the models were evaluated using an evaluation matrix. Time constraints significantly influenced the size of the evaluation dataset. Each fine-tuned model takes a varying amount of time, ranging from 15 seconds to 2 minutes, to generate a single output. Table 12 provides the time each model requires to generate one sample.

Given these time constraints, evaluating a large dataset was not practical. Therefore, this project used representative sampling to create a smaller but comprehensive evaluation dataset of 60 instances. This dataset included a wide variety of shape request cases, ensuring comprehensive evaluation even with fewer data points.

Table 12: Time Taken by Each Model During Evaluation

Model	*Time per sample*	*Time for 60 samples*
Falcon	2 minutes	2 hours
Llama2	3-4 minutes	3-4 hours
Mistral	15-20 seconds	20 minutes

5.3.2 Comparison Based on Loss Function

The initial phase of the comparison involved determining the optimal values for the parameters r and α by minimizing the loss function. This project conducted training sessions for Falcon 7B, Mistral 7B, and Llama2 7B using various combinations of α and r, while maintaining a fixed learning rate of 4e-4, which is the default rate employed during fine-tuning. Figure 24 illustrates the outcomes obtained from training these models with different parameter settings.

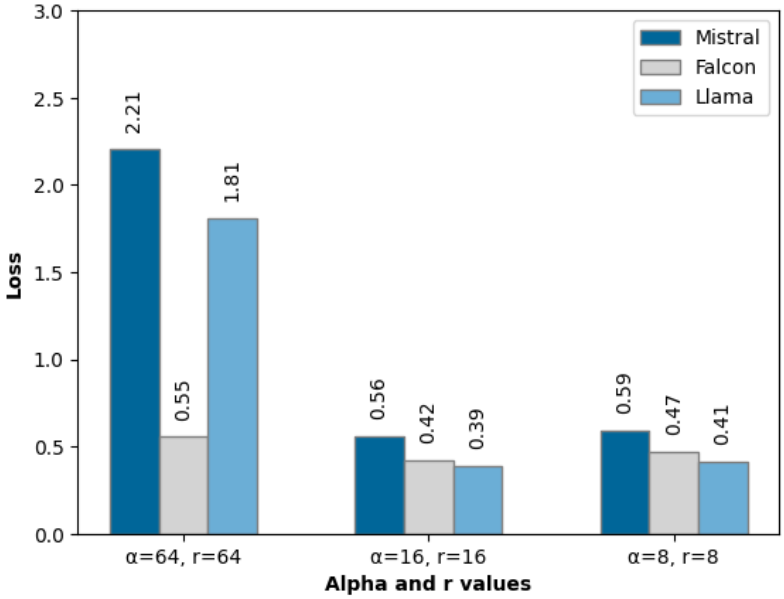

Figure 24: Comparison of Models Based on R and Alpha.

The figure indicates that the loss function varies across different models and parameter configurations. At r=64 and α=64, both Llama and Mistral exhibit considerably higher loss values compared to Falcon. However, reducing α and r to 16 leads to a noticeable decrease in loss for Llama and Mistral. Conversely, further decreasing α and r to 8 results in an increase in loss again. This observation suggests that α=16 and r=16 yield the best loss values for all three models, indicating that these parameter settings are optimal for minimizing loss. The two-loss curves illustrated in Figure 25 and Figure 26 show how higher values of r and α contribute to an unstable curve, characterized by fluctuations and irregularities. However, as the values of r and α decrease, the curve becomes smoother and more stable, suggesting that lower parameter values lead to training stability.

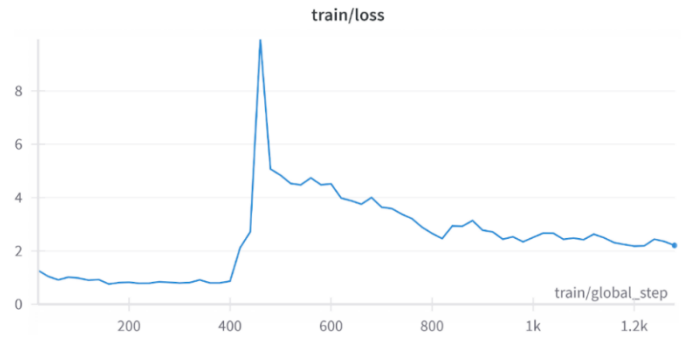

Figure 25: Loss Curve At r=64 and α=64.

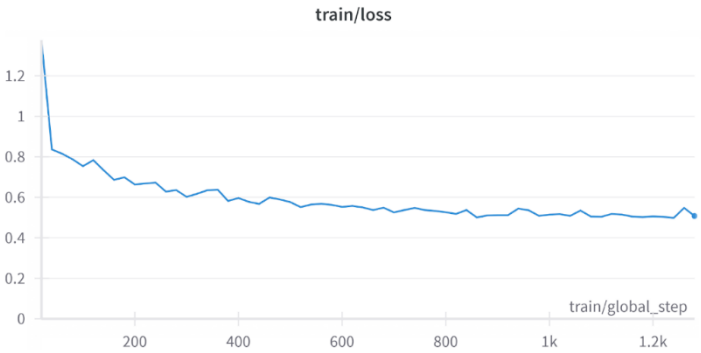

Figure 26: Loss Curve At r=16 and α=16.

5.3.3 Comparison based on Bleu Score

The second phase of comparison involves fixing the values of r and α at 16 and varying the learning rate. Initially set at 4e-4 then, proceeded to decrease it to 5e-5 and 6e-6. The comparison, shown in Figure 27, is based on the BLEU score, which assesses the presence of each word of the predicted output in the actual output. Higher BLEU scores indicate better performance, suggesting that the model successfully generated the function calls with their parameters. However, lower BLEU scores, as observed in the case of llama2 at 5e-5 and 6e-6, and Mistral at 6e-6, signify that the model struggled to generate accurate function calls. Consequently, these models will be excluded from the comparison analysis.

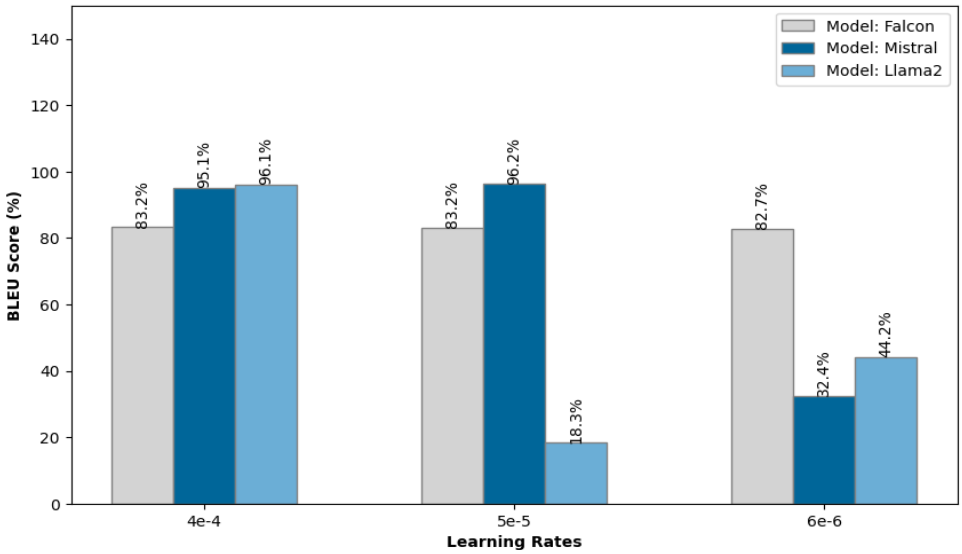

Figure 27: Comparison of the Models Based on the BLEU Score.

5.3.4 Comparison based on Similarity Text

The third step of comparison focused on selecting the learning rate that yielded the highest text similarity score. This metric indicates the degree of similarity between the actual and predicted outputs. Models trained with different learning rates were compared, excluding those with low BLEU scores. Notably, Falcon models across different learning rates presented lower similarity scores. The reason for this is their tendency to generate rubbish data alongside the required function calls, therefore, the text similarity score for Falcon models dropped to around 28% as shown in Figure 28. However, the Mistral and Llama2 models achieved higher similarity scores, averaging around 77%. Despite their relatively better performance, Mistral and Llama2 models included the user prompt within the output alongside the predicted function calls. This caused a slight reduction in the similarity score, as the actual output should ideally contain only the function call. As a result, Falcon models will be filtered out from the comparison.

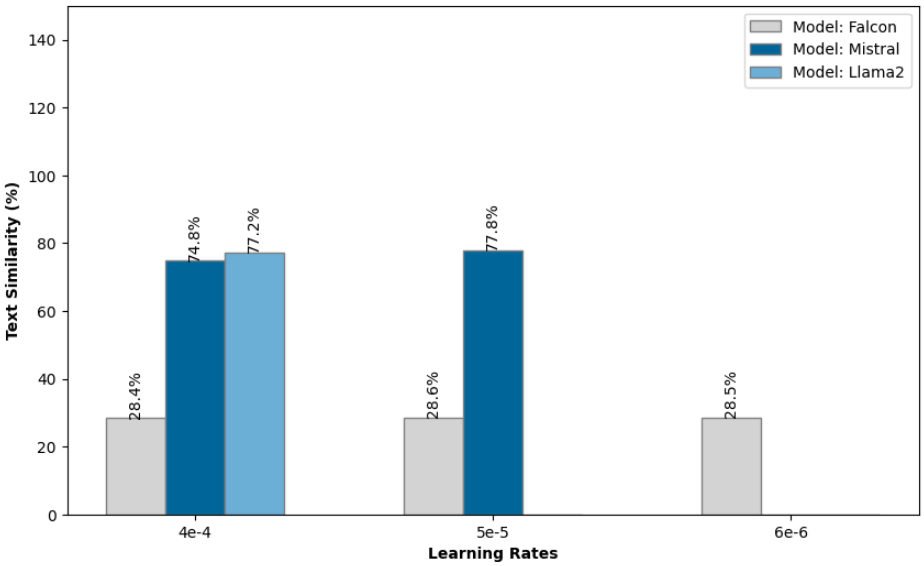

Figure 28: Comparison of the Models Based on Text Similarity against Learning Rates.

Now, the final comparison will be on three models for the final phase: Mistral trained with learning rates of 4e-4 and 5e-5, and Llama2 trained with a learning rate of 4e-4. These models have demonstrated the best performance in the previous steps of comparison, showing relatively higher BLEU scores and text similarity scores compared to other configurations. By focusing on these three models, the aim is to evaluate their overall effectiveness in generating accurate function calls with accurate numeric values for all parameters within the function call.

In conclusion, our comparison phases offered valuable insights into how different language models (LLMs) perform when tasked with generating shapes. By systematically testing various hyperparameters and evaluation metrics, we were able to determine the most effective configurations for each model. Adjusting parameters like learning rate, alpha, and rank in an iterative manner, and validating our findings through representative sampling, ensured a thorough assessment of model performance.

Ultimately, Mistral 7b emerged as the most promising model. When configured with alpha and r values set to 16, alongside a learning rate of 5e-5, Mistral 7b consistently demonstrated superior performance across multiple evaluation metrics. It consistently achieved the lowest loss, highest BLEU score, and highest similarity score compared to other configurations. Additionally, Mistral 7b proved to be the most efficient model, generating output at the fastest rate. These results highlight Mistral 7b's effectiveness and efficiency in shape generation tasks, making it the top-

performing model in our comparison study.

5.3.5 Comparison based on MAPE

In the final phase, it is to compare the remaining models based on their accuracy in predicting the values for each parameter. Table 13 below shows the accuracy of each model. All models achieved 100% accuracy in predicting the shape name, indicating that they could extract the name of the shape from the user prompt accurately. Additionally, the models performed perfectly in predicting the values for height, inner radius, and outer radius, with the exception of Mistral at a learning rate of 4e-4. These models successfully extracted the radius values even when the user provided the diameter instead.

However, when it comes to predicting the area and perimeter of each shape, Mistral at a learning rate of 4e-4 performed poorly. Therefore, the comparison now focuses on Mistral at a learning rate of 5e-5 and Llama2 at a learning rate of 4e-4. As illustrated in the table, Mistral at 5e-5 outperformed Llama2 at 4e-4, showing higher accuracy in predicting the area and perimeter for each shape. This indicates that Mistral at a learning rate of 5e-5 is the most accurate model for predicting these parameters.

Table 13: Accuracy of the Dimensions between the Final Models

Parameter	*Mistral (4e-4)*	*Llama(4e-4)*	*Mistral(5e-5)*
Heights	100	100	100
Outer Radius	100	100	100
Inner Radius	24.8	100	100
Ring Profile Area	72.0	87.9	93.5
Tube Profile Area	49.0	97.4	98.0
Hex Nut Profile Area	42.4	87.6	90.7
Mild Square Bar Profile Area	93.9	99.3	99.8
Shackle Linkage Profile Area	87.0	92.0	96.5
Ring Profile Perimeter	99.3	99.9	99.9
Tube Profile Perimeter	91.9	99.8	99.8
Hex Nut Profile Perimeter	97.6	98.6	99.6
Mild Square Bar Profile Perimeter	99.9	99.1	100
Shackle Linkage Profile Perimeter	87.8	97.0	96.3
Shackle Linkage Side Length	43.5	99.3	99.7
Joint Radius	99.1	100	100
Mild Square Bar Side Length	100	100	100
Shape Accuracy	100	100	0.98

5.4 Filling the JSON file.

In the next step after model evaluation, the output is executed in Python using the exec() function. The exec() function is a powerful built-in Python function that dynamically executes code from a string or object. It allows for the execution of dynamically generated programs and can be used for evaluating dynamic code, implementing scripts, or modifying the global or local namespace.

The desired output from the model is a function call to Generate_file. This function takes the shape name and corresponding parameters, which are generated based on the user prompt, as arguments. These arguments are used to populate the respective JSON files.

As outlined in section 2.2, the JSON templates are derived from the original Fusion 360 dataset but have been modified and simplified for better human readability and easier file handling. Additionally, random IDs have been replaced to streamline the filling process, which is based on the equations specified in section 3.4.3. Each JSON template is fixed per shape, with only the values being updated based on the provided parameters, Figure 29 shows the process of filling the JSON file by python code.

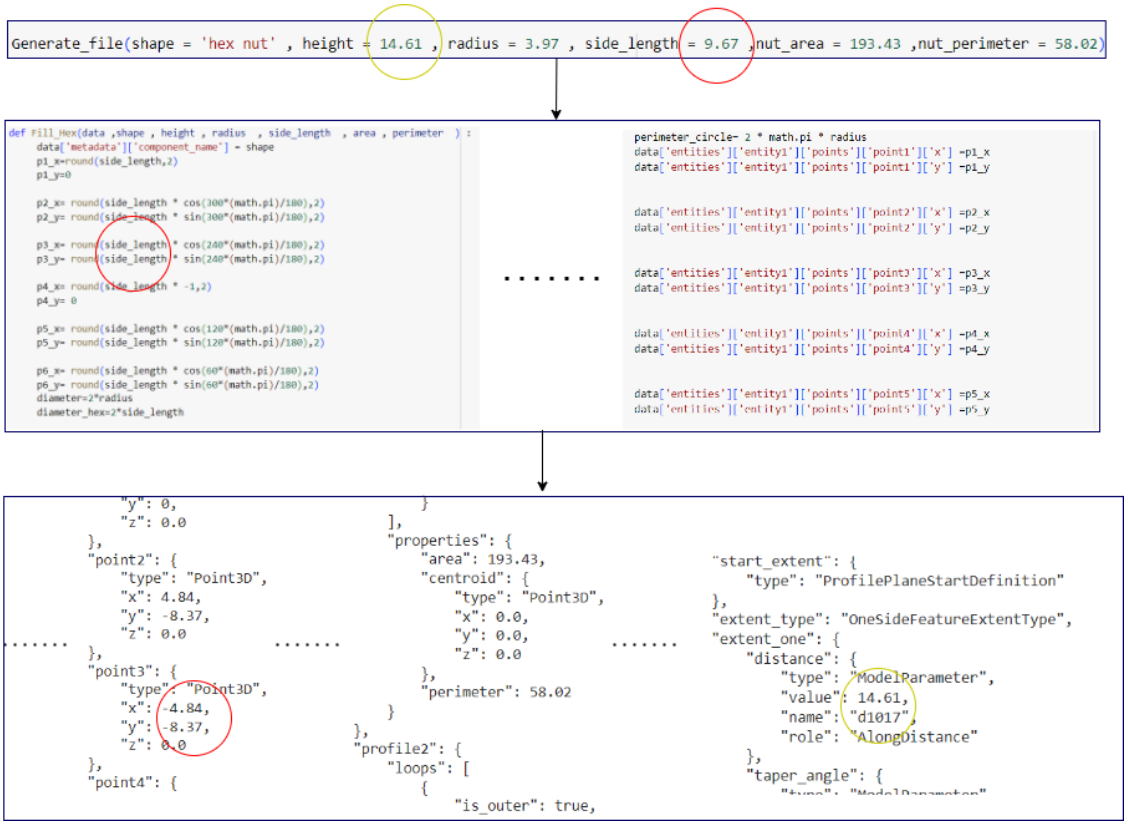

Figure 29: Filling the JSON File.

5.5 CadQuery Conversion

The next step is converting the JSON file into CadQuery code. This is done by iterating through the entire file to extract and save points, curves, and profile information for sketch entities. The process begins with sketching the profiles that are in the **extrudeFeature** entity by building the

profile using CadQuery commands.

First, a work plane is created with **cq.Workplane("XY").** Then, the curves in the extruded profiles are combined into one profile. The generated code is dynamic, based on the curves in the extruded profile as shown in Figure 30a. For the relevant shapes, only three curve types are used:

- **Circle**: Created with **.moveTo(center_point).circle(radius)**
- **Arc**: Created with **.moveTo(start_point).radiusArc(endpoint, radius)**
- **Line**: Created with **.moveTo(start_point).lineTo(endpoint)**

Once the profile is created, it is extruded using the **.extrude(extrude_value)** method. If the profile contains arcs or lines, it must end with the **.close()** method to ensure the path is closed by drawing a straight line back to the starting point.

The values used are filled based on the information extracted from the JSON file. The generated code includes important import statements and dependency installations. To be user-friendly, comments are added to explain the generated code as shown in Figure 30b. The code generates a CadQuery plane and exports it into two formats:

SVG: This is used for visualization with Gradio. The Gradio library in the script creates an interactive web interface that reads and displays the SVG file generated by CadQuery, making it easy to visualize the CAD model in a web browser without specialized CAD software as shown in Figure 30c.

STL: This format is exported to Google Drive for 3D visualization. The STL file can be viewed using 3D viewers and CAD viewers as shown in Figure 30d.

The output designs of SVG and STL can be viewed as shown in Figures 31a and 31b respectively.

```
# Initialize a new cadquery Workplane object
wp = cq.Workplane("XY")

profile = (
  wp
  .moveTo(4.84,8.37)
  .lineTo(9.67,0) #Draw Outer Line1
  .moveTo(9.67,0)
  .lineTo(4.84,-8.37) #Draw Outer Line2
  .moveTo(4.84,-8.37)
  .lineTo(-4.84,-8.37) #Draw Outer Line3
  .moveTo(-4.84,-8.37)
  .lineTo(-9.67,0) #Draw Outer Line4
  .moveTo(-9.67,0)
  .lineTo(-4.83,8.37) #Draw Outer Line5
  .moveTo(-4.83,8.37)
  .lineTo(4.84,8.37) #Draw Outer Line6
  .close()
  .moveTo(0.0,0.0)
  .circle(3.97)# Draw Inner Circle1
)
profile = profile.extrude(14.61)
```

a

```
##Uncomment the following lines if you didn't install cadquery and gradio !!

'''
!pip install cadquery
!pip install gradio
!pip install gradio Pillow
'''

## Important Libraries


import pandas as pd
import cadquery as cq
import gradio as gr
import cadquery as cq
from cadquery import exporters
import math
```

b

```
#For Gradio

# Export the result to a file in SVG format.
# Parameters:
# - profile: The data or object to be exported, typically a graphical representation or a plot.
# - "C:\Users\tibra\shape.svg": The file path where the SVG file will be saved.
#   Note: Ensure the file path is correctly formatted for your operating system (double backslashes for Windows).

exporters.export(profile, "C:\\Users\\tibra\\shape.svg")

def display_svg():
    # Read the SVG content from the file
    # Change the file path as needed
    with open("C:\\Users\\tibra\\shape.svg", "r") as svg_file:
        svg_content = svg_file.read()
    return svg_content

# Set up a Gradio interface to display the SVG content as HTML
iface = gr.Interface(
    fn=display_svg,
    inputs=None,
    outputs="html",
    title="SVG Image Display",
    description="Display an SVG image from the specified path."
)

# Launch the Gradio interface
iface.launch()
```

c

```
# To mount to your Drive
from google.colab import drive
drive.mount('/content/drive')



# Export the result as an STL file to the specified path
exporters.export(profile, '/content/drive/MyDrive/result.stl')
```

d

Figure 30: Generated Code [a, b, c, d].

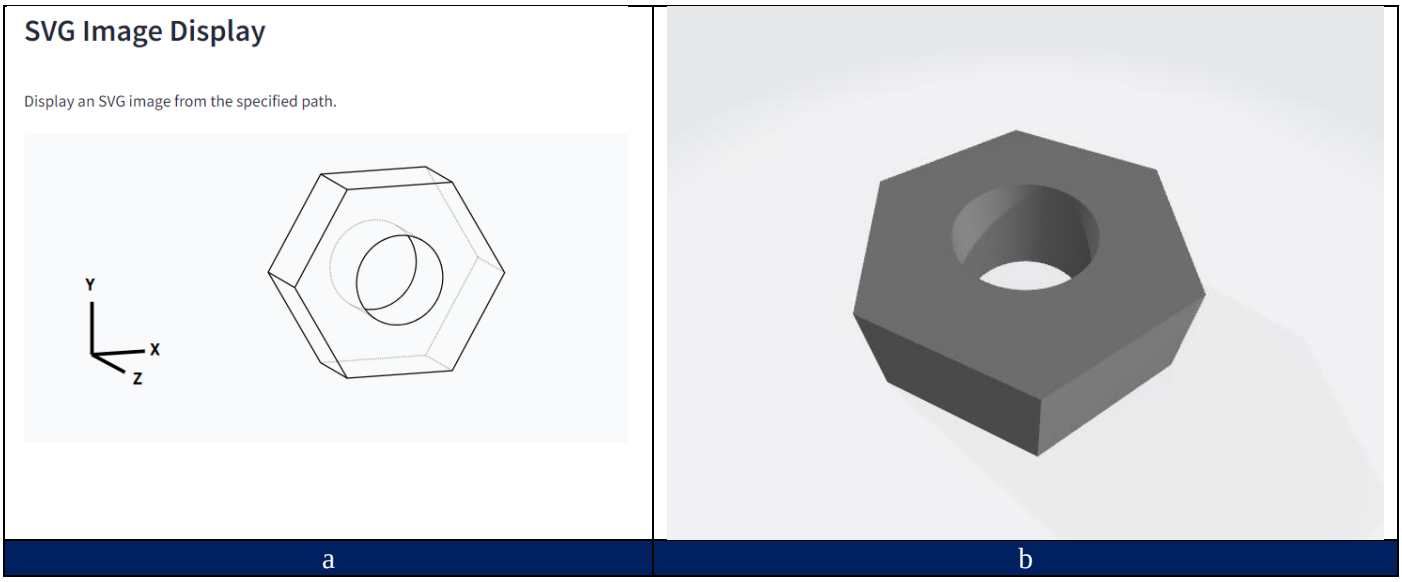

Figure 31: Models 3D View.

5.6 Website Creation

For the website creation, Squarespace is used to build a semi-online platform called CADify. The main concept of CADify is to take user prompts and redirect them to a path that is accessible by the model.

5.6.1 Platform Choice: Squarespace

Squarespace is used for its user-friendly interface and robust features, which allowed us to quickly set up and customize the website without extensive coding.

5.6.2 Website Functionality

User Prompts to JSON: When a user submits a prompt on CADify, it is redirected to a path that is accessible by the model. The prompt must contain the necessary information for creating the 3D models.

User Options: After submission, users have three options:

- **Show JSON File:** Users can view the raw JSON data for the model requested.
- **Show STL File:** Users can download or view the STL file, which is the 3D model ready for printing.

- **Show CadQuery Code:** Users can view the CadQuery code used to generate the 3D model, allowing for further customization and understanding of the model creation process.

5.6.3 About Us page

The "About Us" page introduces CADify, explaining the purpose and functionality of the website. It also includes detailed descriptions of the available shapes and their dimensions, helping users understand the potential applications and benefits of using the platform.

5.6.4 Website design

Figure 32 is the home page that appears when the user clicks on the website link. The home displays a welcome message and a request for the user to enter their names and the requested shape with the requested dimensions. Figure 33 is the page that the website directs the user to when entering the user request. This page provides the user with the available files to download.

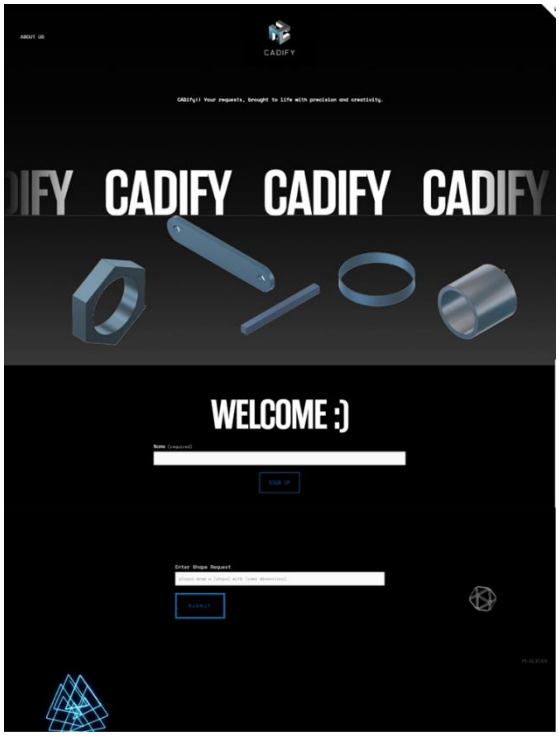

Figure 32: Home Page.

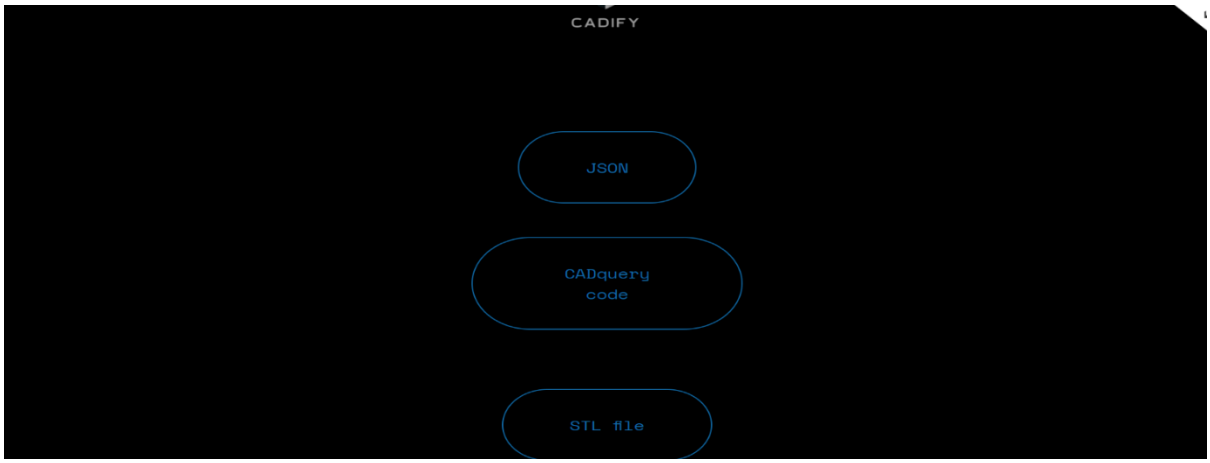

Figure 33: Options for the User.

Figure 34 is the About Us page. On this page, the website explains the purpose of the website, introduces the available shapes, and gives brief information about each shape.

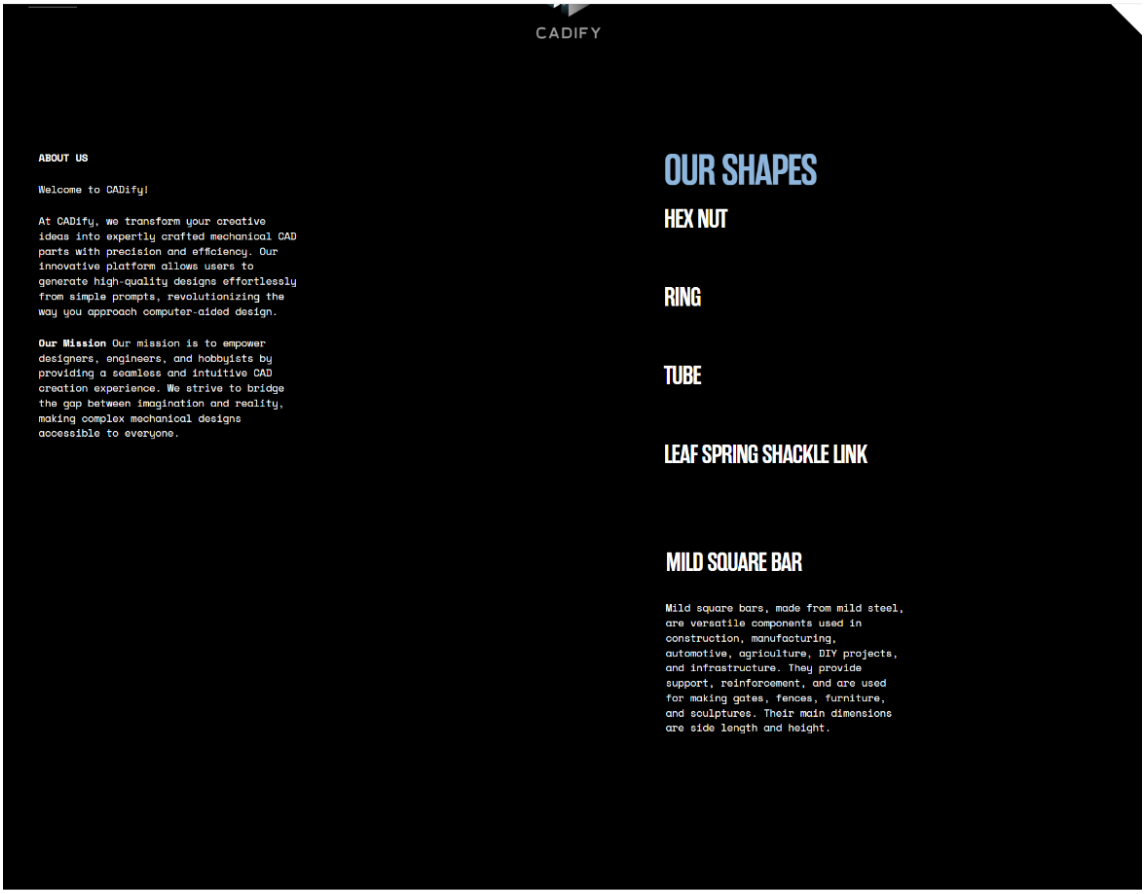

Figure 34: About Us Page.

5.7 Meeting Design Requirements within Realistic Constraints

present the achievement and compliance of this project with the design requirements within the realistic constraints established at the outset of the project, affirming their fulfillment is duly certified.

Table 14: Achievement of Design Requirements

Design Requirements	***Comments***	***Achievements***
The system shall include the development of a user-friendly interface for users to input their design requirements for a mechanical part using natural language commands and prompts.	Website CADify was created as a user-friendly interface. CADify allows users to enter their requests using commands and style of writing chosen by the user.	✓
The system shall exploit LLM's capabilities to analyze and understand the user's description and requirements.	After Fine-tuning the LLMs were able to successfully extract information from the user-prompts.	✓
The work shall include the generation of a dataset of at least 3 mechanical parts.	The dataset included total of 5 shapes.	✓
Each mechanical part within the dataset shall have at least 100 samples.	Each mechanical part in the dataset contained at least approximately 1500 samples.	✓
The pre-trained LLM shall be fine-tuned using the generated dataset.	The LLMs were fine-tuned using the dataset created.	✓
The system shall generate a CadQuery code based on the	The system successfully generates	✓

user's input.	CadQuery code based on user requirements.	
The system shall be able to convert the CadQuery code to a file that is readable by CAD viewers.	The system successfully downloads STL file that can be opened by any 3D viewer.	✓
The system shall be able to display the mechanical part to the user.	The system displays the shape requested by the user on fusion or displayed on 3D viewers using the corresponding STL file.	✓
The system shall be able to determine the mechanical part according to the user's input with an accuracy of at least 70%.	The system can determine the mechanical part requested by the user with an accuracy of 98%.	✓

Table 15: Fulfilment of Realistic Constraints

Realistic Constraints		***Fulfillment***
Economic	The cost shall not exceed 400JDs. This cost will be for using Colab pro which is a subscription plan for Google's Collaboratory platform. It offers advanced GPU and TPU resources for faster model training.	Cost Breakdown: Colab pro plus subscription: 38 JD 2100 GPUs bought at 7 JD per 100 GPU Total: 248 JD
Manufacturability & Sustainability Constraints	Challenges in training Large Language Models (LLMs) for mechanical parts recognition include limited access to diverse datasets featuring numerous samples of various components. Additionally, the computational complexity involved in fine-tuning pre-trained LLMs poses a significant obstacle. Moreover, keeping up with the latest versions of Large Language Models (LLMs) is a constant challenge. As these models evolve with new features and structures, it becomes necessary for practitioners to adjust their existing methods.	System fine-tuned on dataset created specifically for the project. The system fine-tuned the LLMs trained on approximately 7 billion parameters. The system fine-tuned the LLMs using the most updated versions available on Hugging Face.
Ethical	Potential privacy concerns and signing Istari Digital NDA. In the context of potential privacy concerns, ensuring the confidentiality of the project idea is essential. Bysigning Istari Digital's Non-Disclosure Agreement (NDA), all parties commit to preserving the integrity of the project concept.	The project preserves all agreements between the relevant entities.

Chapter 6 Conclusion and Future Work

This project went through a couple of approaches before deciding on the best approach. Initially, the approach was to create a dataset consisting of user prompts alongside respective JSON files taken directly from the Fusion360 dataset. To create this dataset, it was required to manually filter the dataset which consists of approximately 40,000 files to find JSON files of the shapes chosen. Upon filtering all the files, it turned out that most shapes only have a small number of JSON files. For example, the largest number of files found was 250 files for the Ring shape. The second issue discovered was the randomness of the content of the JSON files found. Although JSON files are supposed to be human-readable, they sometimes contain information that is either unnecessary or not clear.

In this case, in each file for a specific shape, the identifiers in the file, the ones responsible for identifying important components such as file name and entities, are given a random 32 integer identifier for each file for each component. This randomness if fed to the pre-trained LLM might raise issues where the model will have difficulties building relationships. Thus, two decisions were made here. Due to the unavailability of enough JSON files for each shape, a template JSON file for each shape was chosen from the dataset and emptied of the numerical values that make up that unique file such as the inner radius and height in case of the Ring.

Then, using a code that has saved the JSON file format, a dataset can be created of any size wished. The values of the dimensions in the JSON file template each time were filled with random values each time to ensure the diversity that was found in the original dataset. It is important to note here that the dimensions area and diameter cannot be filled with any random value but should be consistent with dimensions such as the radius and diameter. For that reason, the mathematical relations to calculate the area and dimension were manually found and filled in the JSON file based on them.

At this point, a dataset consisting of user prompts with JSON file was created and ready for training. Initially, a dataset of sizes ranging from 300 to 600 was created. It is worth noting here that each JSON file contained approximately 4000 characters. On a dataset of such size, fine-tuning the model took a good amount of time approximately 2 hours.

The fine-tuning was done on the free version of Google Colab. However, the results as expected were poor and the output is always printing only half the file. It is natural that a large dataset is needed to adequately fine-tune LLM. A larger dataset of size of approximately 2000 was created for testing purposes and then kept on increasing the size up to 4600, naturally when increasing the dataset size, the time it takes to fine-tune increases. However, the GPU amount the free Colab offers is not enough for training on a dataset of this size. Nonetheless, the training continued with a paid Colab subscription. Alas, the output improved but not remarkably and the output was still printing only half the file.

It was clear that increasing the dataset size was key. Thus, this approach came to a natural halt when the time it takes to fine-tune a dataset of size 4600, which is midsize in number of instances but the total number of characters for a dataset this size was almost 22 million, is almost 6 hours, and the paid for GPUs are exasperated very quickly. As such, for reasons of resource limitations, another approach had to be taken.

The next and final approach is the current one. A solution was created based on the previous limitations. This solution preserves the main objectives of this project. As a reminder, the main objectives of this project are as follows:

- Show the capabilities of LLMs to understand the dataset extract information.
- To prove that LLMs can build mathematical relationships to predict dimensions like area and perimeter.
- Convert the JSON file to CadQuery code.
- Take in a user request for a shape and provide the user with the STL file of the shape.

Thus, the final approach is to change the dataset. Instead of having the dataset include the JSON files themselves, the JSON files are substituted with function calls to the function that builds the JSON files. This is possible because the JSON files of each shape are of the same template regardless of the dataset. In the function call the shape dimensions are included as arguments. These dimensions include the area and the perimeter of each shape.

This approach was the best solution for all previous issues. The final dataset size chosen in this project consisted of approximately 16,700 instances which was three times larger than the dataset of the previous approach, but the total number of characters was almost equal to 5.5 million which is three times less than the previous dataset. Also, through this solution, all objectives of the project are achieved. Thus, proved that the model could understand the dataset and predict the output in the required format. The model was able to extract all necessary information from the user prompt. Finally, the model was proven to be able to predict the area and perimeter of the shapes by building mathematical relations.

Looking ahead, this project can be expanded in several ways to enhance its functionality and applicability. One potential avenue for future work is the integration of more complex shapes that consist of more than one basic component assembled. Furthermore, the project can be fine-tuned on models that are trained on a larger number of parameters. For example, Mistral 70B or Falcon 70B.

With the availability of Faster and Larger GPUs and computational powers, the dataset can directly include the JSON files instead of the function calls. This will allow the model to directly output the JSON files of the requested shape. Also, with better resources, the size of the dataset can increase to include a much greater number of shapes.

References

[1] S. M. Kerner, "What are large language models (LLMs)," May 2024. [Online]. Available: https://www.techtarget.com/whatis/definition/large-language-model-LLM.

[2] E. Beeching, C. Fourrier, N. Habib and e. al, "Open LLM Leaderboard," 2023. [Online]. Available: https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard.

[3] "What are large language models (LLMs)?," [Online]. Available: https://www.ibm.com/topics/large-language-models.

[4] "Large Language Models Explained," [Online]. Available: https://www.nvidia.com/en-us/glossary/large-language-models/.

[5] "Mistral 7B: The best 7B model to date, Apache 2.0," [Online]. Available: https://mistral.ai/news/announcing-mistral-7b/.

[6] "tiiuae/falcon-7b," [Online]. Available: https://huggingface.co/tiiuae/falcon-7b.

[7] "The Falcon has landed in the Hugging Face ecosystem," [Online]. Available: https://huggingface.co/blog/falcon.

[8] "Model Details," [Online]. Available: https://github.com/meta-llama/llama/blob/main/MODEL_CARD.md.

[9] W. Karl, "Fusion360GalleryDataset," [Online]. Available: https://github.com/AutodeskAILab/Fusion360GalleryDataset/.

[10] "SKETCH," [Online]. Available: https://cadquery.readthedocs.io/en/latest/sketch.html.

[11] "Examples," [Online]. Available: https://cadquery.readthedocs.io/en/latest/examples.html.

[12] "Getting started with Fusion," [Online]. Available: https://help.autodesk.com/view/fusion360/ENU/?guid=GUID-7B5A90C8-E94C-48DA-B16B- 430729B734DC.

[13] "Build & Share Delightful Machine Learning Apps," [Online]. Available: https://www.gradio.app/.

[14] K. D. WILLIS, Y. PU, J. LUO, H. CHU, T. DU, J. LAMBOURNE, A. SOLAR-LEZAMA and W. MATUSIK, "Fusion 360 Gallery: A Dataset and Environment for Programmatic CAD," *ACM Transactions on Graphics*, vol. 40, no. 4, 2021.

[15] A. A. Khan, "Deep Learning for Object Detection: Training Data Generation using Parametric CAD Modelling and Gazebo Simulation," 2021.

[16] Y. Rebahi, M. Gharra, L. Rizzi and I. Zournatzis, "Combining Computer Vision, Artificial Intelligence and 3D Printing in Wheelchair Design Customization: The Kyklos 4.0 Approach," *Artificial Intelligence and Applications*, 2023.

[17] Y. Ganin, S. Bartunov, Y. Li, E. Keller and S. Salicet, "Computer-Aided Design as Language," DeepMind, 2021.

[18] M. Jang, G. Kim, S. Park and H. Lee, "Chat 3D: Interactive 3D Reconstruction With Assistance of Large Language Model," *Image and Vision Computing*, vol. 137, p. 11, 2023.

[19] "O Ring," [Online]. Available: https://shorturl.at/XbuIF.

[20] "STANDARD USA SQUARE-RINGS SIZES," [Online]. Available: https://www.marcorubber.com/square-ring-seal-size-chart.htm.

[21] "Steel tubes for mechanical engineering, machine components and general use," [Online]. Available: https://www.steeltube.sk/for-mechanical-engineering-parts-of-machines-and-general-use/.

[22] "HEX JAM NUT," [Online]. Available: https://www.zerofast.com/product/hex-jam-nut.

[23] "MILD STEEL SQUARE BAR," [Online]. Available: https://www.metalsupplies.com/products/mild-steel-square-bar/.

[24] "Heavy Duty Shackle Set Performance Polyurethane; Includes Spring Bushings, Shackle Bushings and Hardware," [Online]. Available:

https://www.rockauto.com/en/moreinfo.php?pk=1137336&cc=0&pt=7484&jsn=3.

[25] "Here's Everything You Need To Know About Encoder-Decoder Architecture," [Online]. Available: https://inc42.com/glossary/encoder-decoder-architecture/.

[26] "NVIDIA: TensorRT-LLM," [Online]. Available: https://github.com/NVIDIA/TensorRT-LLM.

[27] "GPT and other LLM's: decoder only v/s encoder-decoder models?," [Online]. Available: https://medium.com/@ManishChablani/gpt-and-other-llms-are-they-decoder-only-or-encoder-decoder-models-1bdaf23a256a.

[28] I. Jackson and M. J. Saenz, "From Natural Language to Simulations: Applying GPT-3 Codex to Automate Simulation Modeling of Logistics Systems," [Online]. Available: https://shorturl.at/2PZwV.

[29] T. Dettmers and L. Zettlemoyer, "The case for 4-bit precision: k-bit Inference Scaling Laws," 2023.

[30] O. Zem, "Exploring the Impact of Quantization on LLM Performance," [Online]. Available: https://medium.com/@olga.zem/exploring-the-impact-of-quantization-on-llm-performance-5698e16c5564.

[31] A. Sooriyarachchi, "Efficient Fine-Tuning with LoRA: A Guide to Optimal Parameter Selection for Large Language Models," [Online]. Available: https://www.databricks.com/blog/efficient-fine-tuning-lora-guide-llms.

[32] Z. Rauf, "A beginners guide to fine tuning LLM using LoRA," [Online]. Available: https://zohaib.me/a-beginners-guide-to-fine-tuning-llm-using-lora/.

[33] E. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang and W. Chen, "LoRA: Low-Rank Adaptation of Large Language Models," 2021.

[34] M. Andriushchenko, F. D'Angelo, A. Varre and N. Flammarion, "WHY DO WE NEED WEIGHT DECAY IN MODERN DEEP LEARNING?," [Online]. Available: https://arxiv.org/abs/2310.04415.

[35] T. Hu and X.-H. Zhou, "Unveiling LLM Evaluation Focused on," 2024.

[36] K. Papineni, S. Roukos, T. Ward and W.-J. Zhu , "Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics," Association for Computational Linguistics, 2002, pp. 311-318.

[37] M. Jadeja, "Jaccard Similarity Made Simple: A Beginner's Guide to Data Comparison," 2022. [Online]. Available: https://medium.com/@mayurdhvajsinhjadeja/jaccard-similarity-34e2c15fb524.

[38] M. Azam, Y. Chen, M. O. Arowolo, H. Liu, P. Popescu and D. Xu, "A Comprehensive Evaluation of Large Language Models in Mining Gene Interactions and Pathway Knowledge," [Online]. Available: https://www.biorxiv.org/content/10.1101/2024.01.21.576542v1.full.pdf.

[39] "The AI developer platform," [Online]. Available: https://wandb.ai/site.