Systematic Design and Integration of Large Language Model Tools for Engineering Analysis

Gabriel Krüger
Johannes Lundahl

Department of Industrial and Material Science

CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2025

www.chalmers.se

Master's thesis 2025

Systematic Design and Integration of Large Language Model Tools for Engineering Analysis

An investigation, and the development, of large language models tools in analysis engineering

Gabriel Krüger
Johannes Lundahl

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Industrial and Material Science
Chalmers University of Technology
Gothenburg, Sweden 2025

Developing and evaluating large language models to align with engineering tasks and increase their effectiveness
An investigation, and the development, of large language models tools in analysis engineering
Gabriel Krüger, Johannes Lundahl

Supervisors: Alejandro Pradas Gómez, Department of Industrial and Material Science. Najeem Muhammed, GKN Aerospace.
Examiner: Ola Isaksson, Department of Industrial and Material Science.

Master's Thesis 2025
Department of Industrial and Material Science
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: (Insert relevant cover image)

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

Developing and evaluating large language models to align with engineering tasks and increase their effectiveness
An investigation, and the development, of large language models tools in analysis engineering
Gabriel Krüger, Johannes Lundahl
Department of Industrial and Material Science
Chalmers University of Technology

Abstract

Generative AI, and more specifically large language models (LLMs), show great promise in further facilitating and streamlining analysis engineering work. In this thesis, the current limitations for the use of such technologies regarding the incorporation in engineering tasks are investigated. This investigation consists of a comprehensive literature study, in addition to nine interviews with engineers at GKN Aerospace in Trollhättan, Sweden. The results indicate a number of challenges. One being the importance of embedding internal company knowledge when leveraging the capacities of LLMs, while the use of non-local LLMs is linked to considerable issues when it comes to handling sensitive data. With the results of this investigative study as a foundation, an LLM-based tool meant to mitigate these identified issues is developed. The development is done using LangChain, such that the OpenAI API can be used in a Python environment. The developed software is focused on the manipulation, and extracting, of data in CNS files. A file format containing results from finite element simulations. Different agentic systems, leveraging different methods for knowledge embedding such as retrieval augmented generation (RAG), and post-training such as fine-tuning, are investigated. The various architectures are evaluated based on their efficiency and accuracy in solving tasks. The results indicate that LLM-based tools have great potential in the field. The top performing architecture based on this testing is incorporated into a sub-graph architecture, for which usability and validation are examined. The results for efficiency, accuracy, usability testing and validation imply the considerable potential of leveraging LLMs in the domain. Nevertheless, performance is not perfect and a number of considerations in such a development must be taken. The methods for knowledge embedding and post-training seemingly have great impact on the performance, and more sophisticated approaches within RAG and fine-tuning have potential to further improve performance.

Keywords: AI, Large Language Model (LLM), LangChain, Agent, Multi-Agent, RAG, Fine-Tuning, Knowledge-Based Engineering (KBE)

Acknowledgements

First of all, we would like to express our great appreciation of the help we have received throughout this thesis by our supervisors Alejandro Pradas Gómez and Najeem Muhammed. Their knowledge in fields like academic writing and software development has been essential. Additionally, we would like to thank GKN Aerospace Trollhättan and manager Rikard Nedar for having us at the office during this thesis. A special thanks to all engineers at GKN who participated in our interview and usability testing studies.

Gabriel Krüger & Johannes Lundahl, Gothenburg, April 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AI	Artificial Intelligence
API	Application Programming Interface
CAD	Computer Aided Design
CFD	Computational Fluid Dynamics
FEM	Finite Element Method
HITL	Human In The Loop
LLM	Large Language Model
QA	Question & Answer
RAG	Retrieval Augmented Generation
RQ	Research Question

Contents

List of Figures

List of Tables

1
Introduction

The use of AI has, over the course of the last years, shown great promise in further optimizing work procedures in a number of engineering domains. AI is used during R&D phases within fields like the automotive and aerospace industry, for example in the design stage where certain CAD software integrates AI features (Financial Times 2024). Classic simulation methods, like finite element method (FEM) and computational fluid dynamics (CFD) tools, are still significantly more common in engineering than AI and machine learning (ML) techniques (Ragani et al. 2023). Nevertheless, it is considered that the incorporation of AI methods can further streamline these "classic" simulation workflows within the pre- and post-processing stages of data. Consequently, this study investigates how specific analysis engineering procedures can be further streamlined with the help of AI tools, and what the corresponding challenges are.

1.1 Background

GKN Aerospace is a leading company in the aviation industry, collaborating with the world's top manufacturers of aircraft and engine components. The Aero Engines division in Trollhättan, Sweden, specializes in high-performance engine components and have facilities and engineering teams in several places all over the world. Here, advanced parts for aircraft and rocket engines are developed and manufactured, along with engine maintenance services (GKN-Aerospace 2024).

In recent years, AI and particularly generative AI has advanced significantly. Due to extensive research in this field, generative AI now supports various applications within the tech sector, including code generation, product design and content marketing (Fauscette 2023). Central to these applications are large language models (LLMs), a key component of generative AI. LLMs are advanced deep neural networks designed to process and generate content such as text, images and videos. These models are built on a transformer architecture, which is particularly effective at understanding the context within sequences of data (IBM 2023). This architecture enables LLMs to capture meaning more effectively than previous types of neural networks like CNNs or RNNs (NVIDIA 2023). LLMs have thus become essential tools for many tech companies and their employees enabling things like automation of processes and improving product development through advanced analytics (Kietzmann and Park 2024).
To facilitate and optimize the work of employees at GKN, the potential of LLMs

is currently being explored as a support tool in their engineering tasks, particularly for engineers in the solid mechanics department. Previously, at this specific department, LLMs have been tested as tools for assisting in report writing. This was done in a study titled *"Supporting the Generation of Engineering Analysis Reports with Large Language Models"*, written by D. Söderqvist and F. Mare (Söderqvist and Mare 2024). Additionally, the use of LLMs have been investigated at GKN for the preparation of CAD geometries for FE analysis (Naik 2024).

Consequently, there is an interest at GKN regarding how LLMs further can be deployed within analysis engineering for the streamlining of work procedures. In this thesis, it is investigated how LLMs can aid over the course of engineering analysis linked to simulation processes. For an engineering simulation involving a geometry, it generally requires the pre-processing of the geometry with corresponding loads and material properties such that a run script can be set up. After this, the simulation tool can be run, leading to a results file containing simulation data. Subsequently, this data must be analyzed and manipulated. This workflow is illustrated in Figure 1.1.

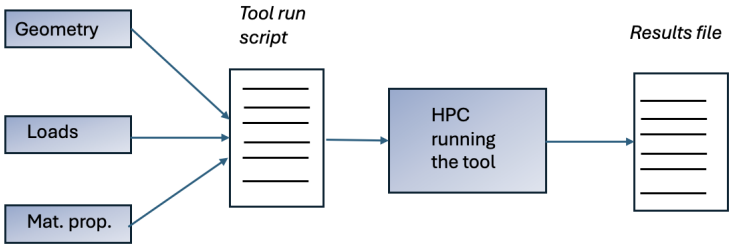


Figure 1.1: General flowchart for solid mechanics analysis using a tool

At the solid mechanics department, this can be a simulation process through a FE analysis tool. Over the course of this procedure, data must be processed before the simulation can be run as well as after its completion. Therefore, the task of an analysis engineer is to both pre-, and post-process data. The incorporation of AI, and more specifically LLMs, in this workflow can be achieved in various stages as indicated by previous studies at GKN. The foundation of the the thesis is therefore to investigate the further incorporation of LLMs in the analysis flow. This is done with a number of research questions, acting as guidelines for the methodology, presented later in this chapter.

1.2 Scope

Given the desire to further investigate the incorporation of LLMs in engineering tasks as GKN, two research questions (RQs) linked to this are defined. These act as a guide for the research metholodgy, such that a well structured The two RQs are:

1. What are the practical limitations for analysis engineers regarding adaptation of generative AI technologies in their engineering tasks?
2. How can an LLM-based tool be systematically designed and evaluated to mitigate the identified limitations?

1.3 Limitations

For this thesis, a number of limitations are considered. These limitations are of time, data security and financial nature. Accordingly, these limitations are as follows:

- Time: the project is limited to a period of about 20 weeks.
- Data Security: No confidential GKN data is considered during the research. This means that development is based on generic data, together with non-confidential insights from GKN engineers.
- Financial: A budget cap of approximately $1300 is given for utilizing cloud-based LLMs.

1.4 Ethical and Environmental Considerations

A number of ethical, and environmental, considerations must be taken into account during the development, and use of, generative AI tools. For example, the current development of LLMs has been criticized. When the Future of Life Institute in March 2023 published the open letter "Pause Giant AI Experiments", signed by a number of well-known persons within the domain of AI research and governance, it pointed out potential LLMs more capable than OpenAI's GPT-4 model and called for a minimum six month pause for the development and training of such models such that AI safety and governance measures could be implemented (Mollen 2025). The signatories expressed their concern, and questions treating topics like "nonhuman minds that might eventually outnumber, outsmart, obsolete and replace us?", and "let machines flood our information channels with propaganda and untruth?" were brought forward (Future of Life Institute 2023). While the scope of this thesis does not concern the development of an LLM, but rather the development of an LLM based tool using existing models, it is important to keep the general ethical and existential questions with respect to LLMs in mind.

There are environmental considerations that must be taken into account during the development, but also use, of LLMs. LLMs have a significant carbon footprint, where one of the reasons is the heavy GPU usage during training (Faiz et al. 2023). For the data centers associated with LLMs, there are numerous factors that determine the environmental impact. Data center energy usage, with the corresponding share of carbon free energy, as well as the embodied carbon footprint of the hardware

must be taken into account (Faiz et al. 2023). For its lifecycle, the carbon footprint of an LLM caused by inference is larger than the one caused by its initial training (Fu et al. 2024). The carbon emissions for GPT-3 caused by training are estimated to 502 tCO2 (Patterson et al. 2021). Furthermore, for GPT-3 the energy demand per query has been estimated to be between 0.002 and 0.005 kWh/query, and with an estimated US carbon intensity of 367 gCO2/kWh the emission per request has been approximated to 1.5 g CO2 (Vanderbauwhede 2024). With millions of requests every day, this highlights the energy needs and therefore the emissions of LLMs not only for training but also for daily use. Consequently, for large scale LLM use the environmental aspect must be taken into account.

The question of ethics is not only important with respect to the subjects the thesis is treating, but also with respect to the applied research methodology. For example, a number of interviews are conducted with engineers at GKN. Ethical considerations within fields such as privacy, sensitive information and correct citation must be taken into account. Storage of recorded interviews must be in line with GDPR, where it states that the storage of data that enables personal identification may only be stored for as long as the specific purpose dictates (GDPR.eu 2025). Furthermore, it should not be possible to determine an interviewee's identity based on the content of the report. Additionally, it is of importance that interviews are conducted such that subjects are not led to disclose irrelevant personal information or sensitive company information.

1.5 Prerequisites

Before presenting the thesis methodology, it is necessary to present a number of prerequisites. These act as the foundation for how the identified issues could be mitigated, and how the desired functionality could be implemented.

1.5.1 AI agents

For tools with AI functionality incorporated, agentic AI systems can ensure that systems or programs within the tool can perform specific tasks through, for example, calling external tools (IBM 2024a). The task of an agent may vary, and it can include tasks like analyzing written texts for grammatical errors to writing computer code. Therefore, there are AI agents of different nature. For example, a so-called "simple reflex agent" does not interact with other AI agents and is pre-programmed such that it performs a specific action given a specific pre-condition (IBM 2024a). There are more sophisticated types of AI agents, and some can determine a specific order to implement certain actions given a specific goal and an optimization algorithm (IBM 2024a). Therefore, the use of AI agents has the goal to optimize LLM processes. This by creating more complex and autonomous systems, that themselves can make independent decisions and leverage external tools. Additionally, an AI agent can reflect on the user input and plan what to do next based on its available tools (IBM 2024a). These tools can be for instance code executing tools, web-search tools or external API calling tools. The agent can also reflect on its own mistakes and correct

those without any human interference. However, in order to make the agents more reliable, it is sometimes useful to make use of active human interaction through the incorporation of human in the loop (HITL) aspects (IBM 2024a).

1.5.1.1 Human in the loop (HITL)

Due to the fact that AI agents are partly or fully autonomous systems, there is a risk that they might make unwanted or even harmful decisions. To address this, it is often useful to implement HITL mechanisms that allow the human to intervene. That is, halting or modifying the agent's actions before it proceeds. This approach not only helps to maintain the reliability and accuracy of the responses, but it also prevents issues such as the LLM entering infinite loops (IBM 2024a).

1.5.2 LangChain

LangChain is an open-source framework designed to simplify the development of applications that leverage LLMs. It provides a standardized and modular approach to integrating LLMs with external data sources, APIs, and computational tools, enabling the creation of more advanced AI-driven workflows beyond basic API calls (Balasubramaniam et al. 2024). By offering a flexible architecture, LangChain supports RAG, memory management, and tool integration, making it particularly valuable for tailored applications requiring contextual awareness and reasoning. LangChain enables developers to design applications that integrate LLM capabilities with real-world operations, and has gained significant adoption in the AI community, establishing itself as a popular tool for building generative AI applications (Balasubramaniam et al. 2024).

1.5.3 LangSmith

LangSmith is an observability platform designed to enhance the development and maintenance of LLM-based applications. Created by the developers of LangChain, it provides essential tools for debugging, monitoring, testing, and optimizing AI systems, improving both reliability and interpretability (Balasubramaniam et al. 2024). The debugging and tracing capabilities enable developers to track input-output interactions, and analyze intermediate steps. Additionally, the available testing and evaluation tools facilitate benchmarking across various use cases, ensuring robust and well-optimized LLM applications (Balasubramaniam et al. 2024).

1.5.4 LangGraph

LangGraph is an open-source library designed for building stateful, multi-actor applications with LLMs. Allowing for the creation of agent and multi-agent workflows. It offers control over both application flow and state while integrating seamlessly with LangChain and LangSmith (LangChain Inc. 2024a). LangGraph uses nodes and edges to model functionality and behavior, effectively linking the various structural components of an agent. The nodes are simply Python functions describing

the behavior of each part of the agent, while edges enables the routing logic between the nodes in the agent (LangChain Inc. 2024c). There are two types of edges. The normal edge, where the routing is defined in one way only is represented by a solid line. The conditional edge on the other hand can take more than one way depending on the condition defined and is represented by dotted lines. These are illustrated in Figure 1.2. Important features of LangGraph, that have been relevant and widely used in this project, are memory persistence, HITL and centralized state management, which keeps track of global state updates (LangChain Inc. 2024a). Another key feature of LangGraph is the LangGraph Studio. LangGraph Studio allows users to visually deploy graphs using the LangGraph API and also supports quick debugging (LangChain Inc. 2024b). In Figure 1.2 a simple visualization of a graph in LangGraph is shown.

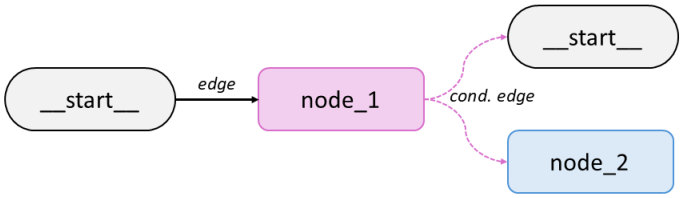


Figure 1.2: Visualization of nodes and edges in LangGraph.

1.5.5 Knowledge embedding methods

The use of LLMs is associated with hallucination due to, for example, that the model has not been trained on certain specific data. Consequently, when using LLMs for fields like engineering there is often a need to embed internal company knowledge. Knowledge that is deemed external to the LLM, since it has not been trained on it. There are different ways to embed external knowledge within an LLM architecture. In the thesis methodology, three different ways are encountered. These are presented here.

1.5.5.1 Entire Context in System Prompt

When using an LLM through an API, one way to embed external knowledge is to explicitly state it in the system prompt of the LLM. How much that can be incorporated in the system prompt is dependent on the context length of the model. Meaning that if you insert an amount of information exceeding the context length, not all of it will be considered. For example, GPT-4o has a context length of 128 000 tokens (OpenAI 2024a). In such an architecture, the model has access to the same specific external knowledge regardless of the prompt defined by the user. This is illustrated in Figure 1.3.

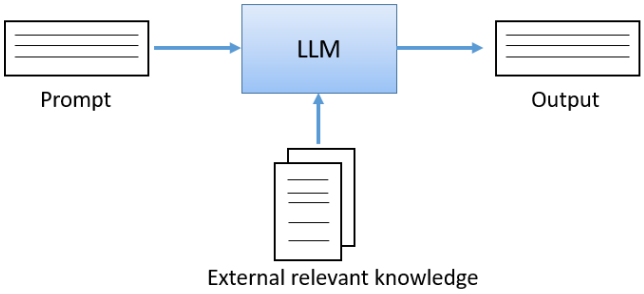


Figure 1.3: Simple architecture with all the external knowledge accessed by the LLM

1.5.5.2 RAG (Retrieval Augmented Generation)

As opposed to providing the entire context in the system prompt, the RAG architecture is a bit more sophisticated in its nature. Its foundation is the splitting of documents and consequently the retrieval from certain parts, given a user question, and not the entire documentation. The information from the knowledge base can be split into several smaller segments through *chunking*, where the chunk size sets the size of the data splits and while a large chunk size may ensure that the data points are coherent the risk is they become too general while a small chunk size risks the loss of coherence within the data points (IBM 2024c). There is no exact answer for what a good chunk size is, and it is dependent on the type of documentation you have. If a chunk contains an excessive amount of data there is a risk that specific knowledge may be overshadowed by other topics, whereas a risk of context loss is associated with a too small chunk size (Stack Overflow 2024). Additionally, when chunking a document it may be split in sections where relevant specific knowledge is divided into two different chunks as opposed to keeping it all in one chunk. To mitigate this, documents can be split into chunks with a certain overlap, ensuring no brusque sudden borders between chunks but rather an overlap between chunk data (MongoDB 2024).

After chunking the knowledge base, the text data can be transformed through an embedding model such that it is vectorized. Embedding is a method for the representation of, for example, text in a numerical way that allows for the use of the data as input to machine learning algorithms since they in general require the input to be low-dimensional numerical (IBM 2025b). As a result, this can improve the performance of the LLM. There are different embedding models, and one example is the OpenAI method used in ChatGPT which allows the model to, as opposed to evaluating each word separately, comprehend how words and categories are linked allowing for improved responses (IBM 2025b). Accordingly, the knowledge base can be transformed to a vector database where all the text is represented through vector embeddings. As opposed to traditional search, where results are explicitly retrieved based on the input word, vector databases enables search based on similarity (IBM 2025a). Meaning that for an input prompt asking a question regarding the word "smartphone", the traditional search retrieves passages explicitly containing

the word "smartphone" whereas a retriever from a vector database would yield results containing similar words such as "cellphone" (IBM 2025a). There are different algorithms for the actual retrieval. Some algorithms are based on *k Nearest Neighbours* (kNN) methods, such that the retriever searches for a number of k vectors that are deemed to be the mathematically closest to the defined query vector (Sawarkar, Mangal, and Solanki 2024). A RAG architecture with text splitting, embedding and retrieval is shown in Figure 1.4.

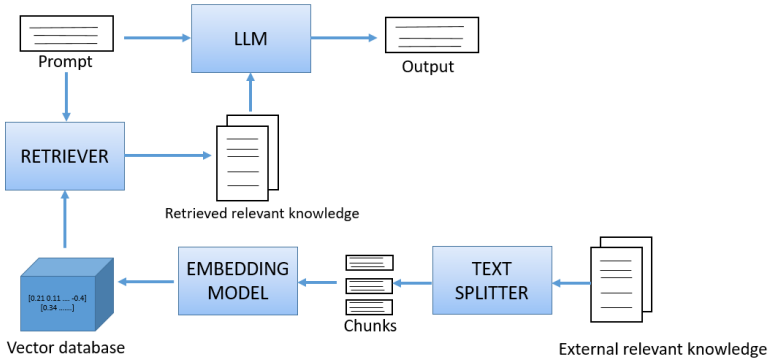


Figure 1.4: RAG architecture

1.5.5.3 Fine-tuning

Fine-tuning, in comparison to the previously mentioned knowledge embedding methods, is a technique to adapt the tailored knowledge through additionally model training on an already pre-trained neural network. However, in a pre-trained model, there might be millions or even billions of parameters that it has been trained on. Changing the weights of all parameters is time-consuming and expensive and also runs a high risk of overfitting the network (IBM 2024b). Instead, fine-tuning makes use of the already pre-trained parameters and changes are only made to a few parameter weights in the model. This enhances optimal behavior of the model since it retains the robustness and complexity of a pre-trained model, while leveraging the customization to a specific use case. Fine-tuning is commonly used when customizing neural networks with a large amount of parameters such as LLMs and computer vision models. Examples of fine-tuning use cases include developing specialized LLMs, such as those designed for code generation or for replicating specific tonalities and writing styles (IBM 2024b). The foundation of fine-tuning, with a base model yielded through an enormous amount of general pre-training data and a fine-tuned model yielded through specifically curated fine-tuning data, is shown in Figure 1.5.

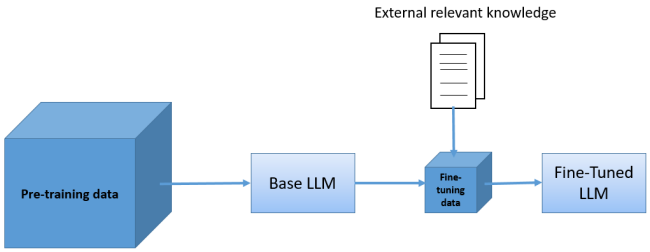


Figure 1.5: Fine-tuning with a data set given internal knowledge

1.5.6 GKN Dummy Data

As presented in section 1.3, no confidential data is used in this project due to GKN's security policies. Consequently, GKN provided a dummy structural analysis case. The structure of this dummy data was created from a real case, ensuring it aligns with real-case analysis while being significantly more compact with a highly simplified geometry. Although the dataset contains less information and only non-sensitive data, its structure and file formats closely emulate real-world data and structure. The file structure of the dummy analysis case is presented in **??**.

1.5.7 PyCNS Knowledge

PyCNS is an internal Python module used at GKN for extracting and manipulating data in CNS files, who are a result of a FE simulation. The module is treated in the thesis, and both source code and an existing user manual was provided. Therefore, the documentation was both of a more descriptive nature and a pure source code nature. An example for how the source code knowledge was given is displayed below. This is meant to show the structure, since some of it has been redacted in order to not display the entire source code.

```
def combine_cns(inputs) -> CNS:
    """Combine multiple CNS objects into a single CNS object.

    Short description of functionality and use case.

    Example:
        new_cns = combine_cns([cns1, cns2, cns3])

    Args:
        inputs: A list of CNS objects to be combined.

    Returns:
        CNS: A combined CNS object.
    """
    # ...

    new_cns = ... # Redacted combination logic

    return new_cns
```

Listing 1.1: Redacted combine_cns function

Instead, the internal PyCNS manual did not contain any source code for the functions but rather short descriptions and examples in a pdf. A somewhat redacted extract from it, for combine_cns, is shown below.

Extract from Internal Manual

1.1 `pycns.combine_cns`
`combine_cns(inputs) → CNS`
Function to combine multiple CNS objects into a single CNS object.
Short description of functionality and use case.
Example:

```
>>> large_cns = combine_cns([cns_list])
```

Parameters:
- Information about inputs

Returns:
- A pycns.CNS object.

Return type: CNS

.

1.5.8 Computational Effort Parameters

When working with an LLM-based tool, it is important to measure the software's efficiency. This can be evaluated based on both the computational effort and the computational time required to complete specific tasks. Computational time can be measured based on the latency associated with a given task, while computational effort can be evaluated by analyzing the number of tokens required to generate the solution. When evaluating the performance on a set of questions, two key latency metrics can be used. The P50 latency, also known as the median latency, measures the response time at the 50th percentile, meaning that half of the responses are faster and half are slower than this value. This metric provides an estimate of a typical response time, giving insight into how quickly an agent generally responds under normal conditions. The P99 latency, on the other hand, represents the 99th percentile latency. This value indicates that 99 percent of all responses were completed in less time, while the slowest 1 percent exceeds this threshold. The P99 latency is particularly useful for assessing worst-case scenarios, ensuring that the system does not experience excessive delays in response generation. This concept is displayed in Figure 1.6

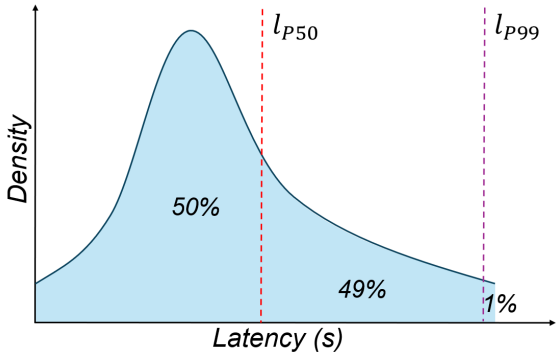


Figure 1.6: P50 and P99 latencies

As mentioned above, the computational effort of an LLM can be measured through the number of tokens that is needed to satisfactory answer a specific request. In natural language processing, text is divided into a certain number of tokens such that the question prompt defined by a user to an LLM is split into different sections (Microsoft Research 2023). The input is tokenized. This means that the input text specified by the user, as well as the output text generated by the LLM, corresponds to a number of tokens. For OpenAI's models, a general rule of thumb is that one token can be approximated to four characters of text (OpenAI 2024c). Consequently, an interaction with an LLM where the user has to correct the LLM and provide more context before receiving an acceptable answer will require more tokens both in the form of input and output, and will consequently be more computationally demanding as opposed to a correct answer yielded directly from a well written prompt. For a user, there is also a financial aspect to the number of tokens used. OpenAI API pricing is based on number of tokens, such that GPT-4o has a cost of \$2.50 / 1 M input tokens and \$10 / 1 M output tokens (OpenAI 2024b).

2
Method

In this chapter, the methodology for answering the two RQs is presented. For answering RQ1, namely *what the practical limitaitions for analysis engineers regarding the adaptation of generative AI are*, this is done through data gathering with the help of a literature review and an interview process conducted with engineers at GKN. The result for RQ1 serves as a base for the methodology when it comes to answering RQ2, namely *how an LLM-based tool can be systematically designed* given the limitations and challenges identified in the RQ1 result. This consists of the software development of an LLM-based tool and corresponding evaluation of it. The methodology workflow is displayed in Figure 2.1.

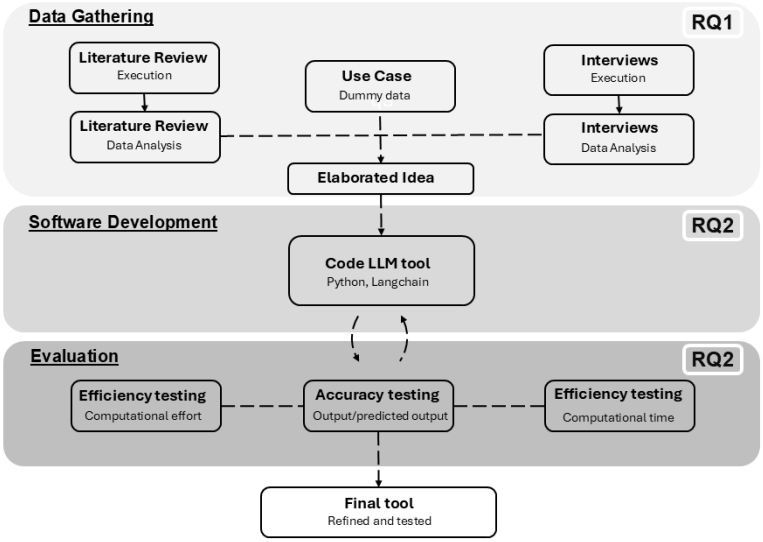


Figure 2.1: Components of the methodology for answering the RQs

2.1 Data Gathering

This section presents the *Data Gathering* process, which primarily consists of a literature review and interviews. For completeness, previously provided dummy data is also incorporated into the flowchart, in Figure 2.1.

2.1.1 Literature Review

A literature review was performed in order to gain a greater understanding of the use of generative AI in engineering, and what the potential challenges in its use may be. To ensure a fruitful and well-structured literature review, the methodology was divided into three main stages. These are *Identification*, *Screening* and *Review*. This methodology is modeled on Page et al. (2021), where an initial number of screened records successively is reduced such that a number of relevant and high quality papers are selected for the final analysis. The methodology for this process is further described in this section.

2.1.1.1 Execution

The basis for the purpose of the literature review was RQ1, namely *what the practical limitations* are for the use of generative AI in an engineering context. In order to perform a structured literature review that can be easily repeated in the greatest possible measure, a number of keywords were considered. These keywords were classified in three major categories. These are *AI Concept*, *Application* and *Embedded knowledge*. The motivation to this was that it facilitates the actual generation of keywords during the thought process, but also structures later database searching when using Boolean operators. The idea was that articles containing at least one keyword from each class were desired. This means that the *OR* operator could be used within a class, and the *AND* operator between classes. The corresponding keywords are presented in Table 2.1.

AI Concept	***Application***	***Embedded Knowledge***
Large Language Model (LLM)	Design Process	Fine-tuning
Generative AI	Knowledge Management	Coupling
	Systems Engineering	Role-specific
	Mechanics	Knowledge-based
	Complex Systems	In-house
	Requirements Engineering	
	Finite Element Method	

Table 2.1: Keywords for literature search

An initial search in Scopus with these keywords yielded a total of 97 results, and while some of these were deemed to be relevant another search was performed such that more records could be screened. The decision was made to only search for the *AI Concept* and *Application* keywords. The screening procedure for the literature review is shown in Figure 2.2. This method is highly influenced by *The PRISMA 2020 statement: an updated guideline for reporting systematic reviews* (Page et al. 2021). This meant that the literature review was divided into three stages. *Identification*, where keywords were chosen and further refined after an initial search in Scopus. *Screening*, when records were excluded based on their titles, abstract, judged overall

quality and number of citations. Lastly, during the *Review* stage the records included for the literature review were analyzed.

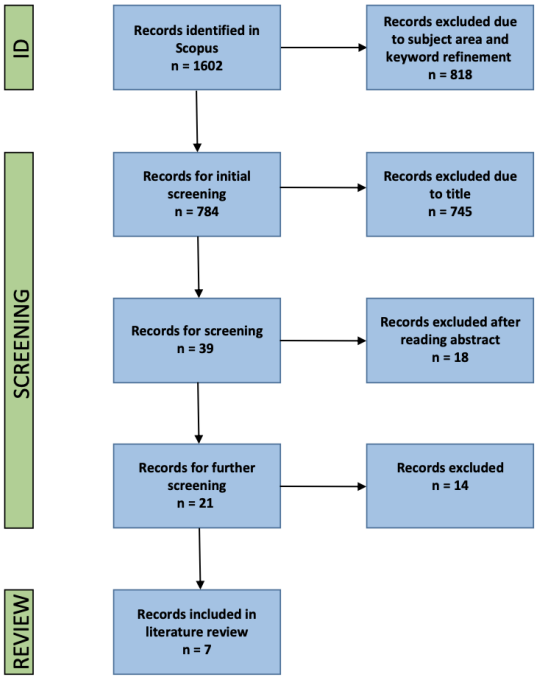


Figure 2.2: Screening procedure in Scopus for the literature review. Highly inspired by Page et al. (2021)

The first keyword search in Scopus led to 1602 results. This was further refined in the Scopus filter by selecting *Engineering* and *Computer Science* as subject areas, English as language, and selecting keywords *language model*, *large language model*, *knowledge management*, *requirement engineering* and *knowledge engineering*. This led to 784 results in Scopus. An initial screening was performed by simply reading the titles. Titles who were deemed relevant with respect to the RQs were of interest. Consequently, titles indicating studies in fields like medicine and linguistics were ignored. Therefore, 39 records were selected for further screening. For each of these 39 records, the abstract was studied. Once again, records deemed relevant with respect to the RQs were selected for further screening. This meant a total of 21 records. These records were subject to the final screening. Here, the entire text was analyzed such that its quality and relevance could be judged. Number of citations were also taken into account, which means that articles ideally should have been deemed relevant and of high quality after an analysis of the text, and well cited. A number of 7 records were chosen for literature review in Scopus. In order to increase this number, snowballing together with limited keyword search was performed on Arxiv.org. Hence, this lead to 11 articles in total for the literature review.

2.1.1.2 Data Analysis

For the data analysis stage of the literature review, a content analysis was performed for each chosen article. The articles were read first such that an initial understand-

ing of research questions, methods and results could be obtained. After this, the articles were read through again and sections that were deemed relevant with respect to the defined RQs were highlighted. After having performed this for all the selected articles, these highlighted sections were divided into different themes. The thematic analysis was performed as presented by Säfsten and Gustavsson (2020). The themes obtained from the thematic analysis included for example *Lack of in-house knowledge*. For each theme, the findings for the articles were summarized in a coherent text. These are presented in section **??**.

2.1.2 Interviews

Such that a further understanding of the topic of the potential and challenges of generative AI in engineering could be established, a number of interviews were performed. The advantages of interviews include flexibility, the ability to tailor questions for specific cases (Säfsten and Gustavsson 2020). In this case, the focus was to collect and obtain relevant data in order to answer RQ1. However, there are also disadvantages that must be taken into consideration. One such disadvantage is the fact that a poorly chosen respondent could lead to deceitful results (Säfsten and Gustavsson 2020). Nevertheless, due to the fact that interviews enable a direct contact with respondents who may hold key answers regarding the research questions it was deemed appropriate. The interview process contained two main stages. These were *Execution* and *Data Analysis*, illustrated in Figure 2.3. The methodology for these is presented in the following sections.

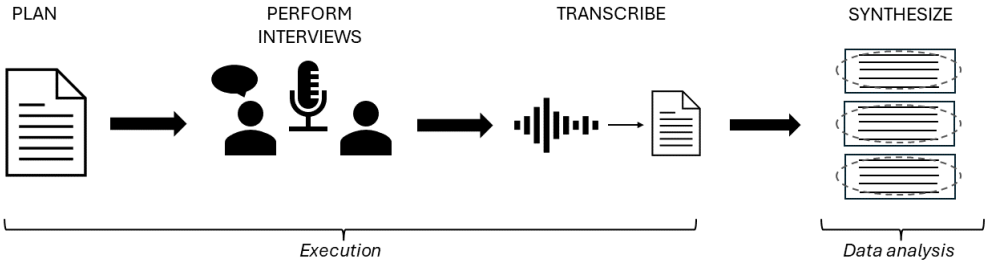


Figure 2.3: Stages for the interviews

2.1.2.1 Execution

The respondents of interest were engineers at GKN Aerospace who have experience, or have previously possessed experience, in simulation based working processes. Ideally, it was desired that the respondents should have had diversified experience in the topic of AI and LLMs, and also regarding the incorporating of generative AI in their work. Nevertheless, a variation in type of respondents was desired since this counteracts biased results and generally widens the perspective in the answers. The interviewees where chosen with help from the supervisor in accordance with the stated requirements. A list of the respondents can be seen in Table 2.2. Some interviews were performed in person at GKN Aerospace in Trollhättan and some were performed digitally.

Semi-structured interviews were performed. This is the mix between the completely structured interview and the unstructured interview. Whereas the structured interview is governed by fixed questions, and the unstructured interview is very open with more of an overall theme deciding the discussion, the semi-structured is a sort of in-between (Säfsten and Gustavsson 2020). The primary reason for conducting semi-structured interviews was the variability in the respondents' fields of expertise, allowing for dynamic adaptation of questions during the interview. Furthermore, the limited prior knowledge of certain processes is also a strong reason to perform semi-structured interviews. Consequently, the interview can be adapted more to the prior knowledge of the topic that the interviewee possesses. The questions for these interviews are presented in section A.1.

In a study made by Griffin and Hauser (1993) the number of interviews needed to identify a given amount of customer needs was investigated. The data was collected and analyzed by professionals in the field. The study revealed that the added value plateaued after approximately five to six interviews, see Figure 2.4. Therefore a number of nine interviews, each lasting approximately 25 to 30 minutes, was deemed relevant for this project.

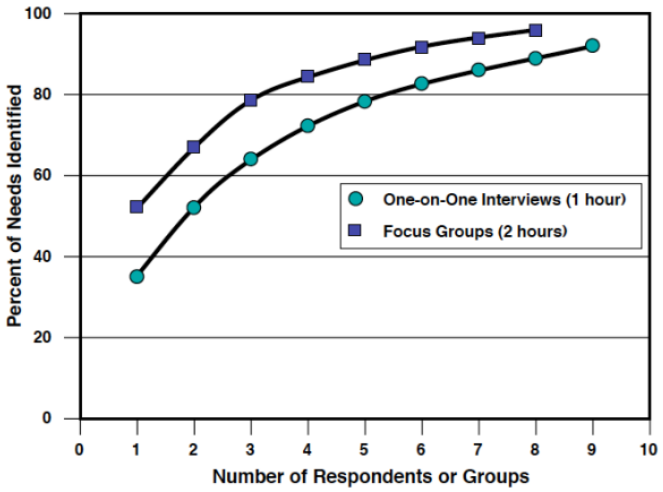


Figure 2.4: Number of interviews in relation to identified needs (Griffin and Hauser 1993)

Nevertheless, there was a question for how these nine interviewees were going to be selected. Therfore, *Sampling in design research: eight key considerations* by Cash et al. (2022) was used as a foundation for the choice of these interviewees. Since the LLM-based tool was going to be developed on a GKN specific case, the sampling was performed within a set of GKN engineers.The consideration *Design framing: what type of impact on practice do you hope to achieve?* presented by Cash et al. (2022) emphasizes the link between the conducted design research and practice, and highlights its importance. Another consideration presented by Cash et al. (2022) is *Theoretical framing: where in the theory-building/theory testing research cycle is current knowledge?*, where relevant domain knowledge within a population must be considered. Interviewees were therefore to be selected based on their knowledge

and experience within analysis engineering, but also knowledge and experience and knowledge within AI. Knowledge connected to both of these domains was highly desired with respect to the RQs. Therefore, these two factors were deemed to provide a solid foundation for gathering relevant existing knowledge within the sample, but also gaining an understanding of the possible impact on current analysis practices that the thesis could yield.

The profiles for the nine interviewees are shown in Table 2.2. The duration of the interviews was deemed sufficient to allow for the collection of comprehensive information while ensuring the project remained manageable within the given time-frame. The interviews were be recorded using a SONY ICD recorder, and the recordings were then transcribed using an offline, locally stored transcription software. This approach ensured that the content remained secure and was not publicly accessible under any circumstances, in accordance with the aspect of personal information discussed in section 1.4. Furthermore, the interviews were conducted in English to streamline the transcription process, as the transcription tool used was better optimized for English speech recognition. After the automatic transcription was done, the content was manually revised to entirely comply with the interview recordings.

With the aim of working in accordance to the policies and statutes of GKN regarding data security, an interview contract was made. Before the interview, the contract was provided to the respondent for signature. The contract contained information about how the recordings were to be managed and served as an additional safeguard between the interviewer and the respondent. This was deemed to be in line with one of the considerations presented by Cash et al. (2022), namely *Good scientific conduct and ethical appropriateness*. The contract can be seen in section A.2.

ID	Title	Role	Site	AI Exp.
1	Research engineer	Research in digitalization and automation. Working with AI dev.	Sweden	Very High
2	Eng. method specialist	Assessing fatigue life in manufacturing; specialist in crackprop.	Sweden	Low
3	Analysis engineer	Analysis lead, supporting hardware design in INC and Pratt & Whitney projects.	Sweden	High
4	Analysis engineer	Solid mechanics analysis and simulations in ANSYS.	Sweden	Low/Medium
5	Analysis engineer	Solid mechanics analysis and simulations in ANSYS.	Sweden	Low/Medium
6	Structural mech. engineer	Simulation and analysis consultant in finite element analysis.	Sweden	Medium
7	Analysis lead	Leads structural engineering team in India and senior engineers at GKN.	Sweden	Low
8	Design engineer	Automating design processes and methods used in the company.	Netherlands	Low
9	Eng. team leader	Working within design principles.	Sweden	Low

Table 2.2: Respondent details, roles, and AI experience.

2.1.2.2 Data analysis

Having transcribed the interviews, a thematic analysis of the interviews was performed. A thematic analysis aims to identify recurring themes within the interview data. These themes represent patterns or insights that may be relevant answers to the RQs (Säfsten and Gustavsson 2020). In order to find themes, it is useful to assign codes to the data. These codes were extracted using a combination of inductive and deductive coding (Fereday and Muir-Cochrane 2006). Hence, a code was assigned to parts of the data deemed to be valuable in answering the RQs, which were then paired and divided into labeled themes. These themes however, in contrast to a pure deductive coding strategy, might be defined in advance in accordance to a inductive coding strategy. The themes are in turn linked to a final theory (Säfsten and Gustavsson 2020). Therefore, the transcribed interviews were read through and relevant comments and responses were highlighted. These highlighted sections represented codes, and from these codes a number of themes were defined. Each code was connected to a corresponding theme, and some codes were connected to more than one theme. These themes are presented in section 3.1.2.

2.2 Software Development Method

After having finished the interview and literature review stages, intended to answer RQ1, the development of an LLM-based tool could be started. This development was intended to answer RQ2, with the results from RQ1 as a foundation. In this section, the development stages are presented. More precisely, how the desired functionality of the LLM-based tool was decided, how an initial strategy was outlined and how it was actually developed.

2.2.1 Development Objective

RQ2 considered *how* an LLM-based tool could be developed and evaluated, given the obtained results from RQ1. These results, presented in chapter 3, highlighted various issues connected to the use of generative AI tools both in a broad industry perspective as well as with respect to analysis engineering practices at GKN. While it would have been preferable to develop an LLM-based tool that could mitigate all these issues and be implemented in the entire analysis workflow displayed in Figure 1.1, this was deemed to be too comprehensive given the thesis timeline. Therefore, the key takeaways from the data gathering were used as a foundation for deciding the development objective. The interview study seemed to give an indication that the post-processing stage of the analysis was of a more cumbersome nature than the pre-processing stage. For example, interviewee 7 emphasized the time-consuming aspect of the post-processing of data in subsection 3.1.5. This meant, among other, identifying relevant data and extracting the relevant parameters to present. Furthermore, the potential of an LLM when it comes to dealing with extensive result data and summarizing key parts was mentioned by interviewee 8 in subsection 3.1.7.

At GKN, there are a number of internal Python modules for facilitating the anal-

ysis and manipulation of certain file formats that are the result of a FE simulation. Therefore, these modules already aim to simplify the cumbersome post-processing stage. Nevertheless, some engineers may be less experienced when it comes to Python. This Python learning curve was outlined by interviewee 6 in subsection 3.1.7. The combination of existing Python scripts for automation, and an LLM-based tool to make these more accessible, was considered as a consequence to this. Interviewee 8 discussed this possible integration in subsection 3.1.7. However, there are such Python modules for the analysis and manipulation of CNS, CDB and UNV files among other. Together with the GKN supervisors, a discussion regarding an appropriate integration between these internal Python modules and an LLM-based tool was held. Based on their knowledge within the area, it was decided to start the focus on the PyCNS module. A module that treats the analysis and manipualtion of CNS files. The PyCNS module was chosen over other existing internal modules since the overall structure of an LLM-based tool would remain largely consistent regardless of the module used. Given that the CNS module is the most comprehensive, it was selected as the starting point. This choice ensured that any future transitions or integrations with other modules could be managed smoothly without significant structural changes. An LLM-based tool tailored for this module was deemed to facilitate the post-processing stage of an analysis, and lowering the threshold for the use of available internal Python modules.
A number of criteria, based on the results from RQ1, for such an LLM-based tool were defined. This was deemed to further structure and facilitate the development phase. These are shown below.

	Criteria
1	Able to manipulate or analyze CNS files using the pycns module.
2	Avoidance of black-box architecture through incorporation of human knowledge in process.
3	Prevention of hallucination and erroneous output.
4	Output of well-structured and well-formatted answers easy to follow.
5	Robust and user-friendly

Table 2.3: Development Criteria

The first criteria highlighted the successful integration of the PyCNS module to the LLM-based tool. The second criteria was meant to ensure a consideration of several findings from the interviews and literature review. Interviewee 7 commented, in subsection 3.1.7 , on the need for transparency in a potential LLM-based tool such that the engineer could easily follow computations. Additionally, this was meant to mitigate the theme of *Over-reliance on AI tools* was identified from the literature study and presented in subsection 3.1.1. The third criteria was stated based on the risk associated with LLMs that is hallucination. This risk was outlined from the theme *Hallucination/Low quality output* identified in the literature review, presented in subsection 3.1.1. Additionally, this was linked to the interview process where interviewee 4 in subsection 3.1.3 commented on the poor quality answers that available models like Copilot and ChatGPT yield when asked on theoretical

questions connected to analysis engineering. The fourth criteria was meant to reduce confusion and time consuming aspects of trying to identify parts of the output. The fifth, and final, criteria was meant to secure the development of a tool that was easy to use and reliable enough to justify the use of it.

2.2.2 Development Strategy

With the actual development objective known, an initial development strategy was defined. While it was known that an LLM-based tool focused on the PyCNS module was going to be developed, it was unknown as to how this was going to be done. Consequently, with the help of the results obtained from the interview stage and the literature study, the development strategy with respect to RQ2 was defined. Since the LLM-based tool was to be focused on PyCNS, an initial logical flow of work with this module was defined. This meant the existence of a specific question related to the data in a CNS file, an initial problem solving strategy and code writing, successful code execution and subsequently the presentation of the desired results in a clear and well-formatted way. Furthermore, this workflow was translated into a graph with respect to the desired behaviour of an LLM-based tool. This initial graph contained a number of nodes, corresponding to specific tasks. These nodes were meant to mitigate the challenges connected to the adaptation of generative AI in engineering, challenges who were outlined both in a broad sense as well as in a more GKN specific sense. These challenges are presented in section 3.1. For example, a human in the loop (HITL) was incorporated in this graph to avoid a black-box behaviour and consequently have the engineer in charge.

With a question related to the pycns module as an input, the graph contained the nodes *Generation*, *Human feedback*, *Code execution*, and *Present result*. For the *Generation* node, a problem solving description and corresponding Python code was to be generated. The *Human feedback* node were to display this reasoning and code to the human engineer, asking for approval to proceed to the *Code execution* node. This key feature was desired due to the need to avoid a black-box architecture and having code run for which the user has not inspected. Thus, if the proposed code was disapproved the human user should have provided more context or commented why it was declined such that it returned to the *Generation* node where a revised description and code could be generated. At the *Code execution* node the proposed code was supposed to be run. In order for the model to present a feasible solution, a check whether the code execution failed or not was necessary. If the code was executed successfully, the present node were to present the results in a clear and well formatted way in the *Present result* node. Nevertheless, if the execution were to fail routing should have been done back to the *Generation* node for the generation of a revised and working code. This graph, with corresponding logic and nodes, was deemed to model the workflow of an engineer performing analysis on CNS objects. This initial graph structure is shown in Figure 2.5.

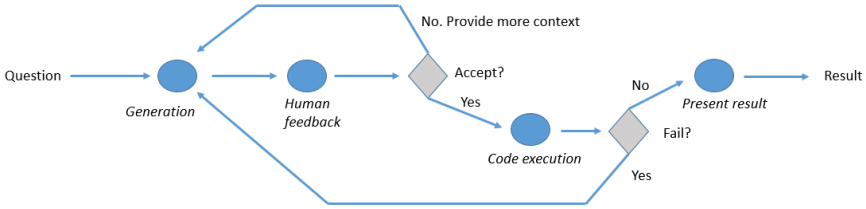


Figure 2.5: Initial graph structure

The nodes were imagined with the criterion in Table 2.3 in mind. For example, the *Human feedback* node was deemed to ensure the second criteria, whereas the *Present result* node was deemed to ensure the fourth criteria. This graph acted as the basis for the actual software development when such an LLM-based tool was programmed.

2.2.3 Development Method

After having performed the initial development strategy phase, it was deemed that some sort of agentic behavior system was required for the development of the sophisticated graph presented in Figure 2.5. The reason for this is that an implementation using manual systems, like chains, without internal decision making and the ability to use external tools would be impossible with respect to the desired graph structure. After thorough research, it was determined that LangGraph offered the most convenient and effective approach for the design of agentic systems compatible with both open-source and state-of-the-art models. The introductory courses provided by LangChain Academy (LangChain Academy 2025) for LangGraph, LangChain, and LangSmith offered the foundational knowledge necessary for assessing complex graph structures in LangGraph and integrating functionalities from both LangChain and LangSmith. Python was selected as the implementation language because of its familiarity and widespread use. Additionally, after gaining a comprehensive understanding of the PyCNS module, the development of programmable software based on the structure, shown in Figure 2.5, could begin.

Nevertheless, such a graph requires the incorporation of internal knowledge. In this case, knowledge about the PyCNS module. The results from the literature study, presented in subsection 3.1.1, highlighted the challenges of poor quality output through hallucination as well as the need for the embedding of internal engineering knowledge. Consequently, four distinct agents were developed based on the knowledge embedding methods described in subsection 1.5.5, namely *Entire Context in System Prompt*, *RAG*, and *Fine-Tuning*. Furthermore, these agents were designed according to the criteria presented in Table 2.3, with the ultimate goal of closely emulating the graph presented in Figure 2.5. A concise overview of these agents is provided in Table 4.2. Finally, the resulting agents are presented in section 3.2.

	Agent 1	Agent 2	Agent 3	Agent 4
Knowledge Embedding	Complete documentation in system prompt	RAG	None	Complete documentation in system prompt
Post-Training	None	None	Fine-tuning	Fine-tuning

Table 2.4: Comparison of Agents across Knowledge Embedding and Post-Training

After developing the four initial agents, an additional main agent was introduced to further improve the robustness of the architecture and to fulfill all the criteria outlined in Table 2.3. Specifically, the initial four agents were primarily designed with the goal of executing code, which is a task that may not always align with user needs. Therefore, to design an agent capable of providing varied responses and especially meeting criterion 5, it became essential to implement functionality that could effectively guide the user. This guidance was achieved through the integration of a routing system capable of distinguishing irrelevant questions and providing explanatory responses without the necessity of executing code. Consequently, the main agent was developed in a modular way, enabling the four initial agents to be integrated in it, hence keeping the code-execution functionality. Ultimately, the agent that performed best, in terms of accuracy and efficiency in the evaluation, was chosen for integration into the main graph. The idea of the main agent architecture is presented in Figure 2.6, while the final implementation result is presented later in section 3.2.

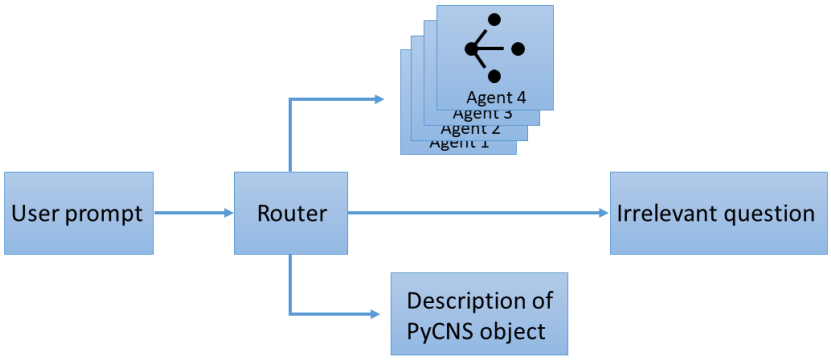


Figure 2.6: Main agent architecture

2.3 Evaluation

After designing the four different agents, it was necessary to evaluate these against each other. This evaluation was done such that one of the four agents could be incorporated in to the main agent. Therefore, the evaluation stage could be divided into two main parts. Namely, the initial evaluation of each of the four agents and the evaluation of the main agent. Both parts will be evaluated on their accuracy and efficiency performance, according to subsection 2.2.3.

2.3.1 Agent 1-4 Evaluation

To assess the performance of the developed agentic systems, a structured QA pair evaluation was conducted. The evaluation focused on two key aspects. One of them was *accuracy*, which measured how well agents responded to predefined questions. The other one was *efficiency*, which measured how effective the computational process for answering the questions was. This was done through the analysis of latency and token data. Each agent was tested using a set of 30 QA pairs that were carefully designed based on a thorough examination of the PyCNS documentation. The questions were selected to ensure diversity in style and execution while maintaining correctness. All the answers in the QA pairs included the computed result, which was pre-calculated manually in Python using the PyCNS module. To enhance the reliability of the evaluation, the experiment was repeated across five separate runs. This allowed for a more robust assessment of the agents' performance and reduced the impact of any variability in individual runs. The entire evaluation process was conducted using LangSmith, which enabled a fully automated testing pipeline. By automating this process, the evaluation became both efficient and reproducible. In the end, all grades were manually inspected by a human to further revise the results.

For agent 2 with its RAG architecture, an initial evaluation was needed to choose the relevant retriever parameters. RAG performance is dependent on a number of different factors such as the choice of embedding method and vector database. A comprehensive investigation as to what RAG implementation was best for this agent was not performed since it is such as an extensive domain. Nevertheless, the chosen chunk size and k value were investigated to a certain extent. The choice of chunk size has been shown to have a considerable effect on the performance in RAG configurations, where a larger chunk size might provide more context while also increasing the time to process the provided information (Wang et al. 2024). The choice of chunk size is also dependent on the density of information in the documentation from which data is retrieved (Zhong et al. 2024). Furthermore, when using a k nearest neighbours (kNN) algorithm for the document retrieval the performance is linked with the selected k value (Leto et al. 2024). Therefore, a somewhat naive, investigation on the effect of chunk size and k value was performed for Agent 2. Since the information density varied in the documentation, two different strategies were outlined. One strategy consisted of setting a smaller k value of 5 while increasing the chunk size between values in the range of 250 to 1750, while the other strategy consisted of setting a smaller chunk size of 200 but increasing the k value in the range of 8 to 20. Initial QA evaluations, consisting of only one repetition, was performed for these different configurations such that one could be selected for the comprehensive QA evaluation for Agent 2.

For fine-tuning, the number of pre-trained parameters of a model can affect the performance (Lu, Luu, and Buehler 2024). Therefore, two different models were evaluated for Agent 3. Since the OpenAI fine-tuning platform was used, these were GPT-4o and GPT-4o-mini. This meant that an initial investigation on how the performance varies between a smaller and a larger model in terms of parameters could be done.

2.3.1.1 Accuracy Evaluation

To measure accuracy, the response of each agent to the 30 questions was compared against a predefined reference answer using an LLM. The grader assessed correctness by assigning a binary value to each response: TRUE if the response was sufficiently close to the reference answer, and FALSE if it contained incorrect or conflicting information. The grading process followed structured evaluation criteria to ensure consistency and objectivity. Specifically, the grader focused solely on factual correctness, ensuring that responses did not contain contradictions. Additionally, responses that included more information than the reference answer were still considered correct, provided they remained factually accurate. The structured evaluation prompt used by the grader was inspired by the LangChain GitHub repository (David 2024), ensuring a well-defined and systematic approach to evaluation. In order to ensure that the grader was trustworthy in its output of TRUE/FALSE answers, a multitude of predicted answers were compared to the provided reference answers with respect to the grading. The QA evaluation framework used in the assessment is illustrated in Figure 2.7.

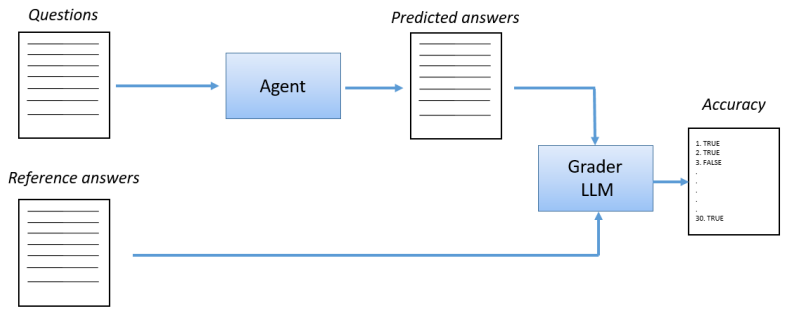


Figure 2.7: QA evaluation for accuracy

2.3.1.2 Efficiency Evaluation

The efficiency evaluation is divided into *latency performance* and *token usage*. These were measured to understand how quickly the agents generated responses and how much computational power it demands, respectively. Latency was recorded in seconds, representing the time taken for an agent to provide an answer after receiving a query. By analyzing both P50 and P99 latencies, the evaluation captured both the expected response times under normal conditions and the potential worst-case scenarios. This ensures a balanced evaluation of the performance. Token usage, on the other hand, was tracked as two single numerical values provided by LangSmith through all the LLM API calls being done inside the graph. One represents the input or prompt tokens, while the other one represents the ouput or completion tokens. Since OpenAI charges based on the number of tokens used, this measurement was an indication of how the utility costs of the different agents differed.

2.3.2 Main Agent Evaluation

The agent evaluation was performed such that an approach for the main graph, displayed in Figure 3.4 and consisting of two subgraphs, could be decided. The agent that was deemed most suitable in terms of accuracy and efficiency, given the results yielded from the QA evaluations, was selected to be included in the main graph. Similar to the agent 1-4 testing, the evaluation of accuracy and efficiency through QA assessment was conducted with reference to the main graph. Nevertheless, this was done on a different set of questions than the agent evaluation. Whereas the agent evaluation consisted of 30 questions based on code generation, the questions for the main graph evaluation were written based on three pillars to capture the variety in style, mentioned in subsection 2.2.3. These were *code generation*, *description generation* and *detection of relevance*. Therefore, 31 questions were written such that the reference answer contained a result from a Python code, a description connected to PyCNS functionality or a standard output *"You asked something that was not related to the PyCNS documentation!"* given an irrelevant question. None of the 31 questions used in this QA evaluation were identical to one of the 30 questions used in the previous QA evaluation.

2.4 Usability Testing

In order to verify if the developed software could be used in an industrial context, its usability was examined through user testing. This user testing, where two analysis engineers at GKN tested the software, used the *Usability* branch in the system acceptability architecture presented by Nielsen (1993) as a foundation. These are displayed in Figure 2.8. The engineers were chosen such that they were familiar with the PyCNS module and had used it within their work previously. The notion of user testing is of importance due to the fact that the software was not developed by intended users.

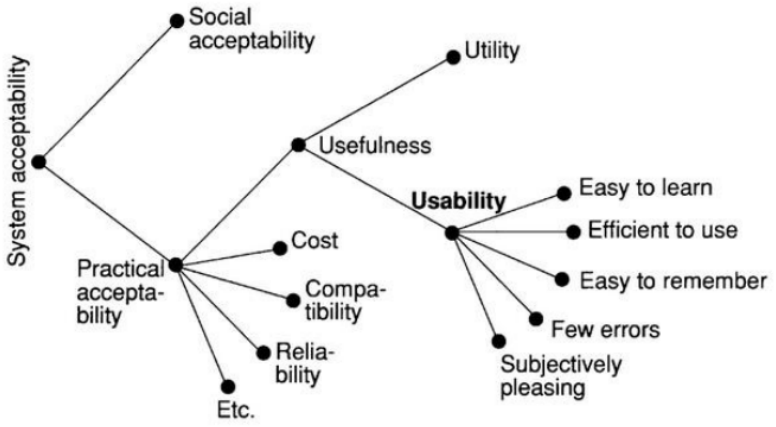


Figure 2.8: The branches of system acceptability (Nielsen 1993, p. 25)

While the method for investigating factors like *Cost* (through token utilization) was presented in section 2.3, important usability factors like *Easy to learn* and *Subjec-*

tively pleasing are more difficult to get an understanding of as a developer. Therefore, these required concrete testing with intended users. The testing methodology was inspired by Farzaneh and Neuner (2019), where a usability evaluation was performed on a software tool in the domain of bio-inspired design. The user testing procedure consisted of a number of steps. Through a provided LangGraph user interface, a brief tutorial of the tool functionality was demonstrated to the engineer. After this, the engineers were asked to use the tool for assignments similar to those they would normally use the PyCNS module in a direct programming interface.

When the engineers were asked to use the tool, this was utilized in the user testing as an observational technique. The engineers were provided with the dummy case presented in section 1.5. This meant that the user behaviour was inspected, and that no interference with the user was done. The advantage of observational techniques is that it is deemed to yield an essentially authentic user environment, and problems connected to the usability can be detected and further addressed (Farzaneh and Neuner 2019). Furthermore, an observation of the user rather than an active instruction of what they were going to do was deemed to reduce the risk of biased results. After this, a brief semi-structured interview was conducted with the engineers to collect their ideas regarding the usability of the tool. This semi-structured interview consisted of six set questions, taken from Srikanth, Hasanuzzaman, and Meem (2024) where the usability of existing state of the art LLMs within the domain of threat intelligence enrichment are investigated. These questions are shown in section C.1. Lastly, the results from the observations and the interviews were analyzed such that an indication of the usability could be obtained. The usability testing method is summarized in Figure 2.9.

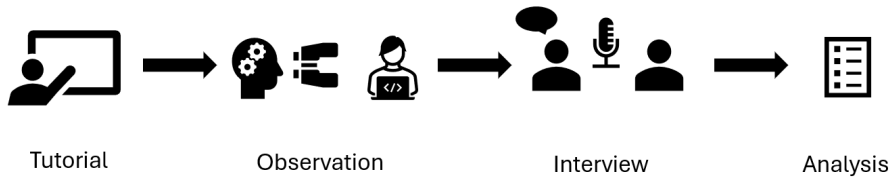


Figure 2.9: The structure of the usability testing

3
Results

3.1 Data Gathering

After finishing the literature review and the interview process, various results relevant to answering RQ1 were obtained. In the following chapter, these results are divided between those explicitly obtained from the literature review in section **??** and those explicitly obtained from the interview process in section 3.1.2. While results from the interview process are generally more applicable towards a specific GKN use case and those obtained from the literature review tend to be of a broader nature, there seems to be an intersection between the obtained results from both stages.

3.1.1 Literature Review

From the literature review a number of challenges connected to the use of generative AI in engineering were identified. These are presented in the following section. The different challenges are often connected to each other, meaning for example that a poor quality output yielded through hallucination may be connected to lack of in-house knowledge and poor prompting. However, generative AI in engineering has its set of limitations and some of these challenges may be chronic, but often it seems that they can be mitigated with the correct strategy.

Applications of LLMs in engineering

The use of generative AI, and more specifically LLMs, show promise in a number of engineering fields. Among these fields are computational mechanics, engineering design and requirements engineering. Here, a brief introduction of LLM applications in engineering based on the reviewed literature is presented.

The use of LLMs could facilitate identification and analysis of data, whether is is related to requirements or information connected to standards. Norheim et al. (2024) investigate the use of LLMs with respect to engineering requirements in complex systems. Arora, Grundy, and Abdelrazek (2024) also consider the use of LLMs in RE, such that a SWOT analysis is performed for the different stages of the RE process. Furthemore, LLMs have the potential to reduce the amount of manual work in product development. Ehring et al. (2024) investigate how LLMs can be used for information classification, such that role-specific identification of information can be

streamlined.

LLMs also show potential when it comes to computational aspects in engineering. A more mechanical engineering focused study on the use of LLMs has been performed by Ni and Buehler (2023), who with the help of LLMs create AI agents, so called *MechAgents*, capable of solving elasticity problems through multi-agent collaboration. Additionally, an article by Alexiadis and Ghiassi (2024) brings forward the use of LLMs integrated to physics-based simulation software. Alexiadis and Ghiassi (2024) investigate how an AI model could be used for geometry generation, mesh set up and material property definition in the simulation software such that the simulation can be run and the AI model additionally could give thorough results to the user. The use of LLMs in computational mechanics is likewise investigated by Brodnik et al. (2023), where the potential and challenges of the use of LLMs in applied mechanics is discussed. One of the possbilities in the field of applied mechanics is assisted programming, where the creation of computational algorithms can be facilitated through code generation and translation (Brodnik et al. 2023).

Engineering design is an area subject to investigation on how LLMs can further streamline engineering processes. Chiarello et al. (2024) analyze 15 355 research papers within engineering design and discuss the potential of LLMs, where the use of LLMs in the four engineering design phases *Problem Definition*, *Conceptual Design*, *Embodiment Design* and *Detailed Design* is treated. Pradas Gomez et al. (2024) investigate the use of LLMs as support for designers in complex systems engineering, where two aerospace cases, one being generation of Python code for an aircraft actuation system UML diagram and the other being code for geometry generation through interaction with a CAD kernel, are investigated. Additionally, LLMs in the automotive industry, using LLMs together with RAG for further automatization of design workflows as well as improvement of software development, is investigated by Zolfaghari et al. (2024) through a comparative study of current available LLMs.

There are various approaches in using LLMs within engineering. For instance, Ni and Buehler (2023) proposing the multi-agent approach where multiple agents collaborate to solve tasks. Similarly, Du et al. (2023) introduce a method for addressing problems, such as arithmetic tasks, using a multi-agent system. In their approach, agents debate the solutions generated by each participant and collaboratively determine a final answer.

The use of LLMs show potential regarding integrated use with external engineering software, as shown by for example Alexiadis and Ghiassi (2024) with the combination of LLMs and physics-based simulation software. Jiang et al. (2024) investigate facilitating the creation of building energy models (BEMs) through LLMs. Such that an "auto-building modeling platform" can transform natural language from a user input prompt to a particular generated building model, via the integration between an LLM architecture and a physics-based simulation (Jiang et al. 2024).

Lack of in-house knowledge

A prominent challenge regarding the use of generative AI, and more precisely LLMs, in an engineering context seems to be the contrast between their general nature and the need for internal competence in tasks. Ehring et al. (2024) highlight the current practice in product development of manual identification and analysis of suitable information from internal documentation, where the inquiry is whether LLMs can make the process more effective. However, the data LLMs have been pre-trained on may vary a lot from internal documentation. Consequently, when using generative AI tools for engineering purposes embedding in-house knowledge is of importance (Ehring et al. 2024). Indeed, as Ehring et al. (2024) write: "without fine-tuning, today's model are unable to classify information in a role-specific way. The models lack too much industry-specific knowledge". A need for fine-tuning could be considered. Nevertheless, fine-tuning is demanding with respect to necessary data (Ehring et al. 2024).

Chiarello et al. (2024) discuss the possibility of using LLMs for facilitating engineering design through combining them with 3D models such that geometry models could be created, although stressing the importance of properly defined material constraints and manufacturing processes. Such constraints could be guided by internal knowledge. LLMs having been subject to fine-tuning could therefore be considered for this (Chiarello et al. 2024). The general nature of LLMs is brought up by Pradas Gomez et al. (2024), as available LLMs are not trained for internal company and project methods. In addition to fine-tuning, RAG can be used to mitigate this challenge (Pradas Gomez et al. 2024).

Issues with sensitive information

Using sensitive information as input to an LLM that is not run locally is problematic, and hence one of the main issues with using current available LLMs in an engineering context. Zolfaghari et al. (2024) mention the current challenge for the use of LLMs in certain engineering fields due to the need for internally stored private data, trade secrets and technical data. For the current available LLMs, the highest performing ones are available through external APIs which is a hinder for the use of LLMs in sensitive domains like defense projects (Pradas Gomez et al. 2024). However, there are LLMs that can be run locally, and therefore eliminating this issue. Examples of such models are LLAMA3 and Mistral, and the ability to run LLMs locally is synonymous to significant benefits in engineering fields where sensitive data is processed (Zolfaghari et al. 2024).

Drawbacks in prompting

The use of LLMs often presents challenges related to prompting. Creating an effective prompt is needed to achieve the desired output. For their use cases, Pradas Gomez et al. (2024) had to experiment with multiple versions and iterations of prompts to obtain outputs that were considered valid responses. Arora, Grundy,

and Abdelrazek (2024) mention the importance of prompt engineering, as the output behaviour of the LLM is strongly reliant on the prompt design. Prompting is in itself a technique which must be mastered for efficient use of LLMs. Prompts could be specified in a pre-defined manner like "Context, Task and Expected Output" (Arora, Grundy, and Abdelrazek 2024). Prompt engineering makes use of the context of the specific task, the language and the known abilities of the LLM (Arora, Grundy, and Abdelrazek 2024). As Arora, Grundy, and Abdelrazek (2024) found in their study: "slightly different prompts can produce very different outputs". Jiang et al. (2024) also emphasize the importance of prompting. Given restricted computational resources the performance of the model can be improved through well defined prompts (Jiang et al. 2024).

The issues related to prompting can display themselves in flawed output, for example a result obtained through hallucination. For reducing the risk of hallucination and erroneous output caused by a multiple step process with faulty reasoning by the LLM, there are methods in prompt engineering like "chain of thought prompting" consisting of clear step-by-step instructions (Brodnik et al. 2023). Mitigation of these errors can also be achieved through RAG, where retrieved parts of documents are added to the input prompt (Brodnik et al. 2023). There are two dimensions to prompt engineering with respect to LLMs, where one approach is the use of APIs, such as the OpenAI API, meaning the use of LLMs is done in a programming environment as opposed to the less technical use in an pre-available user interface (Alexiadis and Ghiassi 2024).

Over-reliance on AI tools/Loss of human engineering competence

As the use of LLMs is associated with a risk of hallucinated or poor quality output, it is of importance to avoid over-reliance on their use. LLMs should not be incorporated such that engineers can use them as a "black-box" solution and exerting blind faith in these, a "best of both worlds" practice should rather be strived for. As Pradas Gomez et al. (2024) note, the current nature of LLMs as "helpful assistants" is contrary to the nature of a competent designer, who is not inclined to always respond to a question given the provided information. The competent designer should question flawed information, and in certain cases express the need for further additions (Pradas Gomez et al. 2024).

As Chiarello et al. (2024) highlight in their example of translating functional models into natural language, the concept of *functional analysis*, described as a specialized language mastered by certain designers, carries a risk of information loss during the translation process. Specific domain insight possessed by a number of engineers could therefore be *lost in translation* as the LLM is invoked. And while LLMs seem to have a significant potential in facilitating tasks, the indication is such that yielded output should be inspected by a competent human actor to mitigate this. The reliance, and more specifically the need for verification of outputs, may differ between different engineering fields. Norheim et al. (2024) investigate the challenges of LLM application to requirements engineering (RE) tasks, and discuss it with respect to RE

tasks like requirement translation and requirement analysis. Furthermore, Norheim et al. (2024) note that a certain level of error is currently inevitable, and there is a need for organizations to determine domains where a certain level of error may be acceptable and domains where it is not. The risk of imperfect LLM performance should be considered in environments with human safety as a major aspect (Norheim et al. 2024).

Hallucination/Low quality output

A common problem with LLMs is their lack of reasoning ability. Hallucination, meaning the LLM gives an output that is factually incorrect but probable from a linguistic perspective, is a challenge with LLMs (Brodnik et al. 2023). Chiarello et al. (2024) mention the danger of hallucinations from an engineering design perspective, where they may cause issues especially when such tools are used by designers with lack of experience.
However, there are ways the challenge of hallucinations can be mitigated. One such is through a multi-agent reasoning model. The multi-agent reasoning model proposed by Ni and Buehler (2023) demonstrated the ability to self-correct hallucinations. For the multi-agent system proposed by Du et al. (2023), the authors found that having agents "debate" with each other lead to a higher accuracy in output. In addition to a multi-agent system, there are other methods to reduce the risk of hallucination. One such way is the use of RAG (Brodnik et al. 2023).

3.1.2 Interviews

The results from the thematic analysis are presented through 6 different themes. These are *Current risks and challenges*, *Cumbersome tasks pre-processing*, *Cumbersome tasks post-processing*, *Factors hindering full automation*, *AI implementation possibilities* and *Risks and challenges with AI*. The themes *Cumbersome tasks pre-processing* and *Cumbersome tasks post-processing* refer to the stages before and after a simulation process. To substantiate the results from the given theme, several *"quotes"* are presented in context to it.

3.1.3 Current Risks and Challenges

During the interviews, a number of points related to risks and challenges of current workflows were identified. These were, for instance, connected to how knowledge is stored, retrieved and used. Interviewees were asked whether they used AI in their work. While existing AI tools are often unsuitable for specific tasks, interviewee 8 noted the limitations of using non-local LLMs as tools from a data secrecy point of view:

> *"You cannot just put anything in there."* – Interviewee 8

Furthermore, these non-local LLMs like ChatGPT perform most effectively on generic questions. However, this generic nature often clashes with analysis procedures where you may need answers to theoretical questions such as on the topic of solid mechan-

ics. As interviewee 4 responded on the question whether he used existing LLMs in his work:

> *"I've tried to use both Copilot and ChatGPT to ask like theoretical questions, but none these models have been trained on the manuals"*
> – Interviewee 4

Consequently, there is a consideration on how to transfer existing knowledge to engineers. This transfer of knowledge seems to be especially important when it comes to less experienced engineers. On the topic of current challenges in the workflow, interviewee 7 commented on this, speaking as an experienced analysis lead:

> *"It's a very changing team. So we always have new people and they need to understand the codes, which is not always easy"* – Interviewee 7

The issue of knowledge embedding is not only relevant to engineers, but also to other workforce. Interviewee 1 investigated the use of LLMs and augmented reality in assembly. Indeed, the potential use of such tools in an assembly context was justified by Interviewee 1 with the example from an assembly site:

> *"They don't keep their inspectors for a long amount of time, they tend to rotate people who they hire quite frequently, so they have to be retrained constantly."* – Interviewee 1

Thus, such a challenge indicated the potential benefit of LLMs as a helping tool. The current risks and challenges in the analysis engineering workflow, as well as in other fields, highlighted the theme of current challenges connected to the storage and retrieval of knowledge.

3.1.4 Cumbersome Tasks Pre-Processing

The interviewees were asked if there were any tasks in their workflow they found extra cumbersome. For analysis engineers, the answers could be divided into cumbersome tasks in the pre-processing stage as well as the post-processing stage. Meaning, during the preparation of a simulation and during the analysis and interpretation of the output. In the case when using Ansys as a simulation software, run scripts are written in APDL. For the pre-processing, interviewee 7 commented on the issue of setting up certain run scripts:

> *"And then some of the problems come from scripts that are not properly set up and then cause problems."* – Interviewee 7

This means time is consumed by inspecting, and fixing code such that the simulation can run properly. These run scripts contain numerous input variables, and interviewee 5 commented on the process of changing these for the specific simulation:

> *"Normally, in some sort of run script, at the start of each run, you have a lot of input variables and today you change these manually."*
> –Interviewee 5

Furthermore, the input to the run scripts needs to be identified and extracted in

an earlier stage. This data can for example be given in Excel, where it must be analyzed and retrieved. Interviewee 5 remarked on this:

> *"Before you can actually give that as an input to the analysis, you need to do a few steps. One of them could be removing duplicates. Others could be formatting [...] So before the run script we could have another script basically prepared to convert that Excel sheet into a format that works for Ansys."* –Interviewee 5

The indications were that the inconvenient aspects of the pre-processing stage were due to the current procedure of certain tasks related to analysis, retrieval and formatting of input data. However, some interviewees mentioned perceiving the pre-processing stage as rather elementary and straight-forward. Interviewee 4 expressed the following:

> *"And I would say that I don't experience that much difficulty and it's actually quite fast to do it"* –Interviewee 5

3.1.5 Cumbersome Tasks Post-Processing

For the post-processing stage, a number of tasks must be performed such that the output script can be transformed to presented data. Interviewee 3 commented on the current nature of the post-processing:

> *"I believe these days we spend most of the time in post-processing. Especially working with data, moving data, moving information from one system to another. From Ansys to text to PowerPoint or Word, or transforming the data in the process and then writing conclusions about the data that we got or the images that we see."* –Interviewee 3

Data must be transferred between different programs, like Ansys and PowerPoint or Excel, which creates a need for the engineer to act as an intermediary, as these systems are not integrated. Interviewee 7 also remarked on the time-consuming nature of post-processing with respect to its engineering team:

> *"And then they also spend a lot of time post-processing the data [...] They need to extract a lot of graphs, time points, load the result files, pick the nodes... All these kind of things."* –Interviewee 7

These repetitive tasks in the post-processing stage, which do not always necessarily demand a lot of thinking, were further commented by interviewee 7:

> *"So right now, I think we are spending a lot of times on things that the computer could do. And there's not a lot of engineering thinking in the current work."* –Interviewee 7

Therefore, it could be considered that the engineer could be relieved from certain of these cumbersome, and repetitive, tasks.

3.1.6 Factors Hindering Full Automation

Although AI has been present for a while with its applications expanding and improving significantly, there still remains a degree of reluctance toward it. The reasons behind it could differ between engineers. One key challenge mentioned in the interviews was getting engineers themselves to trust the transformation towards the incorporation of AI tools. For instance, interviewee 8 remarked on this:

> *"But most of the time comes from getting the engineers that are doing it by hand now to trust the tools that you develop."* –Interviewee 8

Furthermore, a common recurrent opinion was that the complexity in the processes makes it difficult to automate using AI. As discussed in subsection 3.1.8, LLMs can be considered non-deterministic. Given that many workflow processes require individual assessment and the application of common sense, interviewee 2 commented his suspicion against it:

> *"I think a lot of analysis depends on judgement and how do you automate that judgment part?"* –Interviewee 2

It could also be due to that some of the existing software's that are currently used for simulation or calculation are not compatible with automation, as noted by interviewee 3:

> *"If you think about ANSYS only, then ANSYS has its limitations on how much you can actually automate it. Of course, every software will have its own limitation and ANSYS has its set of limitations."* –Interviewee 3

3.1.7 AI Implementation Possibilities

A recurring theme from the interviews highlighted the importance of preserving the knowledge of the engineer and remaining them the central decision-maker in the workflow process. Hence, maintaining oversight and not fully relying on the LLM was considered a crucial element, as interviewee 4 and 7 commented:

> *"Because everyone can kind of do an analysis, but you need an engineer to actually understand what you are putting in and what the output is"* – Interviewee 4

> *"I see it more as like an assistant or like a helper, that the engineer is still in charge."* – Interviewee 8

Several respondents mentioned the idea of letting the tool give feedback to the engineer before proceeding and initializing the simulation or any other action of a more significant nature. This ensures that the control and responsibility remain the engineer as previously discussed. Interviewee 7 suggested following:

> *"The engineer should go, should be able to go and see what the program does and try to understand."* – Interviewee 7

Another suggestion to ensure the engineer retain control of the process was proposed

by Interviewee 1. The idea focuses on establishing clear boundaries for the LLM through a set of predefined actions. This is not only delegating more control to the engineer, but also making the LLM less sensitive to hallucinations.

> *"So instead of having it more freeform, it has to choose from a list of actions. So you have more control of what the LLM is doing rather than directly controlling a software."* – Interviewee 1

A further recurring point was the idea of using the LLM for analyzing result data from the simulations and use it for visualization. Interviewee 8, among other, remarked on this:

> *"So indeed having lots of results, I think a large language model is really good in summarizing that, creating pictures for you, plots, things like that."* – Interviewee 8

Current analysis processes are streamlined with the help of, for example, Python scripts for computations. However, not all engineers are well experienced using Python. As interviewee 6 commented:

> *"And if you start from zero, then it requires time to learn something new. I didn't know Python before."* – Interviewee 6

Therefore, the implementation of LLMs could lower the threshold of using such tools. Interviewee 8 discussed the potential of LLMs with respect to this speaking from their own experience in using Python scripts for automation of certain tasks:

> *"We already try now a little bit with Python scripts to automate [...] Through Python you get the information out. But yeah, it would be even easier if you can just ask a large language model to do that and it will know which script to run basically to get the information out."* - Interviewee 8

3.1.8 Risks and Challenges with AI

Not only risks and challenges with the current workflow was identified during the interviews, but with respect to the implementation and usage of AI and LLM in a working context at GKN. Several of the respondents remarked on the privacy policy within the company and the importance of protecting certain data from dissemination. Interviewee 8 mentioned following:

> *"Well, you know, within GKN, there are a lot of security issues or like rules. You cannot just put anything in there."* – Interviewee 8

The concern regarding LLMs hallucinating and providing misleading information to users was raised by several respondents. Discussions about the reliability of LLM outputs were a recurring theme. Furthermore, Interviewee 3 mentioned the risks associated with the LLM's lack of ability to make determinations in which of the output is relevant or not:

> *"And the biggest risk with LLMs is they are not deterministic, in the case of a calculator at least you can say that there is only one answer."*

– Interviewee 3

Maintaining a balance between transferring knowledge to computer systems and preserving the expertise of engineers is important. It is crucial to understand the importance of keeping the human mind actively engaged to ensure that critical knowledge and problem-solving abilities stay within the company. Interviewee 9 remarked on this:

> *"It's probably possible to ask an AI to do that for us. But in two years, nobody knows how to do it."* – Interviewee 9

In Section 3.1.7, the collaboration between the LLM and the engineer was discussed to ensure reliable and trustful results. Interviewee 2 noted that, while an exchange between the two exists, over-reliance on the LLM remains a problem and could lead to significant failures in a field where margins for error are extremely low.

> *"As soon as a tool is available, we tend to sort of trust it bit too much than we should and in aerospace industry where the safety is of paramount importance, even if you miss two which are going too cause some failure it's too many."* – Interviewee 2

3.2 Software Development

In this section, the results from the software development are presented. This includes the structure and functionality of the four different agents with distinct techniques for knowledge embedding and post-training, as well as that of the main agent consisting of a sub-graph architecture.

3.2.1 Agent 1: *Complete Documentation*

As described in section subsection 2.2.3, this agent involved embedding all relevant contextual information, i.e. the complete PyCNS documentation, directly into the LLM's system prompt. The advantages of this approach are that the LLM receives a unified, comprehensive prompt, which can lead to coherent outputs. However, this method is inherently limited by the prompt's maximum token length, which may constrain the amount of context that can be provided. A too large context window may tend to cause the LLM forget about the beginning of the context window. The final implemented design of Agent 1 consisted of five nodes (apart from the start and end node), which is one further compared to the initial graph structure presented in Figure 2.5. Since the agent has the possibility to retrieve the full set of documentation in this configuration, it was decided to add a further node, `error_reasoning`. This node was deemed to ensure a more robust architecture. All the five nodes are presented in further detail below. In thi end, the graph representing the agent with complete documentation as system prompt context is presented in Figure 3.2.

`cns_generation`: This node generated an initial plan for the agent, outlining its intended actions. This was accomplished using an LLM tailored to generate information from the PyGKN library. The output was strictly structured into three main components, using `Field` from the Pydantic library and `with_structured_output` from the LangChain library. This structured approach, as well as the prompt template for the LLM was heavily inspired by the LangChain code assistant agent from LangChain's GitHub repository (Kim Wee 2024). This node was designed to always route directly to the `human_feedback` node. Below, the design of the structured output and the prompt template for the LLM in this node is presented.

```
    description: str = Field(description="Description of the
       problem and approach")
    imports: str = Field(description="Code block import
       statements")
    code: str = Field(description="Code block not including
       import statements")
```

Listing 3.1: Structured output for the LLM

```
prompt_template_gen = ChatPromptTemplate.from_messages([("
    system","""You are a coding assistant with expertise in
    PyGKN, a library for structural computations.Here is a
    full set of the PyGKN documentation: {context} Answer the
    user question based on the above provided documentation.
    Ensure any code you provide can be executed with all
    required imports and variables defined. Structure your
    answer with a description of the code solution. Then list
    the imports. And finally list the functioning code block.
    Strictly use the methods and functions described in the
    PyGKN documentation and do not rely on other Python
    modules (such as pandas) unless they are explicitly
    mentioned in the documentation. **Ensure that your code
    always assign the printed output to a variable named '
    result' as a concise, descriptive f-string.**
     Here is the user question:"""), ("placeholder", {messages}
        )])
```

Listing 3.2: Python code for generating the prompt template

`human_feedback`: This node integrated an interrupt function from the LangChain library that allowed users to confirm whether a given code block were to be executed. Moreover, it provided an opportunity for users to add any missing information that may not have been included in the initial prompt. This design ensured that human oversight was maintained, preventing the execution of potentially harmful or unwanted code. This was deemed to be of great importance, given that the code runs locally on the machine. Having received the human input. it either continued to `execute_code` or went back to the `cns_generation` node together with the provided additional information from the human.

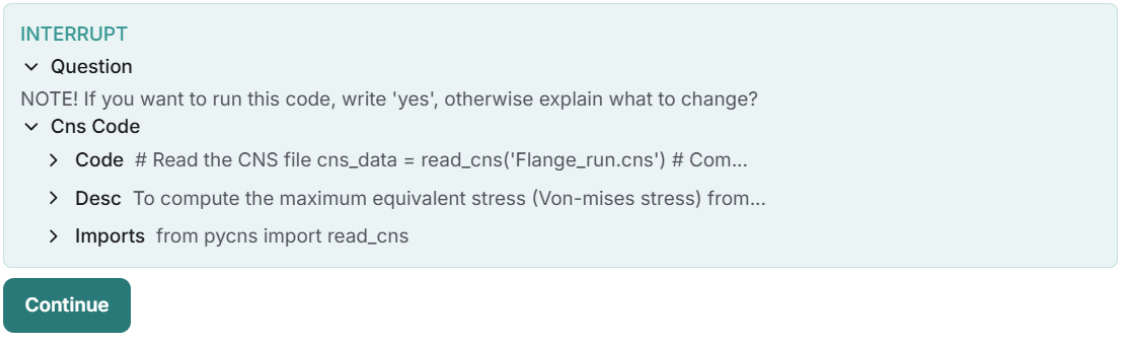


Figure 3.1: Example of a human interrupt block in LangGraph studio

`execute_code`: This node was designed to execute the code generated in `cns_generation`. The execution process consisted of two stages. First, it verified that the import statements ran without errors. If an error occurred during this initial stage, an error message detailing the issue was added to the system messages, and the agent rerouted back to the `cns_generation` node. However, if the imports executed successfully, the node proceeded to run both the import statements and the main code block. If an error was encountered during this stage,

the same error-handling procedure was employed. Finally, if the entire code was executed without any errors, the control was passed to the `present_results` node. This component of the agent drew significant inspiration from the open-source LangGraph GitHub repository (Kim Wee 2024), as it was deemed to be highly relevant with respect to the intended use case. Furthermore, this node introduces a risk of the agent getting stuck in an infinite loop. This occurs when the agent fails to generate an executable coding solution, causing it to repeatedly cycle between the `cns_generation` and `execute_code` nodes. To prevent this, a condition was implemented to force the agent to stop after a predefined number of iterations using the global variable `max_iter`.

`present_results`: This node aimed to present the results of the code execution along with the original details from the `cns_generation` node in a clear and structured way. It uses an LLM tailored to result presentation, ensuring that users could easily understand the outcomes of the executed code together with the connection to the initial plan. This approach ensured clarity in the output while also making the agent more robust since it could be adapted to the changes in the output.
`error_reasoning`: The intention of this node was to add reasoning regarding the error, given an error occurred in the `execute_code` node. It employed an LLM with a specifically designed system prompt for reasoning with respect to errors. It made use of the full documentation together with the error, such that a plan could be designed for what to change during the next generation step. It added the error reasoning message to the global system messages and routed back to the `cns_generation` node.

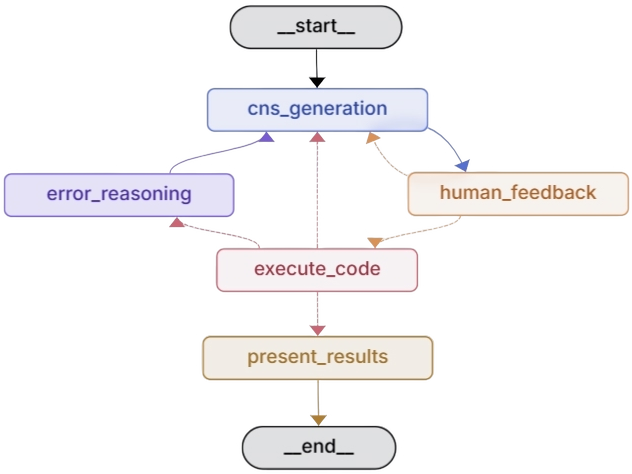


Figure 3.2: Graph structure system prompt context

3.2.2 Agent 2: *RAG*

This agent involved the RAG approach, which entails enhanced context integration by connecting the LLM with a retrieval system. Compared to the configuration with the entire documentation as the context in the prompt, this system dynamically fetched relevant documents or snippets of text that were then incorporated into the system prompt.The difference was therefore what {context} in Listing 3.2 contained. While it was the complete documentation in Agent 1, the RAG configuration meant only certain relevant sections from the complete documentation were retrieved and included. The advantage of this configuration was that it allowed for a broader range of contextual data than what can be embedded directly in the system prompt. However, the performance of such a method depends heavily on the quality and relevance of the retrieved content. The agent consists of the same nodes as Agent 1, however, to enable the retrieval system in the graph, an additional node was added to the graph, presented below.

`vector_retriever`: This node handled the context retrieval through RAG. The PyCNS documentation was first splitted into chunks using `RecursiveCharacterTextSplitter` from the LangChain library, which were then stored in a vector database using `Chromadb` and OpenAI's vector embedding model. The retrieval was also done using `Chromadb`. This node routed to the `cns_generation` node with the retrieved relevant part of the documentation.

This configuration has identical system prompt template as Listing 3.2, except that the retrieved relevant documents are given as `context` instead of the complete documentation. The graph representing the RAG agent is presented in Figure 3.3

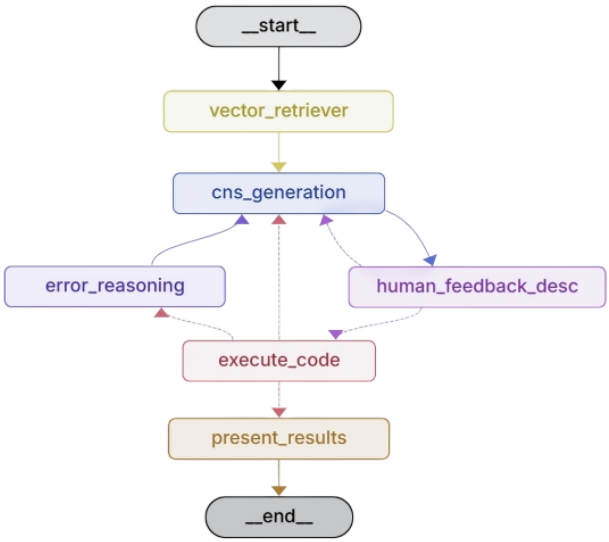


Figure 3.3: LangGraph representing agent with RAG

3.2.3 Agent 3: *Fine-Tuning*

The agent with fine-tuning involved adapting the LLM to a specific domain by re-training it on a specific dataset. The fine tuning was done using the OpenAI

fine-tuning platform (OpenAI 2025). Therefore, this required a large number of examples within a JSONL file. Consequently, each example was in JSON syntax. The format contains a system, user and assistant description. One example being:

```
{
   "messages" : [
    {
       "role" : "system",
       "content" : "You are a coding assistant with expertise
          in PyCNS, a library for structural computations."
    },
    {
       "role" : "user",
       "content" : "How do I read a CNS file using pycns?"
    },
    {
       "role" : "assistant",
       "content" : "To read a CNS file using PyCNS, you can use
           the `read_cns()` function. Here is an example:
       ```from pycns import read_cns
       cns_data = read_cns('/path/to/your/file.cns')
       print(cns_data)```
       This code reads the CNS file and prints the data."
    }
  ]
}
```

In total, a number of 184 such examples were defined and used for the fine-tuning stage. The generation of these examples was based on a number of considerations. Whereas the PyCNS documentation, given in its entirety as context to Agent 1 and through retrieval to Agent 2, contained data for all functions, methods and attributes, this had to be ensured explicitly for the fine-tuning approach. Therefore, it was ensured that the complete functionality of PyCNS was included in the examples for fine-tuning. The advantages of the method were deemed to be that the model becomes highly attuned to the nuances of the target domain, leading to more accurate and context-aware response. Fine-tuned models typically outperform generic models when applied to specialized tasks. On the downside, fine-tuning is resource-intensive, requiring significant computational power and a substantial amount of high-quality training data. Moreover, a fine-tuned model may exhibit reduced flexibility when addressing tasks outside its trained domain.

3.2.4 Agent 4: *Fine-Tuning & Complete Documentation*

This agent combined the configurations detailed in subsection 3.2.2 and subsection 3.2.3. By integrating these approaches, the aim was to explore whether their individual strengths can be leveraged to create an even more effective solution.A new system prompt template has been added to notify the LLM that it has also been fine-tuned on the documentation, see Listing 3.3
```

```
prompt_template_gen = ChatPromptTemplate.from_messages([(
    "system",
    """You are a coding assistant with deep expertise in the pycns
       module, a library for structural computations. Here is some
       documentation to make sure the answer is correct: {context}

When answering the user's question, leverage both your fine-tuned
   expertise and the provided documentation as needed.Ensure that
   any code you provide is fully executablewith all required
   imports and variables defined. Structure your answer by first
   describing your code solution, then listing the necessary
   imports, and finally providing the functioning code block.
   Strictly use the methods and functions described in the PyGKN
   documentation and do not rely on other Python modules (such as
   pandas) unless they are explicitly mentioned in the
   documentation.**Ensure that your code always assign the printed
   output to a variable named 'result' as a concise, descriptive f-
   string.**
When generating any code that references a CNS file, **always use
   the exact file name e.g. "Flange_run.cns"** with no additional
   directories or modifications. Do not hallucinate any file paths
   or prepend directories.
User question:"""), ("placeholder", "{messages}")])
```

Listing 3.3: Python code for generating the prompt template

3.2.5 Main Agent

As described in subsection 2.2.3, the main agent incorporates additional functionality to meet all criteria in Table 4.2. Therefore, some extra nodes were required to obtain this behavior. The nodes `router_user` and `cns_desc_gen` are described in more detail below and can be seen in Figure 3.4. In addition to these two nodes, the `present_results` node was moved to the end of the main graph, rather than being positioned at the end inside each of the 4 agents. This adjustment ensures that results are presented regardless of the path taken by the `router_user` node, making `present_results` a global node within the main agent graph.

`router_user`: This node manages the initial routing of the user prompt, according to Figure 2.6. As described in subsection 2.2.3, Agent 1-4 automatically assumed that the user intended to generate code, attempting to do so regardless of the actual request. To improve flexibility, the router now differentiates between different types of question or user prompts. If the user requests information about a specific function, method, or attribute in the PyCNS library without requiring a computed result, the query is directed to the `cns_desc_gen` node. If a computed result is needed, the request is routed to what is now referred to as `code_gen_graph`. If the query is deemed irrelevant to the PyCNS library, the router redirects to the `present_result` node with the message shown in Listing 3.4.The routing behavior for the LLM is defined by the prompt template provided in Listing 3.5.

```
        result_irrelevant = "You asked something that was not
            related to the PyCNS library!"
```

Listing 3.4: Routing message for irrelevant or non-relevant question

```
route_instructions_template = """You are managing an AI assistant
    that specializes in PyGKN, a library for structural
    computations.
Here is a full set of the PyGKN documentation:  {context}

Your role is to classify user intent into one of three categories
    :
1. 'code_gen_graph': The user wants to generate and execute PyGKN
    code. This includes requests for code snippets,
    implementation guidance, or execution-related queries.
2. 'cns_desc_gen': The user is asking a general question about
    PyGKN documentation. This includes theoretical explanations
    about anything in the documentation , API details, or library
    usage guidance.
3. 'present_results': The user's question is unrelated to PyGKN.
    In this case, return 'present_results' and include a message
    that you can only answer questions related to PyGKN
    documentation.

Based on the following conversation, classify the user's intent
    and return ONLY one of the above three labels.
Do NOT generate a response or attempt to engage in conversation.
"""
```

Listing 3.5: Python code for generating the prompt template

`cns_desc_gen`: This node generates general responses to user prompts related to the PyCNS library. Unlike the previous `cns_generation` node in Agents 1–4, it does not strictly output a format for descriptions, code, or imports. For instance, if a user ask a question about calculating the maximum equivalent stress, the node may provide guiding code, but this is not mandatory. The response is generated using an LLM, and the node always routes directly to `present_results`.

`code_gen_graph`: This node is not a traditional node but rather a subgraph node, which is a graph nested within another graph. It will represent the best performing agent of previously defined Agent 1–4, with the input prompt from the main graph serving as the input to the subgraph.

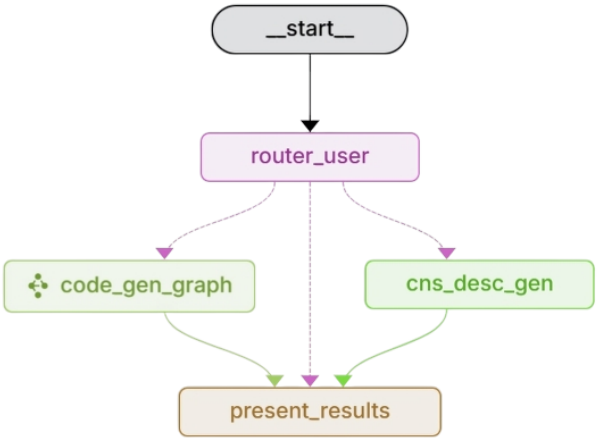


Figure 3.4: Main agent with sub-graph

3.3 Software Evaluation

In this section, the results for the accuracy and efficiency evaluation of the agents in section 3.2 are presented. For the accuracy evaluation, the answers produced by the agents for a set of questions are compared to corresponding reference answers and are assigned a binary TRUE/FALSE values based on their similarity. For the efficiency evaluation, latency data and the number of tokens required for the answering of these questions are measured. Initially, an overview of the results for *Agent 1-4* will be presented, followed by an individual more detailed presentation of each agent. Finally, the evaluation results for the *Main Agent* will be presented.

3.3.1 Comparison Between Agent 1-4

To provide a clear overview comparison of the agents performance, a visual summary of the mean accuracies and latencies is presented in Figure 3.5. As illustrated in the figure, Agent 4 achieved the highest accuracy, whereas Agent 3, using GPT-4o mini as its base model, had the lowest accuracy. In terms of latency, Agent 2a had the lowest P50 latency, though only by a small margin compared to Agent 4. Regarding the P99 latency index, Agent 1 performed the best, although only by approximately

one-tenth of a second over Agent 4. Based on these results, Agent 4 is considered the overall best performing agent in this evaluation.

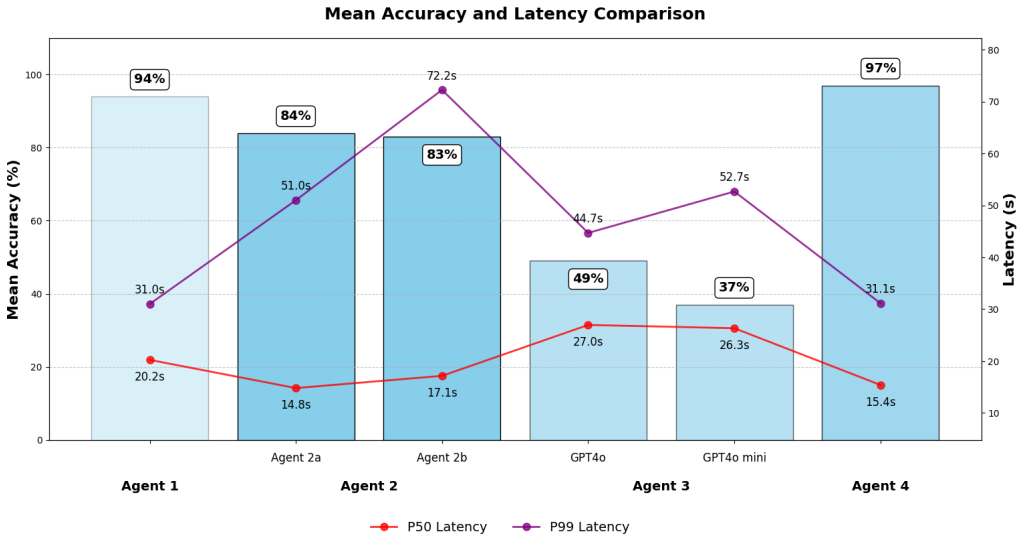
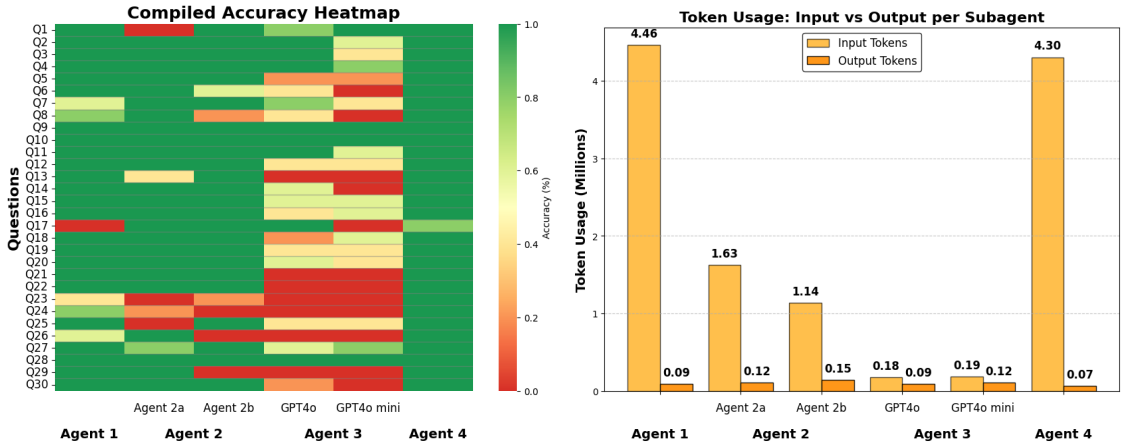


Figure 3.5: Comparison of mean accuracy, latency & token usage between agents

In addition, a compiled accuracy heatmap together with the token usage for each agent was created to display the global performance of all agents on the 30 questions used in the evaluation. The heatmap was generated through averaging each question into a single accuracy score bar. The bar is divided into 30 sections, which are gradient colored from red to green depending on its accuracy. The gradient bar for each agent are put together in a resulting compiled heatmap in order to find recurring patterns across all agents. The compiled heatmap and the token usage can be seen in Figure 3.6a and Figure 3.6b respectively.

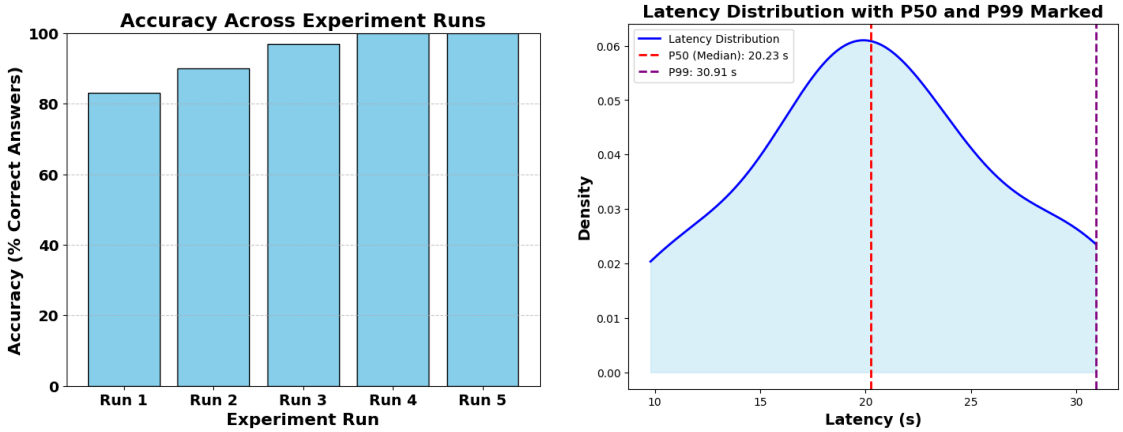


(a) Heat-map summarization **(b)** Token usage all agents

Figure 3.6: Accuracy and token usage results for the four agents.

3.3.2 Agent 1

Following the experimental evaluation, the results from Agent 1 showed to be promising. The model consistently achieved high accuracy across all runs, with two instances where it produced completely correct outputs without any errors, see 3.7a). The mean accuracy across all runs was calculated to be 94%. Additionally, the P50 and P99 latencies were within acceptable ranges, as illustrated in 3.7b. However, while the overall latency was reasonable, a response time of 42 seconds for the final answer may be considered relatively long in practical applications. The input and output token usage, represented as **Input** and **Output** in Figure 3.7b, was recorded as 4.46 million and 93 thousand respectively. The input token usage is very high since the complete PyCNS documentation is provided in the input prompt every time an LLM call is being done.



(a) Accuracy **Agent 1** (b) Latency **Agent 1**

Figure 3.7: Comparison of accuracy and latency performance for Agent 1

Additionally, a heatmap was generated to provide a clear overview of the model's accuracy performance over all runs and questions. This heatmap is shown in Figure 3.8. As seen in the chart, the model struggled the most with question 24, answering it incorrectly more often than correctly. It also had some difficulty with questions 8 and 27.

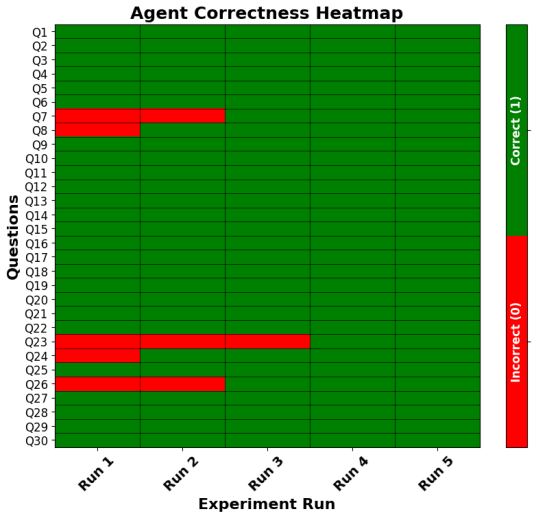


Figure 3.8: Agent correctness across all runs for Agent 1

3.3.3 Agent 2

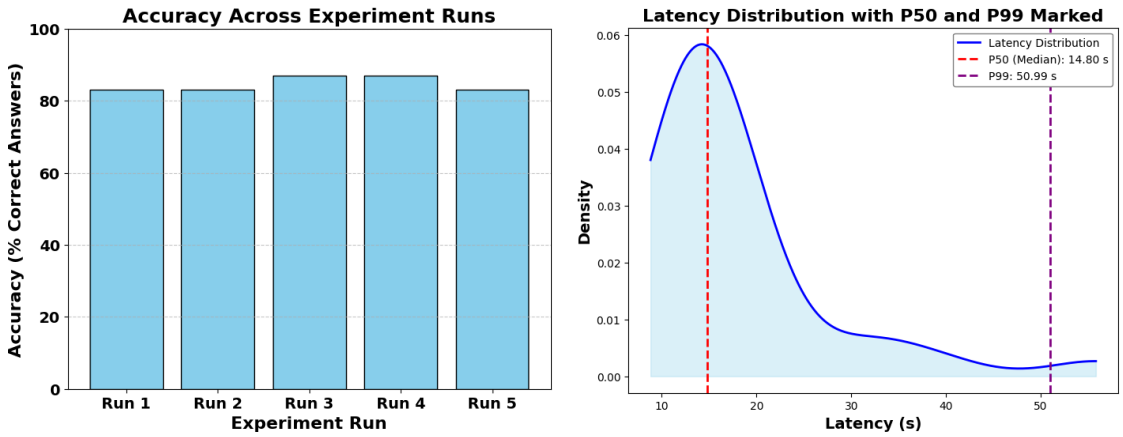
The results yielded from agent 2 include initial accuracy testing for the two different parameter choice strategies, together with accuracy and latency testing across multiple experiment runs for the two top performers from each strategy, which will be refered to as **Agent 2a** and **Agent 2b**. For the initial strategy, with a smaller k value but an increasing chunk size, accuracy data is displayed in Table 3.1.

Chunk Size (CS)	Chunk Overlap (CO)	k	Result
250	50	5	20/30
500	50	5	20/30
750	50	5	18/30
1000	50	5	22/30
1250	50	5	26/30
1500	50	5	25/30
1750	50	5	26/30

Table 3.1: Parameters and accuracy results for RAG strategy 1

The indication from this initial strategy was that an insufficient amount of data was retrieved given a too small chunk size, leading to hallucination and poor output. The results improved during the increase of the chunk size, allowing for more context to be provided. Nevertheless, the accuracy does not seem to increase past a certain point likely due to that the extra data being retrieved is irrelevant with respect to the questions being answered incorrectly. The configurations with chunk sizes of 1250 and 1750 performed equally well for the accuracy, but since 1750 performed slightly better latency wise it was chosen for the more comprehensive QA evaluation. The configuration does not outperform its initial result of 26/30, and the questions it answers incorrectly to are seemingly the same over all the repetitions. The accuracy

evolution, together with the latency data, for **Agent 2a** is shown in Figure 3.9, while the accuracy for each question accross the five runs is presented in the heatmap in Figure 3.10.



(a) Accuracy **Agent 2a** (b) Latency **Agent 2a**

Figure 3.9: Comparison of accuracy and latency performance for Agent 2a

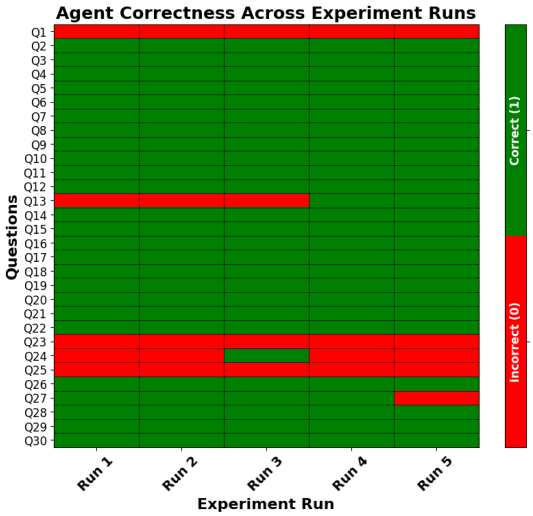


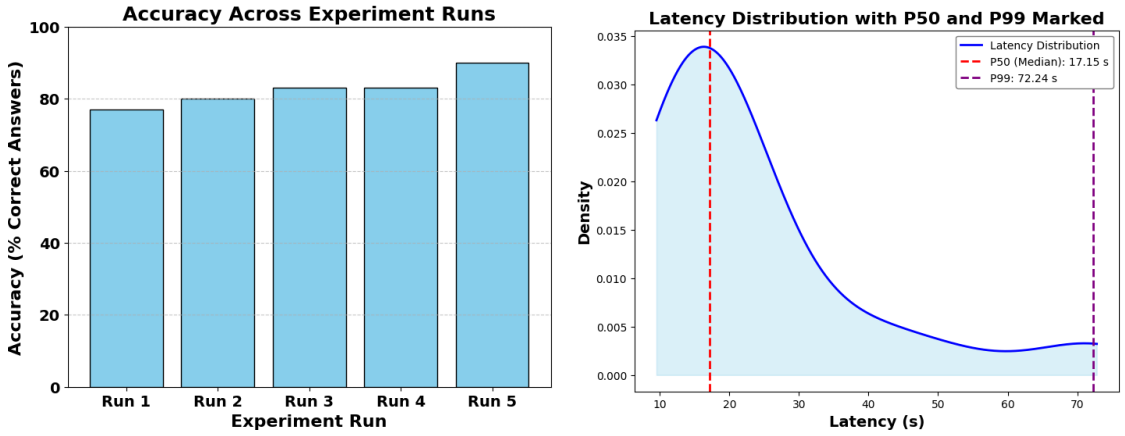
Figure 3.10: Agent correctness heatmap for Agent 2a

Due to the several questions failing to be answered correctly, these are iterated within the graph leading to higher latency and consequently a fairly high P99 latency over the five repetitions. Instead, for a smaller chunk size and overlap but with an increasing k value, the results are presented in Table 3.2.

Chunk Size	Chunk Overlap	k	Result
200	20	8	20/30
200	20	10	20/30
200	20	12	24/30
200	20	14	23/30
200	20	16	25/30
200	20	18	25/30
200	20	20	24/30

Table 3.2: Parameters and accuracy results for RAG strategy 2

Accuracy results improved as the k value increased, allowing for a greater amount of context to be retrieved. However, this increase seems to stop after a certain point. Indicating that the retrieval of less and less relevant data. The configurations with k values of 16 and 18 performed equally accuracy wise, but the one with a k value of 16 performed better latency wise. Therefore, it was chosen for the more comprehensive QA evaluation. The accuracy and latency for this configuration referred to as **Agent 2b** is shown in Figure 3.11. Over the five repetitions, the best accuracy is displayed in the fifth run where it responds correctly to 27/30 questions. The latency distribution has a similar shape as the one in 3.9b.



(a) Accuracy **Agent 2b** (b) Latency **Agent 2b**

Figure 3.11: Comparison of accuracy and latency performance for Agent 2b

The accuracy for five experiment runs, is as before presented in a heatmap which can be seen in Figure 3.12. Similarly to **Agent 2b**, it performed bad on Q23 and Q24. On the other hand it performed worse on Q26 and Q29, but was instead completely correct on Q25 and Q1 compared to **Agent 2a**.

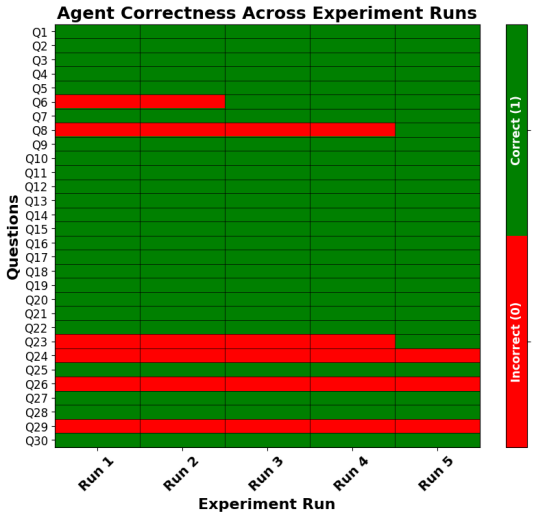
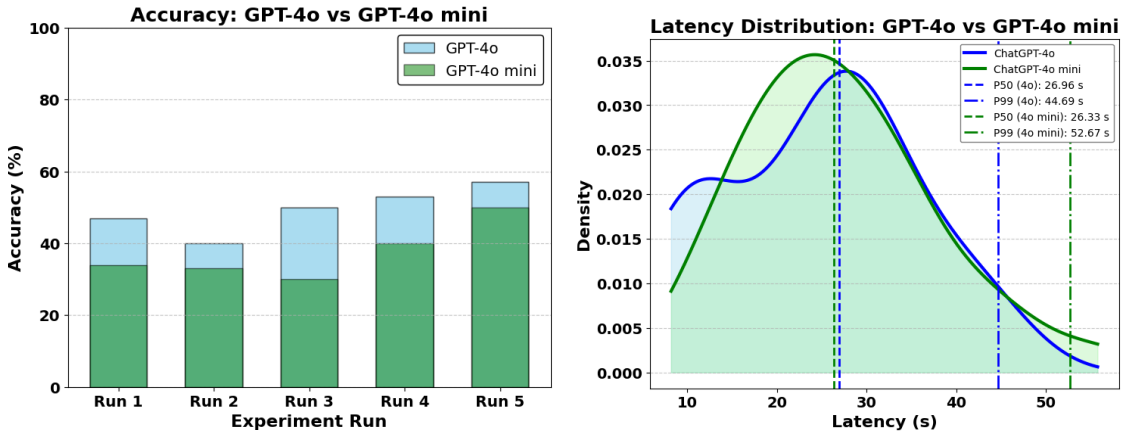


Figure 3.12: Agent correctness heatmap for Agent 2b

3.3.4 Agent 3

The results yielded from agent 3 include an combined accuracy graph for the two different models, GPT-4o and GPT4o-mini as well as an combined latency distribution graph. This can be seen in Figure 3.13a and 3.13b. The accuracy plot shows that the GPT-4o model performed better across all runs. The P50 latencies were almost the same between the to models, with the GPT-4o mini model performing slightly better. The P99 latencies differed a bit more, with the bigger model again being the optimal model.

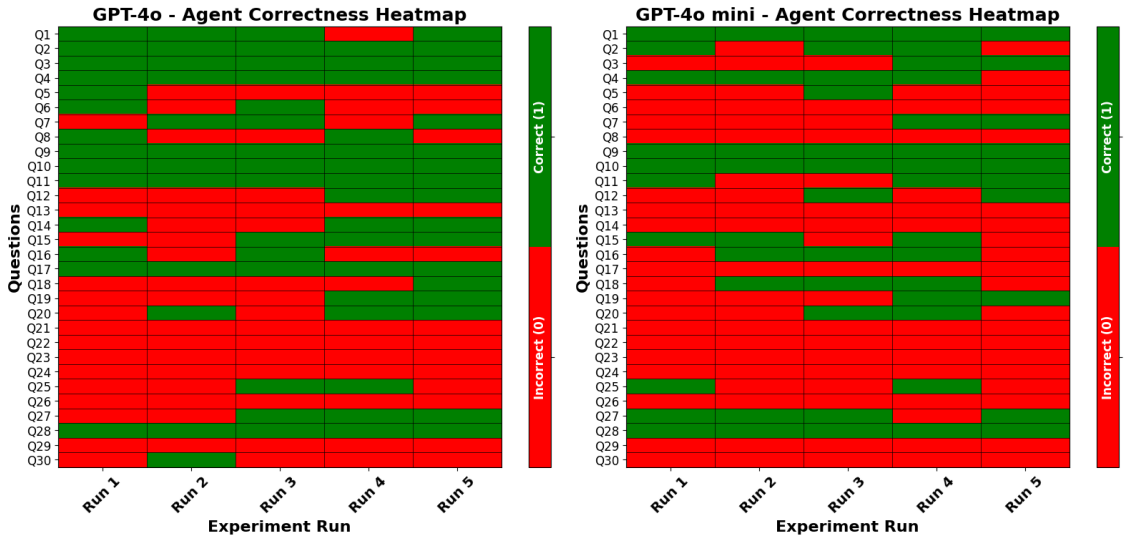


(a) Accuracy Agent 3: 4o & 4o-mini **(b)** Latency Agent 3: 4o & 4o-mini

Figure 3.13: Comparison of accuracy and latency performance for Agent 3 with GPT-4o and GPT-4o mini models

The correctness heat-maps for the two models are presented in Figure 3.14a and Figure 3.14b. Overall, both models performed with relatively low accuracy, resulting in a high number of incorrect responses. However, some notable patterns emerged

when comparing the two heat-maps. Both models performed reasonably well on questions Q1–Q4, Q9–Q11, as well as Q27 and Q28. Conversely, they struggled the most with questions Q18–Q26, consistently giving incorrect responses in this range. An interesting difference was observed for Q17, where GPT-4o answered correctly in all runs, whereas the GPT-4o mini model failed to provide a correct response in any run.



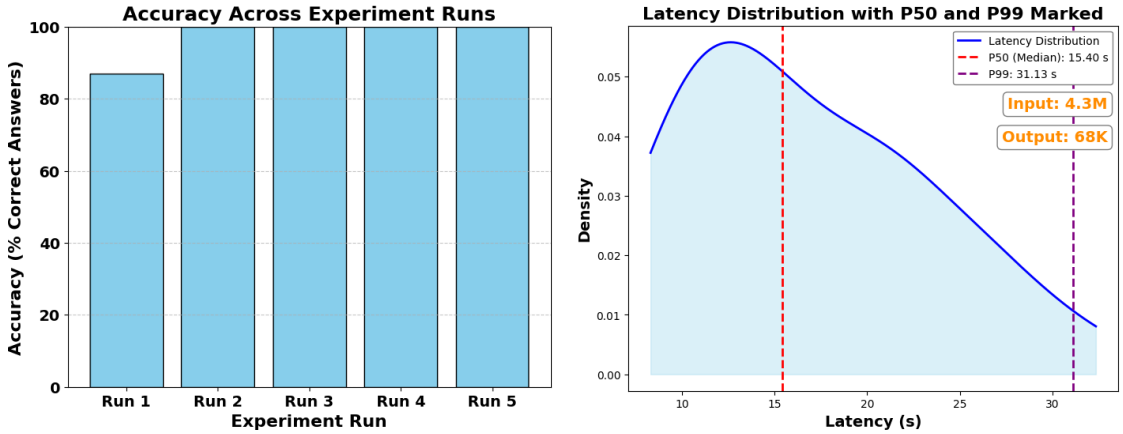
(a) Heatmap Agent 3: GPT-4o **(b)** Heatmap Agent 3: GPT-4o mini

Figure 3.14: Agent correctness heatmap for Agent 3 for GPT-4o and GPT-4o mini

3.3.5 Agent 4

Agent 4, which leverages fine-tuning alongside complete context through the system prompt, demonstrated excellent performance. Across five experimental runs, it produced only a single incorrect response out of 30 QA pairs, achieving an mean accuracy of 99% (see 3.15a). This result is exceptionally strong, indicating that the fine-tuning approach with full context significantly enhances the model's reliability. Beyond accuracy, Agent 4 also provided low P50 latency at 15.4s. However, its P99 latency remained comparable to that of Agent 1 (see 3.7b).

As in previous sections, a heatmap was generated to provide a detailed breakdown of the model's performance on individual questions. The heatmap for this agent, shown in Figure 3.16, reveals that it performed exceptionally well across all questions, with only a single error occurring on question 18 throughout the five experimental runs. Every other question was answered correctly in all instances, demonstrating the model's consistency and reliability. This further reinforces the effectiveness of fine-tuning with complete context in achieving high accuracy across a diverse set of queries.



(a) Accuracy Agent 4 **(b)** Latency Agent 4

Figure 3.15: Comparison of accuracy and latency performance using fine-tuning and complete context in system prompt

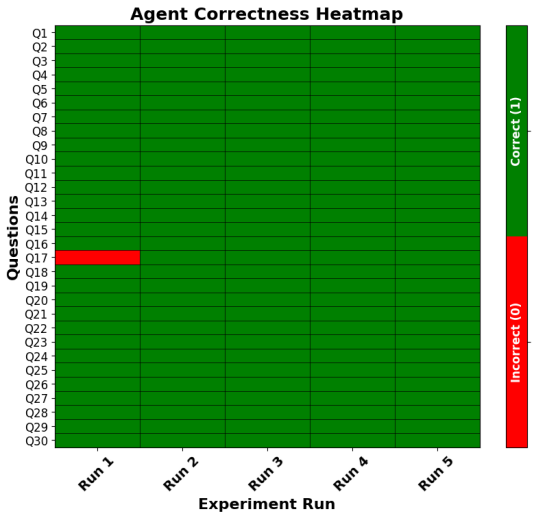
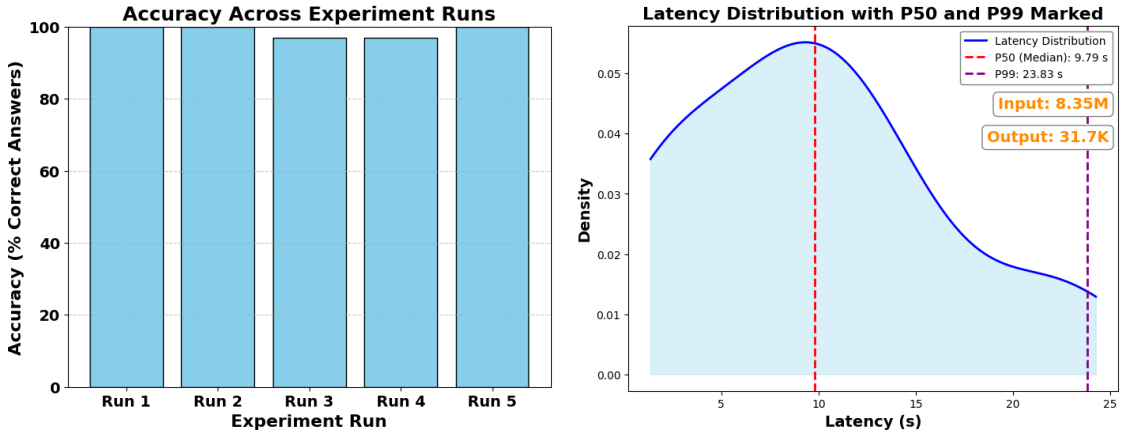


Figure 3.16: Agent correctness heatmap for Agent 4

3.3.6 Main Graph Evaluation

For the main graph, QA evaluation containing 31 pairs was performed through the incorporation of agent 4 in it. This because Agent 4 was considered the best overall in terms of accuracy, latency and token usage. The main graph performed well with respect to accurately answering the set of questions. In general, it handled questions regarding code generation, description generation as well as the filtering of irrelevant questions. For the two questions it answered incorrectly, Q31 and Q18, it hallucinated a description of another PyGKN module not available in the documentation and incorrectly handled a CNS object. The corresponding accuracy and mean latency data from the five repetitions is shown in 3.17a and 3.17b. The latency data for this QA evaluation is seemingly lower than the ones from the agent evaluation. However, this is due to the nature of the 31 questions in this case. For

the questions testing whether it could filter irrelevant questions, it managed well and only took between one and two seconds to output the standard reply indicating the question was irrelevant. Furthermore, the containing descriptive questions were in general less time consuming since there was no need to write and execute code.



(a) Accuracy per run (%) **(b)** P50 & P99 latency per run (s)

Figure 3.17: Comparison of accuracy and latency performance for Agent 4

Looking at the heatmap for the main agent, which can be seen in Figure 3.18, it performed with almost 100% accuracy, except for two errors in Q18 and Q31.

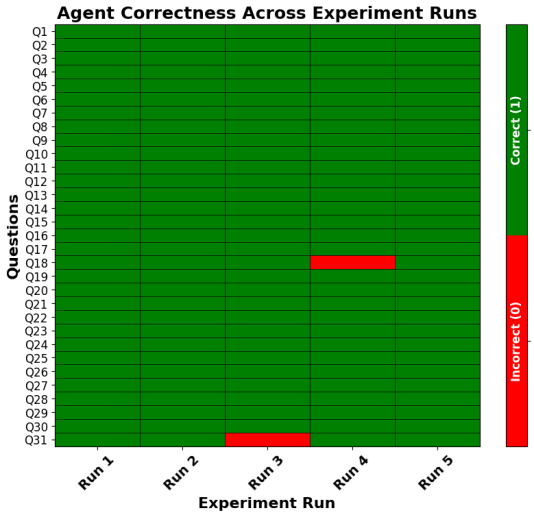


Figure 3.18: Agent correctness heatmap across multiple experiment runs

3.4 Usability Testing

The two analysis engineers subject to the usability testing, denoted Engineer 1 and Engineer 2, used the developed software for similar use cases to those they previously had used PyCNS for. Since they decided the task themselves, and consequently did

the prompting, it was ensured that the usability testing was not biased and highly relevant with respect to real use cases. The software performed well, and showed itself capable of solving the majority of the prompted questions with the testers as judges. Furthermore, the brief interview that followed indicated a considerable potential for the use of similar software to the one developed. One functionality that was highlighted during the interview was the incorporation of HITL.

> *"Since it's running locally on the computer, I like that it asks before it executes anything."* – Engineer 1

Also, the aspect of being able to explain what you want in natural language as opposed to explicitly writing code was a popular feature.

> *"You write your thoughts in text and you receive what you're thinking of."* – Engineer 2

Both engineers were enthusiastic when answering the question whether they would use such a software in their work.

> *"Yes, I could imagine using it."* – Engineer 1

> *"Certainly."* – Engineer 2

On the question what differed between the tested software and current available LLMs, like ChatGPT and Copilot, both engineers underlined the advantage of internally embedded knowledge.

> *"This one is more niche, it's based a lot on what we have done before. That's the downside with the others, that you have to explain more to them. 'This file type looks like this' and so on. That's the big advantage, that you don't have to this every time. "* – Engineer 1

> *"[It's] Better, because it works on what we do. "* – Engineer 2

Nevertheless, the need for the development of a proper user interface was stressed during the testing.

> *"The functionality is there, then there is a question how useful it is. User interface... "* – Engineer 2

4
Validation

The results for the developed tool aim to give an indication whether the tool actually, as stated in RQ2, could *mitigate the identified limitations*. That is, whether it could be used as a tool for analysis engineers at GKN with respect to the limitations disclosed in chapter 3. The validation methodology is based on *Verification and Validation of Simulation Models* by Sargent (2010). And while there are differences between an LLM-based tool like the one developed in this study, and for example a 3D simulation software, it is deemed that the validation methodology is relevant and can be applied in this study. This due to the fact that several parallels can be drawn between the simulation model presented by Sargent (2010) and the model developed in this thesis.

Sargent (2010) defines the validation of a simulation model as "the substantiation that a model within its domain of applicability possesses a satisfactory range of accuracy", while the definition of model verification is stated as "ensuring that the computer program of the computerized model and its implementation are correct". Sargent (2010) treats the notions of *Problem Entity*, *Conceptual Model* and *Computerized Model*, with corresponding validation and verification procedures between these. The corresponding links between these are shown in Figure 4.1.

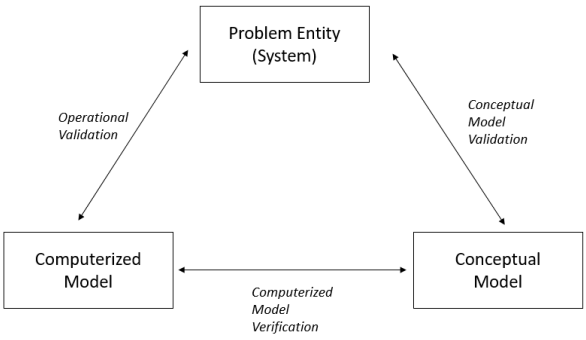


Figure 4.1: Validation architecture (Sargent 2010).

These three entities are the pillars of the simulation model. The actual system that one desires to model (*Problem Entity*), the portrayal of this system in a manner through, for example, a graphical or mathematical depiction (*Conceptual Model*) and the developed computer representation of this (*Computerized Model*) (Sargent 2010). For this study, these three entities have been determined and developed during the quest to answer the RQs. The *Problem Entity* is an analysis engineer trying

to analyze and manipulate CNS objects. The *Conceptual Model* is the graphical and logical representation of this, described in subsection 2.2.2 and displayed graphically in Figure 2.5. Lastly, the *Computerized Model* is the developed LLM-based tool outlined in subsection 2.2.3. Consequently, in order to validate the developed LLM-based tool these three entities must be examined in relation to each other.

Sargent Definition	**Thesis Equivalent**
Problem Entity (System)	Analysis engineer wanting to extract and manipulate data from a CNS object after a FE simulation
Conceptual Model	Initial graphical representation of CNS object analysis workflow shown in Figure 2.5
Computerized Model	Developed software. LLM-based tool described in subsection 3.2.5

Table 4.1: The three pillars of a simulation model defined by Sargent (2010) and their corresponding parts in the thesis

The goal of the conceptual model validation has two main aspects. The assumptions and theoretical foundation acting as the groundwork for it must be valid, and relationships within the portrayal of the model with respect to factors like model structure and logic must be valid (Sargent 2010). A method for this validation described by Sargent (2010) is *Face validity*, where persons experienced with respect to the system (*Problem Entity*) are interrogated regarding the reasonability of it. Since the conceptual model was defined together with the supervisors at GKN, who are deemed to be well knowledgeable within the domain, given the results from the data gathering stage it is considered reliable and therefore validated. Furthermore, the computerized model verification acts as a link between the *Conceptual Model* and its representation in the *Computerized Model*. This verification treats whether the computer implementation, for example the programming, is correct, and Sargent (2010) mentions how static and dynamic testing can be used in this verification stage. Static testing of the software was performed in the sense that the constructed graph, constructed using LangGraph with corresponding nodes and logic implemented, was continuously ensured to be similar to the conceptual model. Dynamical testing included running the software, and ensuring that it was free from bugs and that certain outputs were guaranteed given specific ouptuts. A clear example of this was the QA evaluation, where a clear accuracy value could be obtained. While the computerized model verification cannot be deemed to have been satisfied 100 %, due to the existence of dissatisfactory answers as shown in subsection 3.3.6, it is deemed that this verification was achieved satisfactory, albeit not entirely perfect.

Finally, the operational validation acts as the link between the *Problem Entity* and the *Computerized Model*. For the simulation model as described by Sargent (2010), this validation stage establishes if the output of the software model matches a desired accuracy given the domain of its desired purpose. The operational validation

methodology is dependent on whether a system is observable or non-observable. If data can be collected from the operational behaviour of the system, then the system is deemed to be observable (Sargent 2010). For the system (*Problem Entity*) in this study, consisting of analysis engineers trying to analyze and manipulate CNS objects, it is difficult to say whether it is observable or non-observable. There are deterministic aspects in such processes, given that the engineer may calculate certain stress data or extract specific values. For such use cases data can clearly be extracted, and operational validation can be performed through the direct comparison of reference data from a use case and the answer given by the developed software tool. Just like for the computerized model verification, the QA evaluation is therefore a part of the operational validation and indicates satisfactory results. Nevertheless, the system has a non-observable aspect as well since there is no clear and mechanized way all analysis engineers work with CNS objects. There may be different ways to extract, manipulate and present data. For simulation models, non-observable systems can be validated through the analysis of output by domain experts (Sargent 2010).
As mentioned, the system in this study is not subject to a clear-cut simulation model as opposed to, for example, a 3D simulation model for the system of beam deformation under certain conditions. However, the non-observable nature in this case is validated through user evaluation. This can be compared to the system expert analysis mentioned by Sargent (2010). Namely, how experienced analysis engineers deem the accuracy and usability of the developed tool. For this aspect, the user evaluation gives an indication whether the model can be validated or not. For the user evaluation, the results showed promise when it comes to the application of the software in real use cases. The complete validation can be synthesized to a number of questions connected to the entities, a corresponding validation or verification method and an assessment. For the developed software, this is summarized below.

Question	**Validation/Verification Method**	**Assessment**
Is the modeled initial graph of an engineering problem involving analysis and manipulation of CNS files plausible?	Conceptual validation. The graph was constructed together with supervisors at GKN deemed experts within domain.	Validated
Is the software behaving correctly with respect to its task?	Computer model verification. QA evaluation investigated accuracy and efficiency of software.	Verified
Is the software relevant and usable with respect to the defined problem?	Operational model validation. Usability testing with engineers at GKN.	Partly validated. More extensive usability testing and better UI needed.

Table 4.2: Summary of questions and methodology for the software validation

In conclusion, relevant validation measures are deemed to have been achieved such that there is an indication that the developed tool is validated with respect to the problem. Nonetheless, it cannot be said that it is 100 % validated. Indeed, as

Sargent (2010) write: "It is often too costly and time-consuming to determine that a model is absolutely valid over the complete domain of its intended capability".

5
Discussion

In the following chapter, the methodology and gathered results from thesis are discussed. The identified key takeaways are outlined, together with their implications and a critical review of their truthfulness. Lastly, the two RQs are answered and possible further research on the topic is discussed.

5.1 Literature Review

LLMs have various different fields of application in engineering. Nevertheless, it seems that the same type of challenges applies to different fields. In order for a company to use LLMs as an engineering tool, numerous aspects must be taken into account. For the case of on-site LLMs capable of handling sensitive information, they require internal documentation for them to be used efficiently. Fine-tuning and RAG can be used to reduce this challenge. However, fine-tuning is data demanding and the question is how this data should be curated or if enough available data exists. Furthermore, different engineering applications require different types of data. Using LLMs for integration with psysics-based simulation software may require tailoring towards specific file formats and programming languages, whereas LLMs as an engineering design help could require training on internal documentation. Consequently, when a company applies LLMs in its engineering work there is a question whether it should be tailored for a smaller domain or general in the sense that it can be used in various engineering fields.

The aspect of sensitive information may be of varying importance depending on the nature of the company. A defense company is generally subject to stricter regulations than a civilian one, but all companies wish to reduce the risk of sensitive information being leaked. As the results from the literature review indicated, this challenge can be mitigated by using locally run LLMs. But for such local LLMs, there is a financial consideration that must be taken into account with for example hardware purchases.

Besides internal compatibility and sensitive information, a challenge regarding the use of LLMs is over-reliance. An LLM architecture for engineering should be designed such that it facilitates the engineering work by performing repetitive tasks. As the results from the literature review indicated, the LLM should act as an assistant to the engineer rather than as a replacement to it. LLMs are associated with a risk of hallucinated outputs which an experienced engineer may notice but an unexperienced may miss, and therefore a black box solution should be avoided. While

the risk of hallucination can be mitigated through fine-tuning, RAG and multi-agent models the actual engineering brain cannot be replaced.

The literature review was conducted through analysis of 11 research papers, where seven of those originated from keyword search in Scopus and the rest through snowballing and search on Arxiv.org. This number could have been increased such that the robustness of the obtained results could have been improved. Nevertheless, it was desired for the majority of analyzed records to originate from a well structured and easily reproducible literature review. The number of records could have been increased through snowballing, but at the expense of having a well structured literature review. Furthermore, the number of identified records could have been increased with a different keyword search. A number of various keyword sets were tested, where some sets yielded too many records to screen and some sets yielded too few. The keywords used are deemed to be relevant, and having lead to applicable and high-quality papers. Still, it is deemed that a slight tweak of the keywords could lead to an increased number of relevant records and therefore an even more robust literature review.

5.2 Interviews

The conducted qualitative interviews gave numerous important insights to the project. Regarding cumbersome tasks in the pre- and post-processing stages, a common factor was that the majority of these tasks were considered repetitive and rather undemanding. In that sense, these tasks present opportunities for automation and an LLM may be a feasible solution in streamlining these processes. Furthermore, a greater amount of the respondents perceived the post-processing stage as more time-consuming and inefficient than the pre-processing stage. Hence, this could be seen as a finding substantiating the fact that it is a process of greater potential of development compared to the former. Still, it must be taken into account that the sampling size of the interviewees is not large enough for this conclusion to be deemed entirely certain. There is certainly a possibility that an experienced engineer may find both the pre- and post-processing stages to be fairly simple whereas a lesser experienced engineer may find one of them to be more difficult.

In the context of risks and challenges with respect to the current workflow, and in relation to the risks and challenges regarding the implementation of AI, there are several significant observations. Initially, the addressed risks and challenges using AI related to internal processes at GKN was an important aspect for the development stage of the tool. Although several interviewees were hesitant about the risks and challenges in the implementation of an LLM tool, there were still more prominent concerns regarding the inefficiency and inconvenience in current workflow processes. Furthermore, it also appears that some sort of inquisitiveness exists in using AI to enhance processing efficiency. Some respondents have already tried using cloud-based LLM models in order to ease their work but experienced poor results due to the models being non tailored to the specific needs.

Additionally, it emerged that some processes might be too complex or even dangerous to incorporate with an LLM due to numerous reasons. This was a consideration that was brought into the software development stage. Although, as was presented in subsection 3.1.7 there were different applicable suggestions on how to make, according to them, possible processes automated through an LLM and how that potentially could look. This was also valuable for the software development.

The results gathered from the interview process are, in certain cases, highly subjective. One such example is in section 3.1.4 where interviewee 5 stated that it did not find the pre-processing stage difficult. Different engineers have different backgrounds, and consequently what one might find simple the other might find difficult. Therefore, it is important to not draw too substantial conclusions based on individual interviewee quotes. Rather, these should be taken into account in a broad way together with the findings from the literature review.
Moreover, conducting semi-structured interviews was a necessity considering the large variance in the type of respondents participating in the interviews. This enabled the gathering of more relevant and useful answers. However, a downside with this is that it also made it harder to find common themes when performing the thematic analysis due to the wide spread in the answers. Additionally, it was noted that the interviews included a significant amount of content that was ultimately unused or deemed irrelevant to the project. Therefore, a more focused approach with well thought questions could have made the interviews more concise and hence saved a lot of valuable time.

5.3 The Role of Knowledge Embedding

Two different methods for knowledge embedding were investigated, a context-based method with the incorporation of all relevant knowledge in the system prompt, and a RAG implementation. These are of significantly different nature, and the results indicated a better performance for the context-based agent. Indeed, the context-based method ensures that the accessed information is sufficient for solving the specified question in the QA evaluation stage. As long as the context window is not exceeded, this is certain. However, in the RAG architecture the retriever must fetch a number of specific documents for the LLM to have sufficient knowledge for solving the question. This retrieval process is dependent on a number of factors. As indicated in the results, the accuracy during the QA evaluation is dependent on chunk size, chunk overlap and k value. This is reasonable when you reflect on the nature of the PyCNS documentation with its heterogeneous nature. If the documentation was constructed of a large number of compact examples, consisting of a short explanation and corresponding Python code, it might have been beneficial with a smaller chunk size and a large k. Ideally, such a case would ensure that the LLM is provided with a variety of use cases for the function, or functions, that are to be used. Instead, the documentation used in this study also contains longer passages of source code for the PyCNS functions. This source code contains relevant information regarding input arguments, and written comments. But the source code

also contains underlying information of the functions that is not relevant for the questions. It is desired for the LLM to be able to use the PyCNS module properly, not to produce code for similar Python functions.

One might consider an example where, to answer a question properly, the LLM needs to use functions x, y and z from the PyCNS module. Information on these may be scattered across the documentation, and for a RAG architecture to yield an appropriate answer it needs to retrieve passages containing information on all three. If it only retrieves information for x and y, it will likely hallucinate for the implementation when z is to be used. For the questions the RAG agent responded incorrectly to, this was a recurring issue. Certain functions may have intuitive names with respect to the user question their implementation is needed for. A user may ask how many nodes a CNS object contains, and the retriever will reasonably fetch a passage containing information on the function *read_cns_size*. But all functions are not as direct. The user may ask the LLM to present certain data from the CNS object in a table, requiring the *envelope* function. One of the questions the RAG architecture struggled with was such a question. Namely, as illustrated in Figure 3.10 and Figure 3.12, Q24 where the PyCNS implementation required the *envelope* function after having read the local CNS file with the *read_cns* function. Nevertheless, when the retriever fetched a number of five larger documents only one contained information on this function. Additionally, when a number of 16 smaller documents were retrieved only one of these contained information regarding *envelope*. As long as irrelevant data is provided to the LLM, it will output poor answers.

An interesting indication from the role of knowledge embedding based on the results is that of latency, and the corresponding tokens required to answer a question. To get an indication of the time required for the agents to yield answers for the questions they answered correctly, the P50 latency can be consulted. For both RAG configurations in Agent 2, this value is lower than the one computed in Agent 1. Furthermore, when inspecting the total number of tokens required over the five tries Agent 2 uses fewer tokens than Agent 1 and 3. This is logical since the system prompt, containing all the context, in Agent 1 has a great deal of text and therefore a great deal of tokens. While Agent 1 only provides pieces of this documentation to the system prompt. Consequently, there seems to be a trade-off in Agent 1 and 2 between computational effort and accuracy. Ideally, an improved RAG configuration can deliver a high accuracy while keeping the computational effort low.

Knowledge embedding is a vital part to ensure correctness of output. The context-based method provides the LLM with significant amounts of irrelevant data, leading to an important computational effort and a risk of over-saturation with respect to the context. A RAG configuration is seemingly the more appropriate method for further development. However, this would require a more thorough study on the effects of the different RAG parameters together with a new curated compact and homogeneous documentation.

5.4 The Role of Post-Training

Fine-tuning on its own, as indicated by the results for Agent 3, yielded disappointing results. Despite that the fine-tuning data consisted of 284 well formulated examples, carefully chosen to include all the functionality of the PyCNS module. Therefore, there is a question regarding how many examples are needed to yield a satisfactory accuracy. This was not further investigated, due to the time-consuming nature of example generation. Even if LLMs are used for example generation, these must be examined to ensure they are correct and the fine-tuning data is diverse within the PyCNS module. As a result of this, when developing such an LLM-based tool a decision must be made whether the incorporation of internal knowledge should be emphasized on direct knowledge embedding (RAG, context-based) or post-training. The results are not sufficient to completely reject fine-tuning as a main strategy, but there is certainly an indication that better performance can be achieved by focusing on the knowledge embedding part.

Two models were investigated for the fine-tuning, GPT-4o and GPT-4o mini. The results indicate a somewhat better performance when using GPT-4o. Fine-tuning on the larger LLM therefore led to a better result, but the results are not comprehensive enough to state that it is preferable to fine-tune on a larger model. Since the OpenAI fine-tuning platform was used, these were the two chosen models. Nevertheless, smaller models could be investigated as well to gain a further understanding.

With respect to the computational effort, Agent 3 has a substantially smaller amount of tokens produced during the QA evaluation. For a well performing fine-tuned model, this lesser computational effort is a significant advantage. Nevertheless, considering the poor performance of the fine-tuned model on its own, the token count in the QA evaluation for Agent 4 is not noticeably smaller than that of Agent 1. Due to the lesser required computational effort for the fine-tuned model and RAG, it might be interesting to conduct a thorough study on finding well performing RAG and fine-tuning agents such that fine-tuning, RAG and RAG + fine-tuning can be compared in QA evaluations.

5.5 Knowledge Limitations

As discussed in section 5.3 and section 5.4, if there are limitations in the knowledge provided to the LLM either through the knowledge embedding or the post-training, it will lead to poor quality results. The limitations may display themselves through lackluster retrieval, insufficient documentation or inadequate fine-tuning training data. But perhaps most importantly, the authors of this thesis did not have any prior background working with structural analysis in an industrial context, and therefore no prior knowledge of the PyCNS module. Questions and reference answers for the QA evaluation, together with examples for the fine-tuning, were based solely on the documentation. In the case of the QA evaluation, where the questions are defined in such a way that they are deemed to be those that potential analy-

sis engineers may ask such a tool, there is a risk that this heavy reliance on the documentation does not entirely mirror actual use cases. An unconscious bias may lead these questions to be too direct with respect to the documentation and not nuanced enough in their nature to mirror a real user. A solution for this would have been either to have an experienced PyCNS user approve the questions, or help the authors to write them. There is no doubt that the questions are correct with respect to PyCNS functionality, since they have been tested. Rather, there is a question to be made regarding their proximity to the questions potential users are going to ask.

Furthermore, as discussed in section 5.3, there is a case to be made for a revised documentation. This documentation can contain short descriptions for all PyCNS functions and attributes, together with a significant amount of compact examples where these are used together for specific cases. This would be beneficial both from a RAG perspective, since the RAG parameters can be tailored for this uniform documentation, and that an LLM can generate a large number of similar examples for fine-tuning data given such a comprehensive and well-structured documentation.

5.6 The Rapidly Improving Field of AI

As mentioned in section 1.1, the field of AI and particularly generative AI is a rapidly growing and improving field of the tech industry. This was something that was definitely noticed during several phases of the thesis, which has been both of a restricting and advantageous nature. One of the initial challenges that was encountered was the rapid evolution of the LangChain software, which outpaced the updates to its internal courses and documentation. Many modules and functions changed name or location, making it time-consuming and complex to navigate this rapidly evolving environment. Frequently, there was a reliance on online forums where other users had faced similar issues, often hoping to find guidance from a LangChain developer in the discussion. An advantage of this rapid development was the release of LangChain's long requested Human-in-the-loop feature in the middle of the development stage. This was a functionality that was highly desired in the LLM-tool, and just in time LangChain decided to release this feature, or at least a solution that was entirely working. Furthermore, a point was reached where this feature was again limiting the intended agent behaviour. When it was desired to include the human-in-the-loop feature inside the main agent including a subgraph, it was not possible at the time. Fortunately, after a few weeks, this issue was resolved with a new software update. However, the continuous impact of rapid changes of such software, both in terms of outdated resources and limitations due to unreleased functionalities, has been a time-consuming and challenging aspect of the thesis.

Additionally, by the time the thesis was started, the OpenAI ChatGPT-4o model was by far the most powerful LLM on the market. Hence, it was used as the base model in the development of the agents. However, towards the end of the thesis, a lot of things have happened in the LLM market. One of the biggest industry news was the release of the DeepSeek-R1 model (DeepSeek-AI 2024). This open-source model

appeared to compete with GPT-4o across all benchmarks and could be implemented on-premise or run locally. Unfortunately, there was not enough time to set it up and re-run all evaluations using this model. Given the rapid pace of development, with frequently improvements, it is possible that some aspects of this project could have been done more efficiently and effectively within just a few months in the future.

5.7 Reliability of Results and Sources of Error

The reliability of the results in this study is influenced by various factors, which can be categorized as controllable or uncontrollable sources of error. Controllable errors come from aspects of the methodology that can be refined, whereas uncontrollable errors come from external influences without direct control. One major controllable source of error is the human grading being done to further revise the LLM grading. There might be cases where the LLM misinterprets a valid alternative answer as incorrect or erroneously correct an already valid response, which has to be detected by human correction. Although, with over 750 outputs to control from the QA evaluation, subjective judgment or simply the human factor may risk occasional misjudgments affecting the reported accuracy to remain. Another controllable factor is the predefined QA pairs, which serve as ground-truth references in the evaluation. These answers were calculated by "conventionally" with the use of the PyCNS library. If these answers contain errors, they may incorrectly penalize valid responses.

Among the uncontrollable factors, internet latency played a role in performance evaluation. Since API calls were conducted over a network, response times varied due to fluctuations in connectivity. Although experiments aimed to be performed under stable conditions, preventing all interference was impossible, meaning recorded latency values may not be fully representative or comparable. While this did not impact response accuracy, it only introduced variability in efficiency evaluation. Efforts were made to minimize controllable errors through structured grading, dataset validation, and verification of LLM evaluations. However, uncontrollable factors such as model biases and network fluctuations introduce a degree of uncertainty.

5.8 GKN Usability

The results from the usability testing, presented in section 3.4, gave an initial indication to the value of the developed software in a GKN context. For the niche domain of manipulating, and extracting, data from CNS files, the use of LLMs seemingly show great promise in facilitating engineering tasks and making their work more effective. Nevertheless, it is clear that a number of aspects must be considered and answered before such an LLM-based tool can be put into widespread use. First and foremost, a local model would be ideal from a data privacy point of view. Various models exist who can be run locally, and while this thesis did not consider these, a decision must be taken as to which model to use. As mentioned, the developed software handled a fairly limited use case. Consequently, GKN would need to clearly outline the intended functionality of an LLM-based tool. Both in consideration of

the computational burden put on such a tool, and the amount of internal company knowledge that must be incorporated. This incorporation of knowledge can also be done through various methods, with various ways of leveraging both RAG and fine-tuning. Additionally, a well developed user interface is of importance such that the engineers at GKN actually feels compelled to use the tool. As the results from the usability testing showed, the main weakness of the developed software in its current state was the incorporation of it to the somewhat non-intuitive available LangGraph user interface.

Considering

5.9 Answers to the RQs

The foundation of the thesis was the two RQs, for which the methodology was structured such that answers for them could be obtained. In this section, the RQs are answered based on the results presented in chapter 3 and the previous discussion points in this chapter. For the sake of simplicity, the two RQs are explicitly repeated here:

1. What are the practical limitations for analysis engineers regarding adaptation of generative AI technologies in their engineering tasks?
2. How can an LLM-based tool be systematically designed and evaluated to mitigate the identified limitations?

5.9.1 RQ1

The results from the data gathering, meaning the literature review and the interview process, provided a number clear picture of practical limitations for analysis engineers when it comes to the adaptation of generative AI technologies in their engineering tasks. These are discussed more in detail in section 5.1 and section 5.2. An engineer using generative AI tools, and specifically LLMs, in their work are faced with the limitation that the necessary knowledge for solving tasks is not embedded within available tools. These may be state-of-the-art LLMs like ChatGPT. There is another significant limitation with respect to the use of these, namely the fact that their non-local nature drastically reduces the number of tasks they can be used for. Some companies may have implemented local LLMs where internal information can be used as input, but this is not the case everywhere. A prominent challenge as a consequence of this is the fact that LLMs may hallucinate answers to specific questions, leading to low-quality output and low trust-levels towards LLMs. Lastly, when incorporating tools like LLMs in engineering tasks, a fear of over-reliance and loss of competence seems to be associated to it. While this is a relevant concern, there is a risk that this showcases itself as a reluctance to implement a tool that could make engineering work significantly more efficient.

5.9.2 RQ2

The results from the literature review largely acted as the foundation for the limitations to mitigate, while the interview process both provided information on limitations and possible use-cases for which an LLM-based tool could be developed. It has been shown that specific internal knowledge can be incorporated both through direct knowledge embedding by explicitly stating all relevant info in the system prompt of an LLM, or through RAG, as well as in the domain of post-training where LLMs can be fine-tuned on internal data. Agentic systems with graphical architectures, like those presented in section 3.2, can increase performance of LLM-based tools. For example, with the help of error reasoning nodes or nodes leveraging active human interaction. The incorporation of active human interaction, such as having the engineer review code before it being run and eventually providing more context, has the potential to significantly reduce the risk of over-reliance and a loss of the engineer being in control. Lastly, an LLM-based tool developed using local LLMs can eliminate the challenge associated to company internal data. Nevertheless, this was unfortunately not investigated in the development due to lack of time.

Evaluation, ensuring that the identified limitations have been mitigated, can be done in several ways. In this thesis, QA evaluation investigated the performance for the developed software with respect to accuracy and efficiency. Accuracy ensured that the practical limitation of integrating internal knowledge had been mitigated, while efficiency ensured that it was worthwhile using the LLM-based tool instead of more conventional approaches. Furthermore, user evaluation can be done in order to investigate whether the developed software has a value and if it seems to take the risk of over-reliance into account.

5.10 Further Research

There are a number of fields in this thesis that would benefit from further research, and have not been possible to examine more extensively due to a limited time frame. The developed software focused on the post-processing stage, and more precisely the analysis and manipulation of data from CNS files. It is thought that a more comprehensive LLM-based tool can be developed such that it involves other parts of the post-processing stage, like the incorporation of the internal Python modules tailored for UNV and CDB files. Additionally, an extensive software may even be tailored towards certain stages in the pre-processing stage. As the interview results indicated, there seems to be cumbersome aspects to this as well.

The incorporation of internal knowledge is of paramount importance, and the thesis the effect of different knowledge embedding (RAG, context-based) and post-processing (fine-tuning through GPT-4o and 4-o mini) techniques. Nevertheless, these aspects can be examined more thoroughly. For RAG, the effect of different embedding methods, vector databases and a more sophisticated investigation to RAG parameters could be beneficial. One interesting RAG implementation would be through agentic chunking, such that an LLM-based technique for text splitting

is investigated. Especially since RAG showed itself to have potential when it comes to reducing the computational effort compared to explicitly stating all knowledge in the system prompt. The fine-tuning examined in this thesis only made use of two OpenAI models, and would probably require a lot more training data to perform well. Furthermore, when it comes to the models investigated in this thesis, they are not local. From an industry perspective it would probably be more interesting to investigate if an implementation can be done running models like Mistral or Llama locally through Ollama.

6
Conclusion

This thesis aimed to investigate the potential use of LLMs in an industrial context within analysis engineering. This was done with the help of two RQs as the foundation, where the first one aimed to answer what the current challenges in implementation are, and the second one how an LLM-based tool could be developed given these challenges. Since the thesis was written in collaboration with GKN Aerospace, valuable industry-specific knowledge could be leveraged both in the initial investigative phase as well in the development and evaluation phase. The purpose of the thesis has shown itself to be highly relevant given the results obtained over the course of the work. LLMs have great potential in further facilitating and streamline analysis engineering work, one example being extraction and summary of data in extensive results files. Nevertheless, the results highlighted several major limitations regarding the adaptation of them in engineering tasks. There is a need to incorporate internal knowledge and practices, while the use of non-local LLMs are associated with risks connected to leaks of internal data. Especially in sectors like the defense industry. Additionally, the loss of engineering competence is a fear as tasks become increasingly automated. However, the developed LLM-based tool in this thesis indicates that these issues can be mitigated. Internal knowledge can be leveraged using methods like RAG and fine-tuning, the human engineer can stay in control through the implementation of HITL architectures, and while local LLMs were not investigated in this thesis, they can mitigate the security concerns.

The performance of the LLM-based tool, developed in a relatively short time and with relatively few resources, show the immense potential generative AI has when it comes to engineering tasks. The threshold for the use of the internal Python module PyCNS has been lowered dramatically, and the developed software allows for the further incorporation of other internal Python modules. And while the performance is not yet perfect, and the efficiency can further be improved due to its heavy token usage, the vast promise is clearly shown in the results. Furthermore, this promise has a great possibility to grow given the rapid improvement within AI and potential further work. More sophisticated RAG and fine-tuning approaches have the potential to further improve performance while increasing efficiency. Moreover, the implementation through a local model at GKN would be a big step towards an actual utilization of such a software in the engineering tasks.

Bibliography

[1] Gustaver, M. (2020) A Chalmers University of Technology Master's thesis template for LaTeX. Unpublished.

Bibliography

Alexiadis, Alessio, and Bahman Ghiassi. 2024. "From text to tech: Shaping the future of physics-based simulations with AI-driven generative models." *Results in Engineering* 21:101721.

Arora, Chetan, John Grundy, and Mohamed Abdelrazek. 2024. "Advancing requirements engineering through generative ai: Assessing the role of llms." In *Generative AI for Effective Software Development,* 129–148. Springer.

Balasubramaniam, S., Seifedine Kadry, Aruchamy Prasanth, and Rajesh Kumar Dhanaraj. 2024. *Generative AI and LLMs: Natural Language Processing and Generative Adversarial Networks.* Berlin, Boston: De Gruyter. https://doi.org/10.1515/9783111425078.

Brodnik, Neal R, Samuel Carton, Caelin Muir, Satanu Ghosh, Doug Downey, McLean P Echlin, Tresa M Pollock, and Samantha Daly. 2023. "Perspective: Large language models in applied mechanics." *Journal of Applied Mechanics* 90 (10): 101008.

Cash, Philip, Ola Isaksson, Anja Maier, and Joshua Summers. 2022. "Sampling in design research: eight key considerations" [in English]. *Design Studies* 78 (January). ISSN: 0142-694X. https://doi.org/10.1016/j.destud.2021.101077.

Chiarello, Filippo, Simone Barandoni, Marija Majda Škec, and Gualtiero Fantoni. 2024. "Generative large language models in engineering design: opportunities and challenges." *Proceedings of the Design Society* 4:1959–1968.

David, LangChain AI. 2024. *The Judge: Agent Evaluation Notebook.* https://github.com/langchain-ai/the-judge/blob/main/build-eval-agent/agent-eval.ipynb.

DeepSeek-AI. 2024. *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning.* Technical Report. Accessed: 2025-03-06. https://api-docs.deepseek.com/news/news250120.

Du, Yilun, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. 2023. *Improving Factuality and Reasoning in Language Models through Multiagent Debate.* arXiv: 2305.14325 [cs.CL].

Ehring, Dominik, Ismail Menekse, Janosch Luttmer, and Arun Nagarajah. 2024. "Automatic identification of role-specific information in product development: a critical review on large language models." *Proceedings of the Design Society* 4:2009–2018.

Faiz, Ahmad, Sotaro Kaneda, Ruhan Wang, Rita Osi, Prateek Sharma, Fan Chen, and Lei Jiang. 2023. "Llmcarbon: Modeling the end-to-end carbon footprint of large language models." *arXiv preprint arXiv:2309.14393.*

Farzaneh, Helena Hashemi, and Lorenz Neuner. 2019. "Usability evaluation of software tools for engineering design." In *Proceedings of the Design Society: International Conference on Engineering Design,* 1:1303–1312. 1. Cambridge University Press.

Fauscette, Michael. 2023. *The Use of Generative AI in High Tech.* Accessed: 2024-10-28. https://www.arionresearch.com/blog/the-use-of-generative-ai-in-high-tech.

Fereday, Jennifer, and Eimear Muir-Cochrane. 2006. "Demonstrating Rigor Using Thematic Analysis: A Hybrid Approach of Inductive and Deductive Coding and Theme Development." *International Journal of Qualitative Methods* 5 (1): 80–92. https://doi.org/10.1177/160940690600500107. eprint: https://doi.org/10.1177/160940690600500107. https://doi.org/10.1177/160940690600500107.

Financial Times. 2024. "How AI is changing research and development." Accessed: 2025-03-04, https://www.ft.com/content/648046c1-7fcd-43fb-819b-841f104396d9.

Fu, Zhenxiao, Fan Chen, Shan Zhou, Haitong Li, and Lei Jiang. 2024. "LLMCO2: Advancing Accurate Carbon Footprint Prediction for LLM Inferences." *arXiv preprint arXiv:2410.02950.*

Future of Life Institute. 2023. *Pause Giant AI Experiments: An Open Letter.* Accessed: 2025-02-11. https://futureoflife.org/open-letter/pause-giant-ai-experiments/.

GDPR.eu. 2025. *What is GDPR?* Accessed: 2025-02-11. https://gdpr.eu/what-is-gdpr/.

GKN-Aerospace. 2024. *GKN Aerospace.* Accessed: 2024-10-31. https://www.gknaerospace.com/se/.

Griffin, Abbie, and John R. Hauser. 1993. "The Voice of the Customer." *Marketing Science* 12 (1): 1–27.

IBM. 2023. *What are Large Language Models?* Accessed: 2024-10-28. https://www.ibm.com/topics/large-language-models.

———. 2024a. "AI Agents," July. Accessed February 3, 2025. https://www.ibm.com/think/topics/ai-agents.

———. 2024b. "Fine-Tuning," March. Accessed February 3, 2025. https://www.ibm.com/think/topics/fine-tuning.

———. 2024c. *What is RAG (retrieval augmented generation)?* IBM Think, October. https://www.ibm.com/think/topics/retrieval-augmented-generation.

———. 2025a. *What is a vector database?* Accessed: 2025-02-03. https://www.ibm.com/think/topics/vector-database.

———. 2025b. *What is Embedding?* Accessed: 2025-02-03. https://www.ibm.com/think/topics/embedding.

Jiang, Gang, Zhihao Ma, Liang Zhang, and Jianli Chen. 2024. "EPlus-LLM: A large language model-based computing platform for automated building energy modeling." *Applied Energy* 367:123431. ISSN: 0306-2619. https://doi.org/https://doi.org/10.1016/j.apenergy.2024.123431. https://www.sciencedirect.com/science/article/pii/S0306261924008146.

Kietzmann, Jan, and Andrew Park. 2024. "Written by ChatGPT: AI, large language models, conversational chatbots, and their place in society and business." SPECIAL ISSUE: WRITTEN BY CHATGPT, *Business Horizons* 67 (5): 453–459. ISSN: 0007-6813. https://doi.org/https://doi.org/10.1016/j.bushor.2024.06.002. https://www.sciencedirect.com/science/article/pii/S0007681324000879.

Kim Wee, LangChain Community Ambassador. 2024. *Langgraph Code Assistant Tutorial.* https://github.com/langchain-ai/langgraph/blob/main/docs/docs/tutorials/code_assistant/langgraph_code_assistant.ipynb. Accessed: 2025-02-12.

LangChain Academy. 2025. *LangChain Academy.* Accessed: March 11, 2025. https://academy.langchain.com/collections.

LangChain Inc. 2024a. *LangGraph Documentation.* https://langchain-ai.github.io/langgraph/.

———. 2024b. *LangGraph Studio Concepts.* Accessed: 2025-01-31. https://langchain-ai.github.io/langgraph/concepts/langgraph_studio/?h=studio.

———. 2024c. *LangGraph: Low-Level Concepts - Nodes and Edges.* Accessed: 2024-01-31. https://langchain-ai.github.io/langgraph/concepts/low_level/?h=nodes+edges#nodes.

Leto, Alexandria, Cecilia Aguerrebere, Ishwar Bhati, Ted Willke, Mariano Tepper, and Vy Ai Vo. 2024. *Toward Optimal Search and Retrieval for RAG.* arXiv: 2411.07396 [cs.CL]. https://arxiv.org/abs/2411.07396.

Lu, Wei, Rachel K. Luu, and Markus J. Buehler. 2024. "Fine-tuning large language models for domain adaptation: Exploration of training strategies, scaling, model merging and synergistic capabilities." *arXiv preprint arXiv:2409.03444,* https://arxiv.org/abs/2409.03444.

Microsoft Research. 2023. "How to Evaluate LLMs: A Complete Metric Framework." Accessed: 2025-03-11, https://www.microsoft.com/en-us/research/articles/how-to-evaluate-llms-a-complete-metric-framework/.

Mollen, Joost. 2025. "LLMs beyond the lab: the ethics and epistemics of real-world AI research." *Ethics and Information Technology* 27 (1): 1–11.

MongoDB. 2024. *How to Choose the Right Chunking Strategy for Your LLM Application.* Accessed: 2025-02-20. https://www.mongodb.com/developer/products/atlas/choosing-chunking-strategy-rag/?utm_source=chatgpt.com.

Naik, Atharva. 2024. "Artificial Intelligence and Large Language Models in CAD." Master's thesis, Chalmers University of Technology.

Ni, Bo, and Markus J. Buehler. 2023. *MechAgents: Large language model multi-agent collaborations can solve mechanics problems, generate new data, and integrate knowledge.* arXiv: 2311.08166 [cs.AI].

Nielsen, Jakob. 1993. *Usability engineering.* Morgan Kaufmann.

Norheim, Johannes J, Eric Rebentisch, Dekai Xiao, Lorenz Draeger, Alain Kerbrat, and Olivier L de Weck. 2024. "Challenges in applying large language models to requirements engineering tasks." *Design Science* 10:e16.

NVIDIA. 2023. *What is a Transformer Model?* Accessed: 2024-10-28. https://blogs.nvidia.com/blog/what-is-a-transformer-model/.

OpenAI. 2024a. *GPT-4o.* Accessed: 2025-03-11. https://platform.openai.com/docs/models/gpt-4o.

———. 2024b. *Pricing.* Accessed: 2025-03-11. https://openai.com/api/pricing/.

———. 2024c. *Tokenizer.* Accessed: 2025-03-11. https://platform.openai.com/tokenizer.

———. 2025. *Fine-tuning Guide.* https://platform.openai.com/docs/guides/fine-tuning. Accessed: 2025-02-14.

Page, Matthew J, Joanne E McKenzie, Patrick M Bossuyt, Isabelle Boutron, Tammy C Hoffmann, Cynthia D Mulrow, Larissa Shamseer, Jennifer M Tetzlaff, Elie A Akl, Sue E Brennan, et al. 2021. "The PRISMA 2020 statement: an updated guideline for reporting systematic reviews." *bmj* 372.

Patterson, D, J Gonzalez, Q Le, C Liang, LM Munguia, D Rothchild, D So, M Texier, and J Dean. 2021. "Carbon emissions and large neural network training. arXiv 2021." *arXiv preprint arXiv:2104.10350.*

Pradas Gomez, Alejandro Pradas Gomez, Petter Krus, Massimo Panarotto, and Ola Isaksson. 2024. "Large language models in complex system design." *Proceedings of the Design Society* 4:2197–2206.

Ragani, Alessandro Faure, Paul Stein, Roger Keene, and Ian Symington. 2023. "Unveiling the Next Frontier of Engineering Simulation." Accessed: 2025-03-04, *McKinsey Company,* https://www.mckinsey.com/capabilities/operations/our-insights/unveiling-the-next-frontier-of-engineering-simulation.

Säfsten, Kristina, and Maria Gustavsson. 2020. *Research methodology: for engineers and other problem-solvers.* Studentlitteratur AB.

Sargent, Robert G. 2010. "Verification and validation of simulation models." In *Proceedings of the 2010 winter simulation conference,* 166–183. IEEE.

Sawarkar, Kunal, Abhilasha Mangal, and Shivam Raj Solanki. 2024. "Blended RAG: Improving RAG (Retriever-Augmented Generation) Accuracy with Semantic Search and Hybrid Query-Based Retrievers." *arXiv preprint arXiv:2404.07220.*

Söderqvist, Daniel, and Felix Mare. 2024. "Supporting the Generation of Engineering Analysis Reports with Large Language Models: A Study of Needs and Exploration of Solutions." Master's thesis, Chalmers University of Technology.

Srikanth, Sanchana, Mohammad Hasanuzzaman, and Farah Tasnur Meem. 2024. "Evaluating the Usability of LLMs in Threat Intelligence Enrichment." *arXiv preprint arXiv:2409.15072.*

Stack Overflow. 2024. *Breaking Up is Hard to Do: Chunking in RAG Applications.* Accessed: 2025-02-20, December. https://stackoverflow.blog/2024/12/27/breaking-up-is-hard-to-do-chunking-in-rag-applications/.

Vanderbauwhede, Wim. 2024. "Estimating the Increase in Emissions caused by AI-augmented Search." *arXiv preprint arXiv:2407.16894.*

Wang, Xiaohua, Zhenghua Wang, Xuan Gao, Feiran Zhang, Yixin Wu, Zhibo Xu, Tianyuan Shi, et al. 2024. "Searching for Best Practices in Retrieval-Augmented Generation." *arXiv preprint arXiv:2407.01219.*

Zhong, Zijie, Hanwen Liu, Xiaoya Cui, Xiaofan Zhang, and Zengchang Qin. 2024. "Mix-of-Granularity: Optimize the Chunking Granularity for Retrieval-Augmented Generation." *arXiv preprint arXiv:2406.00456.*

Zolfaghari, Vahid, Nenad Petrovic, Fengjunjie Pan, Krzysztof Lebioda, and Alois Knoll. 2024. "Adopting RAG for LLM-Aided Future Vehicle Design." *arXiv preprint arXiv:2411.09590.*

A

Data Gathering

A.1 Semi-Structured Interview Questions

Present contract

Present the contract to the interviewee and have them sign it. Explain basics of it.

Begin recording

Having obtained a signed contract, the recording can be started.

A.1.1 Demographic questions

1. Could you briefly summarize your role, title and activities (in 30 seconds) at GKN?
2. What do you work with?
3. How many years working experience in the field?

A.1.2 Current process workflow

1. In your experience as an analysis engineer, what type of simulation processes (eg fluid dynamic simulations, solid mechanics simulations etc) have you experience from?
2. If yes, which simulation tools are you currently using or have used in the past?
3. In your engineering tasks, what are the factors hindering a move towards "full automation"? Meaning, a minimization of human involvement in the analysis process. (From start to completion of analysis)

A.1.3 Interviewee experience regarding AI

1. What are your experiences with Large Language Models/Chatbots/ Generative AI in general?
2. Have you tried using any AI tools in your work? If yes, how and do you feel this makes it more effective? If no, why?

A.1.4 Risks/challenges in incorporation of AI

1. Do you see any threats/challenges regarding the incorporation of generative AI in your tasks?

A.1.5 Potential incorporation of AI in the process

1. Going back to the process workflow we talked about earlier, are there any tasks in this process you feel are extra cumbersome and could benefit from automation?
2. Are there any tasks in this process that are very time consuming with respect to the actual output? That means, a lot of time spent on them resulting in relatively low impact.
3. Do you think incorporation of AI or LLMs could streamline the process and do you might have an idea how? (Experienced interviewee)

A.1.6 General finishing questions

1. We have asked our main questions. Do you have any additional comments, or remarks on what has been said?
2. If you have anything you want to add later, or any other thoughts etc. you can contact us at: gabriel.kruger@gknaerospace.com and johannes.lundahl@gknaerospace.com

A.1.7 Wrap it up

Finish interview. Stop recording and thank them for their participation.

A.2 Interview Contract

Terms of interview
For the master's thesis *Developing and evaluating large language models to align with engineering tasks and increase their effectiveness* a vital part of the data gathering is the interview process. By signing this document, and consequently participating in the project as one of the interviewees, you agree that the interview will be recorded for transcription purposes. In order for the interview findings to be used in the final master's thesis report. This recording will only be stored locally at our (Gabriel Krüger and Johannes Lundahl) computers, and will be deleted upon a completed and approved thesis. When disclosing the results from the interview, you will not be named. Instead, namings such as *interviewee x* will be used. This is the case for both the transcriptions as well as results presented in the report. The presented results from the interview are set to be reviewed by our supervisors Alejandro Pradas Gomez and Najeem Muhammed before being published.

Interviewee	Date

B

Software Development

C

Usability Testing

C.1 Usability Test Interview Questions

1. Did you encounter any issues when using the tool?
2. Was there any functionality of the tool that you particularly liked?
3. Was there any functionality of the tool that you particularly disliked?
4. Were there moments when you were confused?
5. Would you use this tool in your work?
6. Compared to LLMs you have previously used, like ChatGPT and Copilot, how would you compare these with the tested tool?