CHALMERS
UNIVERSITY OF TECHNOLOGY

Artificial Intelligence and Large Language Models in CAD

Exploring the capabilities of AI and Large Language Models in assisting with geometric feature selection and identification in CAD.

Master's thesis in Product Development

ATHARVA NAIK

DEPARTMENT OF INDUSTRIAL AND MATERIAL SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2024

www.chalmers.se

Master's thesis 2024

Exploring the capabilities of Artificial Intelligence and Large Language Models in assisting geometric feature selection and identification in Computer Aided Design.

A master thesis report by

ATHARVA S. NAIK (IMSX30)

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Industrial and Material Sciences
Division of Mechanical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2024

Exploring the capabilities of artificial intelligence and LLMs to assist geometric feature selection and identification in Computer Aided Design.
A Master Thesis in Product Development
ATHARVA S. NAIK

Supervisor: Alejandro Pradas Gómez, Chalmers University of Technology
Najeem Muhammed, GKN Aerospace, Solid Mechanics Department
Examiner: Ola Isaksson, Department of Industrial and Material Sciences

Master's Thesis 2024
Department of Industrial and Material Sciences
Division of Product Development
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: AI & LLMs in Engineering Design, AI generated image using the prompt: Create a colorful, highly detailed, and accurate image depicting AI and Large Language Models (LLMs) identifying and selecting CAD geometry using natural language engineering terminology.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2024

Exploring the capabilities of AI and Large Language Models to assist geometry selection and identification in Engineering Design
A Master Thesis in Product Development
ATHARVA NAIK
Department of Industrial and Material Sciences
Chalmers University of Technology

Abstract

Introduction

The process of Finite Element Analysis (FEA) of components necessitates a CAD model that has been appropriately tagged and prepared. This tagging involves identifying and selecting specific parts of the CAD geometry for the application of boundary conditions, forces, or mesh specifications. While manual tagging is feasible for simple geometries, it becomes extremely complex and time-consuming for the intricate and large-scale components used by GKN Aerospace. To address this challenge, GKN Aerospace developed an in-house automated tool to streamline certain aspects of the tagging process, thereby saving time and effort. However, the tool requires proficiency in YAML, a programming language unfamiliar to many analysis engineers, complicating its usage and highlighting the need for a more user-friendly interface.

The aim of this project was to explore the capabilities of Artificial Intelligence and Large Language Models in assisting with tasks such as comprehending and translating natural language terminology of geometric features and using the capabilities of these generative models to create extraction queries in SQL formatted in YAML. It is hypothesized that the capabilities of generative models, text in this case, could be useful in making the geometry tagging process used by the automated tool much more streamlined and simple to use, as it only requires natural language input and not complex code that may be subject to formatting errors. While there are multiple ways to carry out the tagging process, this project focuses only on the process used by the "Autotag" automated tagging tool.

A set of research questions were formulated to guide the project and ensure it was focused on the right outcomes. The project was conducted over a period of 4 months. during the initial phases of the project, interviews were conducted to better understand the challenges and issues currently being faced by the engineers and GKN. Literature reviews were also conducted to gather information about the advantages and disadvantages about LLMs and AI and ways in which these technologies could be used to solve the issues that were identified.

An application was then developed that addressed these problems, and was integrated into the "Autotag" tool in Sim-Center. The development, carried out with knowledge from literature reviews and GKN engineer interviews utilizes Large Language Models (LLMs) to simplify feature selection and identification. It allows use of natural language to ease communication between the user and the application. While limitations exist due to LLMs and complex CAD models, the project showcased a possible way in which LLMs can be used to augment the tagging process where manual selection may not be possible.

Keywords: Artificial Intelligence, Large language models, CAD, Finite Element Analysis, Engineering Design

Acknowledgements

I would like to express my deepest gratitude to Alex and Najeem for their invaluable guidance, input, and unwavering support throughout the project. Special thanks to Ola for his expert guidance in research and development. I am also very grateful to Kevin and Hugo for their constructive opposition and feedback. Finally, a heartfelt thank you to Rikard Nedar and GKN Aerospace for providing this incredible opportunity.

Atharva Naik, Gothenburg, 6/2024

Acronyms

AE Auto Encoder. 38
AI Artificial Intelligence. 18

CAD Computer Aided Design. 3, 4, 6, 7

LLM Large Language Model. 19

OEM Original Equipment Manufacturer. 3, 4

RAG Retrieval Augmented Generation. 31

SQL Server Query Language. 4

UI User Interface. 24

Contents

List of Figures

List of Tables

1
Introduction

During conceptual and preliminary design phases, various concepts and designs are taken into consideration. When it comes to geometrical modelling, NX is the most used Computer Aided Design (CAD) software for "Engines". These CAD processes are iterative processes that go through multiple revisions and updates in order to get to the final version, and be accepted as a suitable model. Once this process is complete, the generated concept needs to be structurally evaluated for which a Finite Element Model needs to be generated. Generating FE (Finite Element) models in-house involves tagging geometrical entities for analysis.

Structural Analysis of engineered components is a major process at GKN Aerospace Sweden. This process usually consumes about 30 to 60 percent of the resources in the sustained engineering effort. Structural analysis is important for any engineered component, as it provides a way of understanding how a component will behave in its operating environment. This can be simulated with the help of specialized software designed exactly for this purpose, allowing the simulation of loads that the component is supposed to withstand during operation as well as the geometrical constraints that need to be taken into consideration. The analysis process consists of three main steps; creating the CAD model followed by preparing the CAD model by applying loads and other relevant boundary conditions (these are decided based on the type of analysis selected). The final step involves simulating the model and analyzing the results.

Pre-processing the FE model while preparing for analysis is considered as a highly time consuming and labor intensive task at GKN Aerospace. It is estimated that an analysis engineer spends around 25 to 50 percent of their time preparing geometry, applying loads and constraints and making sure that the generated model can "run" on a computer or a server. This project takes a look into one of the key aspects of model pre-processing, known as tagging of the geometry.

The act of tagging a model essentially means selecting a "face" or a "feature" in a CAD model and assigning a name or certain attribute to this geometrical entity, that can later be used to identify it based on its requirement and uses. For example, tagging can be done to identify the location of forces acting on that part or for identifying areas that require specific mesh requirements etc. This simplifies the model and allows for the application of loads, boundary conditions, and specific mesh requirements in designated areas.

GKN Aerospace is currently one of the leading aerospace suppliers (Tier-1) on the market. GKN's Engine department employs about 2000 employees and they continue to invest a significant proportion of their R&D budget on enhancing fuel efficiency and reducing emissions within aircraft (GKN Aerospace Website). GKN Aerospace has design responsibility for some of the major components used in the manufacturing of commercial aircraft engines like turbine structures with engine mounts for the GEnx and GTF engine families along with the design and manufacture of complex fabricated core compressor structures. As of 2023, for various engine OEMs in the industry, GKN provides engineered components, parts repair and commercial aftermarket parts for civil as well as defense aircraft engines along with a 3% share in electrical and structural systems

for air-frame OEMs (mel, 2023).
GKN is engaged in the design, development and certification of fan containment casings as well as fan frames and fan core structures for the civil as well as military aerospace engines (GKN Aerospace Website).

Figure 1.1: The Geometry tagging process

1.1 Background

As shown in figure (1.1), the geometry selection ("tagging") process is done after creation of the CAD model and before the model is sent for FE analysis. The first step involves creating a CAD model, which is followed by tagging the model, and then the model is sent to analysis once the tagging process is complete. The tags are created depending on the purpose and the type of analysis that needs to carried out on the model. The tags may be used for creation of the mesh, for making selection recipes in NX or for mesh refinement and post processing. Tagging can be done by the analysis engineer or the design engineer depending on the requirement. However, the tagging is inconsistent between each iteration of the CAD model requiring analysis engineers to manually identify the required geometries that have been left out.

An automated tool that can be used to tag geometrical entities and surfaces was developed at GKN, which helped automate the tagging process to some extent. Tagging a CAD model using the automated tool is one of the ways in which "tagging" can be carried out. The process used by the tool shown in figure (1.2) uses SQL queries in YAML to ensure that the correct CAD model and geometries are selected. YAML is a data serialization language that is human friendly and is

compatible with most programming languages out there. (YAM).

This "Autotag" tool has the capability to automatically select geometric entities with the help of YAML code, which the user needs to input. This capability, while having significant potential, is severely under-utilized since it lacks an extensive and friendly user interface. At the same time, the use of the tool requires some technical know-how in order to be utilized to its full potential. Similarly, to select the required geometry using YAML, the user needs to be aware of and be able to generate YAML code that accurately represents the various relationships between different surfaces and features, which is not always possible.

When it comes to syntax and format, YAML as a serialization language is unforgiving, which makes YAML code prone to errors related to formatting and syntax that can be difficult to resolve easily. The engineers that use this tool need to be able to type YAML code in the right syntax in order to use its capabilities entirely, something which not all analysis and design engineers are familiar with. This leads to reduced adoption of the tool.

Some example YAML code used for the selection of geometrical features using the tool is given below:

```
1     - filter: select * from faces where type like "Cylindrical" and radius < 2
2     - filter: select * from faces where normal_xangle between 70 and 110 and
      normal_yangle between 70 and 110
3     - filter: select * from faces where x < max(x) - 10 or x > min(x) + 10
```

From the given example it can be seen YAML requires a specific format along with the fact that the user needs to be able to figure out and express the various relationships between geometric features in order to make an accurate selection.

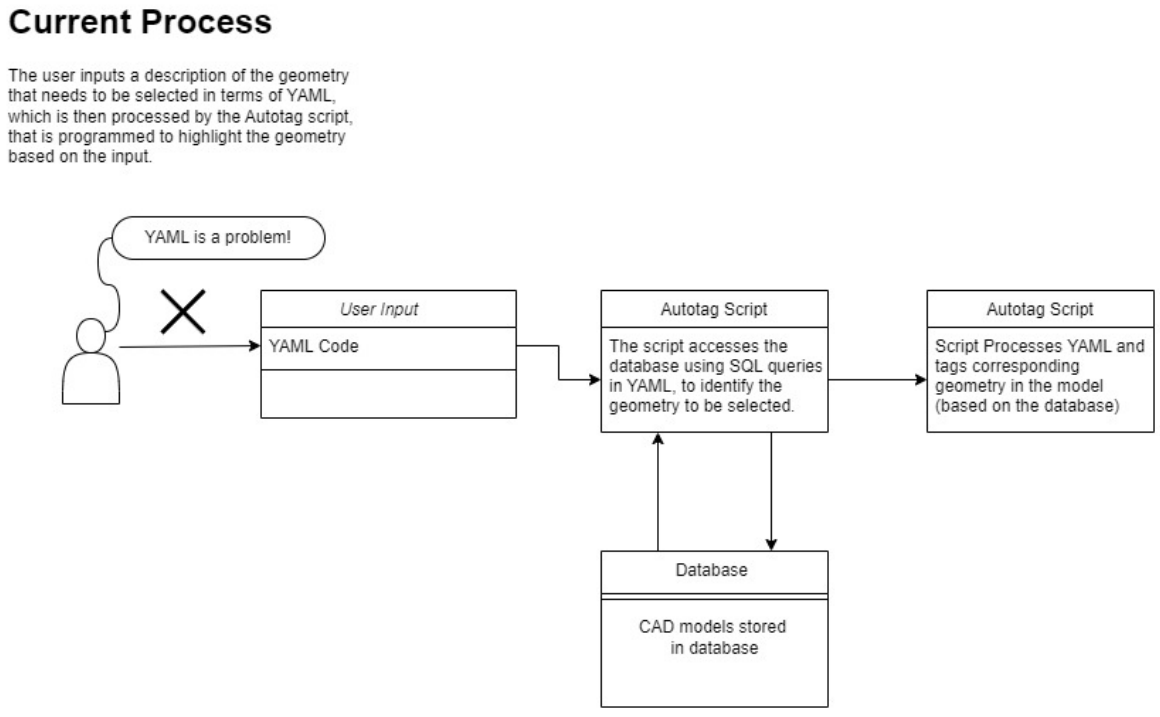

Figure 1.2: Current method in Simcenter

1.2 Aim

The aim as expressed by GKN in the project challenge was to investigate and showcase the possibilities of implementing and integrating a Large language model (LLM) such as GPT 4.0 Or Mixtral to identify geometric features and entities in CAD and generate SQL queries in YAML to extract and select the required geometric features. This would streamline user interaction with the tool by simplifying geometry identification. A successful outcome of the project, along with implementation of the solution will also contribute toward increased adoption of the in-house automation tool (Autotag Script).

With the help of the envisioned solution, the aim of the research can be narrowed to addressing the problems faced by the engineers at GKN today in effectively identifying and tagging the correct geometry with as little error as possible, specifically while using the "autotag" tool. The solution will also be aimed toward improving the user interface of the tool making it simpler to use and improve adoption of the autotag tool.

In order to effectively implement AI, LLMs and other associated technologies it is important to understand the way they work, its advantages, disadvantages etc. This is done with the help of the research questions formulated for the project, as well as relevant information presented in section (3.2.1).

1.3 Limitations of the project

The limitations of the project are highlighted in this section, which sets some boundaries for the areas of focus regarding the research, as well as limitations with respect to the amount of time and financial resources available for the research. It is also important to note that the project only aims to address the tagging process utilized by the "autotag" tool i.e., the use of YAML code for selecting geometrical entities.

- Project duration limited to 20 weeks
- The budget for the project is estimated to 200 USD for LLM services.
- A possibility to explore multi-modal models to handle text as well as 3D models was hypothesized along with training a machine learning model to understand CAD data. However, due to computational restrictions as well as a limited time frame, this method was not pursued.

1.4 Specification of Issue being Investigated

The tagging process can be done in various ways, depending on the user and the software being used for the process. The most common way in which this is carried out is selecting the geometric surfaces and features manually. While manually tagging the geometry can be a fast and efficient process if the CAD model is not complex, this is not always possible since the CAD models being used are complicated, having intricate geometries and inner surfaces that cannot always be manually selected and tagged.
The research explores the the ability of LLMs to not only generate YAML code based on natural language input, but also generate natural language feedback for the generated YAML code, thereby showcasing capabilities of the models to generate code relevant to natural language input.

1.4.1 Research questions

Based on the aim of the project mentioned earlier, the following research questions were formulated with the goal of answering them through the outcome of the project.

1. What are the challenges that engineers face today to generate fast and robust CAD tags for FEM meshes?
2. What are the unique advantages and disadvantages that LLMs face when dealing with engineering tasks?
3. How could AI and LLMs support the identification of CAD geometry using engineering terms?

1.5 Ethical Considerations and Sustainability

Language models can indirectly lead to 'leakage' of data provided to them which is why from a global as well as company perspective, it is important to take some ethical aspects into consideration, especially since in its implementation, confidential data will need to be processed.

1. **Bias:** Since large language models are trained on vast amounts of data that cannot always be supervised, some of the bias and discriminatory tendencies of the data tend to get embedded with the language model itself. This needs to be taken into consideration and avoided in order to maintain ethical integrity of the research (Bender et al., 2021). For example in the case of this research, the output generated by the language model could be affected by the language in which the user interacts with the tool, which could originate from the language's representations in the training data. At the same time, bias could also originate from the fact that the training data may turn out to be relevant for some analyses, and not so relevant for others. In this case if the user tends to trust the generated output without critically evaluating it, could lead to a false impression of trust.

2. **Creativity and ownership:** Due to their very nature, large language models or any generative architecture for that matter cannot be held accountable for the outputs that they generate, which introduces a legal gray area regarding the use of content generated by these models. At the same time, the data which is used to train these models (in case of using pre-trained models) is not transparent, which makes it difficult to verify the integrity in terms of quality and truthfulness of the generated content (Lappin, 2024; Liu et al., 2023).

3. **Privacy and Security:** Large language models require vast amounts of data to be effectively trained, which includes the use of personal as well as company relevant data. This leads to increase in the risk of invasion of privacy and leakage of confidential data. As a way of addressing this issue, it is possible to host and run language models locally, preventing confidential data from ever leaving the network (University, 2023).

Large Language Models require a significant amount of computational and financial resources making them extremely environmentally as well computationally expensive. These costs can be expressed in terms of their impact on the environment. As mentioned in the article (Strubell et al., 2019), the environmental impact of training a single language model, generates carbon emissions which are equivalent to 300 flights from New York to San Francisco. However, this metric applied to training a large language model from scratch. Compared to training a language model, the environmental and financial impact of using a pre-trained model either through an API or by hosting it locally should be of a significantly lower magnitude. However, this will also

depend on the type of model being used, the size and performance capabilities etc. Considering the comparatively smaller environmental impact of singular interactions with the model, these can add up over a period of time leading to an overall non trivial impact (Jiang et al., 2024).

With the improvements in the performance of these models, there have also been developments to training processes as well as awareness of their impacts on the environment. As a result, new tools and methods have been proposed for tracking and reducing the environmental as well as financial impact of creating, training as well as hosting generative models altogether (Ligozat et al., 2022).

2
Methodology

This section talks about the strategy used for collection of data relevant to the project, the ways in which the collected data was processed and interpreted and the methodology for developing the proposed solution.

2.1 Data Collection Methods

The following data collection methods were employed for collecting detailed information about the research as a way of developing a structured and comprehensive understanding of the topic to be addressed.

1. **Interviews:** Interviews are a method of qualitative data collection that is commonly used in research, being one of the frameworks that allow easy recording of collected data in a well structured way. For this project, semi structured interviews were used, as the idea behind this method was to present and answer a set of predetermined questions targeted towards answering the relevant research questions (Jamshed, 2014). Further information about the interview population and selection criteria is given in section (2.2.1).

2. **Literature Reviews:** A literature review is a summary of existing knowledge and information about a topic. The source of this knowledge is generally considered to be articles published by genuine and well accredited scholars or journals that are well known in the scientific community. The purpose of a literature review is to convey the existing ideologies and knowledge about the topic being researched to the reader in a structured and effective way. Literature reviews are a good way to understand the advantages and disadvantages of the research topic, and highlight ways in which this information can be leveraged by the researcher (Taylor, 2024).

2.2 Problem Clarification

The data collection methods highlighted in section 2.1 were used for further clarification of the problem in which:

- Interviews with company engineers and relevant personnel were used for answering Research Question (1)
- Literature Reviews were used for answering Research Question (2) and Question (3)

This helped indicate the capabilities of the technology, along with its advantages and limitations in order to brainstorm solutions.
This phase also involves understanding the process and its related information along with the working of the existing "Autotag" script by looking at company documentation related to the

process and examples that were generated specifically for this project *without* any confidential information.

2.2.1 Interviews

Interviews were selected as a data collection method as it was evident that it would prove effective in understanding the perspective of the company with respect to the tagging process. At the same time, it also provided better understanding of the problems that engineers using the tool on a daily basis are coming across. While surveys and other data collection methods were also considered, they were discarded after considering the fact that not all engineers would have the time to answer surveys online, along with quantitative requirements that would pose a limitation for conduction of surveys.

The interviewees were selected on the basis of their knowledge of the tagging process along with the requirement that they be engineers who use the tagging process at-least in some way. It was not deemed necessary that the engineers should be using the Autotag tool, but the tagging process in general, in any of the FE software used across GKN Aerospace. This changes from project to project since some engineers work with ANSYS and some choose to work with Siemens NX and Sim Center.

At GKN Aerospace, this process is mainly carried out by the Design Engineers who make the CAD model. However, Analysis Engineers also use the tagging process in case they need to make changes to the CAD model before conducting FE analysis. This helped narrow down the criteria for selection of the interviewees.

The criteria was narrowed down to engineers who either tag the CAD models themselves, i.e., the design engineers and the engineers who used pre-tagged models and use the tagging process but not very often, i.e., the Analysis Engineers. Some of these engineers were also ones who did both and also had some firsthand knowledge about the working of Large Language Models. This helped provide a detailed idea and possible foresight in envisioning a solution that could address their problems with LLM.

Criteria	Eng. 1	Eng.2	Eng.3	Eng.4	Eng.5	Eng.6
Does tagging themself						
Uses Pre-tagged models						
Knowledge about LLMs	Medium	None	Little	Medium	Experienced	None

Figure 2.1: Interview Population

Interviews conducted were used to collect contextual information tailored to answering question (1). Each interview lasted for a duration of approximately 30 minutes on average. The interviews were conducted with predefined questions targeted towards collection of maximum possible information regarding the research objective. The questions used for the interview process were open-ended to a certain extent, allowing interviewees to elaborate on points of interest (Denscombe, 2014). The process of formulating and conducting the interviews lasted for about 2 weeks, wherein, as mentioned earlier, engineers and other relevant personnel from 3 different projects were interviewed. The saturation point was achieved after 6 total interviews, since no significantly new information was available at that point.

The interviews were authorized by the necessary authorities and personnel within the organization and the content and collected data was shared with the relevant personnel as well as the interviewees in the event that they asked for it. Export control policies within the organization were followed during processing as well as utilization of the collected data. The interviews mostly took place on company premises, with a few exceptions which were conducted online albeit still on the company network. All the interviews were recorded with the consent of the interviewee, which made it easier to transcribe the same (Denscombe, 2014).

2.2.2 Literature Review

Literature reviews were carried out as a way of data assimilation in order to better understand the quality as well as quantity of information already available. The literature review was targeted towards answering question (2) and (3) mentioned in section (1.4.1) and the findings are presented as a summary in section (4.2) along with the list of articles that were considered, mentioned in appendix (B). The literature that was reviewed was collected from reliable sources including but not limited to Science Direct, Scopus etc. in order to ensure certain aspects about quality of the research articles as well as the information it contained. ArXiv as a source was also taken into used, however with certain criteria about quality in consideration, since articles on ArXiv are not peer reviewed.

The following criteria was used for selecting research articles:

- The name of the article and the author, cross checked to verify if it has been published by an author or a journal that is indexed by Scopus.
- Articles having at-least 30-50 citations if the article has been published within the last two years, and at-least 5-10 if the article has been published within the last 6 months of writing this report will be considered to ensure reliability of the obtained information.

This was useful in effectively understanding current state of the art while also maintaining integrity and quality of the articles reviewed. At the same time similar researches helped provide better idea about potential feasibility issues in conducting the research and achieving effective results.

2.3 Data Analysis

The information acquired from the chosen methods was analysed with respect to the guidelines laid down for analysis of qualitative data (Denscombe, 2014). Reviewed research articles are listed in a table with key search terms. A summary of the findings from the collected articles was utilized as a way to leverage current knowledge on the topic for improved solution development (Denscombe, 2014).
The interview transcription was carried out in the standard format, with annotations and line numbers. The information extracted from the interviews was sorted and summarized according to themes interpreted in the data. At the same time the data was properly catalogued and indexed wherever required while maintaining multiple copies of the data for ensuring safety (Denscombe, 2014).

2.4 Data Validation

Validation of collected data is an important aspect in order to maintain reliability and usability for the intended purpose (Denscombe, 2014). The takeaways from the interviews and literature reviews as mentioned in section 4 were discussed with the supervisors and the department managers at

GKN Aerospace in order to make sure that the issues being addressed are realistic problems that can be tackled. At the same time, a detailed description of the decision making ideology and procedure is included in the report to ensure dependability of the data.

2.5 Solution development approach

The solution development phase was carried out with the aim of addressing the problems that were identified. This laid down the foundation of the first iteration of the application, while successive iterations were based on an agile software development approach, in which each phase (discussed in detail in section 3.2) consisted of developing a version of the solution that aims to solve at least one of the identified problems and then testing that solution as a way to identify advantages and limitations of the approach. These advantages and limitations would then be used to address the next phase of development. This cycle, which follows the principles of agile software development was carried out until the proposed solution and its expected level of robustness was achieved.

With respect to the final solution, the testing of the solution involved analyzing the output of the language model and the developed code with respect to two main categories:

1. Quality of generated output/response
2. Robustness of the generated response, with variables such as models, user as well as CAD geometries. This helped ensure that the results are repeatable as well as reliable.

With respect to the categories mentioned earlier, the test criteria that was needed to analyze the output of the developed solution was generated. The test criteria was such that it needed to take various aspects of Language model 'use' into consideration, such as the amount of time it takes to complete a required task, the cost associated with that task, the consistency with which it accurately completes the task etc. This can be used a way of quantitatively assessing the performance of the language model.

2.6 Discarded Data Collection Methods

This section outlines the data collection methods that were initially taken into consideration as supplementary methods to the two mentioned in the earlier section. However, they were discarded due to the reasons mentioned below:

2.6.1 Surveys

Surveys are an important and somewhat effective method of quantitative data collection. However, this method was not used due to lack of time available for data collection, limited availability of people and the response rate required for effective quantity of collected data would not be sufficient in the available time (Denscombe, 2014). As a result, it has been added as an alternative method to answer questions (1) and (2) as a way of supplementing the information collected through interviews.

2.6.2 Design of Experiments

This method can be an argument for understanding the existing problems in a better way by providing a deeper understanding of the causal factors of the problems. Some of the advantages of this method is that experiments by their very nature are repeatable, which makes it a dependable strategy of effective data collection in this case. Experiments can be designed in a way that helps

understand the performance of the model and collect important insight about the requirement of further development or possible improvements in the models. It should be kept in mind, that ethical considerations need to be more effectively evaluated when experiments come into play due to the inherent nature of this data collection method (Denscombe, 2014). Designing and implementing an experiment that provided quality information about the data needs to be collected would have been a time consuming task. Since the duration of the project was limited to 20 weeks, this method was not used.

2.6.3 Observation

Observation could potentially be used as a method to collect additional data to supplement the primary data collection methods mentioned in section 2.1 in order to get a better way to answer question (3). Being able to observe the engineer perform the task might be able to provide valuable insight related to subconscious practices that vary from engineer to engineer, and also possibly identify and point out any bias that might be affecting the process in itself, but may not be directly visible. This method requires observing the activity being performed. The idea is to talk to the engineers and look for an opportunity to observe the process that is currently used to perform the task at hand. However, since these tasks (tagging of geometric entities in the CAD model using Autotag script) that are not performed everyday, collecting data using this method would be difficult and time consuming.

Figure (2.2) shows an overview of the development phases and the way the obtained information was leveraged to achieve the proposed solution.

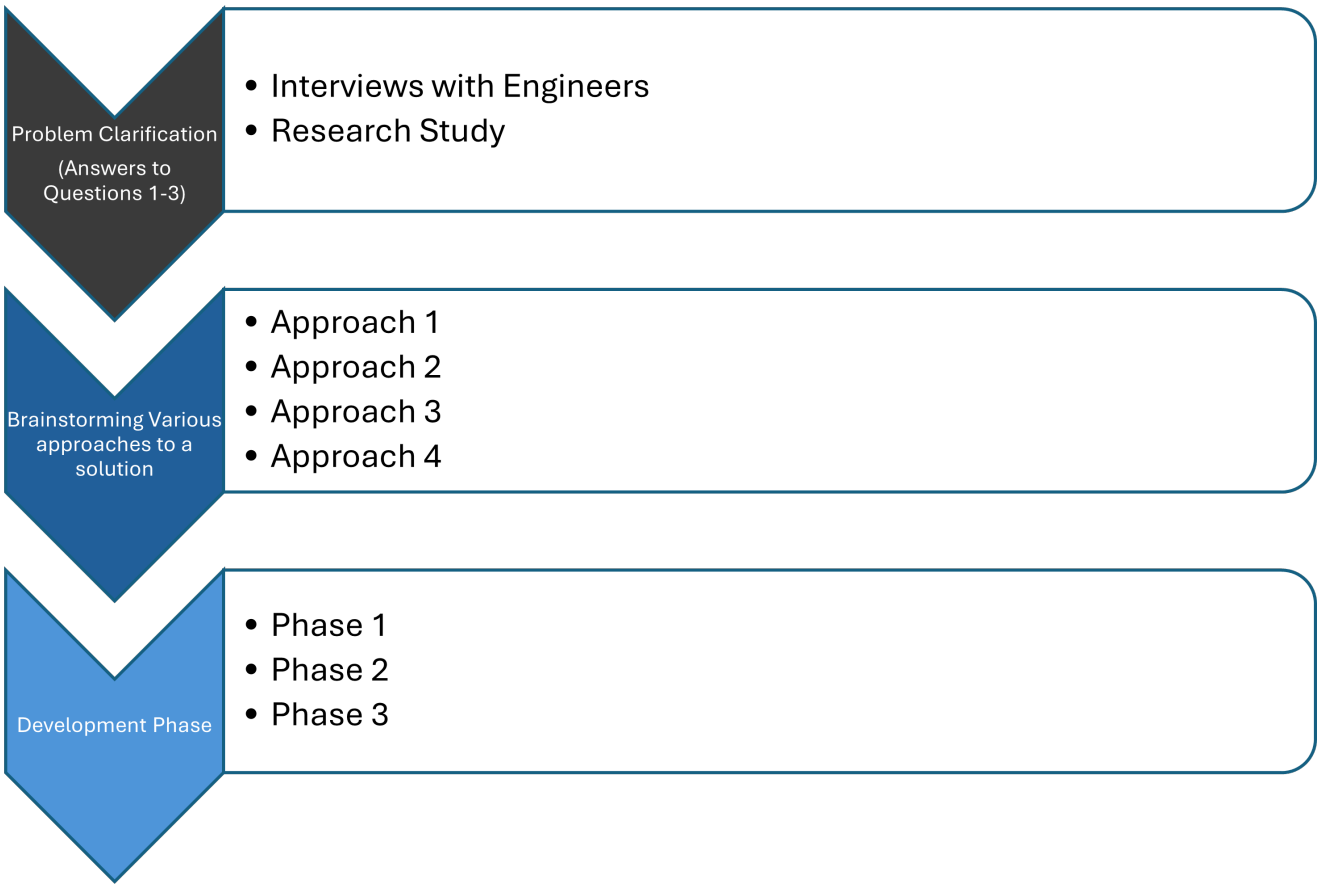

Figure 2.2: Development Process

3
Implementation

This section contains a detailed description of the steps taken during implementation of the proposed methodology. The various stages that the solution progressed through over the duration of the project is outlined in the solution development ideology in section (3.2). Section (3.1) talks about the various approaches used in building the basic architecture of the proposed application, with strengths and drawbacks outlined for each approach.

The completion of the data analysis phase was achieved with the finalization of the problems to be tackled when developing the solution and understanding the ways with which this could be achieved. This was done with the information collected through the interviews (4.1) and literature reviews (4.2) wherein the advantages and disadvantages of the use of artificial intelligence, more specifically large language models were discussed and then potential ways to implement geometry identification that have already been published were reviewed. After careful consideration and discussions with the supervisors of the project and managers at GKN, the issues to be addressed were finalized in section (4.1.1). The possible solutions that take these issues into consideration are mentioned in the following section.

3.1 Considered solutions

Multiple architectures for developing the solutions were brainstormed as a way to find the best possible outcome within the available amount of time. The idea is to implement a solution that addresses as many of the challenges identified through the interviews, and augment the existing capabilities of the "Autotag" tool. This section includes all of the possible ways in which such a solution could be addressed, implying that the solution showcased in section (4.4) is not the only way in which the expected results could be achieved.

3.1.1 Approach 1

One of the solutions, shown in figure 3.1 below, shows the initial approach to generate YAML code that is used to automatically select geometric entities based on a given database that is used to understand the geometric representation of the model in SQL. Since LLMs have the capability to process natural language, this approach uses that ability to convert the user's natural language input to convert that into YAML text that is in the same format as that accepted by SimCenter to automatically select the geometry.

In this case, the YAML output generated by the LLM is directly and automatically populated to the "enter query" section of the tool, which then highlights the geometry depending on the query that is provided. This approach is simple and effective in the sense that it does not change the core process of the way the "autotag" tool handles YAML queries. At the same time, it reduces the amount of effort required from the user for typing and formatting the queries in YAML, while making sure that there are no spelling mistakes or typos in the code. The database schema that

is mentioned in the figure below, as well as anywhere else in the report has been presented in appendix (C). This approach would not require any changes to the existing interface of the tool. The idea was to update a locally stored yaml text file on the computer automatically when a new code is generated.

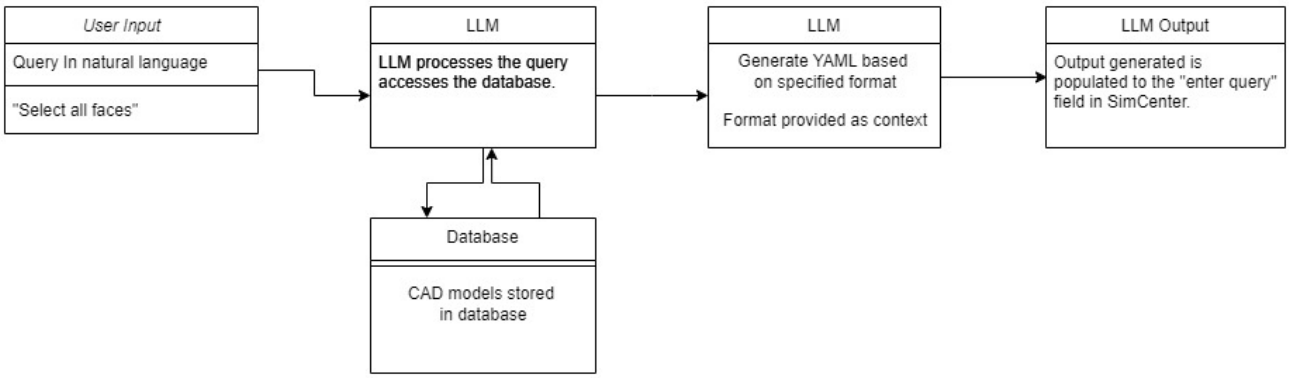

Figure 3.1: The initial approach

3.1.2 Approach 2

The second possible solution involved a similar approach, however with the help of a multi-modal model that has the capability to understand visual and spatial information. In this case, the context is provided to the model, which includes information regarding the database schema, the format of the generated YAML query and a few examples regarding the inputs and the outputs of the model. Along with this context, the model is also provided with images of the geometric model from various angles that allows the generative model to comprehend what the CAD model looks like while having access to the database, which could potentially increase the accuracy with which the model generates the YAML code.

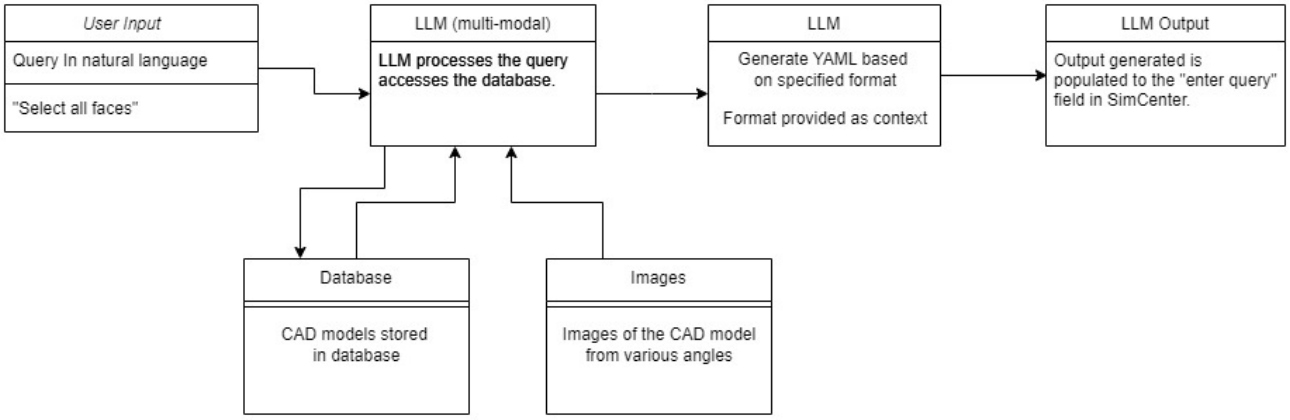

Figure 3.2: Potential solution with multi-modal models

However, in this case the generative model may not be advanced enough to understand the contextual definition of the CAD model and accurately correlate the characteristics of complex features to the database schema based on the natural language prompt of the user.
At the same time, both of the approaches mentioned earlier are simplistic approaches that do not take into consideration any sort of discrepancies in the user input. In this approach, the model would be unable to process the natural language terminology of engineering terms, resulting in a

correctly formatted but invalid YAML query. This occurs because the model cannot understand SQL definitions for words not present in the database schema.

3.1.3 Approach 3

As a result of brainstorming over the solutions and the results of trying to implement the previous solutions, some potential drawbacks and issues with these solutions were uncovered, which led to the development of the next possible approach which is shown in the image below.

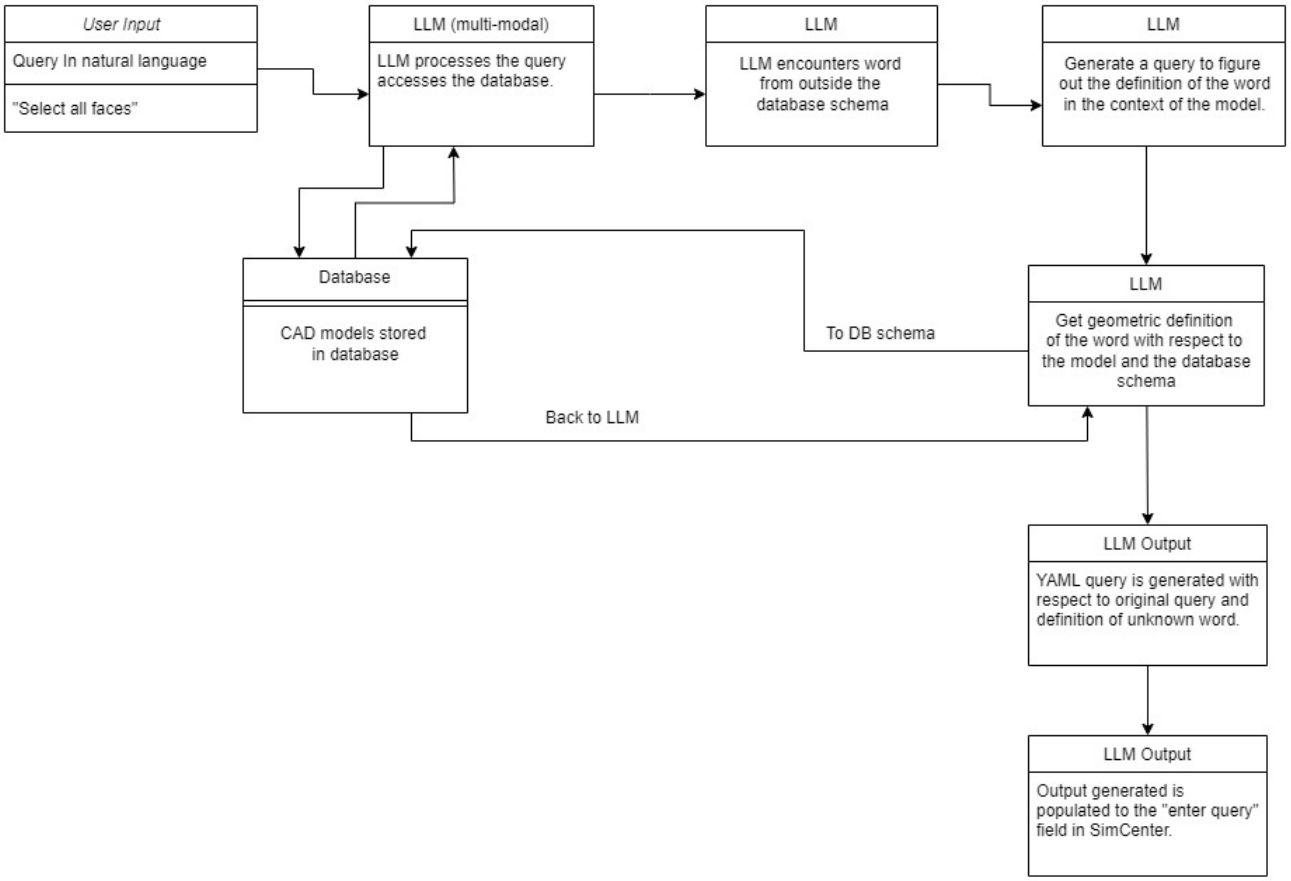

Figure 3.3: Solution approach 3

This approach addresses potential issues with parts of the query that the language model cannot process confidently. To prevent the model from generating incorrect information, it is designed to handle words or parts of the user's query that it does not understand completely. The generated query will then be used to ask the model itself, or perhaps another model, about the meaning of the word or part of the query in the context of the model. This information will be used to create a geometric definition of the term, which is then correlated with the database schema. Subsequently, the YAML text is generated by incorporating both the new definition and the original user query.

This approach could also be coupled with a multi modal model which would essentially look like a combination of approach 2 (fig 3.2) and this approach (fig 3.3), and then fine tuned to further increase the robustness of the generated YAML text. However, this model would not be practical as it would be difficult to provide new images of the model every time something in the geometry changes. The finalized approach to solving this problem is a more tuned version of this approach and is mentioned in the section below.

3.1.4 Approach 4

In this method, the idea was to implement a machine learning model and train on the SQL database in which the CAD models are stored, allowing the model to understand the geometric entities in terms of SQL and make it easier to retrieve them when required. This approach can be combined with a large language model that can then process the natural language query of the user, use the machine learning model to identify the geometry and then generate the YAML text corresponding to that request. However, this approach was discarded due to its complexity, especially with respect to the amount of time available to complete it as well as lack of a large enough dataset containing CAD models that would be required for training and validation of the machine learning model at GKN Aerospace.

3.2 Development Framework and Testing Criteria

The development of the finalized solution was carried out in the following phases, each of which represents a part of the software development that contributed in one way or another, to the findings and breakthroughs that led to the solution as presented. Each phase comprises of a small description of the approach used, the method of implementing the approach, the quality of the output in terms of accuracy and robustness which was measured in reference to the test queries that used to test the performance of the model.

The main criteria that was used to carry out performance testing of the various phases of the developed application are mentioned below:

1. **Consistency of the output:** generated by the model with respect to each call to the language model. This was tested by using a loop that calls the LLM and prompts it the same input prompt a specific number of times to analyze the frequency with which the model responds in the same way.
2. **Accuracy percentage:** Tested with the help of test queries created for this purpose. The test queries are generated such that they are of increasing level of complexity which provides a percentage increase or decrease in the performance of the model. Each query increases in complexity by approximately 10 percent, with a total of 10 queries for testing.
3. **Number of tokens used:** by the model in total for processing the required input string.
4. **Time required:** for the model to complete the run.
5. **Cost:** of each call to the language model API.

3.2.1 Tools and Technologies employed

Artificial Intelligence (AI) refers to the ability of a computer or a machine to simulate the thinking capabilities and reasoning abilities of a human, hence the name artificial intelligence (Britannica, 2024). The goal of developing this technology is to come up with an architecture that is able to replicate the process of decision making and reasoning that humans use, which we call 'thinking'. Recent convergence of advancements in computing power, hardware design, and human psychology has led to rapid and extensive developments in Artificial Intelligence, making the field very rich and diverse. Artificial Intelligence is a broad and complex field that is increasingly being utilized in various research areas. Researchers are continually exploring ways to leverage its expanding capabilities to enhance everyday life (Yongjun Xu, 2021).

Large Language Models (LLM) are a form of Artificial Intelligence, that are used for processing and generating text, among other uses. LLMs are capable of interpreting natural language and generating human like responses in relation to the input provided to them, making them versatile and helpful for applications that need to process or generate natural language in some way. Since these models are trained on vast amounts of data, the capabilities of these models if harnessed in the right way are potentially endless, allowing them to be easily adapted to any requirement. LLMs as a tool are simple to interact with since they only require some form on natural language input, making it easy for users of all experience levels to easily interact with (Bassel Alamrie, 2023).

The developed solution uses a number of different technologies and ideologies, which in a way aid the developed application in achieving the desired result. They are as mentioned below:

The following LLMs were taken into consideration, based on multiple factors such as their quality indices, speed of processing tokens, and cost of use. These models were chosen specifically as they provide two categories of low cost, high performance such as GPT 3.5 and Gemini Pro which have a very high cost to quality ration. On the other hand GPT 4 and Claude 3 Sonnet are flagship and high performing model that are significantly more expensive than that of Gemini Pro and GPT 3.5.

Gemini Pro and GPT 3.5 Turbo are also the models that can be considered as almost equivalent to the performance of smaller open source models. The performance of these models can therefore help give a better idea about the performance of locally hosted open source models as compared to flagship models,since it was currently not possible to implement and test open source models due to technical limitations in their implementation and lack of sufficient time.

1. **Google's Gemini Pro** (gemini-1.0-pro-002) : Gemini Pro has a large context window and low cost, and is a very versatile model when used for small tasks that require low latency. It has a context window of 32800 tokens and a cost of USD 0,8/1M tokens.

2. **OpenAI's GPT 3.5 Turbo** (gpt-3.5-turbo-0125): GPT 3.5 Turbo has a lower context window than Gemini Pro, at 16800 tokens coming in at a cost of USD 0,75/1M tokens which puts it in the same category as that of Gemini Pro.

3. **OpenAI's GPT 4** (gpt-4-0163): GPT 4 is one of OpenAI's most capable models, but however has a small context window of only 8192 tokens with a very high cost of almost USD 37,5/1M tokens making it one of the costliest LLMs of its size.

4. **Anthropic's Claude 3 Sonnet** (claude-3-sonnet-20240229): Claude 3 Sonnet is one of Anthropic's mid range models, having a context window of 200000 tokens and a cost of about USD 6/1M tokens making it extremely well suited for a combination of good performance and comparatively inexpensive.

The above information was obtained from comparing the selected models and is available from (com, 2024).

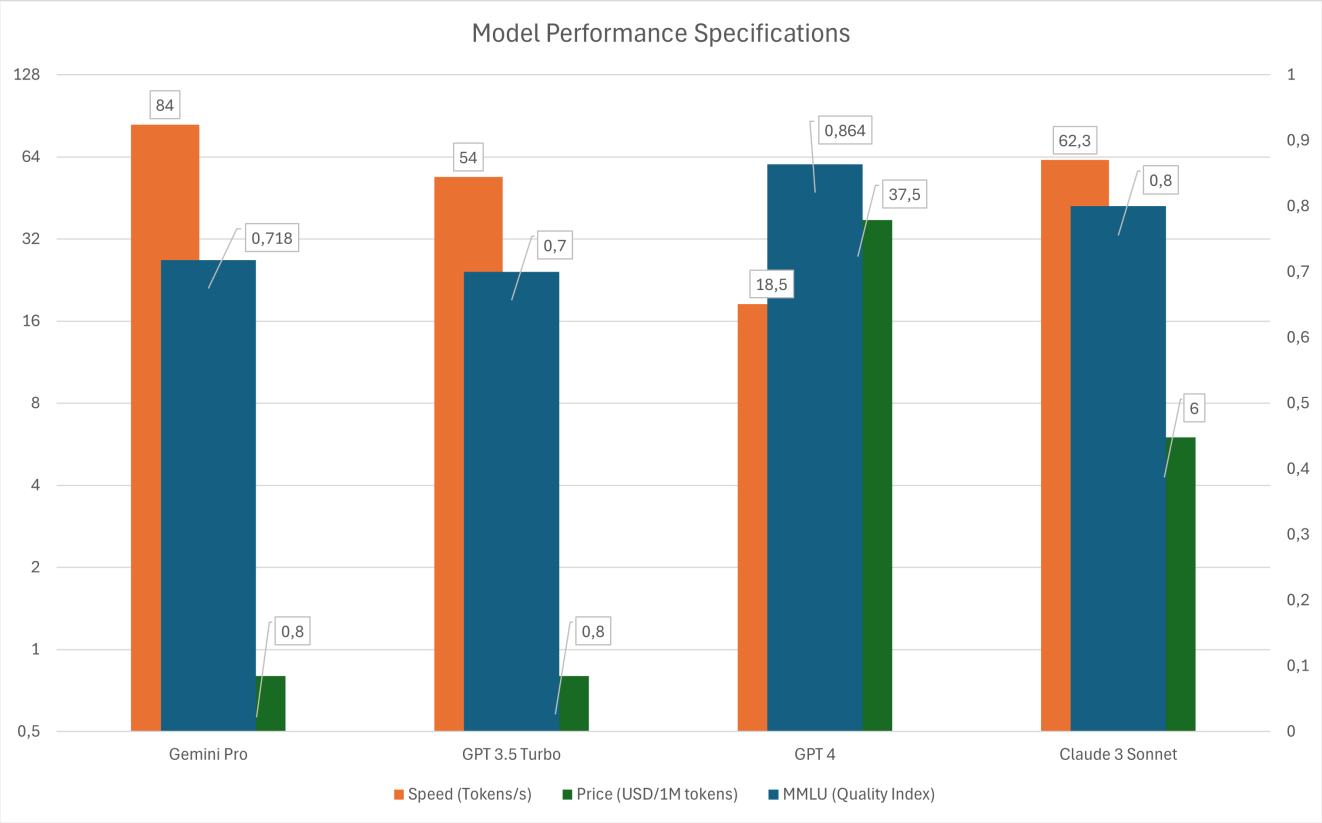

Figure 3.4: LLM specifications

It is understood that there are bigger and more powerful models currently out there and being developed, however the models and the results generated by these models represent a small part of the overall capabilities and possibilities of the use of LLMs for such a purpose.

Langchain: Langchain can be defined as a framework, which is used as a way of developing applications that make the use of generative models, or language models in particular, while allowing developers to easily integrate them into other applications with relative ease. This framework also makes it easier to interact with a language model API as well as other data sources. Langchain also provides a modular approach when it comes to using and testing different language models with varying configurations without requiring major changes in the code (Topsakal and Akinci, 2023).

Langchain utilizes a pre-defined structure for communicating with the language model, which includes two things:

1. A way to provide the model with some context about the task that it is supposed to carry out
2. Provide a response

These actions are mainly carried by prompts. Prompts are defined as a way to provide the model with the user input, as well as a set of instructions and guidelines that tell the model what it is supposed to do. Langchain provides various classes as well prompt templates to help the developer utilize the full potential of the language model as well as the framework (Topsakal and Akinci, 2023).

Figure 3.5 shows a general idea of the process that takes place every time a call to the language model in invoked. Langchain uses a similar architecture to process LLM calls. In some of the development phases mentioned below, every instance of a separate prompt and call to the LLM, a similar architecture is used but with the instructions to the LLM tailored according the expected

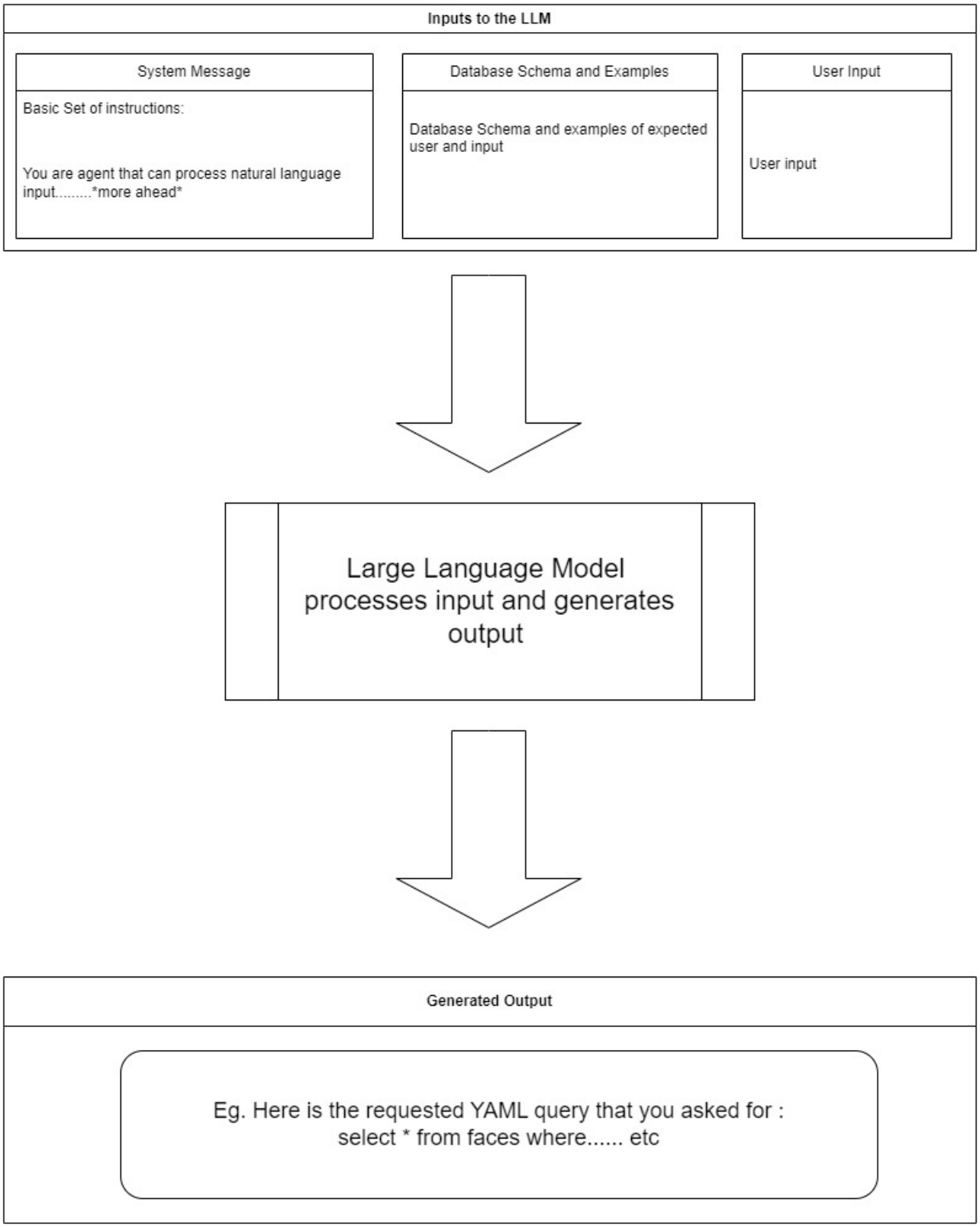

Figure 3.5: LLM Architecture in Langchain

input and required output. This has been represented with the process workflow in each of the phases in better detail.
The framework also has support for local as well as other models, along with the ability to implement custom LLMs if required, which makes it a versatile framework with potentially endless possibilities.

LangGraph: LangGraph is a part of the Langchain framework, which is used to create a network of different actions, and connect them to each other via nodes, essentially creating a network with a series of rules that are followed according to a specific pre defined logic. This process in python is equivalent to creating a flowchart or a feedback loop in which each action is taken as a result of the previous action, guided by a specified logic. With the help of LangGraph, multiple language models and agents with varying configurations can be wired to each other in the same application, allowing creation of cycles or loops in case it is required (lan).

Retrieval Augmented Generation (RAG): Retrieval Augmented generation or RAG is a method of improving the accuracy of a generative model, by providing it with the ability to extract information from external sources, i.e., sources that may or may not be available in its training dataset. In this way, it is possible to provide the model with requirement specific information that allows it to provide better results and reduce the tendency to hallucinate due to lack of enough data (Lewis et al., 2020).

The langchain framework provides a better understanding of the steps that are carried out and the ways in which RAG can be effectively implemented in any LLM application which are:

1. Query Analysis : The query is analyzed and decomposed to provide a better way for the model to extract all relevant information.
2. Routing: This step determines the nature of the query and routes it to the most relevant database.
3. Indexing: A way which consolidates and accesses the required information whenever required.
4. Reason/generate: Once the required documents are extracted and analyzed, the content is summarized and provided to the model in the form of context, which is then used for additional reasoning information.

Vector Databases: A vector database is a type of database that has the capability to store large amounts of data in a higher order form of storage known as vectors. The vectors are generated with the help of an embedding function or another tool, that converts raw data into vectors. A vector is essentially a mathematical or higher order representation of a feature or a text. With this in mind, it can be seen that vectors can be created for almost any form of data, which is then used to create a database from which relevant information can be extracted using technologies such as similarity search.

Similarity search uses the the vector representation of the original content, to look for similar vectors in the database. This information is then used to extract similar information. These techniques also possess the capability to show the degree to which the extracted information is relevant to the input.

In the developed solution, vector databases and RAG techniques are used to extract CAD model specific information that is locally stored in a vector-store to provide the model with additional information whenever required.

3.3 Solution Development

The upcoming sections of the report discuss the various phases of developing the solution, giving an overall idea of the way in which the approach to developing the application changed based on the takeaways from each phase. Each phase represents the progress in development based on the advantages and disadvantages outlined in the previous phase, thus making sure that the next phase of the development addressed major concerns with respect to the test criteria such as rate of success, consistency etc.

The developed and implemented solution bases its foundation on using the answers to the 3 research questions, which ensures that the solution and its use helps showcase the capabilities of this technology and at the same time also showcases the ability and extent to which the research questions have been answered. The ways and extent to which all of the research questions are answered is discussed in section (5)

3.3.1 Phase 1 (Model: Gemini Pro on Google cloud Vertex-AI)

This phase was the first iteration, which was developed over two versions, the descriptions of each version and its characteristics are given below.

3.3.1.1 Phase 1, Version 1:

This version was essentially a trial phase, which involved understanding whether Large Language Models possess the capability to generate the YAML code in the required format. This first trial was conducted directly on the Google Bard website, wherein it is possible to interact with the model in a simple way and the existing website UI. This was a singular attempt at understanding the generative capabilities of the model and the way in which it handles instructions, if provided. Due to the more rudimentary and introductory nature of this phase, the model's ability to handle complex queries was not tested.

3.3.1.2 Phase 1, Version 2:

In the second version of this phase, the project utilized Google Cloud and the VertexAI API to facilitate the generation of SQL queries in YAML from basic user queries. A context file was introduced, which provided instructions and format requirements to the model in the form of a 'system message'. A notable challenge arose with the need to install the Google Cloud CLI on every computer where the code was to be executed, in addition to the requirement of linking a Google account to the CLI. With the help of additional context and guided instructions, the model was able to generate YAML in the required format, indicating the language model's ability to follow instructions.

One notable limitation was the model's inability to identify geometrical features without explicit examples. Additionally, the model required more precise instructions to avoid including unnecessary content, such as code fences and conversational feedback, in the output. To address this issue, a Python function was developed to filter out unwanted content from the generated output. By refining the context file with clearer instructions and incorporating more correctly formatted examples, the model's ability to generate YAML text in the required format improved. Although there was a visible enhancement in the quality and reliability of the generated YAML output, the model still struggled with consistency, often failing to produce identical YAML text for queries with the same meaning. This indicated a need for further improvements to enhance the model's ability to maintain context and fully utilize its capabilities

Advantages:

1. Reliable Output
2. Fast response
3. Low token count

Disadvantages:

1. Requires Google Cloud CLI
2. Inability to maintain consistency of output
3. Cannot identify geometric features unless explicitly mentioned in the database schema.

The context file (System Message) contains the following items, which helped the language model better understand its task:

1. Basic set of instructions
2. Database Schema (Appendix C) (Tables containing information about various features and entities)

3. Examples for specifying format of the output

The input is stored in the environment as a text file that is later converted to a string and passed on the language model when the model is called.

3.3.2 Phase 2: Implementation of Langchain

In the second phase of solution development, to address the disadvantages and limitations of the previous phase, a new framework called Langchain, designed for building LLM-powered applications, was utilized alongside OpenAI's GPT-3.5 Turbo as the language model for developmental testing. Additional information about the framework as well as the various tools and technologies used in the solution have been mentioned in section (3.2.1).
The use of Langchain made it significantly easier to test different models with varying configurations without having to make significant changes to the code itself. Langchain introduces a certain modularity to the code, allowing the models to be switched out by changing a single parameter in the code.

```
from langchain_openai import ChatOpenAI
from langchain_google_vertexai import ChatVertexAI

    # for gpt 3.5 or gpt 4 use
chat = ChatOpenAI(model="gpt-3.5-turbo", temperature = 0.5)
    #for gemini-pro use
chat = ChatVertexAI(model="gemini-1.0-pro", temperature = 0.5)
```

As shown in the example above, changing the chat variable changes the model that is used, allowing the overall structure of the code to remain the same regardless of the LLM model used. With the help of langchain, it was also possible to use few shot prompting, which is a way of providing the model with examples of inputs and outputs, outside of the system message. This makes sure that the LLM has an idea about the kind of user input it should expect, and the response that is expected by the user in return. Few shot prompting as a technique has been found to significantly improve the quality of the output generated by the model especially in terms of consistency and robustness (Schick and Schütze, 2022). As expected, the integration with Langchain as well as use of advanced prompting techniques in the code, allowed the model to better understand the generation task.

By using OpenAI, we overcame the limitation of requiring the Gcloud CLI, as OpenAI API keys are sufficient to run the model without additional installations outside the Python environment. The table below compares the performance of Gemini Pro with GPT-3.5, based on the test criteria mentioned in section 3.2. The performance of Gemini-Pro, GPT 4 and Claude 3 Sonnet are also tested in this phase regardless.

Advantages:

1. Few shot prompting improved model's ability to maintain the required format for the generated output.
2. Easy to implement and test due to langchain, allowing models to be changed without having to make changes in the code.

Disadvantages:

1. Unable to identify geometric entities using natural language

3.3.3 Phase 3: Query Analysis, Reflection and Retrieval Augmented Generation (RAG)

This phase as it turns out is a pivotal step in the direction of achieving a model that is able to tackle the problem of geometry identification using natural language engineering terminology.

3.3.3.1 Phase 3, Version 1:

The initial part of this phase focused on testing a framework called "reflection." This framework allows the model to iterate over the generated output, giving it multiple opportunities to improve the results by looping through the generation prompts. Once the reflection capability was successfully implemented and the generation of YAML became robust and consistent enough to handle complex queries, the next step was to find a method for incorporating geometry identification using natural language engineering terminology. The implementation of the reflection framework is shown in the flowchart in figure (3.6).

3.3.3.2 Phase 3, Version 2:

Earlier approaches discussed addressing the problem of identifying geometry and geometric features with the help of a machine learning model. However, due to limitations mentioned in section (3.1.4), it was important to find a way to address them in a more efficient and time saving way. This method builds upon approach (3) described in section (3.1.3) by integrating it with the developed reflection agent. This combined approach allows the LLM application to

1. Generate a definition for a geometric feature not directly found in the database schema
2. Use the generated definition along with the database schema as reference to generate YAML Text that selects the required geometry accurately.

The way it does this, is shown in the figure (3.6). The application shows the use of 4 different prompts each with its own set of instructions, such as analyzing queries, creating definitions as well as the base and reflection chains. The query analyzer runs first and analyzes the input from the user, looking for words that it does not understand. These words are filtered and sent to the LLM for which sub-queries are generated. For example, if the user says "Select all flanges", the LLM will then relate to the Database schema provided to it in the System message (system2.txt file in the repository) and identify that the word "flanges" does not exist in any of the tables provided. This word is then filtered out, generating a sub-query for it, which is a question "What is the definition of a flange?" and is then passed on to the definer. The definer is another LLM that is instructed to process questions as input and generate a CAD-related definition of any geometric feature in relation to the available information in the database schema.

The base chain shown in the figure, is the third LLM model incorporated for generating the first iteration of the YAML text, based on the direct input of the user query. This model is targeted at utilizing a way of processing known as 'system 1' thinking. It is derived from a process in human psychology known as dual process thinking, which states that the human brain uses two types of thinking processes one which is instinctive, happens as fast as possible with minimum effort to think about possible outcomes and consequences. In system 1 thinking, the decision making process involves a lot of our human instincts that have been developed over the course of our lives, allowing the human brain to apply minimal effort in making a decision. On the other hand, system 2 processes involve a more conscious effort in thinking, taking multiple outcomes and possibilities into consideration, allowing for a more reliable and accurate result, at the cost of processing time

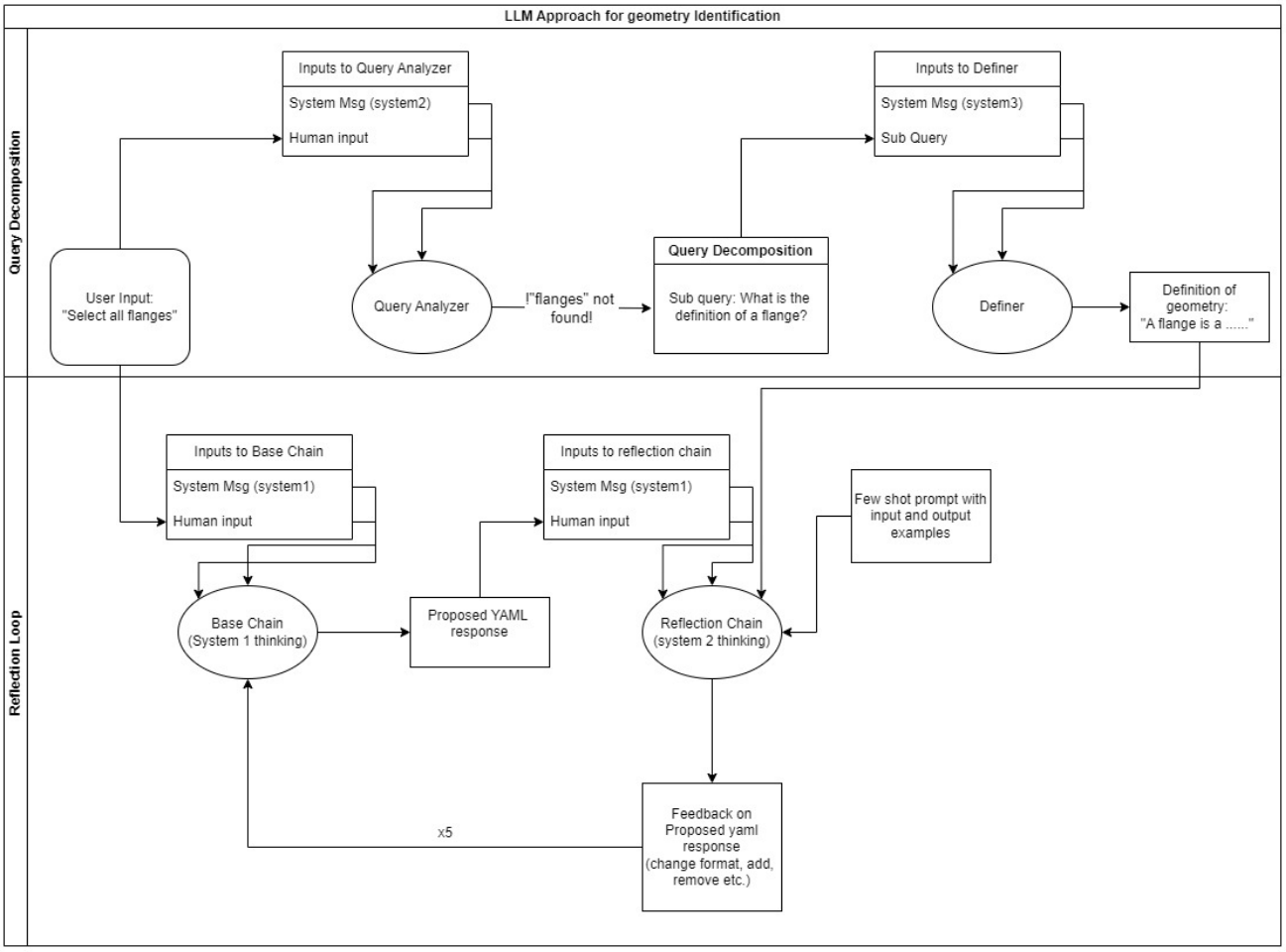

Figure 3.6: Combination of reflection and query decomposition

(T. Hagendorff and Kosinski, 2023; langchain-blog).

Similarly, the base chain uses a process that resembles system 1 thinking to generate the first iteration of the YAML text that is then sent to the reflection chain as shown in figure (3.6). The reflection chain uses a process resembling system 2 thinking, taking input from various examples and instructions that state what the input and output should look like in terms of accuracy and format. At the same time, the LLM in the reflection chain is also instructed to carefully analyze the input, provide feedback on the previous iteration and then generate a new YAML text taking the feedback into consideration. This process can be carried out in a loop as many times as necessary until the desired quality of the output is achieved.

Advantages:

1. This approach is able to provide significant advantages in terms of robustness of the output and accuracy of its responses compared to the previous versions of the code. For example, it is now possible to handle complex user queries with a considerable degree of accuracy as can be seen in the comparison table shown below, which shows a comparison of various models with and without reflection agents and query analysis.
2. Another advantage in this case is the fact that the model was now able to understand the logic behind selecting geometric features on the basis of its natural language engineering terminology. This presents a significant step forward in addressing problems that were earlier mentioned about the model being unable to understand positional references and geometrical features. The language model can also provide feedback about the best possible way the

geometry can be selected, since it is able to generate geometric definitions of the features it does not directly understand.

Disadvantages:

1. One of the major disadvantages of this approach lies in the sole use of "query decomposition" as a method to break down the user query and use that to generate a definition of unknown features. The problem in this case lies in the fact the LLM is unable to effectively understand which parts of the user query are supposed to be decomposed and sent to the model for getting additional information, which results in the model being called for geometric features that it does in fact understand, for eg., radius, plane etc.
2. The geometric definitions that the LLM generates are generic definitions that it uses from the internet and its trained data. Since these definitions are not directly related to the model being used, they tend to introduce a certain degree of uncertainty in the generated output. This means that there could be a possible improvement in the output generated, if the language model were to have access to CAD model specific definitions of features that it needs to identify.

3.3.3.3 Phase 3, Version 3: Retrieval Augmented Generation

The previous implementation can be further improved by improving the context file provided to the "definer", and adding the model specific definitions of the geometry, that will ensure that the correct definition of the geometry is selected, especially in reference to the model. Since the application is designed for tagging a CAD model where changes in the model's design implementation won't affect the core concept. As a result, the underlying geometry and its features remain consistent, allowing for easy correlation whenever needed. Like for example, even if the model and its design changes, a vane in reference to the model will always remain the same. This is expected to improve the performance of the application in terms of geometry identification, since in this way it is guaranteed that the definition that was generated by the definer is dependable, and therefore the resulting YAML text generated in reference to that definition is reliable and accurate as well.

Recognizing limitations in the prior approach's use of generic geometry definitions, a new approach utilizing Retrieval-Augmented Generation (RAG) with localized vector stores has been developed. This new approach overcomes these limitations and offers a more effective method for handling unknown geometries. These vector stores are used to store and retrieve any and all information related to the model that is being tagged. Whatever information is required from these vector stores can be easily and quickly retrieved whenever required using a technique known as similarity search.

Similarity search scans the vector stores for similarities in the data, returns groups of similar data and summarizes them, which is then fed to the Language model, in the form of additional context, making it easier for the language model to generate a more accurate and robust output. This method replaces the query decomposition part shown in fig 3.6 with a separate query analyzer module, that is able to generate a yes or no response analyzing whether the geometry or feature mentioned by the user in the original user query is directly available in the database. This response is then processed using python logic to decide whether or not additional supplementary information from RAG is required.

This phase started the proper testing of the code, with respect to different test queries having

varying levels of complexity used as input to the application, in an attempt to test the performance and accuracy of GPT 3.5 Turbo and Mixtral as mentioned in the table below. Based on the above model, the following advantages and disadvantages in terms of performance and implementation of the model were found:

Advantages:

1. With the help of retrieval augmented generation, the model is now able to provide with exact feature related definitions and other relevant data that is already provided to it, which are stored in the vector database. This data is CAD related definitions of features that are not too specific, but are relevant to the cad model being used, which prevents over fitting of the data, but allows the model to get a better idea.
2. The vector database is maintained locally, which makes it easy to add to the existing data in case changes in certain features or addition of information is required, while also making sure that the data never actually leaves the company network.

Disadvantages:

1. Query Decomposition for this use case was still not perfect in generating the right sub queries and as a result, a different approach to handle it was required.

3.3.3.4 Phase 3, Version 4:

In this version, the disadvantage mentioned in the earlier section is addressed by converting the query decomposition model into a "query analyzer" which instead of breaking down the original user query into a separate sub query, analyzes the user query, looking to extract the names of geometrical features from the user query, and look for those features in the database schema:

```
59  class analyzed_query(BaseModel):
60      """Identifying whether a geometric feature is present in the database and stating the name of the feature"""
61      answer: str= Field(description="YES or NO")
62      features: List= Field(description="Name of the identified geometrical feature (Singular)")
63
```

Figure 3.7: Python class for analyzed query

As a result, the output generated by the model in this case, is a combination of the two responses,

- answer: YES or NO (depending on whether or not the geometric feature that the user wants to select is explicitly mentioned in the database schema)
- features: (A list of the features itself that the user asks to select)

The output generated by the model is shown below for a user query that asks to select all the flanges in the model. The model is able to return the name of the feature(s) it extracted, along with a response stating whether or not it was able to find the feature in the database schema. Since the model is unable to find a geometric feature by the name of flanges explicitly mentioned in the database schema, the response is as shown below:

[analyzed_query(answer='NO', features=['flanges'])]

This output is later handled using python logic, which then accesses the Pydantic object that is generated by the query analyzer and looks at the 'answer'. If the answer is Yes, then it directly invokes the base chain which is the same as the one in the earlier version (figure 3.6) and generates the YAML string and sends it to the reflection agent for further processing.
On the other hand, if the response in 'answer' is NO, then the response is forwarded to the RAG chain, that looks for the definition of the unknown feature in the vector database containing the definitions of all required features, extracts all available information and summarizes it. This

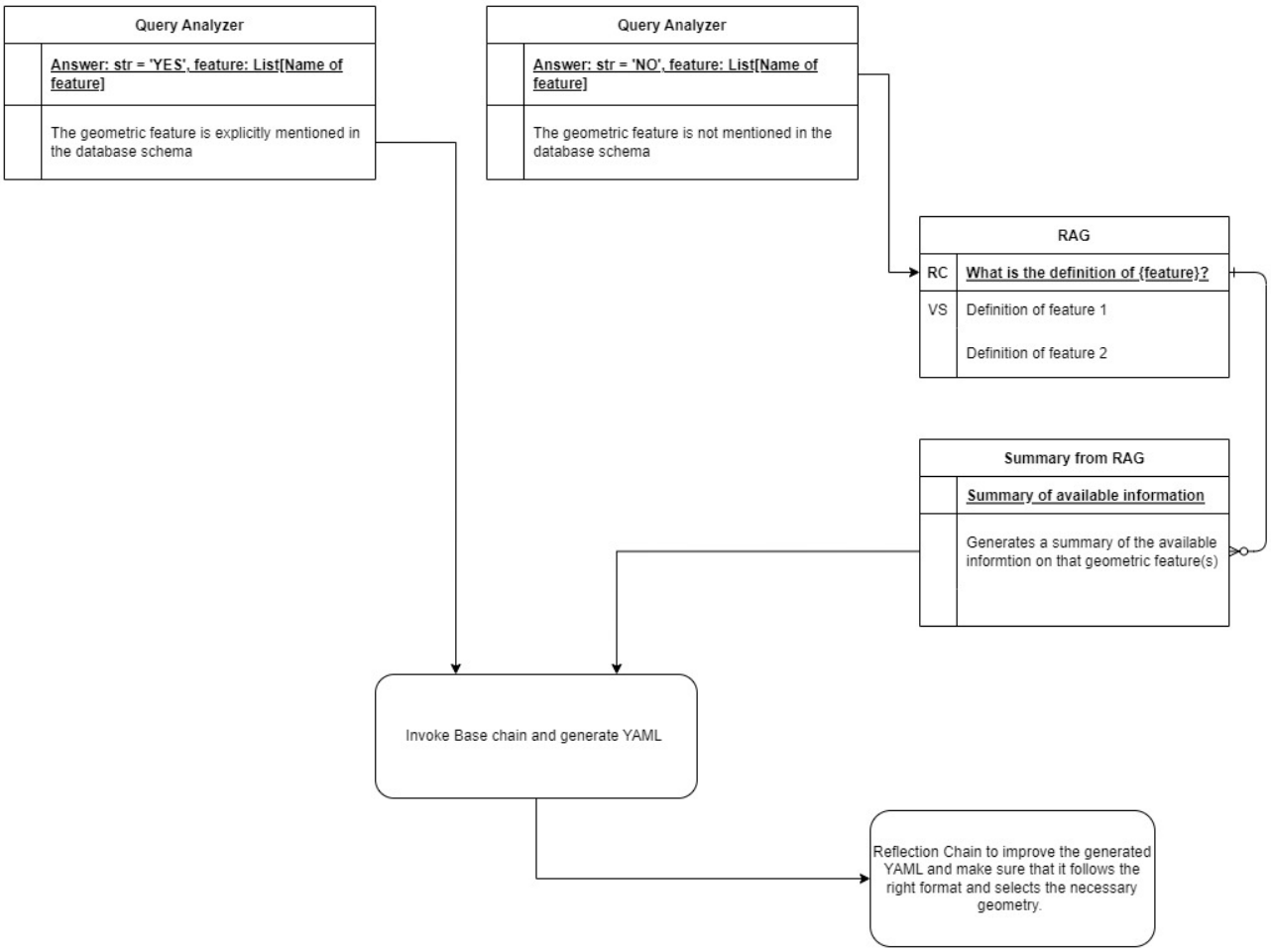

Figure 3.8: Query Analysis Logic

additional information is then provided to the base chain as supplementary information to help the model understand exactly what needs to be selected and the best way to select those features. The decision making logic is shown in figure (3.8).

Advantages:

1. Query analysis along with the implemented python logic ensures that whenever the model comes across a query that requires additional information, RAG is automatically used to provide the model with the relevant information.
2. Compared to query decomposition, the improved version using query analysis is able to correctly identify the geometric features from the users query and pass it on to the next stage with an accuracy rate of almost 95%. This was possible because of the changes made in the pydantic class and the way the pydantic object is handled by the LLM.

Disadvantages:

1. The main drawback of this approach is the reliance on RAG. Using natural language to define geometrical features often leads to over-fitting the definition to the specific model in use, making it work perfectly for that model. However, since the model will not always be the same, the geometric feature definition must be precise enough for the language model to understand which feature to select. Simultaneously, the definition needs to be general

enough to ensure that changes in the model's geometry do not affect feature selection. While testing the application in this phase, instead of the reflection loop iterating over the generated YAML code multiple times as shown in figure (3.6), the reflection task was changed to a single iteration saving computational time and cost of calling the LLM. This was done as it was identified that running the loop five times did not provide any noticeable improvements in the code as compared to asking the model to review the generated YAML just once. The changed reflection prompt along with all the other prompts used in the final application are shown in appendix (D.1).

The testing phase mentioned in the results, analyzes the performance of each of these phases, across four different models. However, this testing was carried out only for the last (most mature) version of each phase to save time. The results are shown in section (4.3).

4
Results

This chapter outlines the results of the problem clarification process with the outcomes as well as the extent to which the the process was able to answer the questions at hand. At the end of the chapter, the details about the developed application that was proposed to GKN in the form of a solution addressing the problems has also been presented.

4.1 Interviews

Over a period of two weeks, (including interview preparation) a series of 6 interviews were conducted with various engineers at GKN Aerospace Sweden, as mentioned in section 2.2.1. These were semi structured interviews aimed at understanding the various problems that are faced by engineers today while tagging of geometrical entities and identifying them.

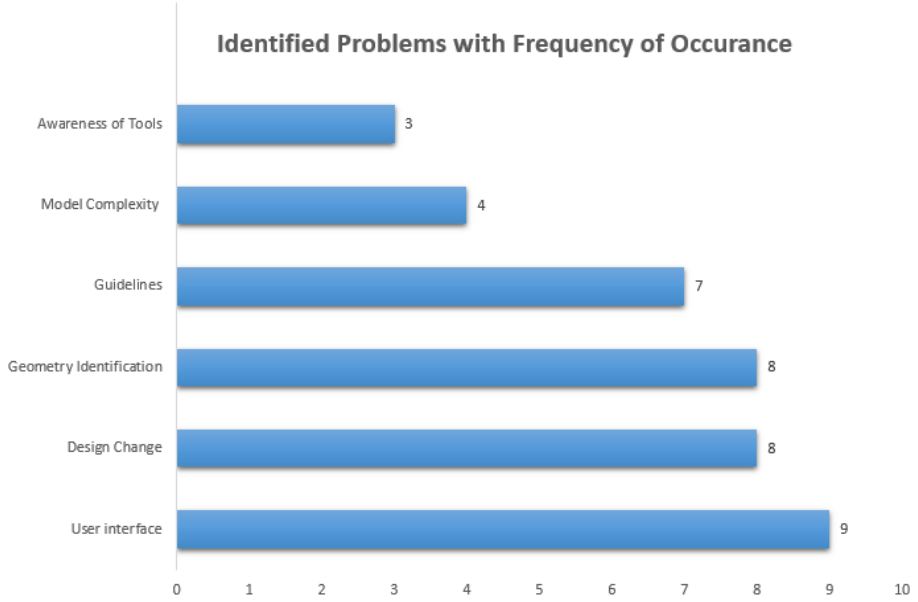

Figure 4.1: Problems identified and their frequency

The above figure shows a graphical representation of the problems that were identified with the help of the interviews and the frequency with which they were mentioned. In figure 4.1, frequency of occurrance indicates the total number of times each problem was mentioned over all of the interviews combined. This gave an idea about the frequency with which they usually occur. Detailed descriptions of the Problems along with relevant quotes from the engineers are given below.

P1: Lack of user friendly interface to effectively interact with the tool.

Based on the problem, it was understood that users are facing issues with the interface due to lack of user friendliness. At the same time the current user interface requires the user to write the

code themselves, something which not all engineers are comfortable with. Ideally, engineers would prefer to have a system that reduces the amount of manual coding required, making the process more user friendly for engineers that are not very familiar with the code. Implementation of such a system would also reduce the amount of time engineers have to spend in looking for and changing settings in the interface, while also reducing the potential errors in the process of manually writing the code.

> *"We don't really know what to set in the interface, so that might be a problem"* - Engineer 1

> *"If there could be a way where we don't need to write so much code itself, that would be easiest"* - Engineer 1

> *"Current tool is not too user friendly unless you know what you are doing."* - Engineer 2

P2: Difficulty in geometry identification impedes precise surface and feature selection using engineering terms and positional references.

Users would prefer if the tool was able to smartly identify geometry in the model, by taking positional references into account and automatically select relevant parts of the model based on descriptions and location cues. The user also feels that it should be possible to allow entering natural language queries in reference to the geometry that needs to be selected.

> *"How do I identify which areas to mesh and which not to, or how do I identify a leading or trailing edge of a vane?"* - Engineer 1

> *"The tool is not so intelligent to understand that this is where the object is located and this is where the loads are to be applied."* -Engineer 3

> *"If we are trying to script certain things, then instead of manually selecting certain geometry, if I can say 'located at this location', that would be nice."* - Engineer 2

P3: Changes in design require the model to be re-tagged.

The users say that when changing geometry, including addition or removal of new surfaces or features makes the earlier tags irrelevant as the tags are attached to the previous characteristics of the model and if certain aspects of those characteristics change, then those tags become irrelevant. This requires the tagging process to be carried out again, which is inefficient terms of productivity or doing the process manually.

> *"Robustness of the tagging process is affected by changes in the geometry"* -Engineer 1

> *"Creation or addition of new surfaces messes up existing tags which requires re-doing the entire tagging process."* -Engineer 2

> *"All of a sudden I realize that the geometry has changed too much and now I can't find any of the old tags"* -Engineer 3

P4: Lack of well defined engineering guideline for the use of the automated tagging tool

Engineers feel that the lack of a well defined engineering guideline with respect to such automated tagging tools reduces its adoption in the company. Such a guideline would be helpful to have as it reduces the amount of time that is required to learn features and capabilities of the automated tool.

> *"There is a lack of a proper guideline and/or an engineering practice on how to do this."* -Engineer 2

> *"If I don't find it easy, that does not mean that the code itself is bad, it just means that we need more training."* -Engineer 4

> *I don't do it often, so I don't know the syntax. So in that sense if you can use natural language, that would be a big difference."* -Engineer 1

P5: Complex Models are difficult to tag.

Engineers find it tedious to manually carry out the tagging process, especially for models that are complex in nature. Complex geometries having tiny details such as overlapping surfaces and edges make it difficult to effectively identify and tag such surfaces with a considerable degree of accuracy. At the same time, internal cavities or other CAD related quirks make the process increasingly complicated. These challenges become more pronounced when dealing with a batch of models that have varying geometry and topological features, which increases the risks of accidentally selecting and generating the wrong tags for the selected geometrical entity.

> *"When it comes to complicated geometries, it is important to not miss any areas."* -Engineer 4

> *"Handling topologically dissimilar models in a batch could lead to problems with the tags."* -Engineer 2

> *"The manual tagging process can be difficult, especially when it comes to models that have internal volumes or cavities."* -Engineer 4

P6: Lack of awareness regarding the existence of intelligent tools for tagging.

It was evident from the data collected in the interviews that sometimes engineers are unaware of the existence of intelligent tools that are available for tagging, or in the least, unaware of the full capabilities of the tool. Even if engineers are aware of the existence of such tools, not all engineers want to step outside the comfort zone of using a software they are familiar with just to use a tool.

> *"Both Design and analysis engineers do the tagging, the problem lies mainly with the familiarity with the use of the tool."* -Engineer 5

> *"There is little awareness of the existence of intelligent tagging tools in NX/SimCenter, which prevents analysis engineers from using it early on in the process."* -Engineer 5

> *"Engineers might not want to step outside their comfort zones, which limits the use of new software."* -Engineer 4

4.1.1 Addressed Challenges

Based on this collected information, the issues mentioned below are the ones that all of the interviewed engineers faced most commonly, as identified after thematic analysis (Denscombe, 2014) of all of the interview transcripts, and were deemed as the issues that were most important to be addressed specifically within this research.

- Problem 1: The users claim, when changes are made in the geometry, the tagging process needs to be re-done as the model no longer recognizes the old tags.
- Problem 2: Geometry identification is not possible which makes it difficult to select select surfaces and features based on engineering terms and positional references.
- Problem 3: Engineers feel that there is no user friendly interface to effectively interact with the tool.
- Problem 4: Lack of a well-defined engineering guideline for use of the automated tool for tagging reduces its adoption.

It is important to note here that this list highlights problems that occur most often, and at the same time is not a complete list of the problems with using the automated tagging tool or the tagging process in general. There are other problems that users face with tagging of the geometry, however due to limited scope the project, and specific focus on the use of the "Autotag" script, other problems especially ones in Ansys have not been taken into consideration, since as of writing this report, the Autotag tool exists only for Siemens NX (Simcenter 3D).
However, it is assumed that should the tool be available for Ansys in the future, the solution developed for Simcenter 3D should be applicable in Ansys as well.

In image (4.1) shown above, there are a number of problems under the un-categorized label. These are problems that are faced by the engineers, but they are not related to the tagging process, or if they are, they remain outside the scope of the project. Some of them are mentioned below:

1. Model visualization is difficult when the tagging process is automated: A potential solution in the case of this problem would require changing the way Simcenter or Ansys visually represents the model.
2. It is difficult to verify the accuracy of the selected geometry unless the user knows exactly what needs to be selected: In this case if the engineer does not know that the tool should be selecting a certain number of faces, there is no way to tell whether the right number of faces have been selected by the model.

4.2 Literature reviews

As a way of gathering structured information to supplement the understanding of existing knowledge and current state of the art, a literature research task was undertaken in an attempt to understand and answer two research questions mentioned earlier in the report. This was carried out according to the guidelines of structured research (Denscombe, 2014).

4.2.1 Literature review for research question 2

The research question is about understanding the advantages and disadvantages of the use of large language models for engineering design related activities. Artificial intelligence is a field of computer science that is constantly evolving and like everything, has certain advantages and disadvantages when it comes to considering the use of such a tool for large scale industrial applications.

The following articles were reviewed to get an idea of the existing knowledge highlighting the advantages and disadvantages of using Large language models (LLMs). The articles were sourced from reputable and dependable sources as mentioned earlier in section (2.2.2). The complete list of articles considered is mentioned in appendix (B).
On the basis of these articles, the following advantages and disadvantages are identified:

No.	Advantage	Reference
1	Helpful in creative thinking in Product Development	(Göpfert et al.)
2	Better at NLP than humans since they learn faster than human	(Lappin, 2024)
3	Are better at fast generation of code snippets	(Chen et al., 2021)
4	LLMs are capable of understanding intent based prompts	(Belzner et al., 2024)
5	LLMs can augment human productivity by automating tasks, such as code generation, bug detection etc.	(Chen et al., 2021; Göpfert et al.; Chen et al., 2023)
6	Effective documentation and information retrieval	(Göpfert et al.)
7	Small learning curve compared to code or industrial software	(Mishra et al., 2023; Chen et al., 2023)
8	Creativity or randomness of these models can be controlled	(Bender et al., 2021)

Table 4.1: Advantages of using LLMs

The above table clearly states the advantages that LLMs bring to the table showing potential to bring about a significant increase in the amount of productivity for completion of a task while serving as a useful tool, complementing the skill of a professional if used in the right way. When it comes to tasks which involve generation or debugging of code, LLMs are very effective in understanding the user's needs if utilized and prompted in the right way and can effectively lead to an increase in the quality of the generated at the code and at the same time reduce the amount of time required to spend debugging the generated code since these models can be fine tuned to comb through huge amounts of code, find highlight and suggest ways to overcome the issues much faster than any human can, thanks to their skills in text identification, information retrieval and ability to understand user intent. Keeping that in mind, there are some limitations about these tools that should also be taken into consideration, as mentioned in the table below.

No.	Disadvantages	Reference
1	Hallucinations and difficult verify integrity of generated data	(Lappin, 2024; Liu et al., 2023; Bender et al., 2021; Chen et al., 2023),
2	Financially and environmentally expensive	(Bender et al., 2021)
3	Lack of transparency when it comes to learning methods and reasoning.	(Lappin, 2024)
4	LLMs are unable to understand contextual semantics because they are trained on structural semantics.	(Lappin, 2024; Mishra et al., 2023)
5	Capacity of the context window is limited.	(Li et al.; Lahiri et al.)
6	Time required for setting up of these models and optimizing their output for the given use case might end up being greater than the value that comes with using them.	(Belzner et al., 2024)
7	Legal uncertainty and copyright issues along with privacy concerns.	(Belzner et al., 2024; Inc., 2024)
8	Consistency of output might be hard to control.	(Mishra et al., 2023; Chen et al., 2023)

Table 4.2: Disadvantages of using LLMs

The content generated by large language models is difficult to verify as they do not generate any references their answers, nor can these systems be held accountable for the data that they generate (Lappin, 2024; Liu et al., 2023; Bender et al., 2021). The methods that these models use for learning and reasoning are not transparent (Lappin, 2024) to the user either which makes it difficult to trust these models without having any prior information about the subject at hand. Even then, LLMs are prone to significant hallucinations (Lappin, 2024; Liu et al., 2023; Bender et al., 2021), especially when subjected to a topic that is unfamiliar to the model.
The other disadvantages surrounding the use of LLMs can be easily addressed by using different tools and methods such as fine-tuning of the models, using prompting techniques (Mishra et al., 2023) which have proved to be immensely successful in improving the quality of the output from LLMs. Concerns regarding legality of use and privacy can be addressed by using locally run versions of these models that do not send any diagnostic or other data outside of company servers, which increases their safety as well as reliability as the information that they generate and retrieve would in that case be based on existing company data provided to these models during fine-tuning.

4.2.2 Literature Review for Research Question 3

The third research question "How could LLMs and AI support the identification of CAD geometry and features using engineering terms?" mentioned in section (1.4.1) aims to understand the various ways in which Artificial Intelligence and Large language models could assist in geometry identification and selection using engineering terminology. Based on the articles collected for this task, it was found out that there are 2 possible ways in which Artificial Intelligence could be used:

1. Use of Auto encoders combined with deep learning architectures (Krahe et al., 2022)
2. Using Multi Modal models (Rios et al., 2023)

According to (Achlioptas et al., 2018), it should be possible to analyze and identify geometrical features from 3D point cloud models with the help of deep AE (Auto Encoder) networks having the capability to reconstruct and generalize point cloud models effectively. However, in order to implement this method it would require that the 3D model be converted to Point clouds, which could lead to a significant loss of detail and model information which is not ideal, especially in the case of complex aerospace components. The approach shown in (Achlioptas et al., 2018) utilizes 3D point cloud models to train a generative auto encoder network which then shows the ability to generate shapes that are at-least as detailed as the shapes it was trained on.

Similarly, (Krahe et al., 2022) presents the ability to perform similarity search operations with respect to geometric features. The paper mentions that it is possible to develop an auto encoder network that has the capability to, with the help of deep learning identify the CAD model that is most similar to any of the models in the database. This means that the network possesses the capability to understand, look for and process geometric features. This uses an auto encoder point cloud for extracting model features, after which a similarity search is performed to look for models with similar features and characteristics. This similarity search is done with the help of vector data about the model stored in the latent space. Paper (Chow et al., 2015) discusses about detecting and identifying geometric features as a way of automatically de-featuring CAD models for simplification. This process could theoretically help in identifying the same geometric features for "tagging". However whether the proposed method is effective in analyzing and extracting intricate and detailed geometrical features still remains to be seen.

Using multi modal LLMs and Generative models is another way which has been presented in articles (Rios et al., 2023; Jun and Nichol, 2023; Nichol et al., 2022) outlining the capabilities of

generative models to create 3D models and point clouds based on natural language input. This capability of the models could be used in extracting geometric features from existing 3D models.

These methods are the current state of the art when it comes to utilization of artificial intelligence in engineering design and simulation. Using 3D machine learning networks and deep learning architectures presented in this section requires a significant amount of computational resources along with a high amount of high quality data for training and testing these models. At the same time the amount of time required for successful training and implementation of such frameworks is very high. Due to a lack of sufficient training data in terms of CAD models, it was important to find a way to implement feature identification without using high amounts of training data. As a result, it was decided to use the high reasoning capabilities of large language models by using the CAD model database and text base definition of geometric features to perform feature identification and extraction with a considerable degree of accuracy.
Even though these methods were not explored directly in this research, the information and insights obtained from this literature review was useful in providing an idea about the capabilities of the technology, and guide the development of the proposed solution.
A detailed list of the scientific articles taken into consideration for this literature review is presented in appendix B.2.

4.3 Testing and Verification

Testing and verification of the application was carried out for the final version at of each phase as version to version improvements in each phase are minimal and would not produce any significant differences in the performance of the model.
A set of test queries, which were formulated with the help of the knowledge acquired while testing the application during its development.This developmental testing and the findings from it gave an idea about the kind of queries that the application is able to handle. The idea behind generating this set of test queries is that each query increases in complexity with a certain level, thereby giving a better idea about which models are able to perform the best with respect to more and less complicated queries.

The complexity of the queries is determined based on the type of selection that the language model is expected to carry out. For example, a query that asks to select all the flanges in a CAD model is more complicated because such a query involves selection of geometrical features that are not explicitly mentioned in the database schema. This means that in order to accurately select only the flange, the model needs to look for a definition of the feature, establish relationships between the flanges and surrounding geometrical entities and use that information contextually to ensure that only flanges are selected.

1. Select all faces with radius less than 10
2. Select all faces with radius equal to 5 and then select their corresponding edges.
3. Select circular edges with a radius of 6mm and then select their adjacent faces.
4. Select all faces in the lower half of the model that have a radius less than 10 that are blends.
5. Select all faces in the XY plane that have a radius greater than 5mm
6. Sort all blends by their Z position and then select the last two blends.
7. Select all boltholes in the model.
8. Select all flanges in the model.
9. Select all struts in the model and then select their leading edges.
10. Select all strut fillets in the aftmost section of the model.

Based on the test criteria outlined earlier, the tests were carried out on the 4 different language models mentioned in section (3.2.1); each having varying sizes and capabilities along with costs.

During the testing of the developed application, newer and more capable versions of these models were introduced. However these were not tested due to time constraints. Detailed specifications of these models along with their performance capabilities are also mentioned in section (3.2.1). The performance column in the tables shown below indicates the number of times out of 5, that the model is able to accurately select the expected geometry, which gives an idea about the percentage of success. The serial numbers in the test tables ranging from 1-10 indicate the test queries in that order.
The testing was carried out manually, which means each test query was manually run 5 times for each model and the required aspects such as tokens, cost and time required were noted.

4.3.1 Phase 1

As mentioned in section (3.3.1), this phase was a trial to assess if the LLMs could generate YAML code in the required format. Conducted on the Google Bard website, it aimed to understand the model's generative capabilities and how it handles instructions. This initial phase did not test the model's ability to handle complex queries.

	Gemini 1.0 Pro			
Sr. No	Tokens	Cost	Time	Performanc
1	3808	0,057120	10	1,00
2	4000	0,060000	6	0,80
3	3730	0,055950	9	0,80
4	3783	0,056745	7	0,60
5	3720	0,055800	13	0,20
6	3789	0,056835	9	0,00
7	3956	0,059340	11	0,00
8	4116	0,061740	10	0,00
9	4349	0,065235	9,5	0,00
10	3915	0,058725	10	0,00
	3916,6	**$ 0,059**	**9,45**	**0,34**

Figure 4.2: Phase 1 Test Results

As shown in figure 4.2, it can be seen that with minimum information available, the language model is able to generate YAML code for the simpler queries according to the criteria mentioned in section (4.3). However, with the limited amount of information that is provided to the model in this case, it is unable to generate accurate YAML code as the complexity of the test queries increases. The final row of the table in figure 4.2 shows the average performance accuracy, cost and number tokens required for this phase.

4.3.2 Phase 2: Implementation of Langchain

In the second phase 3.3.2, Langchain and OpenAI's GPT-3.5 Turbo were used to address the previous phase's limitations. Langchain allowed testing different models with various configurations by allowing modular code changes. More details on the tools and technologies used are in section (3.2.1).

Figures 4.3, 4.4, 4.5, 4.6 show the performance of Gemini Pro, GPT 3.5 Turbo, GPT 4 and Claude 3 Sonnet respectively in Phase 2.

	Gemini 1.0 Pro			
Sr. No	Tokens	Cost	Time	Performanc
1	2310	0,034650	1,8	1,00
2	2322	0,034830	1,9	1,00
3	2329	0,034935	2	1,00
4	2335	0,035025	2,2	1,00
5	2332	0,034980	2,2	0,80
6	2342	0,035130	2,05	0,00
7	2376	0,035640	2,5	0,00
8	2340	0,035100	2,05	0,00
9	2330	0,034950	2,11	0,00
10	2330	0,034950	2,05	0,00
	2334,6	**$ 0,035**	**2,086**	**0,48**

Figure 4.3: Gemini Pro Phase 2

	GPT 3.5 Turbo			
Sr. No	Tokens	Cost	Time	Performance
1	2321	$ 0,001172	2	1,00
2	2333	$ 0,0011840	1,5	1,00
3	2340	$ 0,0011940	1,34	1,00
4	2345	$ 0,0011950	1,05	1,00
5	2337	$ 0,0011940	1,2	0,40
6	2335	$ 0,0011870	1,2	1,00
7	2367	$ 0,0012270	1,4	0,00
8	2326	$ 0,0011820	3,5	0,00
9	2333	$ 0,0011850	1,2	0,00
10	2334	$ 0,0011840	1,22	0,00
	2337,1	**$ 0,001**	**1,511**	**0,54**

Figure 4.4: GPT 3.5 Turbo Phase 2

	GPT 4			
Sr. No	Tokens	Cost	Time	Performance
1	2320	$ 0,069960	2,5	1,00
2	2334	$ 0,070560	1,5	1,00
3	2343	$ 0,071250	2,2	0,60
4	2346	$ 0,071070	2	1,00
5	2350	$ 0,072120	2,7	1,00
6	2334	$ 0,070620	2,2	1,00
7	2390	$ 0,073440	4,3	0,00
8	2346	$ 0,071580	3,11	0,00
9	2345	$ 0,071340	2,4	0,00
10	2364	$ 0,072540	3,9	0,00
	2347,2	**$ 0,071**	**2,681**	**0,56**

Figure 4.5: GPT 4 Phase 2

	Claude 3 Sonnet			
Sr. No	Tokens	Cost	Time	Performanc
1	2310	$ 0,00693	1,8	1,00
2	2321	$ 0,00696	1,85	1,00
3	2329	$ 0,00699	2,05	1,00
4	2334	$ 0,00700	2,04	1,00
5	2327	$ 0,00698	2,04	1,00
6	2340	$ 0,00702	2,31	1,00
7	2362	$ 0,00709	2,5	0,00
8	2334	$ 0,00700	2,7	0,00
9	2336	$ 0,00701	2,5	0,00
10	2353	$ 0,00706	2,7	0,00
	2334,6	**$ 0,007**	**2,249**	**0,6**

Figure 4.6: Claude 3 Sonnet Phase 2

Based on the test results in this phase, it can be clearly seen that the implementation of langchain in phase 2, along with the introduction of additional context provided to the model improves the ability of the language model to accurately generate the correct YAML code for selection of the required geometry.
At the same time, while the performance of the model is seen to have increased here compared to the previous phase, the language model or the application is still unable to effectively identify and select geometrical features based on natural language terminology.

4.3.3 Phase 3: Query Analysis, Reflection and RAG

The first part of this phase tested a "reflection" framework, allowing the model to iterate and improve outputs by looping over generation prompts. After successful implementation and robust YAML generation, the next step was incorporating geometry identification using natural language engineering terminology, which is where Query Analysis and RAG were implemented.

	Gemini 1.0 Pro			
Sr. No	Tokens	Cost	Time	Performanc
a	11328	0,169920	18,9	1,00
b	11549	0,173235	14,4	1,00
c	11407	0,171105	18,84	0,80
d	11635	0,174525	18,72	1,00
e	13000	0,195000	18,7	0,60
f	12620	0,189300	21,1	0,00
g	12700	0,190500	19,42	0,20
h	12900	0,193500	21	0,00
i	13600	0,204000	24,7	0,00
j	11744	0,176160	19,1	0,00
	12248,3	**$ 0,184**	**19,488**	**0,46**

Figure 4.7: Gemini Pro Phase 3

	Claude 3 Sonnet			
Sr. No	Tokens	Cost	Time	Performanc
1	11355	$ 0,03407	19,3	1,00
2	11713	$ 0,03514	21,5	1,00
3	11816	$ 0,03545	19	1,00
4	11015	$ 0,03305	13,1	1,00
5	11622	$ 0,03487	21	1,00
6	11551	$ 0,03465	18,9	1,00
7	13304	$ 0,03991	29	0,80
8	13510	$ 0,04053	32	0,80
9	12960	$ 0,03888	28	0,00
10	13017	$ 0,03905	29,5	0,60
	12186,3	**$ 0,037**	**23,13**	**0,82**

Figure 4.8: Claude 3 Sonnet phase 3

	GPT 3.5 Turbo			
Sr. No	Tokens	Cost	Time	Performance
1	10700	$ 0,006000	10	1,00
2	9500	$ 0,0058935	11,1	1,00
3	11267	$ 0,0059865	10,8	1,00
4	11135	$ 0,0058585	9,5	1,00
5	11400	$ 0,0061640	11,5	0,80
6	11500	$ 0,0063195	9,92	0,80
7	11350	$ 0,0060810	15	0,20
8	12100	$ 0,0069695	17	0,00
9	12280	$ 0,0068180	16,1	0,00
10	12200	$ 0,0068635	15,3	0,00
	11343,2	**$ 0,006**	**12,656**	**0,58**

Figure 4.9: GPT 3.5 Turbo Phase 3

	GPT 4			
Sr. No	Tokens	Cost	Time	Performance
1	10800	$ 0,329970	13	1,00
2	11175	$ 0,349230	17,6	1,00
3	11550	$ 0,360600	20	1,00
4	11320	$ 0,352830	20	1,00
5	11180	$ 0,352590	16	1,00
6	10750	$ 0,337500	22	1,00
7	12200	$ 0,363610	24	0,80
8	13100	$ 0,434970	30	0,80
9	12100	$ 0,378390	25,4	0,00
10	12059	$ 0,379110	25,5	0,00
	11623,4	**$ 0,364**	**21,35**	**0,76**

Figure 4.10: GPT 4 Phase 3

Similarly with the testing and implementation of the RAG and query analysis in phase 3, figures 4.7, 4.8, 4.9, 4.10 show the improvements in the performance of the application which are brought about by RAG and Query analysis. In each of the tables shown in the figures above, an ability to identify certain geometrical features can be seen, albeit at a low level of accuracy. This can be attributed to the definition of the geometric feature provided in the Vector database, implying that the ability to identify and select geometric features depends on the level of detail provided in the definition. An example of the definition that is provided to the language model for help with feature identification is provided in appendix D.1.

4.4 Proposed Solution

The finalized solution is a combination of reflection agents, query analysis and retrieval augmented generation, with the architecture of the developed application using approach (3.1.3) as its foundation. The frameworks and tools used in the application and over the duration of the development phase are described in section 3.2.1. The developed application has been presented in this section along with the results of the testing and validation of the same, with respect to additional test criteria such as testing with various engineers' input, changing the CAD models and other non-pristine test conditions. This helps in assessing the robustness of the entire application as a whole.

4.4.1 Core Problem Resolution

User interface: The proposed solution was incorporated into SimCenter API along with an intuitive and user friendly "AI" button, which incorporates the LLM functionality directly into SimCenter. For final use of the tool, proper supporting documentation will also be created. The tool can be accessed in a way similar to the existing tool, but now the user can input a natural language prompt directly in Simcenter which is then processed by the developed application, and the outcome is highlighted in the CAD model.

Geometry Identification: The implementation of Retrieval Augmented Generation and fine tuning of the contexts provided to the model with the ability to process CAD model specific information. This addresses the problem of geometry identification and identifying geometry in terms of positional references, since with the help of these methods, the language models are able to better process the natural language prompt with the help of the additional information. The additional information that was provided to the model (example shown in appendix D.1 includes but is not limited to natural language definitions of the various geometric features of the CAD model, as well as a natural language representation of the CAD model itself.

Changes in the geometry: This problem was addressed in the testing phase, in which various physical aspects of the model were changed as a way of testing the robustness of the developed solution. The application was tested on 3 different CAD models, along with changing the geometrical dimensions of the CAD models. Since the overall idea of the CAD model remains the same, the application is able to effectively select the required geometrical features within a 10 percent margin of error. This section talks about the ways in which the proposed solution should address the problems it was developed to address.

It is important to point out here, that the contribution of the developed application in the tagging process is only towards the identification and selection of the relevant CAD geometry or features. The assigning of the required tags to the selected geometry can be done with the help of the "Autotag" tool, which has features built into it for this purpose.

4.4.2 Increasing Application Performance:

It can be clearly seen from the results of testing in phase 3 mentioned in section (4.3.3), the models are not entirely capable of identifying geometrical features such as struts and leading edges etc, since these features are not particularly easy to define, without having spatial representations of the CAD model as well as relative properties. Using model specific properties in the definition of these features would result in successful selection for that specific model, but would not be scale-able to other models having similar features but with varying properties.

The implemented solution was presented to engineers at GKN in an attempt to get feedback and further suggestions to improve the application. Based on the feedback and suggestions gathered, it was decided that it would be helpful to integrate an additional feature that allows the user to manually select some sort of geometrical entity/features in the model, and then query the LLM to select additional geometric features based on the pre-selected geometry. This was done by using the Simcenter API, allowing the user to select certain geometrical features. These selected geometrical features are then provided to the LLM as contextual information, along with the necessary instructions to process that information in relation to the CAD model.

The properties of the selected feature are in the form of a python dictionary, the contents of which are provided to the LLM, along with the system message, making it easier for the LLM to

understand the instructions as well as relative positions of similar features. The basic prompt to the LLM is the same as the base prompt that is used to generate the YAML code as shown in section (3.3.3), with an additional set of instructions that are considered only when some manually selected geometry is present. Figure 4.11 shown below depicts the feature in use.
The ability to select geometrical features by providing the LLM contextual and positional references about the required geometrical features solves this issue to a great extent, increases the success level of the application in the high 90 percent range. Extensive testing of this feature was not carried out with respect to the criteria mentioned earlier, however testing the feature during development showed significant improvements in the ability to handle complicated selection queries with almost 90-95% level of success, especially when using a well performing LLM like GPT 4 or Claude 3.

4.4.3 Integration into Simcenter

The developed application was integrated into SimCenter 3D with the help of the NX-Open API, which allows the development and integration of python apps into Simcenter. A minimal UI was developed and added as an option in the existing "Autotag" tool. The figure (4.12) shown below, shows the "AI" button which launches the application allowing the user use the LLM (shown in image (4.11)).

4.4.3.1 The User Interface

The User interface of the developed application as seen in figure 4.11 consists of 3 main parts:

1. Select Items: This part allows the user to manually select parts of the CAD model, either Bodies, Faces or Edges or a combination of the three.
2. Prompt: This provides a text field where the natural language prompt is provided, along with a button to run the prompt.
3. Response from AI: This section consists of two fields, one which provides the selection query that the LLM generates for the natural language prompt and another field which has the capability to output some form of natural language feedback from the LLM based on the user's request. This field was mainly used to analyze the model's thinking process during testing, and may or may not be included in the application if it is implemented.

The user is also provided with the option to edit and make additions to the generated YAML query string without having to run the LLM again for small changes. The "feedback from AI" section is used to output the natural language feedback from the language model with respect to the generated YAML code, its functions and how it could be improved, giving the user a better idea about the reasoning decisions taken by the language model while generating the code.
The following representation depicts a use of the tool in which the user selects one face in the CAD model, that is automatically provided to the LLM when it is selected. Based on that selection, the user then asks the application to select all similar faces that have a similar X-position. The LLM also takes into consideration that 'similar' in this case indicates that there might be slight variations in the properties of the other faces that it has to select, and as a result maintains a tolerance of 5% when generating the selection query and updating the selection.

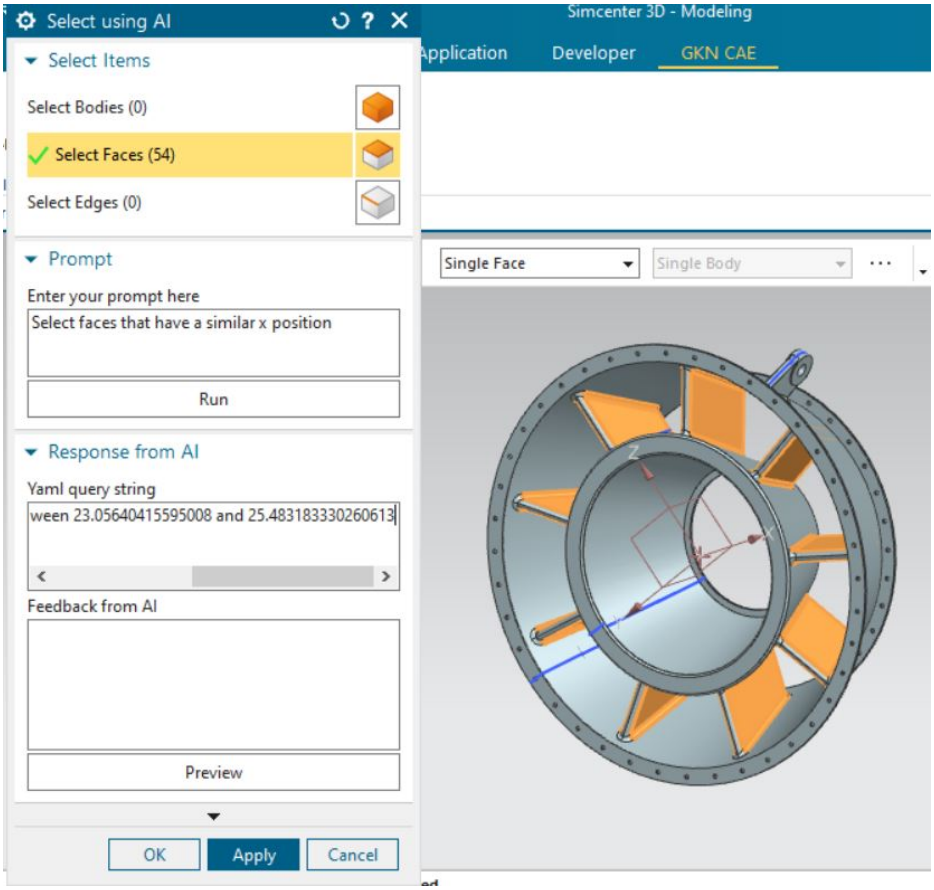

Figure 4.11: AI-Selection Process

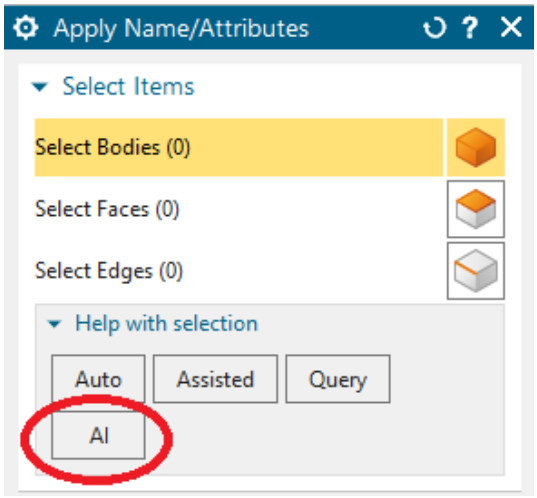

Figure 4.12: AI button in Simcenter

5
Discussion

This chapter discusses the answers to the research questions in section (5.1) along with the developed solution, interpreting and discussing the results of the performance testing of the developed application. Section (5.2) discusses the challenges that need to be addressed further and the ways in which they could be possibly approached.

5.1 Answers to Research Questions

As mentioned in section (1.4.1) a set of research questions were formulated as part of this research. The research was conducted as a way to answer those research questions.

Question (1) "What are the challenges that engineers face today to generate fast and robust CAD tags for FEM meshes?" was aimed at finding and understanding the kind of obstacles and difficulties that engineers at GKN face when they need to tag a CAD model. These challenges were identified with the help of interviews with engineers at GKN. The list of identified Problems is given below:

1. Lack of User friendly interface
2. Difficulty in geometry identification impeded precise surface and feature selection using engineering terms and positional references.
3. Changes in design may require the model to be re-tagged.
4. Lack of well defined engineering guideline for use of the tool.
5. Complex models are difficult to tag
6. Lack of awareness regarding the existence of intelligent tools for tagging.

A summary of the interviews helped highlight the most pressing issues that were relevant to the research (4.1.1) that were eventually addressed by the proposed solution. Detailed descriptions of the identified problems is available in section (4.1).

Question (2) "What are the unique advantages and disadvantages that LLMs face when dealing with engineering tasks?" was directed towards understanding the pros and cons of using a technology that is currently very volatile and new, undergoing constant development. This question was answered with the help of a literature review, collecting and summarizing relevant scientific articles.

Through the summarized information, it was identified that LLMs are helpful in assisting with creative thinking in the product development process. Similarly, language model capabilities can be used for generation of code snippets quickly and easily, along with documentation and information retrieval. Multiple sources have mentioned that language models have the capability to augment productivity by helping automate tasks such as generation of code and detection of bugs in the code etc.

On the other hand, language models are subject to hallucinations, which makes it difficult to verify the integrity of the generated data. While using language models might not be as environmentally

and financially expensive as training a model, the cost (both environmental and financial) adds up over a period of time. The use of content generated by language models is subject to legal uncertainty in terms of ownership and copyright issues as well as privacy concerns in case of sensitive data. A detailed list of all advantages and disadvantages has been provided in tables 4.1 and 4.2.

Question (3), "How could AI and LLMs support the identification of CAD geometry using engineering terms?" aimed to enhance understanding of the current knowledge and research about this technology in relation to engineering design and CAD. With the help of relevant research articles collected categorically, a summary of the relevant work out there was made. As discussed in section (4.2.2), there are two main ways in which AI and LLMs can be used for this purpose; one of them being an auto encoder network combined with deep learning architectures and the other being multi modal generative models.

The first method is a generalized method, which can be implemented in multiple ways, like with the help of 3D point clouds as discussed in (Achlioptas et al., 2018) or using similarity search with a model trained to identify CAD models and their features from a database, as discussed in (Krahe et al., 2022). This information was crucial for brainstorming the solution and developing the foundational ideas of the application architectures mentioned in section (3.1).

The second method discusses the the use of multi modal generative models as mentioned in (Rios et al., 2023; Jun and Nichol, 2023; Nichol et al., 2022) wherein a generative model can be designed and trained that has the capability to generate a 3D model using natural language input. The multi modal generative model also has the capability to create point clouds based on natural language input, which means it could be combined with the earlier method to generate Point cloud models using the capability of the auto encoders and deep learning architectures.

The answers to all three of the research questions played a vital role in the development of the application. The challenges being faced by the engineers at GKN provided a direction to the research, as this question answered the need of developing such an application. The answers to the second and third research question helped understand the advantages and disadvantages of a new technology, which provided insight into the development process of the application. While the methods mentioned in the answer to the third research question were not explored due to the limitations of the project, the application showcases one of the possible ways in which AI and LLMs could help in geometric feature and entity identification and selection. A detailed description of the findings has been presented in section (4.2.2)

5.1.1 The developed Solution

The developed solution consists of 3 different phases The tests on the model were conducted according to the criteria mentioned in section (4.3) and based on the results, the performance of the models in each phase can be visualized in the table shown below.
Figure 5.1 clearly shows increasing performance with each phase, and we can also see the difference in the performance and selection capability of each language model in comparison to the other. It was found that Gemini Pro, while considered as a well performing and powerful model in general, seems be consistently inconsistent with performing the requested task, as can be seen in the performance of the model over the 3 different phases. Every model showed increased performance with the developments made in the code-base with each new phase, however Gemini-pro was the only language model that showed a decline in the performance as well the quality of the generated output with the final version of the code.

	Phase 1	Phase 2				Phase 3			
Sr. No	Gemini Pro	Gemini Pro	GPT 3.5 Turbo	GPT 4	Claude 3 Sonnet	Gemini Pro	GPT 3.5 Turbo	GPT 4	Claude 3 Sonnet
a	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
b	0,80	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
c	0,80	1,00	1,00	0,60	1,00	0,80	1,00	1,00	1,00
d	0,60	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
e	0,20	0,80	0,40	1,00	1,00	0,60	0,80	1,00	1,00
f	0,00	0,00	1,00	1,00	1,00	0,00	0,80	1,00	1,00
g	0,00	0,00	0,00	0,00	0,00	0,20	0,20	0,80	0,80
h	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,80	0,80
i	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
j	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,60
	0,34	**0,48**	**0,54**	**0,56**	**0,6**	**0,46**	**0,58**	**0,76**	**0,82**

Figure 5.1: Phase-wise Model Performance

On the other hand GPT 3.5 Turbo and GPT 4 show increasing performance in the final phase, with Claude 3 Sonnet having the best overall performance in terms of quality of generated output as well as consistency in being able to select the expected geometrical features. While most of the models tested can successfully process simpler queries at low cost, it was identified that successfully understanding and processing more complex queries, especially geometry identification related queries having complicated features such as struts and leading edges were unable to be processed as shown by the red color in the table. While models like Claude and GPT 4 are not completely successful either, they were the only models who were with a considerable degree of accuracy able to select these geometrical features with the same amount of information available to them as the other models. This tells us that the use of even more powerful models, such as GPT 4 Turbo or Claude 3 Opus could lead to better scores of performance, albeit at an increased cost.

Building on the testing conditions in Section (2.5), the developed solution was released to engineers for testing with various CAD models, including complex ones previously deemed too complex. In all of these tests, the application performed exactly as expected, producing results that can be considered in the same category of success as the tests which were conducted.

This validates that the developed and proposed solution is robust within the performance brackets that it performs well in, i.e., for simple and moderately complicated queries and shows a lot of promise when considered in combination with the additional feature mentioned in section (4.4.2).

The test data was also analyzed in regards to looking for a relationship between model performance along the Y axis and cost of executing the application in USD along the X axis. As shown in figures 5.2, an increase in the performance of the model is visible. This increase can be attributed to intermediate steps used in each phase along with improved reasoning logic. However, this increased performance comes at the expense of increased token usage leading to increased cost of executing the application. The average increase and the direct relationship between these two factors can be better visualized with the help of figure 5.3 shown below, which shows the average cost against average performance of all models combined, over the three development phases.

Similarly, there also exists a relationship between the number of tokens a language model utilizes and the performance of the model, which can be seen in figures 5.4 and 5.5 shown below. This also shows that as the number of tokens used by the LLM increases, it leads to improved performance as it can be argued that increase in the number of tokens indicates an increase in the amount of contextual information that is available to the LLM, causing a visible improvement the reasoning capability of the model. A better visualization of this relationship is presented in figure 5.5, which shows the trend of average token use against model performance in each phase for all of the models tested.

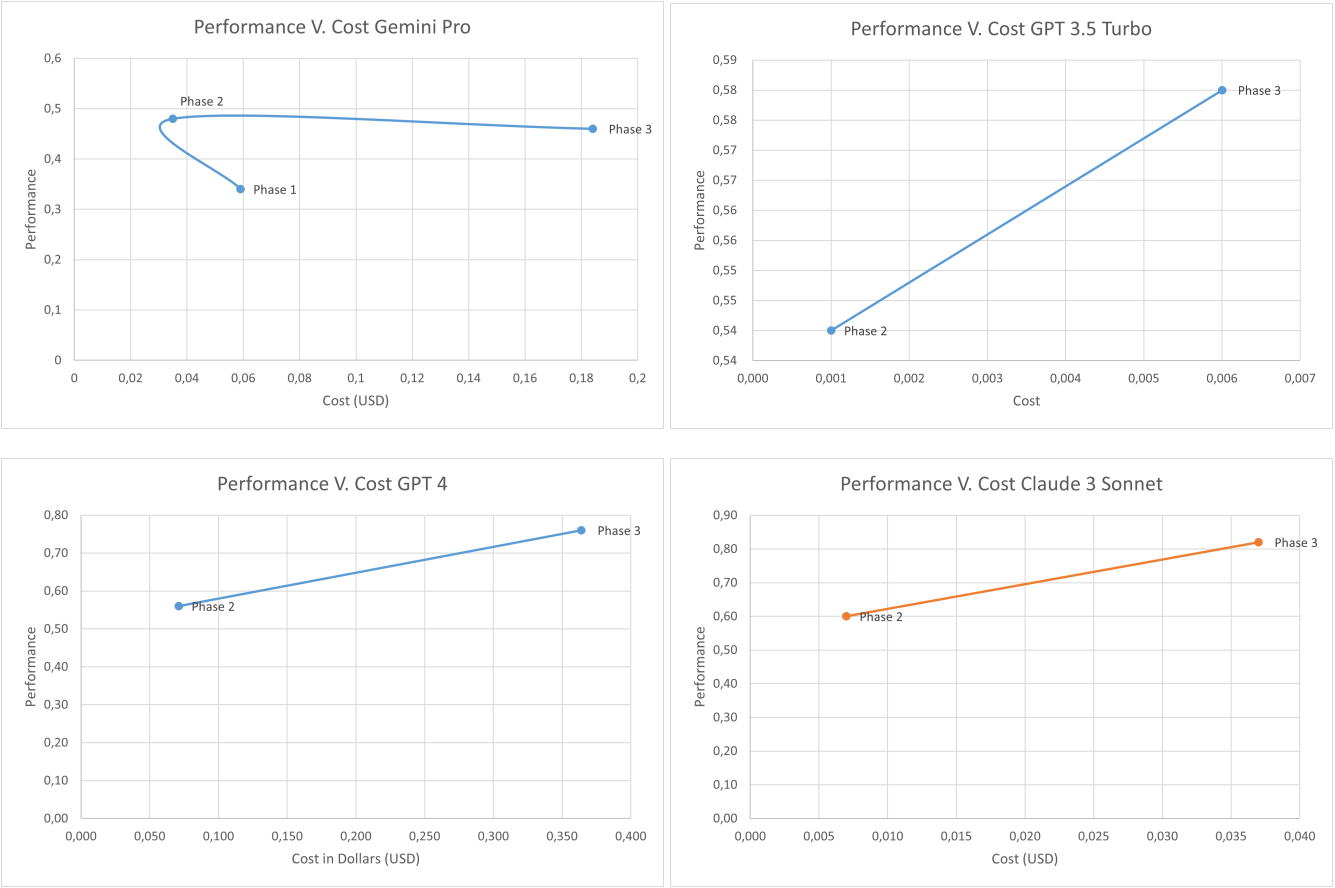

Figure 5.2: Performance Vs. Cost

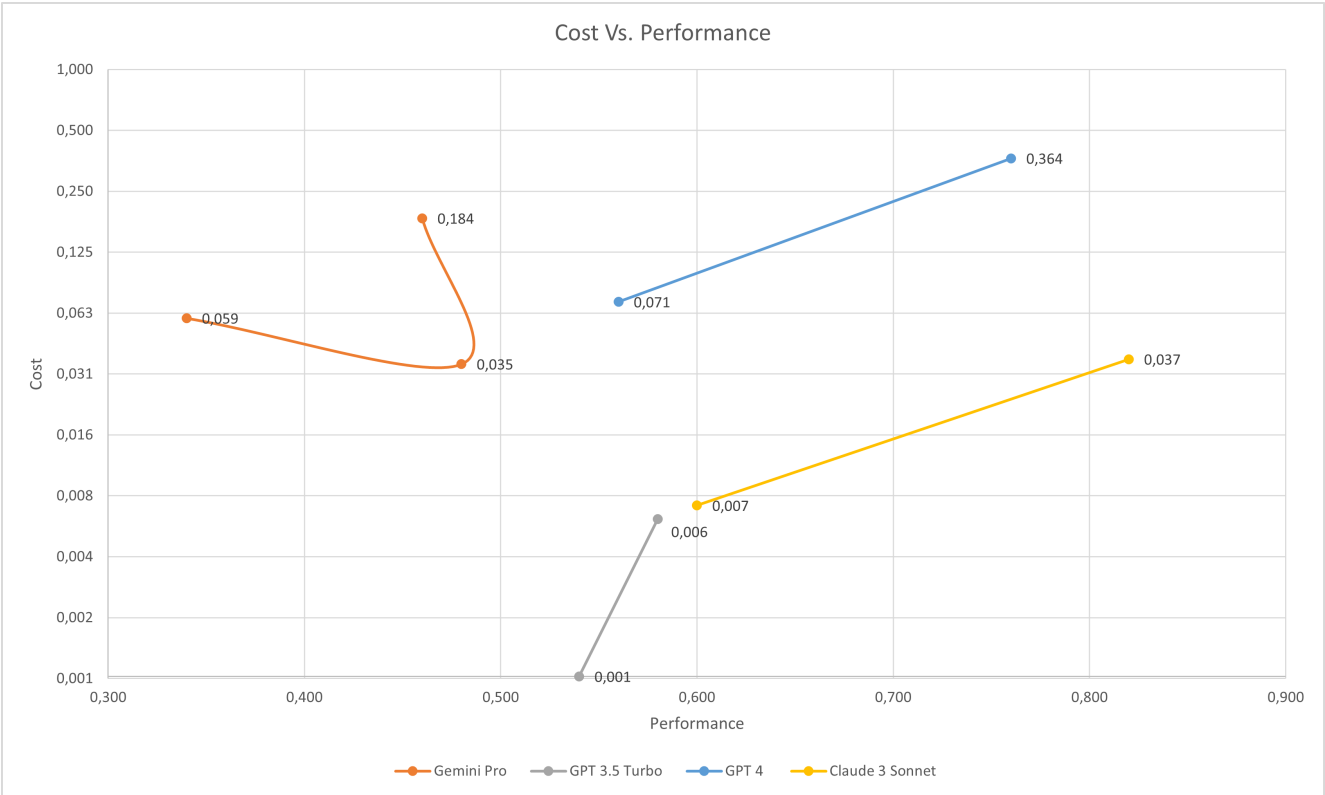

Figure 5.3: Performance Over Cost Relationship between each phase

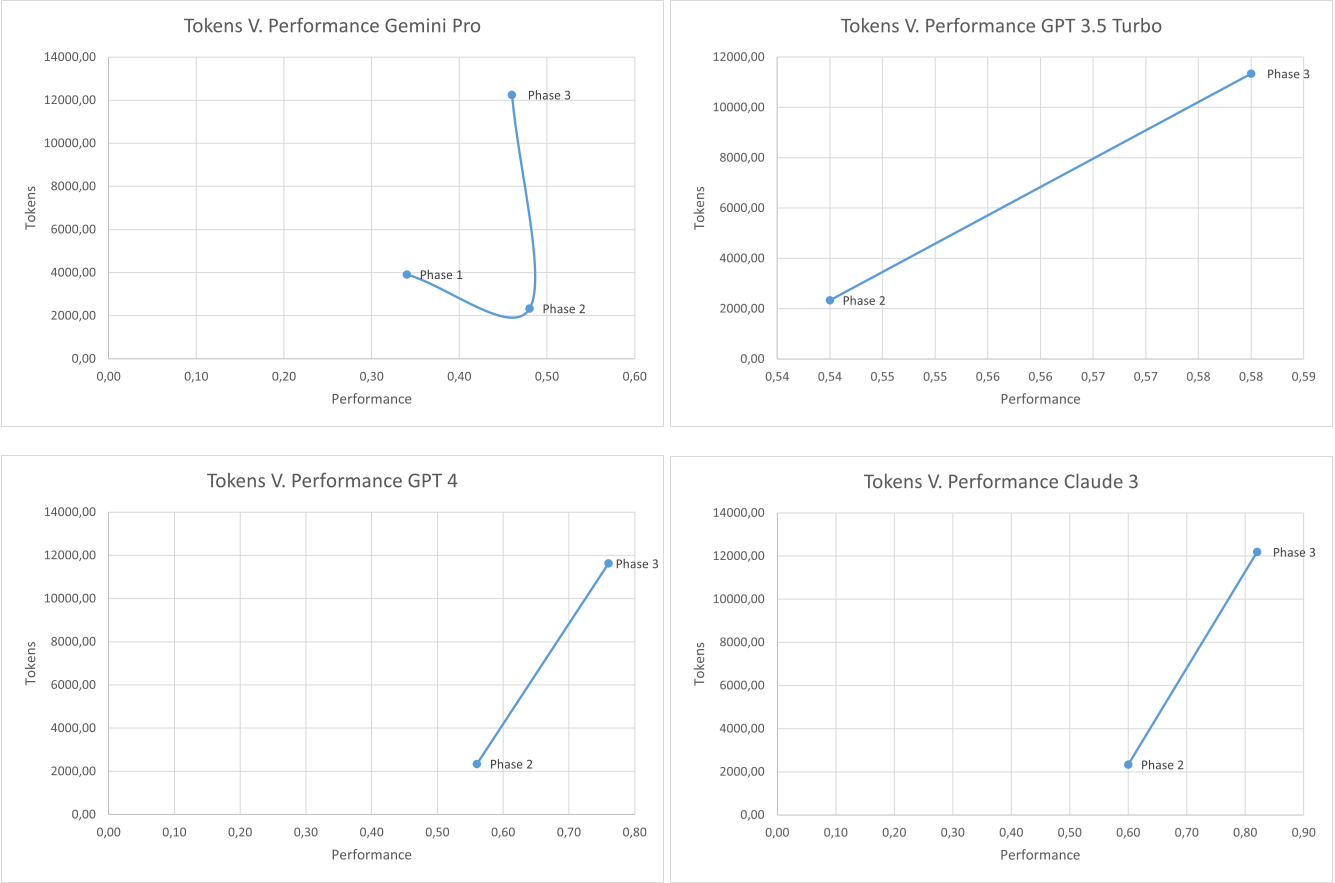

Figure 5.4: Token Use Vs. Performance of the LLM

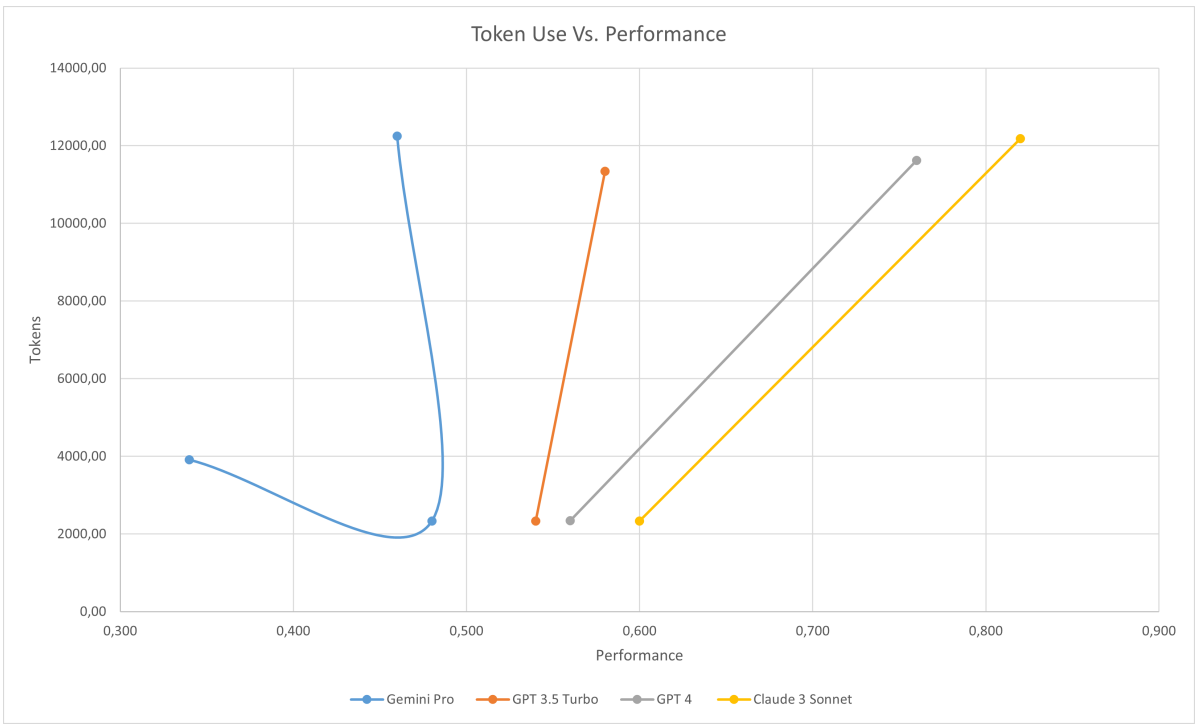

Figure 5.5: Token Use Vs. Performance relationship between each phase

In figure 5.3, the ideal combination of performance and cost would be found in the lower right corner of the graph, relating to high performance low cost.

5.2 Remaining Challenges

Language models have a fixed amount of tokens that they can process for a single call, which is called as the context length of the language model. This context length varies from model to model, depending on the model parameters and capabilities of the language model. When implementing the additional feature allowing manual selection of CAD geometry in the application, the number of faces that can be selected to provide context to the model is limited because selecting a high number of geometrical entities makes the list of properties provided to the model extremely large. As a result, the entire set of properties cannot fit within the context window of the model, which stops the application from completing the task.

This issue can be addressed in two possible ways:

1. Use a language model that has a very high context window such as the newly introduced GPT 4o, which has a context window of 128000 tokens as compared to GPT 4 which had a context window of 8192 tokens (com, 2024). However using a model with a significantly large context window can be extremely expensive to run locally in terms of cost as well as hardware requirements.
2. Create a function in python that can consolidate the properties of the selected geometry into a single python dictionary, which will always make sure that the properties fit within the context window of any model. This can be done by filtering out the common properties of the selected features and ensuring that only the unique properties are included and further consolidated, without losing any essential information.

The tool effectively selects relevant geometry based on specified properties. However, testing revealed that any ambiguity in the query regarding these properties decreases the overall accuracy of the results. Therefore, user queries must include precise instructions about which aspects of the provided properties should be considered when selecting new geometry. This challenge needs to be addressed by providing better contextual information to the language model.

As with all language models, there are some cases when using the application causes the LLM to hallucinate, resulting in selection of geometry or faces that are not similar in any way to the provided properties. Although it must be noted here that this tends to happen only about 1 out of every 10 times of using the application. This issue can be completely eradicated by further fine tuning the system messages and instructions provided to the model.

Performance testing of the application also uncovered the fact that the language model is unable to keep a track of the relationship between the various surfaces. This means that the language model is only able to process what is present in the database schema related to the CAD model, and uses that information for generating the YAML code whenever required. In this case, providing the language model with the ability to maintain a record of the relationships between various features and geometries within the CAD model, outside of the database schema could improve the performance.

6
Conclusion

6.1 Outcome of the thesis

In conclusion, the study was conducted as a way to explore and showcase the capabilities of AI and LLMs in identifying geometric features. This foundation of the research was created with the help of the information collected from semi structured interviews with engineers at GKN, which helped understand the perspectives of GKN as well as engineers with respect to the tagging process for CAD geometry. The project also studied the various advantages and disadvantages of using artificial intelligence and large language models for performing engineering activities.

Similarly, a short literature review was conducted in relation to the research questions as a way of understanding existing technology, its advantages and disadvantages and also to help brainstorm possible ideas for the developed solution. All of this collected information from the interviews and literature reviews was used as a foundation for developing the proposed solution. As a result, the developed application is able to support the identification as well selection of geometrical features and entities in a CAD model.

The developed and implemented application is able to successfully showcase one of the ways in which geometric feature identification and selection can be carried out using LLMs and AI. Based on the feedback gathered from the engineers who had the opportunity to use the application in their day to day work, the application tends to save a lot of time that was earlier required to manually select surfaces and geometric entities while tagging. At the same time, use of the new application improved the capabilities of the "autotag" tool. The application lets users easily select geometry that was previously hard for the tool to pick automatically, since it uses a language model's ability to find connections between the properties of the geometry.
The developed application has the following key characteristics:

1. LLM: The application uses a large language model to process user input in natural language and relate it to a database schema for generating YAML text that is used quick and efficient selection and identification of geometrical features and entities in CAD models.
2. The language model used is a pre-trained model, which means that very little optimization may be required for implementing the application on any existing system.
3. The application now provides a user friendly interface, allowing the user to interact with it much more easily when compared to the "Autotag" tool that was used previously.
4. Cloud maintained application means it does not need to be separately installed on every machine. Anyone who already has access to SimCenter can utilize the application once it is made available, which also ensures that it can be updated frequently whenever required.

However, there are also certain aspects of the code that need to be improved such as:

1. The implementation of Retrieval Augmented Generation could be improved by adding better

definitions of the geometrical features wherever required such that the model is able to identify features such as struts and leading edges and trailing edges etc. as it currently cannot identify those and other similar complicated features.

2. Models with improved context windows need to be implemented that can process more information at a lower cost, allowing the application to understand the CAD model properties in a better and more comprehensive sense.
3. The LLM tends to hallucinate at times when it comes across a user query with a certain ambiguity to it, which causes it to select incorrect geometry. A disadvantage of this issue is the fact that the user might not always be aware of incorrect selections in cases where there is no way for the user to know what geometry is correct and what is not.

It is also important to know here, that while the implemented application is successful to some extent, this is not the only way in which such a task can be carried out with the help of LLMs, and that other ways also need to be explored.

6.2 Recommendations

6.2.1 Recommendations for further research

Language models are versatile tools that can be easily adapted to the specific task at hand. The application that is proposed in its current version does not allow the end user to make changes to the instructions that are provided to the LLM directly. Which means that users cannot add to the functionality of the application. In future versions of the application, users could benefit from a feature where they could make small changes or additions to the set of instructions that are given to the language model directly from within Simcenter, providing some sort of basic user level customization to the tool.

Since LLMs are updated very frequently, the performance of the application could also be tested and improved with the help of updated models that are optimized for running locally or models with higher number of parameters along with multi modality as mentioned earlier, which would then allow the generative model to "see" what has already been selected and get a further idea about what can be improved in the geometry. Generative models could also be used in the FEM, wherein they can process the results of a simulation and suggest changes to the properties of the CAD model, to improve the FE result and improve the overall performance of the model.

As mentioned in section (4.3), the testing process for this application was carried out manually for this project, by manually running each query multiple times to check the consistency of the output as well as its success levels. In order to avoid testing manually in the future, it is recommended to set up a Langchain Evaluation tool that can automatically perform the specified tests and log the results. A similar tool in langchain already exists, that is currently used to benchmark Language models and can be adapted to this application for testing different language models and their performance with respect to specified test criteria.

Going further, once it is possible it is recommended to carry out a company wide testing of the application allowing engineers to provide extensive feedback which could then be considered and implemented. This would help get a better idea about the features that engineers prefer. The testing will also help understand if the engineers actually prefer using the automated application or doing the tagging process manually. An extensive evaluation of the impacts of the tool could help shape further improvements in the capabilities of the application more effectively.

6.2.2 Recommendations to GKN

With the help of this project and the accompanying research, it was showcased that Large Language Models and Artificial Intelligence possess the capabilities to augment and improve the tagging process to a certain extent, aiding in geometry identification and selection.

Investing in a multi modal model would also provide the application with the ability to see the CAD model before processing the user input. However, since multi modal models with this kind of capability are expensive to run outside of the company network, and would also require company specific data to leave the secure network, such an implementation would require investing in suitable hardware that is advanced and powerful enough to handle a company wide implementation of a power multi modal generative model. A model that would be suitable to run locally but also provide good performance similar to the performance of the "Claude 3 Opus" in theory would be ideal. Once the implementation of the model is complete, it also recommended to continue refining the application and update the model regularly to ensure improved performance.

This project emphasized on the use of LLMs specifically due to the nature of the data being handled in this case, especially due to the fact that a large enough dataset is not available to effectively train a machine learning model to understand the CAD models and its relevant properties. However, if in the future such a dataset containing potentially hundreds or thousands of such models and their properties, it is recommended to try and train a machine learning model, which in theory would be much more effective in understanding the features and properties that the current application struggles with.

Bibliography

The official YAML web site. URL https://yaml.org/.

Langgraph. URL https://python.langchain.com/docs/langgraph.

Aerospace: World class business with superior margins and strongly growing aftermarket, 2023. URL https://www.melroseplc.net/media/3147/cme-gkn-aerospace-17-may-2023.pdf.

Comparison of ai llm models, 2024. URL https://artificialanalysis.ai/models/gemini-pro.

Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. In *Proceedings of the 35th International Conference on Machine Learning*, pages 40–49. PMLR, 2018. URL https://proceedings.mlr.press/v80/achlioptas18a.html. ISSN: 2640-3498.

Haluk Akay and Sang-Gook Kim. Design transcription: Deep learning based design feature representation. 69(1):141–144, 2020. ISSN 0007-8506. doi: 10.1016/j.cirp.2020.04.084. URL https://www.sciencedirect.com/science/article/pii/S0007850620301062.

et.al Bassel Alamrie. The user of large language models in science: Opportunities and challenges. *National Library of Medincine*, 2023.

Lenz Belzner, Thomas Gabor, and Martin Wirsing. Large language model assisted software engineering: Prospects, challenges, and a case study. In Bernhard Steffen, editor, *Bridging the Gap Between AI and Reality*, pages 355–374. Springer Nature Switzerland, 2024. ISBN 978-3-031-46002-9.

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pages 610–623. ACM, 2021. ISBN 978-1-4503-8309-7. doi: 10.1145/3442188.3445922. URL https://dl.acm.org/doi/10.1145/3442188.3445922.

Britannica. Artificial intelligence, 2024. URL https://www.britannica.com/technology/artificial-intelligence.

Banghao Chen, Zhaofeng Zhang, Nicolas Langrené, and Shengxin Zhu. Unleashing the potential of prompt engineering in large language models: a comprehensive review. 2023. doi: 10.48550/ARXIV.2310.14735. URL https://arxiv.org/abs/2310.14735.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert,

Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL http://arxiv.org/abs/2107.03374.

Peter Chow, Tetsuyuki Kubota, and Serban Georgescu. Automatic detection of geometric features in CAD models by characteristics. 12(6):784–793, 2015. ISSN 1686-4360. doi: 10.1080/16864360.2015.1033345. URL http://www.cad-journal.net/files/vol_12/Vol12No6.html.

Martyn Denscombe. *The Good Research Guide*. Maidenhead, Open University Press, 2014.

GKN Aerospace Website. Gkn aerospace, about us. URL https://www.gknaerospace.com/en/about-gkn-aerospace/.

Jan Göpfert, Jann M. Weinand, Patrick Kuckertz, and Detlef Stolten. Opportunities for large language models and discourse in engineering design. URL http://arxiv.org/abs/2306.09169.

Google Inc. Gemini API additional terms of service, 2024. URL https://ai.google.dev/terms_preview.

S. Jamshed. Qualitative research method-interviewing and observation. *Journal of basic and clinical pharmacy, 5(4), 87–88.*, 2014. doi: https://doi.org/10.4103/0976-0105.141942.

Peng Jiang, Christian Sonne, Wangliang Li, Fengqi You, and Siming You. Preventing the immense increase in the life-cycle energy and carbon footprints of llm-powered intelligent chatbots. *Engineering*, 2024. ISSN 2095-8099. doi: https://doi.org/10.1016/j.eng.2024.04.002. URL https://www.sciencedirect.com/science/article/pii/S2095809924002315.

Heewoo Jun and Alex Nichol. Shap-e: Generating conditional 3d implicit functions, 2023. URL http://arxiv.org/abs/2305.02463.

Sang-Gook Kim, Sang Min Yoon, Maria Yang, Jungwoo Choi, Haluk Akay, and Edward Burnell. AI for design: Virtual design assistant. 68(1):141–144, 2019. ISSN 0007-8506. doi: 10.1016/j.cirp.2019.03.024. URL https://www.sciencedirect.com/science/article/pii/S0007850619300289.

Carmen Krahe, Antonio Bräunche, Alexander Jacob, Nicole Stricker, and Gisela Lanza. Deep learning for automated product design. 91:3–8, 2020. ISSN 2212-8271. doi: 10.1016/j.procir.2020.01.135. URL https://www.sciencedirect.com/science/article/pii/S2212827120307769.

Carmen Krahe, Milan Marinov, Theresa Schmutz, Yannik Hermann, Mike Bonny, Marvin May, and Gisela Lanza. AI based geometric similarity search supporting component reuse in engineering design. 109:275–280, 2022. ISSN 2212-8271. doi: 10.1016/j.procir.2022.05.249. URL https://www.sciencedirect.com/science/article/pii/S2212827122006989.

Shuvendu K. Lahiri, Sarah Fakhoury, Aaditya Naik, Georgios Sakkas, Saikat Chakraborty, Madanlal Musuvathi, Piali Choudhury, Curtis von Veh, Jeevana Priya Inala, Chenglong Wang, and Jianfeng Gao. Interactive code generation via test-driven user-intent formalization. URL http://arxiv.org/abs/2208.05950.

langchain-blog. Reflection agents. URL https://blog.langchain.dev/reflection-agents/.

Shalom Lappin. Assessing the strengths and weaknesses of large language models. 33(1):9–20, 2024. ISSN 1572-9583. doi: 10.1007/s10849-023-09409-x. URL https://doi.org/10.1007/s10849-023-09409-x.

Patrick Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. *ArXiv*, abs/2005.11401, 2020. URL https://api.semanticscholar.org/CorpusID:218869575.

Dacheng Li, Rulin Shao, Anze Xie, Ying Sheng, Lianmin Zheng, Joseph Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. How long can context length of open-source LLMs truly promise? URL https://openreview.net/forum?id=LywifFNXV5.

Anne-Laure Ligozat, Julien Lefevre, Aurélie Bugeau, and Jacques Combaz. Unraveling the hidden environmental impacts of ai solutions for environment life cycle assessment of ai solutions. *Sustainability*, 14(9), 2022. ISSN 2071-1050. doi: 10.3390/su14095172. URL https://www.mdpi.com/2071-1050/14/9/5172.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. 2023. doi: 10.48550/ARXIV.2308.03688. URL https://arxiv.org/abs/2308.03688.

Aditi Mishra, Utkarsh Soni, Anjana Arunkumar, Jinbin Huang, Bum Chul Kwon, and Chris Bryan. PromptAid: Prompt exploration, perturbation, testing and iteration using visual analytics for large language models. 2023. doi: 10.48550/ARXIV.2304.01964. URL https://arxiv.org/abs/2304.01964.

Alex Nichol, Heewoo Jun, Prafulla Dhariwal, Pamela Mishkin, and Mark Chen. Point-e: A system for generating 3d point clouds from complex prompts, 2022. URL http://arxiv.org/abs/2212.08751.

Thiago Rios, Stefan Menzel, and Bernhard Sendhoff. Large language and text-to-3d models for engineering design optimization. pages 1704–1711, 2023. doi: 10.1109/SSCI52147.2023.10371898. URL https://ieeexplore.ieee.org/document/10371898/. Conference Name: 2023 IEEE Symposium Series on Computational Intelligence (SSCI) ISBN: 9781665430654 Place: Mexico City, Mexico Publisher: IEEE.

T. Schick and H. Schütze. True few-shot learning with prompts—a real-world perspective. *Transactions of the Association for Computational Linguistics, vol. 10, pp. 716–731*, 2022. doi: 10.1162/tacl_a_00485.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp, 2019.

S. Fabi T. Hagendorff and M. Kosinski. Thinking fast and slow in large language models. 2023. doi: https://doi.org/10.1038/s43588-023-00527-x.

Dena Taylor. Subject and course guides: Research process: Step by step: Literature review, 2024. URL https://libguides.uta.edu/researchprocess/litreview. [Accessed 08-03-2024].

Oguzhan Topsakal and Tahir Cetin Akinci. Creating large language model applications utilizing langchain: A primer on developing llm apps fast. In *International Conference on Applied Engineering and Natural Sciences*, volume 1, pages 1050–1056, 2023.

Capitol Technology University. The ethical considerations of artificial intelligence, 2023. URL https://www.captechu.edu/blog/ethical-considerations-of-artificial-intelligence.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL http://arxiv.org/abs/1706.03762.

Yueqing Wang, Zhige Xie, Kai Xu, Yong Dou, and Yuanwu Lei. An efficient and effective convolutional auto-encoder extreme learning machine network for 3d feature learning. 174: 988–998, 2016. ISSN 0925-2312. doi: 10.1016/j.neucom.2015.10.035. URL https://www.sciencedirect.com/science/article/pii/S0925231215014940.

et.al Yongjun Xu. Artificial intelligence: A powerful paradigm for scientific research. *National Library of Medicine*, 2021.

Hao Lan Zhang, Christian Van Der Velden, Xinghuo Yu, Cees Bil, Tim Jones, and Ian Fieldhouse. Developing a rule engine for automated feature recognition from CAD models. In *2009 35th Annual Conference of IEEE Industrial Electronics*, pages 3925–3930. IEEE, 2009. ISBN 978-1-4244-4648-3. doi: 10.1109/IECON.2009.5415343. URL http://ieeexplore.ieee.org/document/5415343/.

A

Interview Quotes

Problem	Quote 1	Quote 2	Quote 3
Changes in geometry cause issues with existing tags	"Robustness of tagging is affected by changes in geometry."	"Creation or addition of new surfaces messes up existing tags which requires redoing of the entire tagging process"	"All of a sudden I realize that the geometry has changed too much and now I can't find any of the old tags"
Geometry identification is not possible	"How do I identify which areas to mesh and which not to, or how do I identify a leading or trailing edge of a vane?"	"The tool is not so intelligent to understand that this is where the object is located and this is where the loads are to be applied."	"If we are trying to script certain things, then instead of manually selecting certain geometry, if I can say 'located at this location', that would be nice."
Lack of User friendly interface	"We dont really know what to set in the interface, so that might be a problem."	"If there could be a way, where we don't need to write so much code itself, that would the easiest."	"Current tool is not too user friendly unless you know what you are doing."
Lack of Proper Guideline	"There is a lack of like a proper guideline or and engineering practice on how to do this."	"If I don't find it easy, it does not mean that the code is bad, its just that we need more training."	"I dont do it often, so I dont know the syntax.So in that sense if you can use natural language, that will be a big difference."

Table A.1: Quotes by the engineers on problems that are addressed.

B

Research articles

Sr. no	Article Name	Source	Keywords
1	Assessing the strengths and weaknesses of large language models	Springer Link	Deep learning, Transformers, artificial intelligence, natural language processing.
2	Opportunities for large language models and discourse in engineering design.	ArXiV	Product development process, conceptual design, design methodology, design generation, natural language processing, foundation models, multi-modal models
3	Perspective: Large Language models in applied mechanics	ASME (American Society of Mechanical Engineers)	
4	PromptAid: Prompt Exploration, Perturbation, Testing and Iteration using Visual Analytics for Large Language Models	ArXiv	
5	Unleashing the potential of prompt engineering in Large Language models: a comprehensive overview	ArXiv	Prompt engineering, LLM, GPT-4, OpenAI, AIGC, AI agent
6	On the dangers of stochastic parrots: Can language models be too big	ACM Digital Library	Natural language processing, computing methodologies
7	Agent Bench: Evaluating LLMs as agents	ArXiv	
8	Large language model assisted software engineering: Prospects, Challenges and a case study.	Springer Link	Artificial intelligence, machine learning, natural language processing, Deep learning, AI systems, large language models.
9	Gemini API: Additional terms of use	Google	
10	How long can context length of Open source LLMs be?	Openreview	Large language models, Opensource LLMs, Context lengths, Tokens
11	Interactive Code generation via test driven user intent formalization	ArXiv	

Table B.1: Articles for research question 2

Sr. No	Article Name	Source	Keywords
1.	Deep Learning for automated Product Design(Krahe et al., 2020)	ScienceDirect	Artificial Intelligence, Automation, Computer Aided Design, Machine Learning
2.	Learning representations and generative models for 3D point clouds.(Achlioptas et al., 2018)	Proceedings of Machine learning research (1333 citations)	Generative Models, AutoEncoder, Generative Adversarial Networks
3.	AI based geometric similarity search supporting component reuse in engineering design.(Krahe et al., 2022)	Science Direct	Artificial Intelligence, Design, Pattern Recognition, Product Development, Similarity Search
4.	An efficient and effective convolutional auto encoder extreme learning machine network for 3D feature learning.(Wang et al., 2016)	Science Direct	Auto Encoder, Convolutional, Extreme learning machine, Feature learning
5.	AI for design: Virtual Design Assistant(Kim et al., 2019)	Science Direct	Design Method, Hybrid Intelligence, Machine Learning
6.	Attention is all you need(Vaswani et al., 2023)	ArXiv	Computer Science, Computation and Language, Machine Learning
7	Design Transcriptions: Deep learning based design feature recognition(Akay and Kim, 2020)	Science Direct	Artificial Intelligence, axiomatic design, Deep machine learning
8	Developing a rule engine for automated feature recognition from CAD models(Zhang et al., 2009)	IEEE	System based Design, Automated Feature Recognition
9	Automatic Detection of geometric features in CAD models by characteristics.(Chow et al., 2015)	CAD journal (11 citations)	Automated Feature detection, Computer Aided Design
10	Large Language and text-to-3D models for engineering design optimization(Rios et al., 2023)	IEEE	Artificial Intelligence, Large neural networks, text-to-3d Models
11	Shap-E: generating conditional 3D implicit functions.(Jun and Nichol, 2023)	ArXiv	Computer Vision and pattern recognition, Machine Learning
12	Point-E: A system for generating 3D point	ArXiv	Computer vision, Computer science, pattern recognition,

C

Database Schema

Column Name	Column Type	Max	Min	Count	Unique Values
id	INTEGER	1.00000	1.00000	1	
area	REAL	4448048.15882	4448048.15882	1	
volume	REAL	12762317.49510	12762317.49510	1	
x	REAL	4970.39568	4970.39568	1	
y	REAL	35.35630	35.35630	1	
z	REAL	0.64445	0.64445	1	
name	TEXT	0.00000	0.00000	1	
tag	INTEGER	480363.00000	480363.00000	1	
color	TEXT	0.00000	0.00000	1	79
xaxis	REAL	35.36217	35.36217	1	
yaxis	REAL	4970.39572	4970.39572	1	
zaxis	REAL	4970.52143	4970.52143	1	
yzangle	REAL	1.04424	1.04424	1	
zxangle	REAL	89.99257	89.99257	1	
xyangle	REAL	0.40756	0.40756	1	

Figure C.1: Table Name: Bodies

Column Name	Column Type	Column1	Column2	Column3	Column4
id	INTEGER	8901.00000	1.00000	8901	
area	REAL	301647.75591	0.00009	8901	
perimeter	REAL	7757.28962	0.05983	8901	
x	REAL	5162.67700	4800.16820	8901	
y	REAL	441.52143	-439.83776	8901	
z	REAL	440.89613	-440.98030	8901	
radius	REAL	441.52500	0.00000	8901	
radius_minor	REAL	393700786.40157	-1224.27108	8901	
radius_major	REAL	393700786.40157	-154156059.95062	8901	
type	TEXT	0.00000	0.00000	8901	Planar,Cylindrical,Spherical,Blend,Revolved,BSurface,Conical
,Conical		0.00000	0.00000	8901	
name	TEXT	489855.00000	59371.00000	8901	
tag	INTEGER	0.00000	0.00000	8901	79,87,129
color	TEXT	180.00000	0.00000	8901	
normal_xangle	REAL	180.00000	0.00000	8901	
normal_yangle	REAL	180.00000	0.00000	8901	
normal_zangle	REAL	180.00000	0.00000	8901	
normal_xrad	REAL	180.00000	0.00000	8901	
normal_yrad	REAL	180.00000	0.00000	8901	
normal_zrad	REAL	441.52143	0.00000	8901	
xaxis	REAL	5162.67700	4800.16820	8901	
yaxis	REAL	5162.67700	4800.16820	8901	
zaxis	REAL	360.00000	0.00000	8901	
yzangle	REAL	95.12649	84.87448	8901	
zxangle	REAL	359.99961	0.00000	8901	
xyangle	REAL				

Figure C.2: Table Name: Faces

Column Name	Column Type	Column1	Column2	Column3	Column4
id	INTEGER	20960.00000	1.00000	20960	
type	TEXT	0.00000	0.00000	20960	Intersection,Linear
name	TEXT	0.00000	0.00000	20960	
length	REAL	2690.27429	0.00311	20960	
x	REAL	5162.67700	4800.16820	20960	
y	REAL	441.51787	-440.05652	20960	
z	REAL	452.59516	-452.59516	20960	
radius	REAL	441.52500	0.00000	20960	
tag	INTEGER	487103.00000	60928.00000	20960	
color	TEXT	0.00000	0.00000	20960	79
xaxis	REAL	474.80509	0.00000	20960	
yaxis	REAL	5164.01108	4800.16820	20960	
zaxis	REAL	5164.10950	4800.16820	20960	
yzangle	REAL	360.00000	0.00000	20960	
zxangle	REAL	95.16041	84.83959	20960	
xyangle	REAL	359.99961	0.00000	20960	

Figure C.3: Table Name: Edges

Column Type	Column Description
y	Global y coordinate of the centroid of [Body, Face, Edge]
z	Global z coordinate of the centroid of [Body, Face, Edge]
xaxis	Distance from centroid of [Body, Face, Edge] to x axis
yaxis	Distance from centroid of [Body, Face, Edge] to y axis
zaxis	Distance from centroid of [Body, Face, Edge] to z axis
yzangle	Clocking position of centroid of [Body, Face, Edge] in degrees measured about x axis from y axis to z axis
zxangle	Clocking position of centroid of [Body, Face, Edge] in degrees measured about y axis from z axis to x axis
xyangle	Clocking position of centroid of [Body, Face, Edge] in degrees measured about z axis from x axis to y axis
normal_xangle	Angle of face normal (extracted in the middle of the unwrapped face) with xaxis
normal_yangle	Angle of face normal (extracted in the middle of the unwrapped face) with yaxis
normal_zangle	Angle of face normal (extracted in the middle of the unwrapped face) with zaxis
normal_xrad	Angle of face normal with a line drawn from xaxis to the point where face normal is extracted
normal_yrad	Angle of face normal with a line drawn from yaxis to the point where face normal is extracted
normal_zrad	Angle of face normal with a line drawn from zaxis to the point where face normal is extracted
tag	Tag ID of the [Body, Face, Edge]
name	Name of the [Body, Face, Edge] if any
area	Area of the [Body, Face]
volume	Area of the [Body]
perimeter	Perimeter of the [Face]
radius	Radius of the [Face, Edge]. Can be 0 if face/edge is planar/linear.
radius_major	Major radius of the Face at it's centroid. Can be negative or positive depending on the face normal direction.
radius_minor	Minor radius of the Face at it's centroid. Can be negative or positive depending on the face normal direction.
type	Type of the [Face, Edge]

Column Type	Column Description
bodyfaces	Select faces attached to body
facebody	Select body to which face belongs
edgebody	Select body to which edge belongs
bodyedges	Select edges of a body
edgefaces	Select faces connected to edge
tangent	Applies for both faces and edges; select tangent objects
faceedges	Select edges attached to faces
adjacent	Select faces adjacent to selected faces
pocket	Select faces of a pocket which includes the selected faces
rib	Select faces adjacent to selected faces
connectedblend	Select all blends connected to selected blends
faceboundary	Select all edges which define the boundary of selected faces

Figure C.4: Table Name: Expand Options

D
Source Code

The code used in the implementation can be found in the github repository at the link below:

`https://github.com/atharvanaik98/Thesis/tree/Trials_langchain`

D.1 System Messages and Prompts

In the final proposal, the instructions provided to the Language Model in the case of each call are provided in the context files folder in the repository, the links to which are also given below.

Query Analyzer: qa_instructions.txt (Github Link)
Prompt to the query analyzer:

```
1     prompt = qa_instructions + "Answer the user query. Print the output in Json
      format according to the instrcutions. {format_instructions}. {human_input}
2
3     extracted_features = query_analyzer.features
```

Definer:
Prompt for RAG

```
1     "You are an assistant that is an expert at RAG (Retrieval Augmented
      Generation). You have been tasked with looking for the definition of the
      word provided to you as input and generate a summary of the available
      information. You will use only the context provided to you.
2     Context: {context}
3     Query_to_db: {query}
4
5     query = f"What is the definition of {extracted_features}?"
```

Base_chain: system1.txt (Github Link)

Reflection Prompt: refsystem copy.txt (Github Link)
Prompt message:

```
1     "Generate constructive feedback for the YAML code: {base_chain output}.
      Check for selection capability and correctness of the code based on the
      following prompt that was used to generate this code: {human_input}
2
3     Output = feedback
```

LLM prompt for the manual selection feature:

```
1     Manual selection = selected_information
2
3     "The user has already selected the following information: {selected
      information}. Based on this information, the user will ask you to carry out
       the further steps according to the system message {system1}. If the user
      asks you to select similar geometry, then you are yo maintain a tolerance
      of 5% with respect to the asked criteria."
```

Definitions used for RAG: An example of the way the definition for a flange was created in order to help the model identify the feature efficiently is mentioned below:

```
1
2     # Information about the model:
3     The CAD model is a component for an aircraft jet engine.
4
5     ## Orientation
6     The jet engine, and the component are oriented so that the x axis (positive
      ) is in the direction of the airflow. The component has approximatelly
      axial symmetry around the x axis.
7
8     ## Geometrical components
9
10    ### Flanges
11
12    The purpose of flanges is to connect the component to the neighbour
      component using fasteners. Flanges are planar faces.
13
14    There are forward (fwd) and aft types of flanges. The forward flanges have
      a smaller x dimention ('x' property) compared to the aft flanges.
15    Additionally, the forward flanges interface surfaces have an orientation `
      normal_xangle` between 170 and 190 while the aft flanges have an
      orientation `normal_xangle` smaller than 5 or greater than 355.
16    In addition, the forward flanges are the planar surfaces that have the `x`
      coordinate within 50 units of the min `x` coordinate value.  The aft
      flanges are the planar surfaces that have the `x` coordinate within 10
      units of the max `x` coordinate value. The `xaxis` value is irrelevant.
      When selecting `flanges` it is important to not create any new keys, and
      stick to the database. Make sure to include that in the summary.
17
18    There may be one or two faces that are forward or aft flanges. If there are
       more than one, they are subdivided into "inner" and "outer". The inner
      face has a `perimeter` smaller than the outer face.
19
20    ###
21
22    In the properties mentioned above, the x co-ordinate value indicates the
      position of the flange with respect to the origin, and based on the x co-
      ordinates due to the orientation of the model, the component can be divided
       into two parts, forward and aft. The properties given are example
      properties for a flange, and the dimensions such as radius, x coordinate,
      and other positional parameters might change depending on the CAD model
      being used and its orientation.
```

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY