The Role of Generative AI in the Modern DevOps Pipeline

- Master Thesis -

Project Report

Mads Berthelsen (20173802)

Aalborg University
Electronics and IT

Here you can write something about which tools and software you have used for typesetting the document, running simulations and creating figures. If you do not know what to write, either leave this page blank or have a look at the colophon in some of your books.

AALBORG UNIVERSITY
STUDENT REPORT

Electronics and IT
Aalborg University
http://www.aau.dk

Title:
The Role of Generative AI in the Modern DevOps Pipeline

Theme:
DevOps, Prompt Engineering & Code generation

Project Period:
Fall semester 2024

Participant(s):
Mads Berthelsen (20173802)

Supervisor(s):
Henning Olesen
Peter Anglov

Copies: 1

Page Numbers: 90

Date of Completion:
September 5, 2024

Abstract:

The advent of Large Language Models has revolutionized software development, significantly enhancing the efficiency of DevOps processes. Coding practices, in particular, have greatly benefited from the integration of Generative AI, which now assists developers directly within their coding environments, reducing the complexity and duration of repetitive tasks.

This thesis explores further optimization of software development by focusing on the coding stage. It investigates how combining generative AI with prompt engineering can enhance this process. While initial experiments show significant potential, certain delimitations prompted a Conceptual Design approach, leading to necessary changes for truly optimizing code generation.

Acknowledgements

I would like to express my deepest gratitude to the individuals who have directly or indirectly contributed to the completion of my master's thesis. Their support, guidance, and encouragement have been invaluable throughout this journey.

First and foremost, I extend my sincere thanks to my supervisor, Henning Olesen. Henning has been a pivotal figure in my academic journey over the past two years, having supervised several of my projects. His ability to elevate my academic work beyond my own expectations and his constant push for excellence have inspired me to produce results I can truly be proud of.

A special thanks goes to Jan Cordtz from Microsoft for his incredible knowledge and insights on the subject of AI and its integration into DevOps. His willingness to participate in countless meetings and provide an Azure environment for my experiments was instrumental in the success of my research.

I am also deeply grateful to Tom Willy Nielsen from UFST. His participation in the creation and shaping of the experiments conducted within this thesis significantly enhanced the depth and quality of the academic work.

Finally, I want to express my deepest appreciation to Peter Anglov, who entered my life as a course professor during my 3rd semester. Peter's support has been invaluable not only in the completion of my thesis but also in shaping my life ambitions. His expertise and willingness to engage in numerous lengthy meetings and discussions have been instrumental in the quality of this academic product. I am fortunate to consider Peter not just a supervisor but a close friend, whose impact on my life extends far beyond this thesis.

I also wish to extend my heartfelt thanks to my family and girlfriend, who have always been supportive of my choices and understanding of my absence while I pursued my goals. My parents will forever be my biggest inspiration.

Contents

Acronyms

AI Artificial Intelligence. 1

DevOps Development and Operations. 2

EIC European Identity and Cloud Conference. 21

GenAI Generative AI. 1

GPT Generative Pre-Trained Transformers. 17

LLM Large Language Model. 1

ML Machine Learning. 15

NLP Natural Language Processing. 15

QA Quality Assurance. 7

QC Quality Control. 7

RAG Retrieval Augmented Generation. 3

UFST Udviklings- og Forenklingsstyrelsen. 7

VM Virtual Machine. 53

XAI Explainable AI. 2, 3

Chapter 1

Introduction

This chapter introduces the landscape for exploring human-AI collaboration in the future of software development as well as the problems coming alongside it

The field of software development has undergone a drastic shift driven by the integration of Artificial Intelligence (AI) booming since the release of GPT-3 back in November 2022 [7]. Microsoft and OpenAI have strengthened their partnership [3], highlighted by Microsoft's significant investment of billions of dollars in the continued development of Generative AI (GenAI) [2]. Microsoft is not alone in its intense focus on AI development as Google's Large Language Model (LLM) Gemini is also making notable advancements in this domain [58]. This increased focus on advancing AI technology emphasizes the importance for us as users to keep up with this transformation. We must not fall behind the technology but instead, adapt and grow alongside it. The transformation is not only technological but also social changes and expectations are urging developers to re-conceptualize their role in software development. While AI-driven tools currently act as valuable coding assistants, automating repetitive tasks and boosting development efficiency, their future potential goes far beyond mere assistance. As their capabilities and expertise rapidly evolve, these models could transition from assistants to colleagues by translating natural language into code [83]. This potential shift may not replace traditional human coding knowledge entirely but rather reshape it and force developers to adapt accordingly. It is important to remember, that LLMs strength and knowledge are not inherent but harnessed by humans who understand human-to-machine communication [78].

Imagine a future where mastering human-to-machine communication with facilitated tools like LLMs becomes vital for guiding and collaborating with AI as your new colleague [38]. Such a collaborative future built on effective communication through prompt engineering unlocks the true potential of coding and development in upcoming years.

The essence of prompt engineering revolves around the interaction with AI as a "black box", a term that metaphorically represents the non-transparent internal workings of these models [39]. Developers provide an input (the prompt), which is then processed by the AI in ways not entirely visible or understandable from the outside, but ultimately producing an output. This output's quality and relevance are directly correlated with the specificity, clarity, and intent of the input, making prompt engineering a pivotal skill in the modern software developer's toolbox [85].

Generative AI presents significant challenges that need careful attention. One of the most pressing issues is AI hallucination, where the system generates information that appears factual but is often inaccurate [63]. This problem arises because AI models produce outputs based on patterns in their training data without truly understanding the content. As a result, AI can deliver responses that seem convincing but are not grounded in reality, leading to the spread of misleading or false information. Additionally, biases in both the input data and the prompts can further undermine the reliability of AI outputs. To maintain trust and ensure responsible use, it is crucial to independently verify the accuracy of AI-generated information.

In response to these challenges, recent regulatory changes, particularly those introduced by the EU AI Act, have started to focus on making AI systems more transparent and understandable [56]. This marks a shift away from the "black box" approach, where the inner workings of AI are unclear. This shift emphasizes Explainable AI (XAI), which aims to make AI systems more transparent, addressing ethical and legal concerns while building trust in AI technologies [81]. For developers, this means designing AI prompts that not only generate accurate and relevant responses but also enable the AI to explain how it arrived at those answers [15]. With new regulations increasingly requiring AI systems, especially LLMs, to clearly demonstrate their reasoning processes, this focus on transparency is a significant step toward more responsible and accountable AI development [56].

As AI tools become increasingly integrated into software development, the ability to craft precise and thoughtful prompts is becoming a key factor for success. One area within the software development process that can benefit significantly from AI's capabilities is particularly the early stages of the Development and Operations (DevOps) pipeline [30].

The DevOps pipeline represents a classic approach to software development emphasizing continuous development, integration, and operation of software [14]. This process streamlines the development, testing, and deployment phases to enhance efficiency and product quality across all of the teams involved. A key phase within this pipeline is the coding and building stage, traditionally where developers write, test, and integrate code [30]. However, the integration of AI and LLMs is with great certainty going to revolutionize this phase.

The proposal is to shift the focus towards creating detailed and descriptive code documentation at the outset, even before any coding begins [37]. This upfront documentation can then serve as the foundation for generating code through AI and LLMs, transforming the way software is developed by prioritizing planning and design over manual coding. This approach has the potential to flip the traditional software development process on its head. Instead of a workflow where coding is followed by documentation, comprehensive documentation would be crafted first, laying the groundwork for code to be generated automatically using AI tools and frameworks, such as LLMs and prompt engineering.

The emergence of XAI represents another significant advancement in utilizing AI within software development. Retrieval Augmented Generation (RAG) allows users to leverage the natural language comprehension abilities of LLMs to operate on extensive repositories of stored user data. By combining the retrieval of relevant information from a specific dataset with the generative power of LLMs, RAG systems enable developers to work with precise, contextually relevant data rather than relying solely on the generalized knowledge base of the model. This approach enhances the effectiveness of AI-driven development, as it allows for more accurate and context-sensitive code generation and modification.

This thesis delves into LLMs and the complexities of prompt engineering, examining their impact on software development and their implications for the industry's future. Part of the analysis work will attempt to generate code using prompt engineering frameworks to test if it's possible to create code explicitly using natural language and descriptive code documentation. The potential of inviting AI into the actual coding itself will open up use cases of integrating new code as well as editing old code to fit new standards, services, and laws. Having AI deeply involved within your software mainframe would ideally allow it to edit code based on new requirement specifications.

1.1 Motivation

For the initial four and a half years of my academic pursuit, I aimed to pursue a career in academia. This aspiration was clear and remained unchanged until the last six months when my perspective shifted significantly due to the mentorship of Peter Anglov, my professor in security and compliance, who is now also my assisting supervisor. Through several one-on-one meetings, this mentorship challenged my previous ambitions and encouraged me to consider a broader range of possibilities for my future. We discussed not just my career, but also where I might live, my social relationships, and my personal goals. It was this practical and forward-looking mentorship that led me to reconsider my academic trajectory. A few months before commencing my master's thesis I consciously steered away from the path leading to a PhD and instead selected a thesis topic that could serve as a foundational step towards a future career in engineering.

While traditional education has long focused on human-to-human communication, the continued advancement of LLMs such as ChatGPT, introduces a significant shift towards mastering human-to-machine interaction through prompt engineering. This technological advancement spikes my fascination with the transformative role of LLMs in the programming and development sectors. I am convinced that the future developer will benefit immensely from understanding and leveraging prompt engineering. This specialization transcends mere interaction enhancement; it is about harnessing the power of LLMs to revolutionize coding practices, automate complex tasks, and innovate solutions. Educating individuals in prompt engineering is crucial as we enter this new era in AI. It's an essential skill for influencing the future of human-machine communication and reshaping the software development landscape. Therefore, I expect a progressive rise in demand for developers with these attributes.

Exploring the field of LLMs I recognized their impact beyond technological innovation, particularly in creating specialized job opportunities. Research done by LinkedIn highlighted the lucrative potential of prompt engineering, with an average annual salary of $120-390,000 [70]. The demand for job positions with prompt engineering qualities is progressively increasing as well [13]. This discovery fuels my motivation for my master's thesis in prompt engineering. I see this work not just as an academic achievement but as a strategic move towards a career in IT engineering.

1.2 Background

During my undergraduate studies, I was employed at Netcompany, where I worked as an Operation Engineer for nearly two years before returning to university to complete my Master's degree.

In this role, I managed the end of a DevOps pipeline, overseeing batch job execution and software patching. I was responsible for both proactive measures, such as anticipating and preventing fines from GDPR breaches, as well as reactive responses to issues. This involved constant communication with data centers, cloud providers, and development teams to ensure operational readiness for deploying new software and integrating new code with existing services. I actively participated in meetings with developers to anticipate and mitigate potential disruptions, aligning our operations with regulatory requirements to prevent fines for software vulnerabilities. Additionally, I monitored service status using tools like SolarWinds and promptly addressed issues such as database overflow, server downtime, and job failures.

My experience at Netcompany was invaluable in gaining expertise in maintaining critical software infrastructure and collaborating with cross-functional teams. These experiences have significantly influenced my academic pursuits and professional growth. After navigating the complexities of DevOps and witnessing the extensive collaboration it requires, combined with exploring the potential of large language models in code generation, I conceived the idea of transforming the traditional coding process by reversing the sequence, starting with detailed documentation first.

1.2.1 The Central Role of Code Documentation

Building upon the transformative potential of AI in software development, it's crucial to acknowledge the traditionally vital role of comprehensive code documentation. This documentation has served as the backbone of understanding and communication within development teams, ensuring that developers have a clear grasp of the codebase and the impact of any modifications. Recognizing this importance sets the stage for an innovative approach that could fundamentally alter the software development process. However, for this approach to be successful, developers must excel in requirement specifications, crafting clear, detailed, and comprehensive code documentation for AI to generate code from. Achieving this level of precision in documentation is essentially an exercise in prompt engineering, requiring developers to master the art of articulating requirements in a manner that AI can interpret and implement accurately.

1.2.2 A New Approach: Documentation First, Code Second

By reversing the sequence and prioritizing the creation of detailed code documentation before any code is actually written, I want to introduce a method that flips the conventional process on its head. In this paradigm, developers first draft exhaustive and descriptive documentation that outlines the desired functionalities, system architecture, and specific requirements of the application. AI then takes on the role of translating this comprehensive documentation into functioning code, thereby positioning documentation not as a post-development task but as the initial step that guides the entire development lifecycle.

However, it is important to recognize that LLMs are inherently designed to produce varying outputs with each iteration. Running the same prompt multiple times can lead to the generation of different codebases, each with its own structure, logic, and implementation details. This variability necessitates a robust evaluation process to determine which of the generated codebases is most suitable for meeting the outlined requirement specifications.

Since AI is responsible for writing the code from the very beginning, based on thorough documentation, it inherently possesses a deep understanding of the codebase's structure and logic. Consequently, making modifications, whether to introduce new features, apply patches, or adjust functionalities to comply with evolving regulations, becomes inherently more intuitive and efficient. The AI, having 'learned' and trained from the original documentation and its own coding decisions, is fully aware of the implications of any changes. This significantly reduces the need for involvement from multiple people or developers and dramatically reduces the complexity throughout the code's lifecycle, leading to a development process that is far less complex and smoother. The success of this innovative approach underscores the importance of adept prompt engineering and precise requirement specifications.

1.3 Problem Definition & Research Question

This thesis examines how AI technologies can help improve the DevOps lifecycle. The focus is on practical applications, such as how these tools can assist engineers and IT professionals with coding, building, implementing, operating, and maintaining systems. By using LLMs and AI, organizations may find it easier to manage complex tasks, increase productivity, and simplify difficult aspects of development and operations. The main question guiding this research is:

How can generative AI and prompt engineering improve code development and maintenance in a DevOps environment?

To effectively address this overarching inquiry, the research will be structured around several underlying questions that delve into different facets of the topic. These questions will explore the social and coding impacts of integrating LLMs and prompt engineering into the DevOps workflow, aiming to provide a comprehensive understanding of the potential implications and benefits for developers and organizations.

1. How is the growing use of LLMs in IT expected to impact the importance of prompt engineering over traditional coding skills?

2. How can we develop software explicitly using natural language and prompt engineering in low-code and no-code environments? (Follow-up question: How can we verify the quality of code generated by machines?)

3. How can a Retrieval-Augmented Generation solution be developed to streamline the comprehension and analysis of large technical documents in a DevOps

environment?

The subsequent section will outline the anticipated findings and potential implications of this research on the future of code development, maintenance, and overall efficiency in DevOps environments.

1.4 Expected Outcome

By conducting interviews and meetings with IT professionals from respected companies, including Microsoft and Udviklings- og Forenklingsstyrelsen (UFST), the study aims to explore the state of LLM integration from three key perspectives: current status, future direction, and implementation strategies.

Current Integration Status: Insights gathered from numerous meetings with Microsoft, will shed light on the existing landscape of LLM integration. The study aims to map out the current capabilities, challenges, and successes in utilizing AI to support software development and operations, providing a snapshot of where the industry stands today.

Future Directions and Requirements: By investigating use cases that are currently not possible or still in progress, the research will identify the necessary advancements required to further integrate LLMs and AI into DevOps processes. Discussions with UFST will be fundamental in this exploration, providing specific examples of DevOps challenges, such as managing and comprehending large, complex documents. These insights are expected to inform the design and scope of a potential RAG solution.

Implementation Strategies: Building on the identified use cases, the research will explore the development and testing of a RAG solution specifically tailored to optimize the DevOps process provided by UFST. This solution aims to automate the comprehension and retrieval of information from extensive documents, with the practical application being developed in collaboration with Microsoft on their Azure cloud platform. Testing with UFST will ensure the solution meets their standards for Quality Assurance (QA) and Quality Control (QC), providing a practical demonstration of the potential for advanced AI technologies to enhance DevOps workflows.

The expected outcome of this research is to provide a detailed roadmap for integrating LLMs and RAG solutions into DevOps workflows, highlighting the potential benefits, challenges, and strategies for successful implementation. Additionally, the findings are anticipated to provoke further questions regarding quality assurance, control, and other critical considerations, which will be explored in the discussion chapter 7.

1.4.1 Drafting a Conceptual Design

Based on the insights gained from interviews, meetings, and the development process, this research will propose a conceptual design aimed at improving code generation within the DevOps pipeline. This design is heavily inspired by the use cases provided by UFST and reflects my own expectations and contributions to the future of AI integration in DevOps. The focus will be on addressing the limitations identified during the analysis, particularly those related to the constraints on the number of tokens that LLMs can process.

The proposed solution will explore the possibility of adding entire codebases to the vector database of a RAG, rather than just textual information, to enhance the system's understanding and generation capabilities. By including entire codebases, the RAG solution can provide more contextual and relevant code suggestions, thereby improving the accuracy and efficiency of the development process. This approach aims to bypass current limitations and offer a more comprehensive integration of LLMs throughout the DevOps lifecycle.

Chapter 2

Methodology

This chapter will present the various methodologies employed in this research, providing a detailed explanation of each method and the rationale behind their selection.

In shaping the methodology for this thesis, a critical step involved conducting preliminary interviews with key professionals from leading IT companies, including Netcompany, Lego, Dwarf, and Microsoft. These interviews were instrumental in formulating the primary research question: "How can generative AI and prompt engineering improve code development and maintenance in a DevOps environment?" The insights gained from these discussions guided the choice of research methods.

A notable example comes from an interview with Mathias Vilbrad from Netcompany. Mathias highlighted a significant inefficiency within his development team: the excessive amount of time spent in meetings, often comprising up to half of their workday. He proposed an innovative solution involving the use of AI, specifically leveraging transcription tools and a RAG solution. This approach would involve having only the essential participants in meetings while transcribing the content for others to access later. By querying the RAG system, employees could retrieve information pertinent to their roles without needing to attend the meetings in person.

The methodology for this thesis was ultimately shaped by the variety of unique use cases presented during these preliminary interviews. Each company offered distinct challenges and opportunities, making it clear that a one-size-fits-all quantitative approach would not suffice. Unlike scenarios where the majority of variables remain constant—such as producing various flavors of ice cream, where only the flavor changes—the solutions required here demanded diverse strategies and innovative thinking tailored to each specific context. This uniqueness across use cases made a strictly quantitative approach less suitable, as the complexity and individuality of each situation could not be captured through numerical data alone.

As a result, I opted for a qualitative methodology, particularly aimed at optimizing a DevOps process presented by UFST in partnership with Microsoft. This choice was motivated by the need to thoroughly investigate the specific case, comprehend the unique problem, and understand the requirements. The qualitative approach provides a richer, more contextual exploration of how generative AI and prompt engineering can be implemented in this specific setting, accommodating the unique use case and the innovative solutions it demands.

The structure of this research process is illustrated in the figure 2.1 below, presented in chronological order. This figure provides a visual representation of the sequential steps undertaken during the study. By following the timeline depicted in the figure, readers can easily grasp the logical flow of the research and understand how each phase builds upon the previous one. This visual aid serves not only to clarify the research process but also to emphasize the systematic approach taken throughout the study.

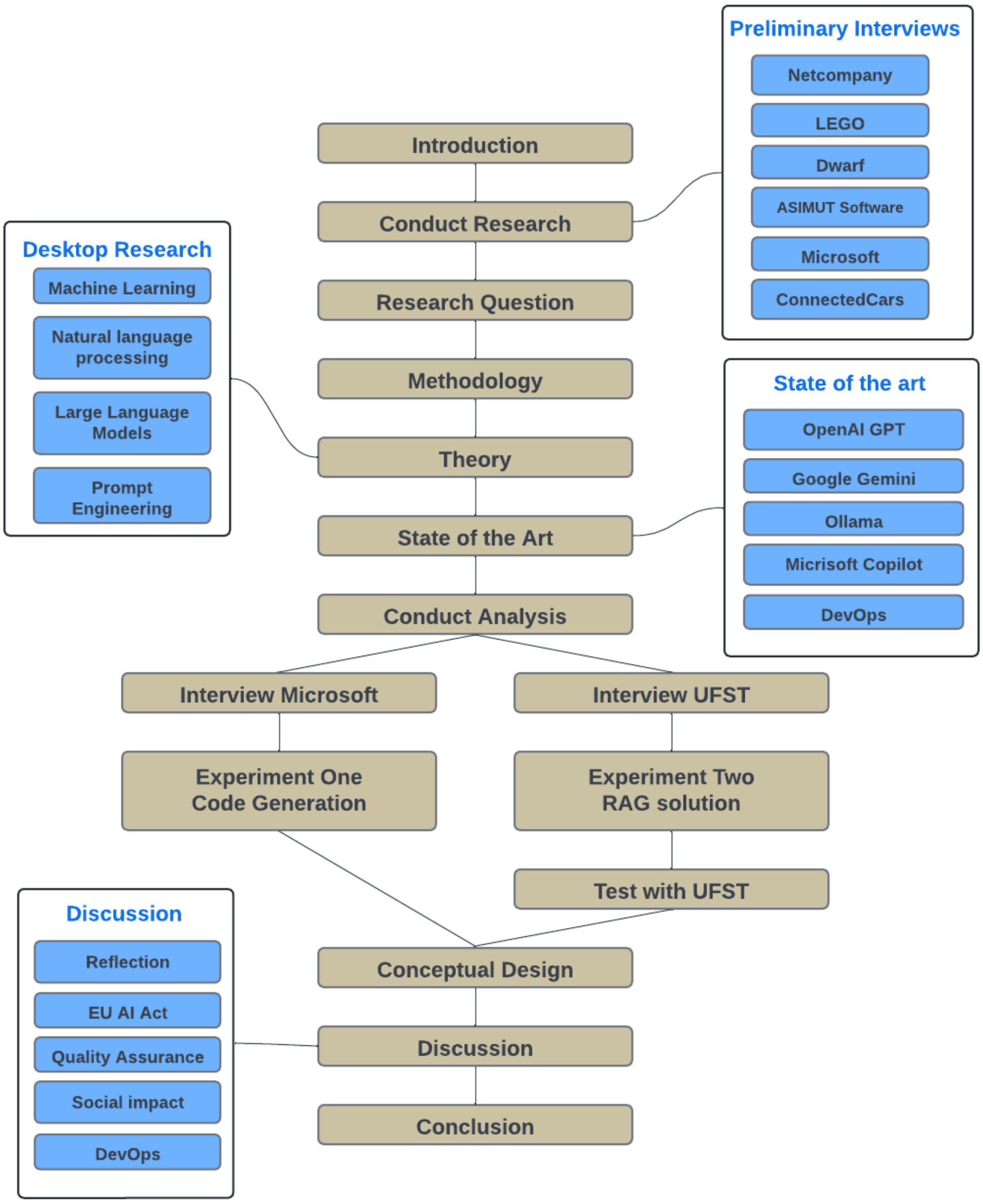


Figure 2.1: A chronological overview of the research process, illustrating the sequential steps. Diagram is created using Lucidchart [40]

2.1 Desktop Research

Desktop research was pivotal in gathering and analyzing information for this paper. This process involved a thorough examination of a broad spectrum of existing data and literature on Machine Learning, Natural Language Processing, Large Language Models, Prompt Engineering, and Generative AI.

The research included reviewing academic journals, industry reports, and online databases to gather relevant information. This comprehensive approach provided a deep understanding of both historical and current developments in these fields. It encompassed not only recent publications but also foundational theories that remain relevant, offering a robust framework to understand the evolution and variations of these technologies.

The research was further enriched by insights from the Machine Learning course material covered during the 2th semester of the ICTE study program. This academic foundation provided a structured and critical perspective, enabling the integration of theoretical concepts with practical applications observed in current studies and real-world implementations, as described in the state-of-the-art section 4.

While AI has recently gained significant attention, it is important to acknowledge its rich history spanning over fifty years. This body of knowledge includes pioneering work, foundational algorithms, and continuous advancements leading to today's sophisticated systems. Leveraging this extensive repository of existing knowledge allowed for a nuanced and in-depth understanding of the subject.

Integrating desktop research into this thesis served several critical functions. It established a comprehensive background and contextual framework essential for understanding the current state and future directions of AI technologies. It also identified gaps and opportunities within the existing literature, guiding the primary research focus and objectives toward DevOps optimization by integrating AI. Lastly, it provided a solid evidence base for the analysis and experiments presented in this thesis, ensuring that the conclusions are well-founded and substantiated by existing knowledge.

2.2 Expert Interview

An expert interview is a qualitative research method used to gather in-depth information, insights, and opinions from individuals with specialized knowledge or experience in a particular field [4]. This method enables researchers to explore complex topics through a conversational approach, allowing for flexibility in the discussion to uncover nuanced understandings and perspectives that are not easily accessible through other data collection methods. There is plenty of existing research and information on machine learning and AI, but the release of generative AI is fairly new. Therefore, relying on academic studies will not be sufficient enough. I find expert interviews particularly valuable when investigating new inventions and their direct impact on the industry as these AI models have the potential to revolutionize how software is developed [71]. Experts are 'forced' to be proactive and updated to stay relevant within the field and avoid being disrupted by the technology but instead adapt accordingly. The software development field is in perpetual motion, with experts offering insights into the present moment that surpass what books and academic studies provide [71]. Unlike the static nature of books and academic studies, the work environment in software development continuously adapts to changes ad hoc, necessitating real-time expertise.

There are two primary interview structures: structured and unstructured [51]. A structured interview follows a predetermined plan, with each interviewee receiving identical questions, akin to a survey, often requiring minimal elaboration. In contrast, an unstructured interview is less formal and encourages free dialogue that captures nuanced points. A third option, semi-structured, blends aspects of both approaches [64]. It involves pre-written questions but allows room for follow-up inquiries, fostering a more organic conversation during the interview process. A semi-structured interview was particularly well-suited for this thesis for several reasons. First, it offered the flexibility to explore topics in-depth, going beyond the scope of the pre-written questions. This approach enabled the discovery of insights that might not have been anticipated at the planning stage. Additionally, the format allowed follow-up questions directly related to the expert's responses. This adaptability was crucial for delving deeper into subjects as they emerged during the conversation. Such an interview acted out like a conversation provides a more comprehensive understanding of the expert's perspectives and experiences.

Interviews are being employed as a key methodology to establish both the current state and future direction of AI integration in DevOps. During the initial analysis phase, these interviews provide valuable insights into the present landscape of AI applications within DevOps processes, helping us understand existing capabilities and challenges. Moreover, discussions about future possibilities guide our vision for how AI can enhance specific DevOps activities. By exploring expert opinions on upcoming trends and potential innovations, we identify key areas for improvement. These insights inform the design of experiments to test AI, LLMs, and cloud technologies. This approach aims to ensure we remain at the forefront of technological advancements.

2.3 Experimental Method

The experimental method is a systematic approach used in scientific research to investigate cause-and-effect relationships between variables [75]. It involves the manipulation of one or more independent variables to observe the effect on a dependent variable while controlling for extraneous variables that could influence the results. The experimental method generally involves several fundamental steps: formulating a hypothesis, performing experiments, gathering data, analyzing the data, and finally, reporting the results [5]. This method lends itself well to this context, particularly because prompts and prompt engineering offer numerous avenues for exploration. The efficacy of the outcomes hinges on the initial input, emphasizing the experimental nature of prompt engineering. Testing hundreds of different prompts and experimenting with various prompt engineering frameworks become essential aspects of this process. When considering the theoretical framework of machine learning in general, the process of fine-tuning models through parameter and weight adjustments aligns closely with experimental practices. This underscores the suitability of employing such a methodology for the project.

The results of a single experiment often serve as a foundation for subsequent ones. In some cases, multiple iterations are required to arrive at a satisfactory answer to a hypothesis [75]. The experiments conducted within this thesis are aimed not at attaining definitive answers, but rather at providing clear insights into the efficacy of various prompt engineering frameworks and AI tools & services in code generation. These insights identify promising candidates for optimizing the coding phase of a DevOps pipeline. The details regarding both the structure and execution of the experiment will be expanded upon in the analysis chapter 5.6.

Chapter 3

Theory

This chapter will outline the existing theories and the technologies necessary for state-of-the-art products to exist.

The structure of this chapter is inspired by the waterfall model, not so much in its way of developing software, but in a way that the following technology could not exist without the previous one. The technologies will be briefly introduced and presented sequentially to get an intuitive and smooth transition from technology to product. Firstly, Machine Learning (ML) will be introduced by explaining the core fundamentals necessary to understand how Natural Language Processing (NLP) and LLMs operate all from the importance of data, training, and their pivotal role in the advancement of AI. Lastly, the three top-end state-of-the-art large language model services OpenAI ChatGPT, Google Gemini and open source model Llama3 will be explored. In case you want to skip the introduction to the various technologies involved in the creation of large language models, continue reading from section 4.1 and forward.

LLMs are tools created in the field of NLP, which helps computers understand and engage in conversations like humans [44]. This technology builds on the foundation of machine learning, a concept that dates back to the 1950s [61]. Think of it as a chain reaction in technology, where each new development stems from the previous one.

3.1 Machine Learning

The vast majority of modern society is engaging with services on an everyday basis which utilizes machine learning. All from recommendation systems in the entertainment domain to tumour analysis in the healthcare industry, not to mention the simple model which helps detect and filter the many spam emails trying to infiltrate our inboxes [69]. The traditional definition of machine learning was articulated by computer scientist Arthur Samuel back in 1959 [6]:

"Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed." [6]

In other words, models can improve from experience gained through the data it's trained upon. It is important to emphasize data when introducing machine learning, as it's the foundational building block of any model. There are different types of learning [69]:

Supervised learning involves training a model on a pre-labeled dataset. This means that the correct output or label accompanies each example in the training dataset [69]. The "label" refers to the desired outcome or answer the model is supposed to predict based on the input data. For instance, in a dataset used for a spam detection system, each email in the training set is labeled as either "spam" or "not spam" ahead of time. It attempts to discover patterns or relationships between the input features (e.g., words in an email) and the output labels (e.g., spam status) [67]. The quality and size of the labeled dataset significantly impact the model's performance [74].

In contrast to supervised learning, where models are trained on pre-labeled data, unsupervised learning is training on data without any labels [68]. This means that the learning algorithm is not given any explicit instructions on what to do with the data. Hence, it must identify patterns, relationships, or structures within the dataset on its own [74]. A classic example of unsupervised learning in action is the identification of clusters of tumours in healthcare to determine if they are benign (healthy) or malignant (unhealthy) [66]. Unlike supervised learning, where a model would be trained with images of tumours already labeled as benign or malignant, in unsupervised learning, the model analyses the features of the tumour images without any prior labels. The algorithm might look for patterns or similarities in the data, such as shapes, sizes, or densities, and attempt to group similar tumours. Incorporating machine learning into tumor analysis can significantly enhance the diagnostic process. By training models on extensive datasets, these systems can identify tumors with high confidence levels, flagging those that are likely malignant due to their pronounced deviation from typical patterns. This capability does not eliminate the need for human evaluation of the tumour. Instead, it streamlines the preliminary screening process. By automatically categorizing some tumors as benign or suspect, machine learning algorithms can optimize doctors' workflow, allowing them to allocate more time to complex cases that require their specialized skills and attention.

3.2 Natural Language Processing

Natural Language Processing (NLP) is a branch of machine learning dedicated to enabling computers to comprehend, interpret, and generate human language. NLP involves a series of technical processes, including Tokenization and Vectorization, that transform raw text data into structured representations for the model to analyze [80]. Sentences are broken down into smaller frames or units to facilitate understanding. Rather than interpreting words in isolation, NLP models analyze the semantic relations between these frames to grasp the meaning of the sentence similar to human comprehension [44]. This approach allows the model to consider the context conveyed by the arrangement of words, capturing the complexity of language beyond the literal meaning of the singular frame. For instance, in the sentence "A guy sitting on a bank looking over the lake watching the glowing sunset," the model recognizes the semantic relations between "guy," "sitting on a bank," "looking over the lake," and "watching the glowing sunset." By understanding the contextual connections between these frames, the model interprets the scene of a person observing a sunset by a lakeside, demonstrating its ability to comprehend language in a manner that mirrors human cognition. External lecturer Allan Hammershøj gave this particular example during one of his lectures. To illustrate this example I've generated two images using the model **DALL·E 3** from the OpenAI Generative Pre-Trained Transformers (GPT) bundle. Please look in the appendix section for the image comparison illustrating ambiguity A.1. Handling this might be relatively straightforward for the majority of NLP models. However, processing larger texts can be slightly more complicated for the model.

The Transformer architecture addresses these issues by employing a "self-attention" mechanism [76]. Self-attention allows the model to weigh the importance of different words in a sequence when processing each word. This means that the model can focus more on relevant words and less on irrelevant ones, enabling it to capture dependencies regardless of their distance in the input sequence. Think of attention as having multiple pairs of eyes looking at different parts of the sentence at the same time [27]. This helps the model understand various relationships within the text more effectively regardless of the input size.

3.2.1 Generative AI

Generative AI refers to AI systems that can generate new content, such as text, images, or music, based on the user prompt and patterns learned from the data they were trained on. In the context of LLMs like GPT, generative AI allows the model to produce human-like text based on the input it receives. Combining Generative AI with the transformer architecture led to the name GPT (Generative Pre-trained Transformer).

3.3 Large Language Model

Large Language Models are tools created in the field of natural language processing, which helps computers understand and engage in conversations like humans [44]. These models are built on deep learning architectures, with transformers being a prominent example [77]. LLMs have the capacity to understand and generate human-like text based on the vast amounts of data they have been trained on. LLMs are neural network-based architectures trained on massive datasets containing text from diverse sources such as books, articles, websites, and social media platforms [29]. The training process involves exposing the model to sequences of text and training it to predict the next word or sequence of words in a given context. This process enables the model to learn the statistical patterns and relationships inherent in natural language, also known as language modelling [79].

The breakthrough with LLMs lies in their ability to capture and leverage complex linguistic structures and semantics through the use of attention mechanisms within the transformer architecture [45]. Attention mechanisms allow the model to focus on relevant parts of the input text when generating responses, facilitating more accurate and contextually appropriate outputs.

Some state-of-the-art models are trained on vast amounts of coding, building, and implementation data, enabling them to understand programming languages and generate code in response to user prompts [83]. These models are often referred to as code-generating language models and can assist developers in writing code more efficiently and accurately. By learning from extensive repositories of code snippets, documentation, and programming tutorials, these models gain a deep understanding of the syntax, structure, and semantics of various programming languages [48]. When provided with a user prompt or description of the desired functionality, the model can generate corresponding code snippets that fulfil the specified requirements.

3.3.1 Fine-tuning

Fine-tuning involves training the model on specific tasks or datasets to adapt it to perform a particular function or generate specific types of content. There are numerous AI code assistants explicitly designed for processing coding data, making them proficient in handling coding tasks but not necessarily adept at general language understanding. These models, while excellent at code processing, need fine-tuning with models such as GPT or Gemini to better understand user inputs and prompts, as these requests are often presented in textual information. Fine-tuning such models with additional language models can refine their capabilities by adjusting parameters to better suit specific tasks or domains, such as coding and implementation. This process not only enhances how the model presents code to the user, potentially improving the user experience and productivity but also amplifies the model's ability to understand user requests, thereby enhancing the efficiency of prompt engineering.

An interesting question arises regarding the order of fine-tuning and its impact on a model's proficiency in various tasks. For instance, does a language model initially trained on text but later fine-tuned with code remain better at language tasks than coding, or does it maintain a balance between language and coding capabilities? Conversely, would a model initially trained on code and subsequently fine-tuned with language tasks exhibit superior coding abilities while being competent in language tasks, but not excel in them? This raises the possibility that a model might develop a primary function and a secondary function based on the sequence of its training and fine-tuning data. For example, a coding model fine-tuned with language data might always be better at coding tasks compared to language tasks, and vice versa. This concept could influence how we approach the development and optimization of language models for specific applications, highlighting the importance of strategic planning in the training process. This question will be challenged in the analysis phase of one of the experiments to determine if two different models (one trained on language and later fine-tuned with code, another trained on coding content and later fine-tuned with language) will perform differently in coding tasks."

3.4 Retrieval Augmented Generation

Retrieval-augmented generation blends two powerful techniques in natural language processing: retrieval-based methods and generative models. By intertwining these approaches, the system enriches the quality and relevance of generated text. In the retrieval phase, the system searches through a vast database or corpus of existing text to find relevant information based on the input or context provided. This retrieved information serves as a foundation for the subsequent generation phase. During the generation phase, the generative model leverages the retrieved information alongside the original input to produce text that is more contextually aligned and meaningful. The fusion of retrieval-based methods and generative models enhances the relevance, diversity, and domain-specificity of the generated text. This approach finds applications across various natural language processing tasks, from dialogue generation to text summarization and beyond, where producing coherent and contextually relevant text is important. An example of this could be a RAG-powered FAQ solution where employees of a company can search for answers using a chatbot. The chatbot would be trained upon NLP and able to retrieve information from all the documents in the vector database and fetch it based on the semantic relation between the prompt and the context of documents [53].

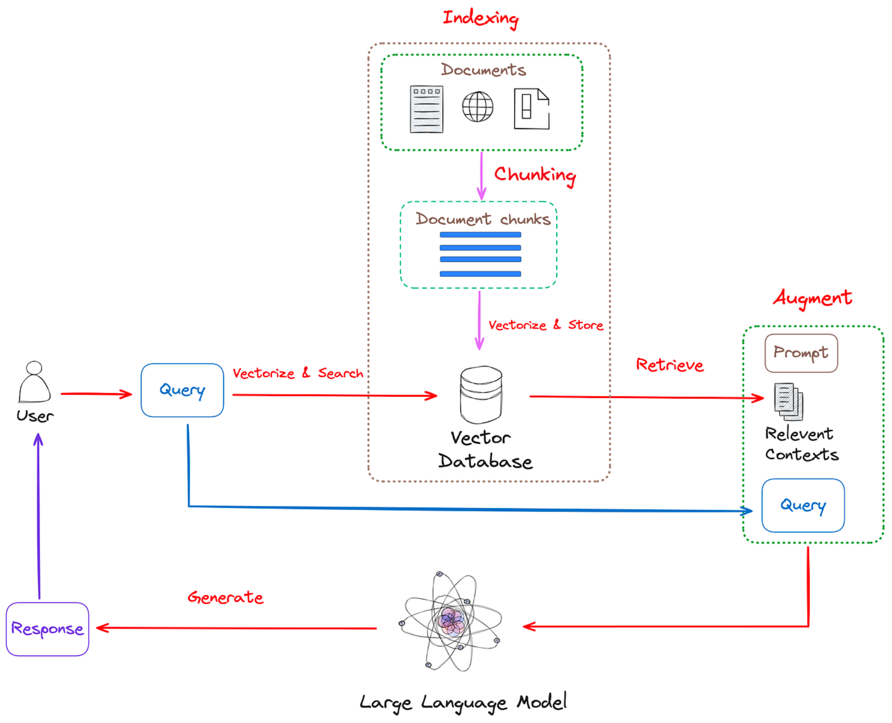


Figure 3.1: RAG diagram from Medium [32]

3.5 Prompt Engineering

Prompt engineering is an essential aspect of interacting with language models, focusing on the process of crafting inputs that guide the model to generate desired outputs [82]. At the core of this practice lies the understanding and utilization of various types of prompts, each serving a unique purpose in the interaction between the user and the AI system. These prompts include:

1. **System Prompt**: This is the initial setup or instruction that a language model, like GPT, uses to understand the context of its operation [62]. It sets the boundaries and guidelines for the AI's responses, often embedded within the model's programming or provided by the developer before user interaction. It ensures that the AI operates within a defined scope and with specific capabilities tailored to its intended use. A system prompt for a model assisting programmers to generate code could be e.g. "Assume the role of a developer with the capability to generate code from code documentation and natural language input".

2. **User Prompt**: These are inputs or questions provided by the user, intended to elicit a response from the language model. User prompts can range from simple questions to complex requests for analysis or creation. The effectiveness of

the response often depends on the clarity and specificity of the user prompt, highlighting the importance of prompt engineering to facilitate effective communication between humans and AI.

3. **History Prompt**: This type of prompt includes the context of previous interactions within a session, allowing the AI to generate responses that are not only relevant to the current question but also consistent with the flow of the conversation. By considering the history prompt, the model can provide more nuanced and coherent responses, taking into account the evolving context of the dialogue. OpenAI has been enhancing GPT across several iterations by expanding the token count for individual messages and thread history, enabling longer conversations and fostering higher-quality interactions with GPT [49].

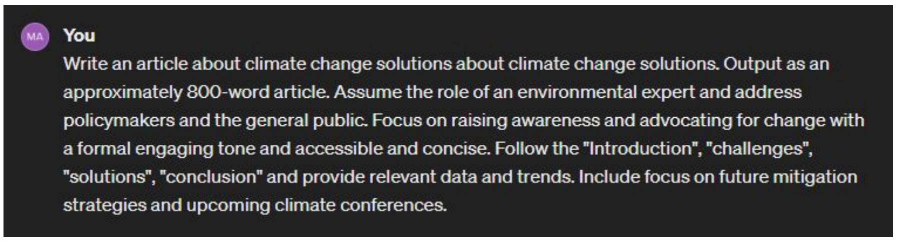
4. **Format Prompt**: This refers to prompts that guide the format or structure of the AI's response. For example, a format prompt could instruct the model to provide an answer in the form of a list, a detailed explanation, or a concise summary. This is particularly useful in tailoring the output to meet specific user needs or presentation requirements, ensuring that the information is delivered in the most useful and accessible manner.

3.5.1 Tagged User Prompting

Patrick Parker was among the pioneering figures to introduce Prompt Engineering during his presentation at the annual tech conference European Identity and Cloud Conference (EIC) in May 2023 [55], occurring approximately six months after the launch of ChatGPT by OpenAI in November 2022 [7]. Parker characterized Prompt Engineering as the means to become a "Bot Whisperer," by producing structured or tagged prompts aimed at unlocking the complete capabilities of this new LLM. Here is an illustration of tagged user prompting:

> Write a **[TASK]** about **[TOPIC]**. Output as **[FORMAT]**. Assume the role of **[ROLE]** and address **[TARGET AUDIENCE]**. Focus on **[GOAL]** with a **[TONE]** and **[LANGUAGE STYLE]**. Follow the **[STRUCTURE]** and provide appropriate **[STATISTICS]**. Include **[ADDITIONAL DETAILS]** [55].

Creating and structuring prompts using a tagged prompt will thoroughly guide the model to produce the desired output. To create an example:


MA You

Write an article about climate change solutions about climate change solutions. Output as an approximately 800-word article. Assume the role of an environmental expert and address policymakers and the general public. Focus on raising awareness and advocating for change with a formal engaging tone and accessible and concise. Follow the "Introduction", "challenges", "solutions", "conclusion" and provide relevant data and trends. Include focus on future mitigation strategies and upcoming climate conferences.

Figure 3.2: Illustration of tagged prompt framework - Answer from ChatGPT can be seen in appendix A.2

3.5.2 Zero-shot & Few-shot Prompting

Zero-shot and Few-shot prompting refers to the ability of an LLM to understand and respond to a task it has never seen before during its training, using only a single prompt without any prior examples or specific training on that task [34]. This is accomplished through the use of general instructions or a natural language description of the task. Zero-shot prompting is particularly valuable because it demonstrates a model's ability to apply its generalized training to specific tasks effectively, showcasing its flexibility and adaptability. This approach is used in various applications, from generating text to answering questions and more, in scenarios where it's impractical to train a model on every possible task it might encounter. The name indicates the model gets no further information or examples as instruction, on how to complete the task [84].

Example: A language model generates a summary of an article without ever being specifically trained on summarizing texts [57].

Few-shot prompting, on the other hand, involves giving the model a very small number of examples (typically between one and a few dozen) during or right before inference, to guide its performance on a specific task [34]. These examples help "tune" the model's response to the task at hand, providing a context or framework within which the model can better understand and generate appropriate outputs.

Example: A language model is given three examples of text summaries before being asked to summarize a new article. These examples act as a direct reference for how to approach the summarization task [57].

Chapter 4

State-of-the-art

This chapter will outline the existing state-of-the-art large language model services & tools, as well as the DevOps lifecycle

In this section, we explore the synergies between state-of-the-art LLMs and contemporary software development methodologies, particularly DevOps. We will encounter an analysis of prominent LLMs and their potential to enhance various facets of the DevOps workflow. The intersection of these two domains reveals insights into how LLMs can optimize and refine software development practices.

4.1 OpenAI ChatGPT

OpenAI's ChatGPT is an advanced language generation model designed to understand and produce human-like text based on the input it receives. ChatGPT was initially released in November 2022 followed by a series of iterations and improvements over subsequent versions, with the latest being GPT-4 updated in March 2023 [7]. GPT-4 is not just a singular model; it's a suite of large language models, each with distinct abilities. This includes DALL·E, which can create images from text descriptions, Whisper, designed to transcribe audio into text, and Codex, adept at understanding and generating code in various programming languages, among others [50]. With every new version, the capabilities of GPT models have expanded, thanks to the growth in the number of parameters and nodes within their transformer architecture. More parameters mean a more efficient model. Below is a concise overview of the various iterations and their specifications:

GPT-3 [50]
Released in June 2020 by OpenAI, GPT-3 marked a monumental step forward in language model technology. With an architecture boasting 175 billion parameters, it was capable of processing texts up to 2048 tokens in length, making it one of the most

advanced AI models of its time. GPT-3's vast knowledge base and sophisticated understanding of language allowed for an unprecedented range of applications, from generating readable text to creating user-friendly chatbot responses. Its release set a new standard for what was possible in natural language processing and opened up new opportunities by offering GPT services as an API to developers.

GPT-3.5 [50]
In March 2022, OpenAI released GPT-3.5 as an incremental update to its predecessor. While it maintained a similar token-handling capacity, this version introduced refinements in the model's understanding and context retention abilities, which affected the quality of history prompting. The update didn't significantly expand the model's size but rather focused on optimizing its existing framework for better performance. GPT-3.5 was a testament to OpenAI's commitment to continuous improvement, offering enhancements in text generation quality and coherence.

GPT-4 [50]
March 2023 saw the introduction of GPT-4, a revolutionary update that vastly extended the model's capabilities. GPT-4 introduced an ability to process texts up to 8192 tokens in length, coupled with a massive increase in parameters, venturing into the trillions. This version introduced multimodal capabilities, allowing the model to interpret and generate responses not just to text but also to images, code and audio showcasing an impressive leap in versatility. GPT-4's enhanced reasoning and deeper understanding of complex queries solidified its position as a pivotal tool across numerous industries, from enhancing creative writing to streamlining customer support systems. Its introduction marked a significant milestone in the journey towards more intuitive and capable AI systems.

OpenAI has plenty of things in its pipeline, amongst the most notable is a text-to-video model [72]. Sora is an AI model that can create realistic and imaginative scenes from text instructions. It can generate videos up to a minute long while maintaining visual quality and adherence to the user's prompt. Sora has not yet been released but rather in a "testing phase" where safety and quality are being assessed by a limited number of testers that include red teamers who will try to find ways to misuse the model through prompt engineering, and visual artists, designers, and filmmakers who will provide feedback on how to make the model more helpful.

4.2 Google Gemini (Bard)

In March 2023, Google released its own large language model called Bard [58]. Designed as a powerful chatbot, Bard offered users the full potential of Google AI. Almost a year later, Bard was rebranded as Gemini, transforming from a single-purpose language model to a versatile multi-modal tool. Gemini can now understand and respond in various formats, including text-to-speech (Chirp), text-to-image (Imagen) and text-to-code (Codey), similar to GPT [17]. Google envisioned Gemini not just as an end-user service but also as a development platform for others to build upon [1].

While Google's LLM arrived after ChatGPT, they have a longer history of developing AI models across the many products they offer. This includes natural language processing and translation tools like Google Lens, healthcare automation advancements with MedLM, and the extensive search optimization algorithms throughout all of Google's products [23].

Gemini has multiple high-end versions of its LLM available depending on the task it has to assist with, each with a high parameter increase. Gemini-base has a total of 270 billion parameters which is significantly higher than what GPT-3 offered back in 2020. Gemini-mid and Ultra move into the trillion category with 1.2 and 1.56 trillion parameters. Each iteration of Gemini also increases the number of tokens which improves contextual information contained within the history prompt. Many Google products can function offline, handling basic AI tasks on your device. This is achieved by using a smaller library of data compared to the full online version. For instance, Google Maps can still provide navigation and some features even without an internet connection. Similarly, Google Lens can translate text offline. Following this approach, Google offers Gemini Nano, an offline version of Gemini that brings core functionalities to your device for use without internet access [19].

Gemini 1.0 [59] Google's Gemini 1.0, a pioneering LLM drawing knowledge from 57 subjects including math, physics, history, law, medicine, and ethics, showcases its unparalleled versatility. Its strength lies in a multimodal design, effortlessly combining text, code, audio, image, and video inputs. With the capacity to process up to 32,000 token inputs and generate 2048 tokens in output, it provides a robust foundation for developers to craft innovative services and software applications [18]. Notably, Gemini 1.0 outshines other leading models, including GPT-4, triumphing in 30 out of 32 tasks, as championed by Google. This achievement solidifies its status as a significant advancement in AI technology [59].

Gemini 1.5 [60]
Google's Gemini 1.5 greatly improves its ability to remember information during a conversation (History prompting). Previously, it could only process 32,000 tokens at a time (as input), but the new version can handle up to 1 million tokens (and even 10 million tokens in research) [16]. This allows it to analyze much larger amounts of data, such as summarizing thousands of pages of documents or writing documentation for entire codebases. This particular feature is exciting concerning the research question,

as code documentation might become the building block of code generation.

4.3 Ollama

Ollama excels in deploying state-of-the-art AI models such as Llama3, Mistral, and Phi 3 [47]. These models are recognized for their capabilities in both natural language processing and code generation. What sets Ollama apart is its robust support for models that are primarily trained on coding data. This means that while most language models, like those in the GPT family, are trained on diverse text data and rely on a complex modular architecture to handle various tasks, Ollama models are deeply integrated and specialized in understanding and producing code. This specialization makes Ollama particularly powerful for tasks involving programming languages, allowing it to generate precise and contextually relevant code snippets and solutions.

Ollama's open-source nature is another significant feature. It enables users to freely access, modify, and deploy the models according to their needs. This openness not only fosters a collaborative community of developers and researchers but also accelerates the innovation process. By being open-source, Ollama invites contributions and customizations, leading to a continually evolving platform that meets the diverse needs of its user base.

A key differentiator for Ollama is its capability to function offline without an internet connection. Users can download and run models entirely on their local machines, ensuring that sensitive data remains secure and operations can continue uninterrupted even without internet access. This offline functionality is crucial for environments where data privacy is a priority or internet connectivity is unreliable. Despite operating locally, Ollama doesn't compromise on performance, making it a robust choice for on-premises AI deployments.

In addition to its offline strengths, Ollama is highly adaptable for online applications. It can be seamlessly integrated into popular coding environments such as Visual Studio Code, where it can assist with real-time code generation and debugging. This integration allows developers to leverage Ollama's advanced capabilities directly within their development workflow, enhancing productivity and efficiency. Ollama's support for API calls further extends its utility, enabling it to handle a wide range of tasks and applications.

4.4 Microsoft Copilots

Microsoft Copilots are advanced AI tools embedded across the Office 365 suite, designed to assist users with a variety of tasks using the power of GPT [42]. These Copilots seamlessly work across the Microsoft product line, including Word, Excel, PowerPoint, Edge, and Teams [41]. By making API calls on behalf of the user, Copilots streamline workflows and enhance productivity. For example, a user can create a PowerPoint presentation that integrates data and insights from Excel spreadsheets and content from Word documents, enabling a more cohesive and efficient work process. Microsoft Copilots are available as a subscription service.

One of the critical challenges Microsoft and other companies faced with using LLMs like GPT was the tendency of these models to provide responses even when uncertain, which could lead to misinformation or inaccurate data. To address this, Microsoft introduced the Microsoft Semantic Kernel, a QC feature [54]. This mechanism functions similarly to a plugin that executes microservices to fetch "fresh data" rather than relying solely on potentially outdated or incorrect general answers from the LLM. For instance, if a user requests a PowerPoint presentation based on specific organizational data, the Semantic Kernel can execute a function in Excel to retrieve the most up-to-date data, ensuring accuracy and relevance. This approach, however, requires that the data reside within Microsoft products, such as Excel, and not third-party services outside of Microsoft's ecosystem.

The Microsoft Semantic Kernel and its plugins are crucial in obtaining accurate and fresh data, significantly enhancing the trustworthiness of the responses provided by Copilot. Microsoft has also implemented measures to ensure QA, such as providing references to the sources of information instead of generic answers. When conducting searches, Copilot utilizes Bing to retrieve and reference the specific sources, aligning with the requirements of the AI Act on explainable AI. While Microsoft does not guarantee complete factual accuracy, they advise users to double-check the responses, especially when the information is sourced from third-party websites.

4.5 DevOps

The term DevOps originates from the fusion of "Development" and "Operations," signifying a holistic approach to software development and deployment [30]. It centers on fostering collaboration, integration, and communication among all teams involved in the software lifecycle. As software typically undergoes multiple development iterations, the DevOps pipeline recurs through these cycles [14]. The fundamental aim of DevOps is to optimize the software lifecycle from conception (idea phase) to full functionality (operation) by bridging the gap between teams and stakeholders [26].

When a customer proposes a new software idea, it is crucial for both software developers and operations engineers to be well-informed. Extensive planning is essential to ensure smooth operation, whether the service targets 1.000, 10.000, or 1 million users and whether it's a mobile application or a cloud-based service. Deployment planning involves extensive preparation, influenced by the developers' implementation choices. Below is an illustration of the stages in a DevOps pipeline in figure 4.1 followed by the specified building blocks [26].

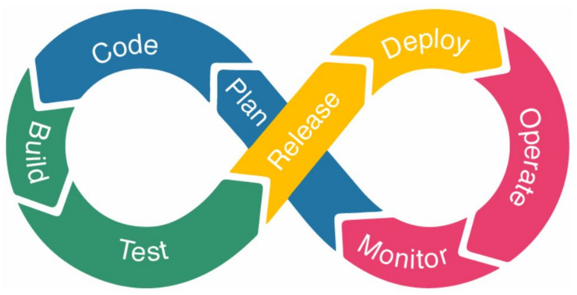


Figure 4.1: Figure illustrating the steps involved in a DevOps pipeline [52]

1. **Plan**
   The Plan stage involves defining the scope, objectives, and requirements of the software project. It includes tasks such as outlining features, prioritizing work items, and estimating timelines and resources needed for development. Effective planning sets the foundation for the entire DevOps pipeline, ensuring alignment with business goals and stakeholder expectations.

2. **Code**
   In the Code stage, developers write, review, and modify the source code of the software. This stage encompasses activities such as implementing new features, fixing bugs, and optimizing code for performance and maintainability. **GitHub copilot** [22] does a magnificent job assisting the developer ad-hoc while coding. Version control systems like **Git** [21] are commonly used to manage changes to the codebase, enabling collaboration among team members and tracking code history. This code history includes well-written code documentation for each iteration and reasons for choosing methods, microservices, etc.

3. **Build**
   The Build stage involves compiling the source code into executable or deployable artifacts. This process typically includes tasks such as dependency resolution, compiling code into binaries, and packaging application components. Automated build tools like **Jenkins** [31] or **Azure Pipelines** [36] streamline the build

process, ensuring consistency and reproducibility across development environments.

4. **Test**
In the Test stage, automated tests are executed to validate the functionality, performance, and reliability of the software. This includes running unit tests to verify individual components, integration tests to ensure proper interaction between modules, and end-to-end tests to simulate user scenarios. Test automation frameworks like **Selenium** [65] and **JUnit** [33] facilitate efficient and thorough testing, helping to identify and address defects early in the development cycle.

5. **Release**
The Release stage involves preparing the software for deployment to a production environment. This includes finalizing testing activities, updating documentation, and generating release notes to communicate changes to users. Release management tools such as **GitLab CI/CD** (Continuous Integration/Continuous Deployment) [20] or **Octopus Deploy** [12] involve ongoing cycles of building, testing, deploying, and monitoring incremental code modifications. This iterative method not only minimizes the risk of building upon flawed or unsuccessful iterations but also enables GitLab CI/CD to detect bugs early in the development phase, ensuring that all deployed code aligns with predefined coding standards. This helps orchestrate the release process, ensuring smooth and controlled deployments with minimal downtime or disruption to users.

6. **Deploy**
In the Deploy stage, the software is pushed to the production environment and made available for end-users to access. This involves tasks such as configuring servers, provisioning infrastructure resources, and deploying application artifacts. Continuous deployment pipelines automate the deployment process, enabling rapid and reliable delivery of updates to production environments.

7. **Operate**
The Operate stage focuses on monitoring and managing the software in the production environment. This includes activities such as monitoring system performance, handling user requests and incidents, and ensuring the availability and reliability of the application. DevOps teams use monitoring tools like **Grafana** [24] to collect and analyze metrics, enabling proactive identification and resolution of issues.

8. **Monitor**
The Monitor stage involves collecting and analyzing data on system performance, user behavior, and application health. This data is used to gain insights into the overall health and performance of the software, identify areas for im-

provement, and inform future development efforts. Monitoring tools like **SolarWind** [46] and other dashboards provide real-time visibility into key metrics, enabling DevOps teams to make data-driven decisions and continuously optimize the software delivery process.

4.5.1 Improving DevOps via LLM Integration

Room for improvement during the Code stage
I suggest there's significant potential for enhancement in this phase through the integration of an LLM. We could potentially streamline development by leveraging AI to generate code based on comprehensive high-quality code documentation written **prior** to implementation. This could involve creating a mock-up of the expected codebase for the AI model to reference as a few-shot prompting example as discussed in section 3.5.2. It might even achieve success through zero-shot prompting, relying solely on the instruction prompt without providing any examples to the language model.

Room for improvement during the Build stage
The Build stage could be significantly streamlined and consolidated with the Code stage through the integration of an LLM. The process could be more cohesive, allowing the model to handle both stages within the same prompting thread, provided with sufficient information. For example, if the objective is to develop cloud-native software leveraging Microsoft Azure, the model could ensure the build process optimally aligns with this integration, facilitating a smoother transition to a cloud environment.

Chapter 5

Analysis

This chapter will analyze the insights gained from interviews with IT professionals and conduct several experiments, aimed at identifying strategies to further optimize DevOps practices.

This chapter begins by examining the current state of AI integration into DevOps, a crucial step in understanding the broader implications and potential for improvement within this field. As outlined in the methodology chapter, a series of preliminary interviews were conducted to gather foundational insights and identify key areas for enhancement. These interviews not only provided valuable knowledge but also served as a guiding framework for defining specific experiments. The focus of these experiments was on code generation and the comprehension of large-scale data documents through AI-driven solutions. The findings from these experiments will be critically analyzed in this chapter, offering a deeper understanding of the current capabilities and highlighting avenues for future advancements.

5.1 Preliminary interview with Microsoft

Microsoft is by far one of the leading companies when it comes to system and software development. Hence, interviewing an expert from Microsoft would provide invaluable insights into the practical applications and implications of LLMs in the industry. Microsoft's pioneering work in leveraging LLMs for enhancing productivity tools demonstrates the potential of these technologies to transform traditional software development practices. An expert with hands-on experience in this area could provide extensive information about:

- The current state of the software development field, from the perspective of a leading IT company.
- The challenges and opportunities occurring from integrating LLMs into the software development practices.

- Outlining the different sectors and how the employees of these sectors might be affected by the integration of AI into software development.
- A better understanding of what the future might look like, if we were to continue down the path of integrating AI into software development.

5.1.1 Structure & logistics

The interview took place online, facilitated by Microsoft Teams. As previously noted, Microsoft Office has integrated various AI technologies to improve the user experience, including a feature being utilized during this interview. Microsoft Teams offers a transcription service that accurately identifies and documents each participant's words at specific timestamps. The entire transcription is too extensive to add in the appendix section; instead, it will be attached as a Word file during the hand-in of the project.

The interview was semi-structured, guided by some pre-written questions, and allowing for follow-up questions to explore points of interest that aligned with the thesis topic. The interview with Jan Cordtz, which took place on 2nd February 2024, was initially intended as a preliminary discussion before the research question was fully defined. This conversation aimed to establish the context of the project and assist in formulating the research question and objectives of the thesis. This interview was the first of many interactions I had with Jan Cordtz throughout the duration of my master thesis, including subsequent conversations, meetings, and email exchanges.

Here is the list of the questions written before the first interview took place:

1. What will the distribution between human-written code and generated code look like in the near future?
2. What will be the primary building blocks for enabling AI to generate code? Is it code documentation? What other factors are involved?
3. As we progressively use LLMs in software development, how important do you think it is for a developer to seek skills in prompt engineering as opposed to basic coding?
4. Which areas in a DevOps pipeline can be optimized by using various LLM solutions?

5.1.2 The Interview with Jan Cordtz

I was privileged to interview Jan Cordtz, a senior cloud solution architect with a heavy tech-literate background. Jan has formerly been employed at IBM, Unisys and Oracle for a combined 26 years before landing his job at Microsoft, where he has remained for the last eight years. The agenda for this meeting was to establish a problem objective in code generation using LLMs. The meeting with Jan provided valuable insights into the integration of GPT within Microsoft Office products to automate workflows through human language. This initiative is set to transform how tasks are performed in Office applications.

The discussion primarily revolved around the proportion of code generated by machines versus humans and the potential for future shifts in this balance. Cordtz noted that he estimates currently 60% of code is machine-generated, with the remaining 40% produced by humans. This dynamic underscores the substantial role AI plays in coding, alongside the critical need for human oversight to ensure the output's relevance and quality. The conversation highlighted the preference for models specifically trained on high-quality data sources, such as GitHub, to maximize the efficiency and accuracy of machine-generated code. Cordtz envisions a possible adjustment in ratio to an 80% machine-generated versus 20% human-generated code. This expectation stems from the belief in the enhanced capabilities of models trained on robust, high-quality coding datasets - "We are not quite there yet, but the models are getting smarter" - As Jan states it. Furthermore, Jan believes we will soon see a significant deployment of RAG solutions, which he considers to be among the most useful applications of LLMs. Microsoft is actively developing several RAG solutions for its cloud infrastructure, illustrating the company's commitment to leveraging this technology to enhance its services and offerings.

The dialogue also touched upon the exciting potential for generating code in various languages, like Python for Excel, illustrating the AI's capability to simplify complex tasks and make sophisticated functionalities accessible to a wider user base. This opens up possibilities for users to specify in natural language what they want Excel to do for them, thus eliminating the need to master Excel itself. Instead, mastering human language and prompt engineering allows users to execute complex commands they likely could not have managed without the GPT copilot, making sophisticated software functionalities available to those without extensive technical expertise. Jan believes this extends to coding as well by stating: "Because of models like GPT my grandmother would have been able to master Excel as well as code, she just doesn't know it yet". This can be illustrated in both GitHub [22] and Visual Studio Code [43] where the copilot can assist users to generate code based on language and prompting. This facilitates a no-code & low-code environment where traditional code knowledge is not necessary but rather a complementary asset. That being said, understanding code is always beneficial and will allow a developer to be critical and do quality control of the copilot's output. The distance between an end-user and development has just shrunk significantly due to the integration of ChatGPT into Microsoft's products, as it enables a bridge between language and code.

An important part of the discussion addressed the impact of LLMs on different areas of software development, particularly comparing frontend and backend development. When asked whether frontend and backend development would be affected differently by the integration of LLMs for coding, and if there would be a varying distribution of machine versus human code between the two, Cordtz shared his perspective. He expects the distribution of machine-generated and human-written code to remain consistent across both frontend and backend development. However, he anticipates a greater need for human supervision over the generated code in the backend, due to the complexity of microservices and API calls that could potentially introduce system vulnerabilities or fail to meet expectations. As such, the backend will require developers to engage more intensively in validating, testing, and verifying the code compared to frontend tasks.

Cordtz emphasized the shifting landscape of software development skills. The importance of prompt engineering over basic coding skills was highlighted, stating, "We need people with these skills NOW." This reflects a critical need for professionals who can effectively interact with AI to generate code. Moreover, Cordtz stated that the main building blocks for generating code using LLMs would be thorough code documentation, system design, and service architecture. Investing more time in the design phase and outsourcing the implementation to a model is seen as ideal. This approach underscores the evolving roles within software development, where both innovative engagement with AI and foundational coding expertise are essential for delivering high-quality software solutions. However, QA of machine-generated code remains paramount, meaning that while prompt engineering should be used for coding, building, and implementation, basic coding skills are indispensable for verifying that the code performs as intended. Thus, in a DevOps pipeline, prompt engineering is for the coding, building, and initial implementation phases, complemented by basic coding skills for verification, testing, and integrating new code with existing systems. This nuanced approach highlights the importance of a balanced skill set in the modern software development environment, emphasizing both the innovative potential of AI and the enduring value of foundational coding knowledge.

He noted the resistance from the 'older' generation of software developers, who often express skepticism towards integrating LLMs into their workflow with remarks like, "I know very well how to code a backend, I do not need some machine to do it for me." Jan argues that the embrace of LLMs in software development is not just a trend but an inevitable evolution. He strongly advocates for the newer generation of developers, especially those fresh from university, to embrace LLMs and master prompt engineering. The key message from Jan is the importance of staying adaptable within the software development field, urging professionals not to cling to outdated practices or the notion that they can outperform an LLM. According to Jan, LLMs have already surpassed the capabilities and efficiency of the best coders he knows across most programming languages. He believes that even the most skilled coders can learn from AI. Jan encourages viewing LLMs as an extension of our capabilities rather than a replacement or threat to the coding profession, emphasizing the potential for AI to enhance rather than diminish the role of human developers.

Jan Cordtz extended a remarkable offer, presenting the opportunity to collaborate directly with him and Microsoft on building a RAG solution. This collaboration will take place within a comprehensive cloud environment provided by Microsoft Azure, specifically tailored for this project. This environment will not only facilitate the development of the RAG solution but also serve as a personal testing ground for generating code, utilizing code documentation as the foundational building block. The commencement of this collaborative coding session is scheduled for Tuesday, 27th February, marking the beginning of a hands-on phase that will significantly contribute to the analysis chapter. This partnership underscores the practical application of the discussed concepts and the tangible steps being taken toward harnessing the power of LLMs and RAG solutions in real-world scenarios.

5.2 Interview with Udviklings- og Forenklingsstyrelsen (UFST)

In this interview, I had the opportunity to speak with Tom Willy Nielsen, an IT Domain Architect at UFST. The primary agenda of our discussion was to identify a specific DevOps process that could be optimized through the integration of AI. Tom, along with his team, plays a crucial role in providing consultancy services to various organizations. Their responsibilities include maintaining, operating, and analyzing both new and legacy codebases, ensuring these systems run efficiently and effectively.

5.2.1 Structure & logistics

The interview with Tom was conducted online via Microsoft Teams. The primary focus of the discussion was to explore how DevOps processes at UFST could be optimized by leveraging the capabilities of AI, LLM, and RAG. The interview was conducted in a semi-structured format featuring targeted questions about the existing DevOps workflow processes that Tom and his colleagues were engaged in. This approach allowed for a comprehensive understanding of how advanced AI technologies could potentially enhance and streamline their operations. While a few pre-prepared questions were used to guide the interview, much like in our previous discussion with Microsoft, the aim was to foster an open-ended conversation to explore potential experimental ideas to test. Here is the list of the questions written before the interview took place:

1. Can you describe your job role and the key responsibilities that you and your team handle?
2. Which DevOps tasks or processes in your workflow currently appear redundant or tedious and might be candidates for optimization through AI technologies?
3. How do you manage the operations and maintenance of legacy systems within your team, and what challenges do you encounter in this process?

5.2.2 The Interview with Tom Willy Nielsen

Tom Willy Nielsen and his team are responsible for consulting on and overseeing approximately 250 systems and services. Many of these systems are built using legacy code, making them challenging to maintain due to the scarcity of developers proficient in languages like COBOL and HPS. This has led Tom to explore AI-driven solutions for modernizing these systems.

During the interview, Tom highlighted the difficulty in maintaining systems coded in legacy languages due to the significant challenge of finding active coders proficient in these older languages. He suggested that AI could potentially translate entire codebases to more modern languages like Java, Python, Rust, and C++, which have broader support and a larger pool of skilled developers. The aim is to make these systems easier to manage and maintain by converting them into more widely-used programming languages.

Another significant point raised by Tom was the redundancy in functionalities among various systems they supervise. Some systems have overlapping functionalities, and combining these systems could be beneficial. However, this consolidation process is complex and requires careful review to ensure that logic, functionality, security, and compliance are not compromised.

During the conversation, Tom and I also discussed the 3270 program, a terminal interface created by IBM that banks use to observe the data input and output of ATMs [28]. The data from these ATMs is handled by a large mainframe, with 3270 acting as a gateway for banks to monitor transactions. Systems like 3270 and the ATMs themselves are very difficult to replace with modern versions, such as those written in languages like Java. One of the primary reasons for this difficulty is the sheer scale of the existing infrastructure. With over 3.2 million ATMs installed worldwide, most of which are running on COBOL and connected to mainframes, replacing these systems would be an incredibly complex and resource-intensive task [73]. Changing millions of ATMs across the globe would not only be logistically challenging but also risk disrupting a well-established and functional system. Therefore, it might be more practical to maintain and support these legacy systems rather than attempting to reconstruct them using more modern coding languages, emphasizing the need for careful consideration in modernization efforts.

Tom believes that addressing these issues with AI can greatly enhance DevOps processes. By automating code translation and system consolidation, AI can streamline operations and improve efficiency. The goal is to create a more manageable and modern codebase that aligns with current development practices and standards.

A specific use case of AI that could greatly improve workflow at UFST involves the analysis and review of extensive EMCS documents. An important part of Tom and his team's job is to analyze and review these very large documents, which can be up to 100 PDF pages long and contain numerous cross-references, making them tedious and challenging to read. To streamline this process, UFST plans to experiment with building a RAG model in Microsoft Azure. This model will include a vector database of EMCS documents and be designed to handle specific prompts to assess its viability in optimizing Tom and his team's tasks. The AI should be capable of summarizing the content of the EMCS documents and answering specific questions related to the documents, while also referencing where the information can be found within the document. Making this procedure more convenient for Tom and his team would significantly improve their workflow and efficiency.

The insights from this interview with Tom Willy Nielsen provided valuable perspectives on the practical challenges of maintaining legacy systems and the potential role of AI in addressing these issues. By translating legacy codebases and consolidating system functionalities, AI can significantly optimize DevOps workflows at UFST, ensuring better maintainability and efficiency.

5.3 Generating Experiments: Collaborative meeting between UFST and Microsoft on AI for DevOps usecases

On May 16th and May 28th, representatives from UFST (Tom and Toraj) and Microsoft (Jan Cordtz and Rasmus) convened for a two-part meeting series aimed at establishing a collaboration to explore and develop AI-driven solutions within an Azure environment. I was privileged to participate in both meetings, providing valuable context for my thesis project and helping to generate the ideas and use cases that emerged from these discussions. This partnership is envisioned to enhance UFST's DevOps processes, offering innovative tools to streamline their workflow.

Initially, I believed my role in these meetings would be that of a spectator, observing the discussions to gather insights for my project. However, Peter Anglov introduced me as a prodigy in the field of AI integration, which encouraged me to take on a more active role. Instead of merely observing, I was invited to participate as a respected consultant, sharing my perspective and knowledge on the subjects being discussed. My extensive understanding of NLP, LLMs, fine-tuning techniques, and RAG systems, having built one prior to this project, proved useful throughout the conversations.

The first meeting on May 16th focused on presenting the overarching ideas and possibilities that AI could bring to UFST. Tom from UFST shared his vision of how AI could revolutionize their operations, outlining a future where AI functions not merely as a tool but as a collaborative colleague. He elaborated on AI's potential to provide a "helicopter perspective" on various tasks, including analyzing complex systems, constructing and understanding code, generating documentation, and facilitating the transformation of legacy COBOL code into more modern languages such as Java. His approach emphasized the need for AI to integrate seamlessly into the daily activities of his team, enhancing their capabilities without replacing their essential human insight.

In his pursuit of these ideas, Tom had already embarked on preliminary experiments using the LLM Ollama (model Llama3). He had installed this model on a mini computer, similar to a Raspberry Pi, to test its capabilities in optimizing some of the use cases he envisioned. While this setup provided some initial insights, the results were only moderately satisfactory and did not fully meet the expansive requirements of his ambitious projects. This experience underscored the need for more robust and scalable solutions, paving the way for the collaboration with Microsoft.

The follow-up meeting on May 28th delved deeper into the technical aspects of the proposed AI solutions. Rasmus and Jan from Microsoft brought their expertise to the table, critically evaluating Tom's proposals with a focus on their practical implementation. Rasmus, noted for his extensive knowledge and experience in AI applications within Microsoft, provided valuable insights into the feasibility of these ideas. His feedback was instrumental in grounding the discussion in the current capabilities of Microsoft's AI technologies, including their various AI copilots.

During the technical meeting, Rasmus and Jan explored the practicalities and potential challenges of implementing the suggested AI solutions. They acknowledged the ambitious scope of Tom's vision while delineating which aspects could be realistically achieved with existing technology and which might require further advancements. This candid exchange helped to clarify the possibilities and set realistic expectations for the collaboration.

The dialogue highlighted the role of Microsoft's AI copilots in supporting UFST's objectives. These copilots could potentially be utilized to automate routine coding tasks, provide real-time code analysis, and assist in documentation, all within an Azure environment tailored to UFST's needs. The discussion also touched on future possibilities, such as more advanced AI capabilities that could emerge as technology evolves, potentially aligning with UFST's long-term goals.

This series of meetings marked the beginning of a promising partnership between UFST and Microsoft. It established a framework for ongoing collaboration, where UFST's vision for AI-enhanced DevOps processes could be progressively realized with Microsoft's support and expertise. Moving forward, the use cases discussed will be thoroughly documented, and some will be actively pursued as part of my ongoing project. This effort aims to bridge the gap between Tom's innovative ideas and the practical applications of AI in enhancing UFST's workflow, setting a precedent for how organizations can leverage cutting-edge technology to achieve operational DevOps goals.

5.4 Outlining Experiments

Following an insightful meeting with industry professionals, we engaged in a brainstorming session to explore the future potential and current capabilities of AI in enhancing DevOps workflows. The objective was to delineate which scenarios were feasible with current technology and which were more aspirational. This collaborative effort aimed to identify specific experiments for my master's thesis that would demonstrate how generative AI can address existing challenges in DevOps.

The four experiments outlined in this research originated from ideas I conceived as I continuously processed new information and opportunities during my thesis. I was particularly excited by the beauty of how new ideas and experiments naturally evolved from these opportunities, transforming initial concepts into more refined and ambitious experiments. Initially, the focus was on exploring the possibilities of generating code from detailed documentation and integrating new code into existing codebases. However, as the thesis progressed, my engagement with key stakeholders, such as Tom from UFST, further expanded the scope of my research. Tom's interest in optimizing specific processes within UFST, potentially solvable with a RAG solution, sparked a surge of creativity in me. This led to the inception of additional experiments that I was eager to include in the study. These experiments were driven by the thrill of exploring uncharted territory in the intersection of AI and DevOps and were fueled by real-life issues brought directly to my attention by IT professionals themselves.

Under the guidance of Rasmus from Microsoft, an esteemed expert in AI often referred to as an "oracle" or "wizard" in the field, the experiments were listed in order of difficulty and feasibility. While Rasmus provided valuable insights into the practicality of each scenario, ensuring they were achievable within the scope of my research, it was my vision and proactive approach that shaped the direction of these experiments. The collaborative nature of our discussions, alongside the opportunities presented by the Microsoft Azure environment provided by Jan Cordtz, solidified the final selection of experiments. Each experiment represents a distinct aspect of how AI can be integrated into DevOps practices, driven by the ideas and strategies I developed throughout the thesis.

5.4.1 Experiment One - Code Generation Based on Code Documentation

Objective: To evaluate the feasibility of generating complete codebases from detailed documentation using LLMs.

- **Process:** Instead of the traditional approach of writing code first and documenting it afterward, this experiment proposes a reverse methodology. The service architecture, functionalities, logic, purpose, datatypes, and methods will be thoroughly described in natural language documentation.
- **Implementation:** LLMs, such as GPT-4, will be utilized to generate the entire codebase from these detailed descriptions. The generated code will then be evaluated for accuracy, efficiency, and adherence to the described specifications.
- **Expected Outcome:** This experiment aims to assess whether LLMs can reliably interpret comprehensive documentation to produce functional and efficient code, potentially revolutionizing the initial stages of software development.

5.4.2 Experiment Two - Simplifying and Translating Lengthy Technical Documents

Objective: To enhance the accessibility and comprehension of lengthy technical documents through summarization and translation.

- **Process:** As described by Tom from UFST, reviewing extensive and complex EU documents is a time-consuming and challenging task, particularly for non-native English speakers. These documents are often 100 PDF pages long, structured in a way that includes numerous cross-references, making them exhausting to read and not very intuitive. Readers frequently encounter a significant amount of seemingly irrelevant information, detracting from the core purpose of their review.
- **Implementation:** AI models will be used to process these lengthy PDF documents, generating concise summaries and translating the content from English

to Danish. The AI will be tasked with identifying and extracting the most pertinent information while maintaining the context and coherence of the original document.

- **Expected Outcome:** This experiment is expected to demonstrate how AI can streamline the review process for technical documents, saving time and improving accessibility for a broader audience within organizations like UFST. The AI-generated summaries and translations should make these documents more digestible and relevant, particularly for non-native English speakers.

5.4.3 Experiment Three - Generating New Code for Existing Codebases

Objective: To test the capability of AI in integrating new features into existing codebases, emphasizing the Testing and Deployment phases.

- **Process:** Developers often need to add new features to existing services, which involves modifying the codebase and ensuring compatibility with existing code. This experiment focuses on using AI to generate new code that integrates seamlessly with the current codebase.
- **Implementation:** The AI will generate the required feature code, which will then be run through the DevOps pipeline. The emphasis will be on the Testing and Deployment phases to identify and resolve any integration issues.
- **Expected Outcome:** The experiment seeks to demonstrate AI's potential in simplifying and accelerating the process of adding new features to existing systems, reducing the likelihood of integration problems.

5.4.4 Experiment Four - Reconstructing Legacy Codebases

Objective: To investigate the ability of AI to modernize old codebases by converting them into newer, supported programming languages without compromising the functionality, logic, security and compliance.

- **Process:** Many organizations maintain legacy codebases written in outdated languages such as COBOL or HPS. This experiment explores using AI to reconstruct these codebases into modern languages like Java, Python, or Rust.
- **Implementation:** The AI will translate legacy code into the target modern language, ensuring that the logic and functionality remain unchanged. Additionally, the new code must meet security and compliance standards verified by a professional software engineer.
- **Expected Outcome:** The goal is to validate AI's ability to facilitate the transition from legacy systems to modern, maintainable codebases, enhancing system longevity and compliance with current standards.

5.5 Assessing Experiment Prioritization

Experiments Two and Four were inspired during an insightful interview with Tom, where the focus was on innovative applications of language models. Specifically, Experiment Two explores the feasibility of creating a RAG solution. This solution is designed to manage Excise Movement and Control System (EMCS) documents by leveraging a LLM to search through a vector database containing vectorized information. This approach aims to enhance the retrieval and comprehension of complex document sets through advanced language processing techniques.

On the other hand, Experiments One and Three were derived from the research question of how software can be developed using purely linguistic approaches. This inquiry seeks to investigate the capabilities and limitations of using natural language as the primary tool for software development, pushing the boundaries of how we interact with and create software systems.

While both Experiments One and Two are feasible within the current technological landscape, Experiments Three and Four present significant challenges. Rasmus from Microsoft pointed out several key obstacles related to the limitations of current LLM capabilities and the complexities of coding standards. LLMs, such as GPT and Gemini, are trained primarily on text and natural language, which makes it difficult for them to handle heavy coding tasks, especially when adhering to rigorous standards like ISO, NIS2 or a set of requirement specifications. Developing software that complies with these standards and meets detailed requirement specifications is particularly challenging.

A critical complication with Experiments Three and Four is the limitation imposed by the number of tokens that an LLM can process. For a model to completely understand an entire codebase and accurately create new code for existing services, it needs to process extensive contextual information. The current token limits of LLMs constrain their ability to encompass large codebases in their entirety, which can lead to errors and incomplete understanding when generating new code. However, as discussed in the state-of-the-art chapter 4, the number of tokens that language models can handle is incrementally increasing with each newer version. This trend suggests that, in the near future, LLMs will be able to process more contextual information, potentially enabling them to grasp and manage the vast amounts of data that a codebase contains.

Integrating new code into existing codebases could be highly beneficial if the model is familiar with the entire codebase. However, this would necessitate downloading the general model and further training it on specific organizational data. While this approach might be relatively straightforward for languages like Java and Python thanks to the abundance of examples on platforms like GitHub, it is less feasible for COBOL. COBOL's scarcity of open-source code pose significant hurdles in obtaining a sufficiently large and diverse dataset for training.

Additionally, using actual company codebases for training could lead to complications with contracts and confidentiality agreements, as seen in the relationship between UFST and their clients. This restricts the ability to freely use proprietary code, making it even more challenging to implement effective solutions within the existing frameworks.

Experiments One and Two will be conducted within the scope of this thesis due to their feasibility with current LLM technologies, while Experiments Three and Four will be explored in a conceptual framework. The idea is to extend the capabilities of RAG solutions beyond textual information to entire codebases. This would involve using a LLM specifically trained on diverse coding languages and later fine-tuned with natural language processing capabilities. Such an approach could eventually enable LLMs to handle complex coding tasks, integrate new code into existing systems, and meet standards like ISO and NIS2. This vision suggests a future where RAG solutions are not just powerful tools for information retrieval but also pivotal in software development and maintenance, overcoming the challenges posed by today's token limitations and the complexity of coding standards.

5.6 Experiment: Code Generation Using Prompt Engineering in Large Language Models

Jan's estimations underscore the importance of experimentation to attain 80% machine-generated code in future code-generation scenarios. It's crucial to assess whether existing state-of-the-art models can generate code meeting our quality standards. Consequently, I aim to merge theoretical insights from the theory section with practical techniques from prompt engineering to conduct this experiment.

The original plan was to conduct the experiment using a single LLM. However, when I began working on one of my projects with Llama3, I noticed distinct differences in quality compared to GitHub Copilot, which is built on GPT. The variations between Llama's open-source models and GPT-based models required reconsideration. The fine-tuning process, as well as the sequence in which it's applied, significantly affects a model's proficiency in specific tasks, such as understanding language or generating code, far more than initially anticipated. As a result, the experiment was adjusted. Instead of generating code from documentation using only one model, I will now use both GPT and Llama. By comparing their outputs, I aim to assess how fine-tuning impacts a model's proficiency in coding versus language comprehension.

5.6.1 GPT vs. Ollama

One of the unique aspects of models supported by Ollama is their specialized training and fine-tuning process, which significantly impacts their performance in specific tasks. Generally, LLMs can be trained initially on a broad dataset of textual information, encompassing a wide range of language usage, including literature, conversations, and technical documents. This foundational training provides the model with a deep understanding of language structures, semantics, and context, making it proficient in generating and comprehending text. This would be the case for GPT.

After this initial training phase, the model can be fine-tuned with additional datasets focused on coding. This fine-tuning process involves adjusting the model's parameters based on code repositories, programming language documentation, and coding practices. As a result, the model, which was primarily efficient in natural language tasks due to its initial training, becomes adept at handling coding tasks. It learns to recognize code syntax, understand programming concepts, and generate code snippets effectively. However, its core remains deeply rooted in language comprehension, allowing it to excel in tasks that require both coding and textual responses.

Conversely, a model can be trained initially on coding data. This approach focuses on the model's ability to understand and generate code from the outset. The model absorbs vast amounts of information related to programming languages, coding standards, and software development techniques. After establishing this strong coding foundation, the model can then be fine-tuned with a general language model, such as GPT, which is highly proficient in natural language processing. This subsequent fine-tuning enhances the model's ability to interpret user prompts and provide coherent and contextually appropriate responses in natural language, even though its primary strength lies in coding. Thus, while it excels in coding tasks, it also maintains the capability to understand and interact with users in a natural language effectively. This would be the case for Ollama.

This dual training approach whether starting with language and adding coding or with coding and adding language; tailors the model's capabilities to specific needs. In the case of Ollama, models are typically fine-tuned in a way that emphasizes their utility in coding environments, making them particularly useful for developers who need sophisticated AI support in writing, debugging, and understanding code.

5.6.2 Prompt Creation

In my endeavor to enhance the capabilities of LLMs in generating code, I have extended the existing tag-prompt framework included in the theory section 3.5.1 of tagged prompts to cater specifically to coding requirements. The original framework, exemplified by tags such as [TASK], [TOPIC], [FORMAT], and others, has been instrumental in guiding language models to produce coherent and contextually appropriate textual content. Recognizing the need for similar guidance in the domain of code generation, I have introduced a new set of tags tailored for this purpose.

These new tags are designed to encapsulate the specific requirements and constraints often encountered in coding tasks. They include:

- **[PROGRAMMING LANGUAGE]:** Specifies the programming language to be used, ensuring compatibility and adherence to project standards.
- **[LIBRARY/FRAMEWORK]:** Identifies any specific libraries or frameworks, aiding in the utilization of existing tools and resources.
- **[FUNCTIONALITY]:** Describes the specific functionality to be implemented, providing clear direction for the task.
- **[ERROR HANDLING]:** Details the error handling mechanisms, ensuring robust and fault-tolerant code.
- **[TEST CASES]:** Outlines test cases to verify the code's correctness, promoting reliability and accuracy.
- **[CODE STRUCTURE]:** Defines the overall structure or architecture of the code, promoting organization and maintainability.
- **[API CALL]:** Specifies the API calls to be made, facilitating integration with external services and data sources.
- **[REQUIREMENT SPECIFICATION]:** Provides a detailed description of the requirements, ensuring that all aspects of the task are covered comprehensively.

By incorporating these tags into prompts, we can significantly improve the precision and relevance of the code generated by LLMs. This approach not only enhances the quality and functionality of the output but also allows us as developers to still be in full control of the output.

The following example demonstrates a coding prompt, utilizing the new tags, for a LLM to generate code that retrieves the top 10 cryptocurrencies and their exchange rates in dollars from an external website using an API key. This example will be used to test the prompt's effectiveness in both GPT and Llama3 for my experiment.

> Assume the role of **a software developer**. Write a **script** to **display the top 10 cryptocurrencies by market capitalization** with output as **Python code**. Use the **requests library** for **API calls** and **json** for data handling. Find and use the necessary **API endpoints** from the existing **Coinmarketcap** service. Include "INSERT YOUR API KEY" at the specific part of the code where I should insert my personal API key. Focus on **fetching** and **displaying the exchange rates in dollars** with a **concise and clear code structure**. Include **code documentation** and **error handling** for **network issues** and **invalid API responses**.

- **[PROGRAMMING LANGUAGE]:** Python
- **[LIBRARY/FRAMEWORK]:** Use the `requests` library for API calls and `json` for data handling.
- **[FUNCTIONALITY]:** Fetch the top 10 cryptocurrencies by market capitalization and display their exchange rates in dollars.
- **[ERROR HANDLING]:** Include error handling for network issues and invalid API responses.
- **[TEST CASES]:** Write test cases to verify that the script correctly fetches and displays the data.
- **[CODE STRUCTURE]:** Organize the code into functions for fetching data, processing data, and displaying results.
- **[API CALL]:** Use the provided API key to access the cryptocurrency data from an external website.
- **[REQUIREMENT SPECIFICATION]:** Ensure that the script is able to connect to the external website using an API key, fetch the top 10 cryptocurrencies by market capitalization, and display their names and exchange rates in dollars.

5.6.3 Analyzing the Performance of GPT and Llama3 in Code Generation via Prompt Engineering

In this experiment, we investigated the efficacy of prompt engineering to generate functional codebases using LLMs. The study was conducted in three phases. The first phase involved constructing a prompt designed to create a functional codebase, using Patrick Parker's "Tagged Prompts" framework as a foundation outlined in the theory section 3.5.1.

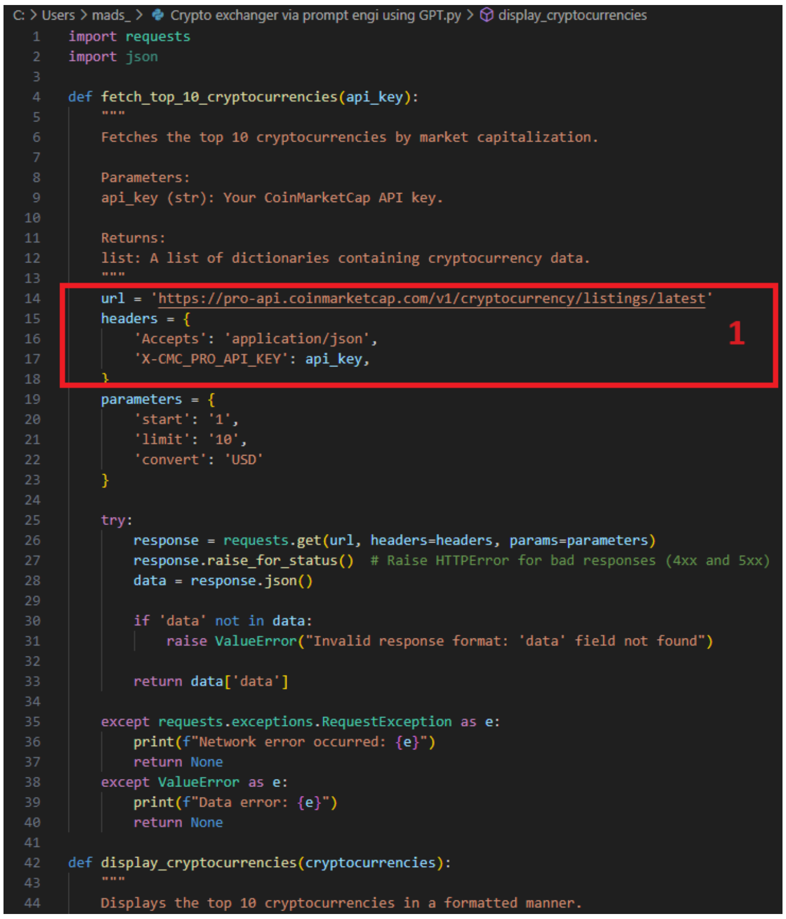
Phase one: Prompt Construction

The initial phase focused on the development of a specialized coding prompt using the "Tagged Prompts" framework. The modifications included creating new tags inspired by programming constructs, such as function definitions & requirements, API calls, error handling, and a task to obtain "fresh data" in the form of an API endpoint and key. This approach was tested and yielded successful results, demonstrating the potential of the modified framework in code generation.

Phase two: Output Analysis

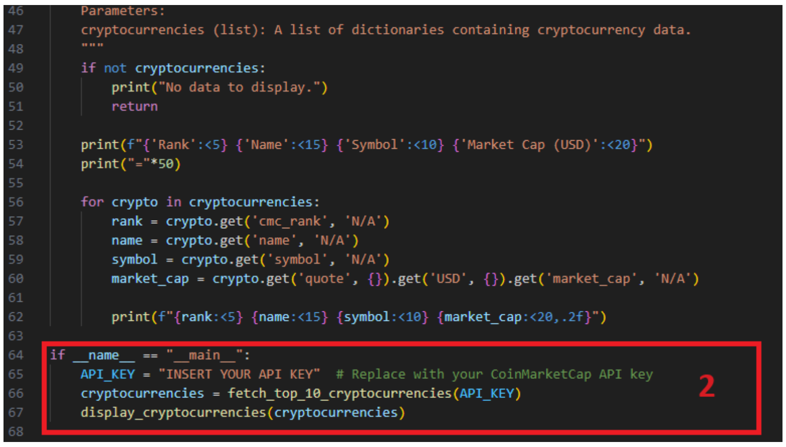
The second phase involved analyzing the outputs generated by two different models: GPT and Llama3. The hypothesis was that Llama3, being primarily trained on code, would outperform GPT, which was initially trained on text and later fine-tuned for code with the implementation of Codex.

Both models generated functional codebases with clear instructions on where to insert a personal API key and where to find it. This result was particularly impressive for GPT, considering its historical difficulty in handling information post-2021 due to its initial training cutoff. However, recent parameter updates and the ability to execute external processes using plugins have enhanced GPT's capability to obtain and handle fresh data. For instance, when planning a trip, GPT can call plugins to retrieve real-time data from flight search engines like Momondo, Skyscanner, and Google Flights, contrasting its earlier generic and outdated responses.


C: > Users > mads_ > Crypto exchanger via prompt engi using GPT.py > display_cryptocurrencies
1

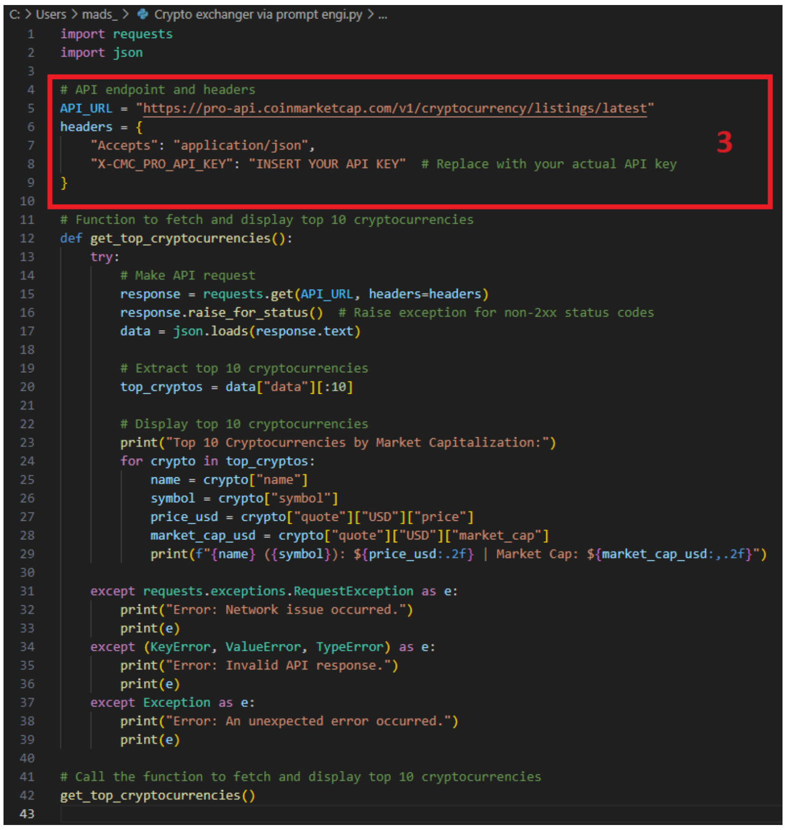
```python
import requests
import json

def fetch_top_10_cryptocurrencies(api_key):
    """
    Fetches the top 10 cryptocurrencies by market capitalization.

    Parameters:
    api_key (str): Your CoinMarketCap API key.

    Returns:
    list: A list of dictionaries containing cryptocurrency data.
    """
    url = 'https://pro-api.coinmarketcap.com/v1/cryptocurrency/listings/latest'
    headers = {
        'Accepts': 'application/json',
        'X-CMC_PRO_API_KEY': api_key,
    }
    parameters = {
        'start': '1',
        'limit': '10',
        'convert': 'USD'
    }

    try:
        response = requests.get(url, headers=headers, params=parameters)
        response.raise_for_status()  # Raise HTTPError for bad responses (4xx and 5xx)
        data = response.json()

        if 'data' not in data:
            raise ValueError("Invalid response format: 'data' field not found")

        return data['data']

    except requests.exceptions.RequestException as e:
        print(f"Network error occurred: {e}")
        return None
    except ValueError as e:
        print(f"Data error: {e}")
        return None

def display_cryptocurrencies(cryptocurrencies):
    """
    Displays the top 10 cryptocurrencies in a formatted manner.
```

Figure 5.1: Figure illustrating how GPT constructs the codebase differently by splitting up the API endpoint and key object into separate codeblocks



```python
    Parameters:
    cryptocurrencies (list): A list of dictionaries containing cryptocurrency data.
    """
    if not cryptocurrencies:
        print("No data to display.")
        return

    print(f"{'Rank':<5} {'Name':<15} {'Symbol':<10} {'Market Cap (USD)':<20}")
    print("="*50)

    for crypto in cryptocurrencies:
        rank = crypto.get('cmc_rank', 'N/A')
        name = crypto.get('name', 'N/A')
        symbol = crypto.get('symbol', 'N/A')
        market_cap = crypto.get('quote', {}).get('USD', {}).get('market_cap', 'N/A')

        print(f"{rank:<5} {name:<15} {symbol:<10} {market_cap:<20,.2f}")

if __name__ == "__main__":
    API_KEY = "INSERT YOUR API KEY"  # Replace with your CoinMarketCap API key
    cryptocurrencies = fetch_top_10_cryptocurrencies(API_KEY)
    display_cryptocurrencies(cryptocurrencies)
```

Figure 5.2: Figure illustrating how GPT constructs the codebase differently by splitting up the API endpoint and key object into separate codeblocks

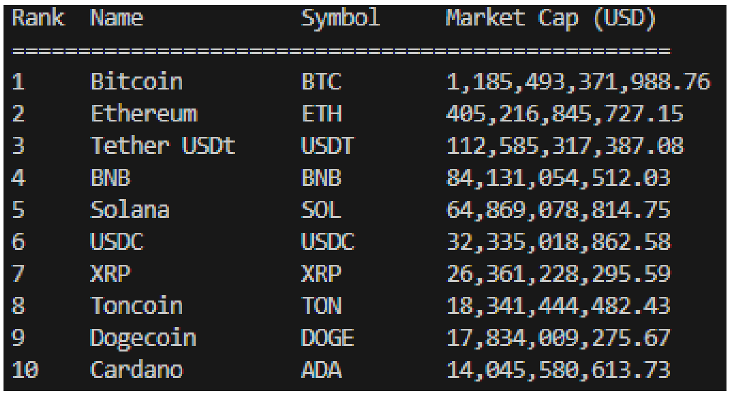


C: > Users > mads_ > Crypto exchanger via prompt engi.py > ...

```python
import requests
import json

# API endpoint and headers
API_URL = "https://pro-api.coinmarketcap.com/v1/cryptocurrency/listings/latest"
headers = {
    "Accepts": "application/json",
    "X-CMC_PRO_API_KEY": "INSERT YOUR API KEY"  # Replace with your actual API key
}

# Function to fetch and display top 10 cryptocurrencies
def get_top_cryptocurrencies():
    try:
        # Make API request
        response = requests.get(API_URL, headers=headers)
        response.raise_for_status()  # Raise exception for non-2xx status codes
        data = json.loads(response.text)

        # Extract top 10 cryptocurrencies
        top_cryptos = data["data"][:10]

        # Display top 10 cryptocurrencies
        print("Top 10 Cryptocurrencies by Market Capitalization:")
        for crypto in top_cryptos:
            name = crypto["name"]
            symbol = crypto["symbol"]
            price_usd = crypto["quote"]["USD"]["price"]
            market_cap_usd = crypto["quote"]["USD"]["market_cap"]
            print(f"{name} ({symbol}): ${price_usd:.2f} | Market Cap: ${market_cap_usd:,.2f}")

    except requests.exceptions.RequestException as e:
        print("Error: Network issue occurred.")
        print(e)
    except (KeyError, ValueError, TypeError) as e:
        print("Error: Invalid API response.")
        print(e)
    except Exception as e:
        print("Error: An unexpected error occurred.")
        print(e)

# Call the function to fetch and display top 10 cryptocurrencies
get_top_cryptocurrencies()
```

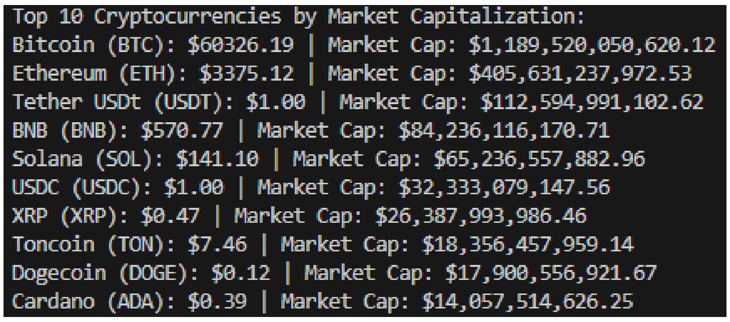
Figure 5.3: Figure illustrating how Llama3 managed to construct the API endpoint & API key in a manageable and more intuitive way for developers to understand

Both models successfully identified the API endpoint at Coinmarketcap and provided the correct web destination for generating a personal API key. This was expected for Llama3 but was an unexpected and pleasing outcome for GPT, which has often struggled with "fresh data" and sometimes provided plausible but incorrect information. The performance of both models in understanding the prompt and producing accurate output was commendable.



```
Rank  Name            Symbol     Market Cap (USD)
================================================
1     Bitcoin         BTC        1,185,493,371,988.76
2     Ethereum        ETH        405,216,845,727.15
3     Tether USDt     USDT       112,585,317,387.08
4     BNB             BNB        84,131,054,512.03
5     Solana          SOL        64,869,078,814.75
6     USDC            USDC       32,335,018,862.58
7     XRP             XRP        26,361,228,295.59
8     Toncoin         TON        18,341,444,482.43
9     Dogecoin        DOGE       17,834,009,275.67
10    Cardano         ADA        14,045,580,613.73
```

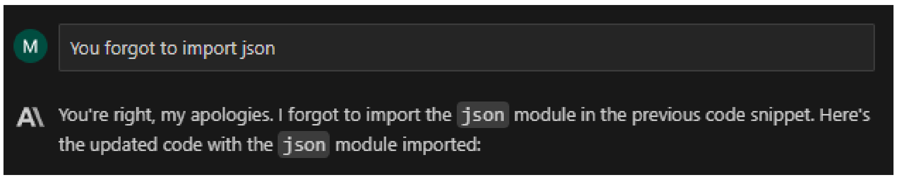
Figure 5.4: The terminal output from running code provided by GPT



```
Top 10 Cryptocurrencies by Market Capitalization:
Bitcoin (BTC): $60326.19 | Market Cap: $1,189,520,050,620.12
Ethereum (ETH): $3375.12 | Market Cap: $405,631,237,972.53
Tether USDt (USDT): $1.00 | Market Cap: $112,594,991,102.62
BNB (BNB): $570.77 | Market Cap: $84,236,116,170.71
Solana (SOL): $141.10 | Market Cap: $65,236,557,882.96
USDC (USDC): $1.00 | Market Cap: $32,333,079,147.56
XRP (XRP): $0.47 | Market Cap: $26,387,993,986.46
Toncoin (TON): $7.46 | Market Cap: $18,356,457,959.14
Dogecoin (DOGE): $0.12 | Market Cap: $17,900,556,921.67
Cardano (ADA): $0.39 | Market Cap: $14,057,514,626.25
```

Figure 5.5: The terminal output from running code provided by Llama3

An interesting discovery was made when Llama3 forgot a simple but crucial line of code: **import JSON**. As the code progressed, it called an object expecting a JSON return without having imported the necessary library at the beginning of the codebase. This is a typical amateur mistake, highlighting a significant aspect of working with LLMs. Upon recognizing the error, I prompted the model with "You forgot to import json," to which it apologized and corrected the mistake in the codebase. This incident underscores the importance of QC when using LLMs, as they can occasionally overlook basic yet essential lines of code despite their advanced capabilities. It serves as a reminder that while these models can perform complex tasks, they are not infallible and require careful review and validation of their outputs. QA and QC of outputs will be further discussed in the discussion section 7.


M You forgot to import json

You're right, my apologies. I forgot to import the `json` module in the previous code snippet. Here's the updated code with the `json` module imported:

Figure 5.6: Figure illustrating Llamas failure to import JSON and its responds when instructed to correct the error

Phase Three: Codebase Comparison

The final phase compared the two generated codebases to determine which model would be a more suitable candidate for assisting programmers in improving their coding, building, and integration workflows. The primary distinction observed was in GPT's handling of the API key object. GPT chose to split the API endpoint and key into two different code blocks, one at the beginning and one at the end of the code. While this was not confusing for this small experiment, it could pose challenges in larger and more complex projects where contiguous code blocks are preferred for readability and maintenance. Especially when multiple developers manage larger codebases as a team.

Experiment Conclusion and Future Work

The experiment demonstrated that both GPT and Llama3 are capable of generating functional codebases with clear instructions, despite their different training focuses. Llama3's training on code provided a slight edge in predictability and organization, while GPT's recent updates have significantly improved its ability to handle real-time data and generate accurate outputs. The findings suggest that both models can be valuable tools for programmers, with considerations for their respective strengths and potential limitations in larger projects.

However, it is important to note that the evidence gathered from this experiment was not sufficient to definitively conclude that Llama3 would consistently outperform GPT in handling larger codebases. Based on my estimations, Llama3 would probably perform slightly better in such scenarios due to its more code-oriented training. This hypothesis could form the basis for upcoming experiments, where a more extensive analysis of larger and more complex codebases could provide clearer insights into the comparative performance of these models.

5.7 Experiment: Simplifying and translating lengthy technical documents via RAG

The primary aim of the meeting with Tom was to explore the potential for optimizing DevOps workflows and processes at UFST through the application of AI. Tom had initially reached out seeking AI solutions to improve the efficiency of handling extensive and complex EMCS documents. These documents, often lengthy and intricate, pose significant challenges in terms of reading, analyzing, and understanding, especially for his colleagues who are less experienced with them. Tom expressed a high interest in making this process more streamlined and manageable.

As Tom elaborated on the difficulties faced in dealing with these documents, it became evident to me that a RAG solution could effectively address his concerns. By creating a vector database populated with EMCS documents and employing an effective system prompt, I realized that we could develop a tool to assist Tom and his colleagues in comprehending and navigating these documents more efficiently.

My initial interview with Jan Cordtz and Peter Anglov led us to acquire a subscription to Microsoft Azure, equipped with the necessary components for developing a RAG solution. Subsequently, Jan, Peter, and I conducted several development sessions to set up the testing environment for this solution.

5.7.1 Setup of the RAG Solution in Microsoft Azure

Microsoft Azure provides an intuitive GUI interface for managing its cloud services. However, the vast number of sections, labels, and icons can be overwhelming without proper guidance. Fortunately, we had the expertise of Jan Cordtz to help us navigate and set up the environment effectively.

The RAG solution I aim to construct can be illustrated by figure 5.7. The EMCS documents gets chunked and vectorized so that a semantic search can be conducted based on the query of the user.

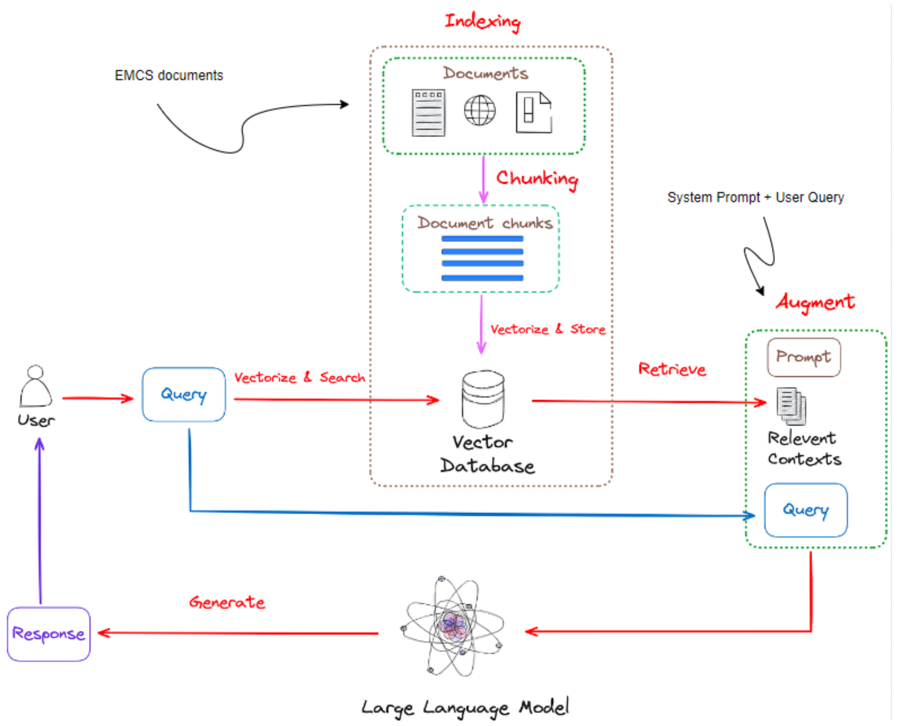


Figure 5.7: Figure illustrating the RAG solution in Azure. Base diagram source [32]

Creating the Virtual Machine and Resource Group

The first step involved booting up a Virtual Machine (VM) tailored for our experiment. This required creating a resource group, which represents the "hardware" components of our virtual machine, such as the network card, RAM (memory), graphics card, and processor. In Azure, these hardware components are emulated as software, allowing for easy scalability. As Jan Cordtz aptly put it, "There is no hardware in Azure; it is all software to the end user. Microsoft delegates the necessary hardware to run the virtual machines." This setup allows for seamless upgrades and scalability by simply configuring a more powerful resource group.

One of the significant advantages of using Azure for this experiment was the ability to hibernate the virtual machine between sessions. This flexibility, in contrast to maintaining a physical system with dedicated hardware, offered substantial cost savings. Running everything in Microsoft Azure cost approximately 0.70 DKK per hour, making it a highly economical solution compared to the expense of a physical system worth thousands of dollars.

Establishing Secure Access

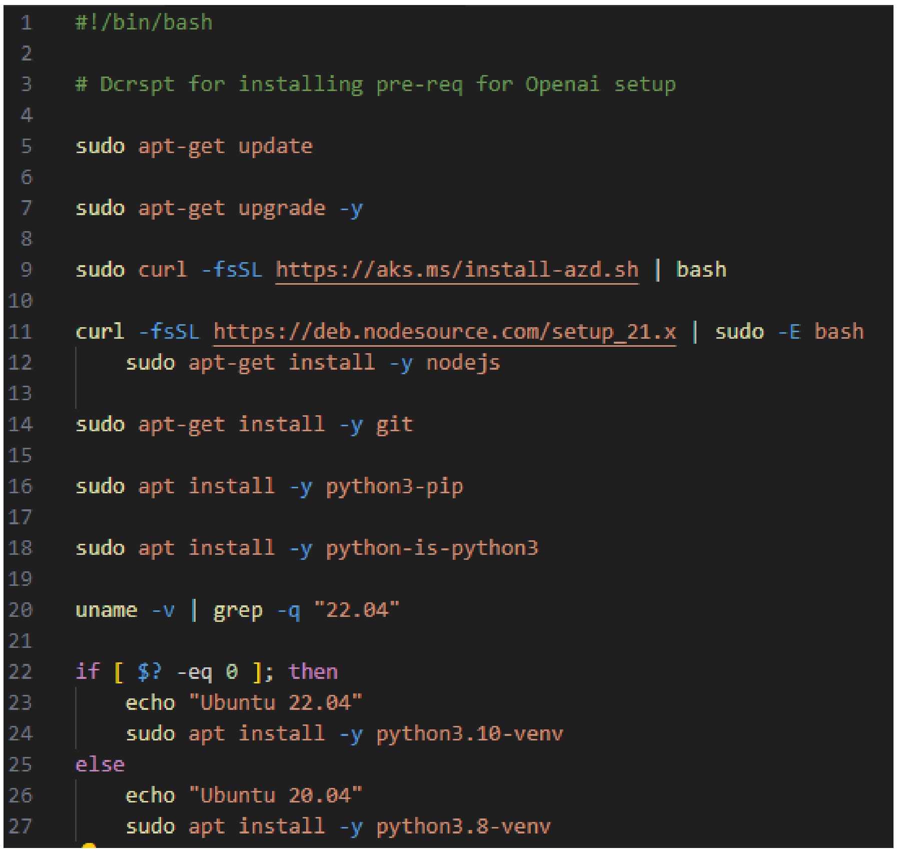
To ensure secure access, we set up a just-in-time connection between Peter Anglov and the virtual machine using SSH. This method of authentication prevented unauthorized access, even if others used the same dedicated hardware. Microsoft employs this as one of its authentication methods to ensure that only authorized users can access their virtual machines.

Choosing the Operating System

We opted to boot up a Linux machine for its open-source nature and high customizability, which facilitated the installation and operation of the necessary OpenAI components with fewer errors. Our initial attempts to set up the system on Windows had failed due to conflicts between OpenAI components and the Windows operating system. Linux, with its "sudo" (super-user do) command, provided greater control and privileges, which were essential for our setup.

For our Linux distribution, we chose UBUNTU. While Red Hat and SUSE offer full support and management tools, they come at a cost. UBUNTU, on the other hand, is free and provides equivalent support, funded through donations. This choice aligned with the philosophy of its creator, a South African businessman, who believed in making Linux accessible to all, including third-world countries, without financial barriers.

Additionally, several installations were necessary to configure our environment correctly. We installed essential Microsoft libraries, Python, git, nodejs, and various Ubuntu packages to ensure compatibility and smooth operation of the OpenAI models. These installations were critical for setting up the infrastructure needed to run our experiments efficiently. The terminal input is displayed in figure 5.8.



```bash
#!/bin/bash

# Dcrspt for installing pre-req for Openai setup

sudo apt-get update

sudo apt-get upgrade -y

sudo curl -fsSL https://aks.ms/install-azd.sh | bash

curl -fsSL https://deb.nodesource.com/setup_21.x | sudo -E bash
    sudo apt-get install -y nodejs

sudo apt-get install -y git

sudo apt install -y python3-pip

sudo apt install -y python-is-python3

uname -v | grep -q "22.04"

if [ $? -eq 0 ]; then
    echo "Ubuntu 22.04"
    sudo apt install -y python3.10-venv
else
    echo "Ubuntu 20.04"
    sudo apt install -y python3.8-venv
```

Figure 5.8: Terminal input used to set up the Linux machine

Launching the Virtual Machine

The next phase involved launching our virtual machine in one of Microsoft Azure's data centers in France. This decision was driven by multiple factors, including the availability and cost of OpenAI services in different locations. At the time of the experiment, France offered OpenAI GPT-3.5 for free, whereas Sweden provided access to GPT-4 at a cost.

Beyond cost considerations, legal and regulatory obligations also played a crucial role in our decision. The regulations governing LLMs can vary significantly depending on the region and country. For instance, launching OpenAI GPT in a data center located in the United States but accessing it from Denmark could pose challenges related to GDPR and other local regulations. To ensure compliance with these regulations and standards, we had a legal obligation to launch our system in France. This choice allowed us to adhere to GDPR requirements and other relevant legal frameworks, even though it meant settling for GPT-3.5 instead of GPT-4. We were instructed to make this decision by Microsoft, which was one of the significant advantages of being supervised by Jan Cordtz. His guidance ensured that our system was set up in the most compliant and efficient manner possible.

This decision had implications for our experiment, particularly regarding the token limits of GPT-3.5 compared to GPT-4. These differences and their impact on the experiment will be discussed further in the evaluation of the results.

5.7.2 Selection of Documents

Many of the optimization possibilities at UFST that Tom and I discussed involved accessing codebases, libraries, and other organization-confidential data. Some of his ideas included managing codebases of legacy mainframe code like COBOL and HPS. However, due to confidentiality and compliance requirements, Tom was unable to provide me with any actual data from these systems.

The case of improving interaction with EMCS documents was more convenient and intuitive because these documents are publicly available through the European Commission. All necessary documents could be easily downloaded as PDF files in various languages from the EU Commission's website, simplifying the data preparation process.

Preparation of EMCS Documents

Three different EMCS documents were selected for the experiment, summing up to 128 PDF pages [11]. These documents were:

1. Directive Regulation [10]
2. Delegated Regulation [8]
3. Implementing Regulation [9]

These documents supposedly cover all aspects related to EMCS, providing a comprehensive dataset for our RAG solution. By uploading these three documents into the vector database, we aimed to test the solution's capability to handle extensive cross-references not only within sections of a single document but also across multiple documents.

Vectorization and vector database Integration

The selected documents were uploaded and vectorized into our vector database as part of the RAG solution. The idea was to enable the RAG to retrieve information with a high success rate in response to any prompt related to the content of EMCS.

This process involved converting the documents into a machine-readable format and indexing their content to facilitate efficient retrieval. By choosing these publicly available EMCS documents, we ensured that our experiment adhered to all confidentiality and compliance standards while providing a robust test case for the capabilities of our RAG solution.

5.7.3 Designing System prompt and User prompts

Designing the system prompt is arguably the most crucial component of a RAG solution, as it provides overall guidance to the model, shaping how it processes future prompts and queries from users. This concept is briefly covered in the Theory section 3.5 but is elaborated upon here as experimentation led to the discovery of a prompt hierarchy. Imagine a hierarchy of prompts where each level dictates the processing of the next, in the following order: **System Prompt** -> **User Prompt** -> **History Prompt**.

For instance, three RAG solutions with identical data registries but different System Prompts would produce distinct answers to the same User Prompt, based on the instructions given by their respective System Prompts. The models' responses to subsequent prompts would still be affected by the initial System Prompt and would further distinguish based on the History Prompt in each of the separate model solutions.

A secondary prompt and every prompt thereafter take the history of the previous prompt plus the model output into consideration, further guiding the following input and outputs, similar to an evolving conversation. This structure ensures that the context from previous interactions influences future responses, enhancing the relevance and accuracy of the information provided.

System Prompt Construction

I constructed the System Prompt based on three key objectives provided by Tom during our meeting in May. The RAG should be able to accomplish the following:

- Capture and understand cross-references within individual documents.
- Bridge information across different documents, offering a more holistic understanding of EMCS-related queries.
- Provide accurate and contextually relevant information efficiently, thereby enhancing the usability and accessibility of these complex documents.

Additionally, the RAG should ideally translate these documents into Danish, as the originals are written in English. Although the European Commission offers translations, Tom finds them poorly executed. For this experiment, I chose to exclude translation capabilities as models perform inconsistently across different languages. Constructing the System Prompt in English but the User Prompt in Danish confused the model and mostly produced incorrect answers. This issue was discovered through experimentation with different System Prompts and mixing languages, akin to asking a colleague a question in English and receiving a response in Danish, something will inevitably get lost in translation. This issue is magnified in AI interactions, as different nodes and parameters are involved in the cognitive process.

Through thorough experimentation, I realized the importance of keeping the System Prompt and User Prompts in the same language to avoid confusing the model.

The System Prompt I created was:

> You are a highly intelligent assistant with a deep understanding of extensive and complex EMCS (Excise Movement and Control Systems) documents. Your primary function is to help the user comprehend and navigate these documents effectively, offering precise, relevant, and actionable insights.

User Prompts Construction

To achieve our objectives, I created three separate User Prompts. The goal was to retrieve answers from each of the documents separately, as well as to capture cross-references where the answer might be found in several places. The prompts would progressively increase in specificity to test the limits of the model's ability to understand the semantic relationship between the question and the retrieved content. The overall aim is for Tom and his colleagues to eventually rely on the model's answers rather than having to search through all three documents themselves. In this way, AI would optimize a general workflow at UFST. The three User Prompts I created were:

1. Provide an overall summary of the EMCS document, focusing on its main objectives and key sections.
2. Explain the concept of 'duty suspension' as described in the EMCS document. What are the conditions and requirements for it?
3. Can you summarize the section on the procedures for electronic administrative documents (e-AD) in the EMCS document?

These prompts were designed to test the efficiency and accuracy of the RAG solution with a progressive increase in difficulty. By using these prompts, we could evaluate whether the model could provide comprehensive and accurate information, thereby validating its usefulness in optimizing workflows at UFST related to EMCS documents.

5.7.4 Testing with Tom from UFST

I scheduled a meeting with Tom to demonstrate the RAG solution and conduct testing with the constructed prompts. Tom's active involvement in the testing process was crucial, as it provided immediate feedback and a practical assessment of the solution's effectiveness.

During the meeting, Tom inquired about the technical aspects of the data selection and the training process of the RAG solution. I explained that no additional training of the model was conducted. Instead, the selected data within the vector database was vectorized and 'exposed' to the GPT-4 model, allowing it to browse and find semantic relationships between the User Prompts and the contents of the vector database, no further testing was necessary.

Tom was curious about the potential for further training or fine-tuning of the model. I clarified that while it is possible to fine-tune the model by exposing it to massive amounts of EMCS documents and other work-related documents, it might not be practical. This is because newer iterations of documents might change some fundamental information, leading to conflicts in the model's training. For example, training a model to understand that 2 plus 2 equals 4, and then later encountering new data that suggests it equals 5, would create confusion and lower the model's confidence score. Instead, a RAG solution allows for the removal of older versions of documents and replacement with newer iterations, keeping the solution fully updated. By regularly updating the vector database, we ensure that the RAG solution remains current and accurate. Similarly, newer versions of GPT with increased parameters and tokens will improve the accuracy and response time of the RAG solution.

Tom mentioned that at UFST, they work with countless documents, with new ones constantly becoming relevant. This makes the process difficult and tedious. However, with a RAG solution, they could execute a script, perhaps at 3 AM, to pull in all new iterations of documents into the vector database, enabling the solution to assist Tom and his colleagues the very next morning. Even though the RAG solution should be able to update regularly with 10-20 new documents each day, the update process can be done seamlessly and automatically.

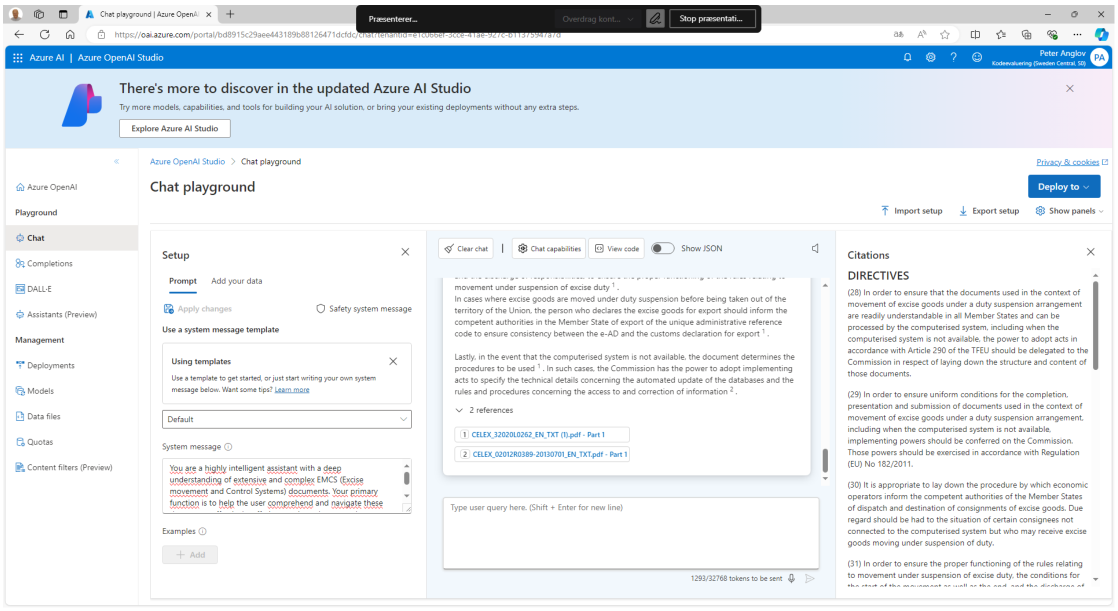
5.7.5 Results of the Experiment & Further Testing

Tom was impressed with the responsiveness of the model as I demonstrated the three pre-constructed User Prompts. I encouraged Tom to ask specific questions about the EMCS documents that he already knew the answers to, in order to test its accuracy and efficiency. Since Tom is not the primary person at UFST who works with EMCS documents, he did not feel qualified to determine whether the model provided proper answers. However, he emphasized the importance of QA of the answers provided by the solution. It is crucial to verify whether the responses are true and accurate. Additionally, it is important to evaluate whether better answers could be found elsewhere in the documents, and to check the cross-references within each document as well as across multiple documents.

To address this, Tom suggested setting up a meeting with his colleague Linnea Nissen, who works extensively with EMCS documents. Linnea's expertise would be invaluable in determining the accuracy and relevance of the model's responses. A meeting with Linnea Nissen was scheduled for the 17th of September. Although this is more than a week after the submission of my thesis, the results of this testing could be included as part of my presentation and defense of my thesis.

During our testing, Tom also posed an interesting question about whether the RAG solution could provide different answers to different people based on their roles within the organization. For instance, he suggested that an analyst and a software developer might require different types of answers, with one expecting a more technical response than the other. Implementing this would be extremely challenging and might necessitate creating different models with explicit System Prompts and different content within the vector database. However, it could potentially be achieved by having multiple System Prompts accessible through a user interface at an outer layer of the model. By configuring the output in the System Prompt to instruct the model to act as an analyst or a software developer, this type of request could be managed. An organizational user interface could be developed where each person has a unique System Prompt tailored to their specific profession and responsibilities.

One of the significant results of the experiment was the model's ability to understand and extract information from cross-references within individual documents and between multiple documents. This capability was tested using one of the user prompts, which was designed to check if the model could pull relevant information from both documents where the answer was distributed. Prior to the testing, I had verified that the necessary information could indeed be found in both documents. The model successfully demonstrated this by providing a response that referenced the appropriate sections from both documents, showcasing its ability to handle complex queries that require synthesizing information across multiple sources. The figure below illustrates one of the responses, displaying the model's capability to extract and integrate information from the vector database effectively.


There's more to discover in the updated Azure AI Studio

Try more models, capabilities, and tools for building your AI solution, or bring your existing deployments without any extra steps.

Chat playground

Setup — Prompt

Use a system message template

Using templates

Use a template to get started, or just start writing your own system message below. Want some tips? Learn more

Default

System message

You are a highly intelligent assistant with a deep understanding of extensive and complex EMCS (Excise movement and Control Systems) documents. Your primary function is to help the user comprehend and navigate these

movement under suspension of excise duty 1 .
In cases where excise goods are moved under duty suspension before being taken out of the territory of the Union, the person who declares the excise goods for export should inform the competent authorities in the Member State of export of the unique administrative reference code to ensure consistency between the e-AD and the customs declaration for export 1 .

Lastly, in the event that the computerised system is not available, the document determines the procedures to be used 1 . In such cases, the Commission has the power to adopt implementing acts to specify the technical details concerning the automated update of the databases and the rules and procedures concerning the access to and correction of information 2 .

2 references

1 CELEX_32020L0262_EN_TXT (1).pdf - Part 1

2 CELEX_02012R0389-20130701_EN_TXT.pdf - Part 1

1293/32768 tokens to be sent

Citations

DIRECTIVES

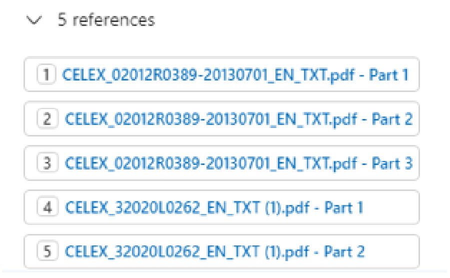
(28) In order to ensure that the documents used in the context of movement of excise goods under a duty suspension arrangement are readily understandable in all Member States and can be processed by the computerised system, including when the computerised system is not available, the power to adopt acts in accordance with Article 290 of the TFEU should be delegated to the Commission in respect of laying down the structure and content of those documents.

(29) In order to ensure uniform conditions for the completion, presentation and submission of documents used in the context of movement of excise goods under a duty suspension arrangement, including when the computerised system is not available, implementing powers should be conferred on the Commission. Those powers should be exercised in accordance with Regulation (EU) No 182/2011.

(30) It is appropriate to lay down the procedure by which economic operators inform the competent authorities of the Member States of dispatch and destination of consignments of excise goods. Due regard should be had to the situation of certain consignees not connected to the computerised system but who may receive excise goods moving under suspension of duty.

(31) In order to ensure the proper functioning of the rules relating to movement under suspension of excise duty, the conditions for

Figure 5.9: Snapshot of RAG solution responding to one of the pre-constructed User Prompts


5 references

1 CELEX_02012R0389-20130701_EN_TXT.pdf - Part 1

2 CELEX_02012R0389-20130701_EN_TXT.pdf - Part 2

3 CELEX_02012R0389-20130701_EN_TXT.pdf - Part 3

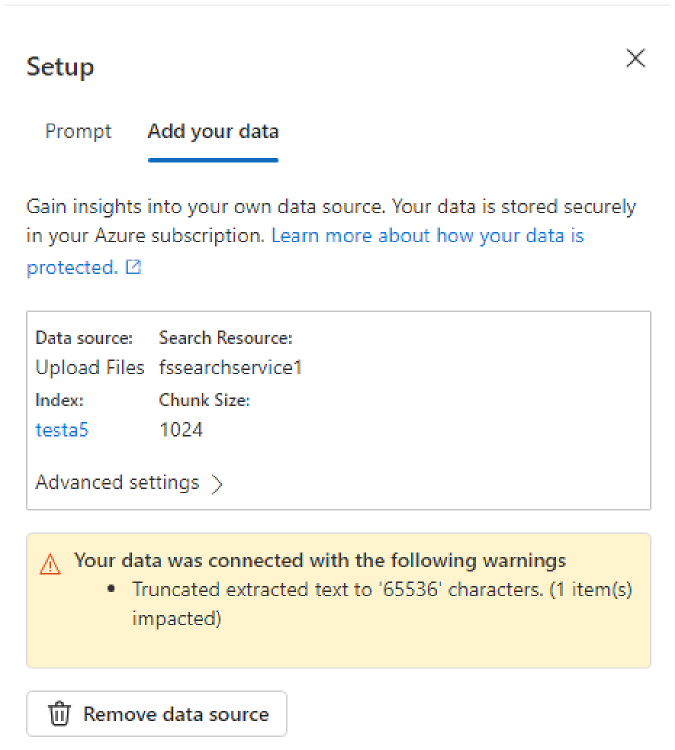
4 CELEX_32020L0262_EN_TXT (1).pdf - Part 1

5 CELEX_32020L0262_EN_TXT (1).pdf - Part 2

Figure 5.10: Figure illustrating the model's capability to cross-reference between documents

However, a minor error was encountered when the User Prompts were too broad, causing the model to attempt to pull too much information from the vector database. This revealed a limitation of the model due to the token constraints inherent in the GPT-4 architecture. The RAG solution requires more tokens than a standard LLM query because the content pulled from the vector database is also tokenized and combined with the user query before being processed by the LLM. When the user query is very broad, the amount of information extracted from the registry can exceed the model's token limit. For instance, GPT-4 has a token limit of 8192 tokens per message. During testing, I encountered an error when the data extracted from the documents exceeded 65536 characters, equivalent to the model's token handling capacity. The model attempted to work around this issue by truncating the input text or its response to fit within the token limit, which means cutting off part of the text, this can be seen in figure 5.11. This truncation can possibly result in incomplete information.

This token limitation is a significant factor that prevented the execution of the third and fourth experiments. This token limitation will be further elaborated on in the following section, which is dedicated to a conceptual design where models have surpassed these token constraints.



Gain insights into your own data source. Your data is stored securely in your Azure subscription. Learn more about how your data is protected.

Data source:	Search Resource:
Upload Files	fssearchservice1
Index:	Chunk Size:
testa5	1024

Advanced settings

Your data was connected with the following warnings
- Truncated extracted text to '65536' characters. (1 item(s) impacted)

Figure 5.11: Figure illustrating the retrieval augmentation process exceeding the token limit of GPT-4

Tom frequently highlighted the importance of QA and control of AI responses. LLMs can sometimes provide inaccurate answers in a very convincing and trustworthy manner. This brings up important discussions about QA and explainable AI, which are crucial aspects of the AI Act and will be further discussed as part of the upcoming discussion chapter 7.

Chapter 6

Conceptual Design

This chapter will explore the potential for further optimization of code generation by combining the technologies from both experiments discussed in the Analysis chapter, leading to the development of a refined conceptual design.

During the analysis and experimentation phase, I realized there are certain capabilities that should become possible in the upcoming years of AI development in the DevOps environment. Current models, including advanced ones like Ollama3, show great promise in generating code from documentation but have significant limitations when it comes to directly interacting with and modifying codebases. Existing RAG architectures can only edit documents by adding or removing them but lack the capability to dynamically modify the document's content through prompt engineering.

The conceptual design I envision involves a dynamic RAG solution with a vector database containing the entire codebases of existing services. New repositories will be generated and stored either locally or within a cloud, depending on the user's IDE support. This design ensures that changes can be made in isolation before being merged, similar to the current development process of branching.

6.1 Integration into IDEs

To enhance usability, the RAG solution will be integrated into the user's IDE of choice as an extension, similar to GitHub Copilot and Codey. This extension will allow users to:

- Choose from a variety of LLMs via the interface of their IDE extension.
- Branch out repositories for editing and make the changes the user requests through prompts.

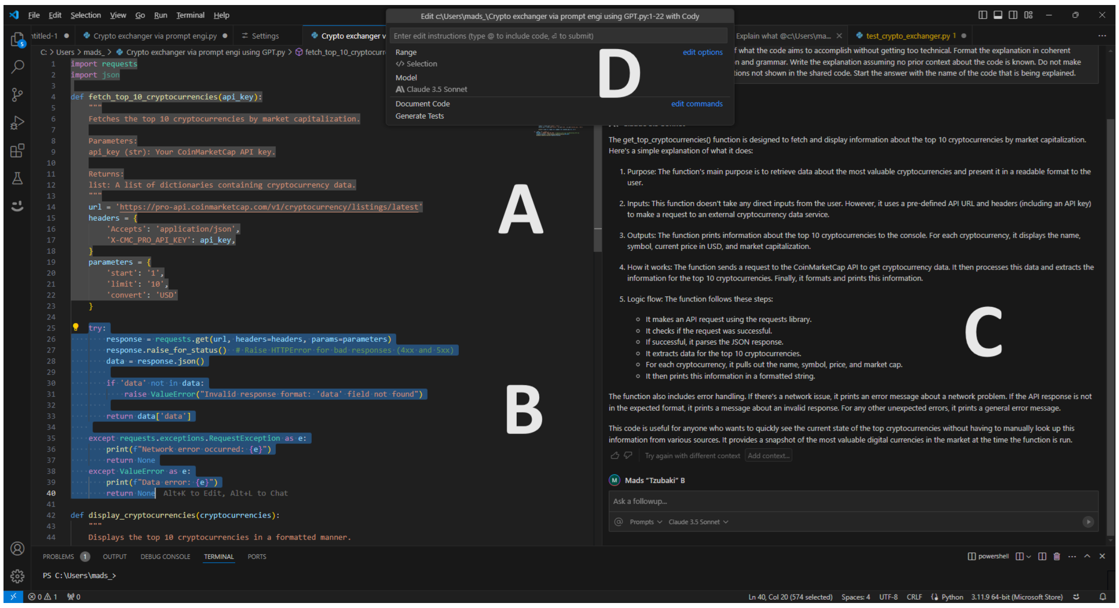
- See immediate feedback within their IDE along with a generated response from the LLM documenting all changes made.
- Highlight changes within the IDE so the user can easily navigate, test, and evaluate modifications to the codebase/repository.

6.1.1 Localized Editing in Comparison to Repository-level Editing

In the figure 6.1, you will see an illustration of my IDE showcasing two different code blocks, Code Block A and Code Block B. The figure also includes the extension interface that allows you to interact with LLMs such as Llama3, GPT, Gemini, or whichever LLM you prefer. This figure and method of creating software are derived from my first experiment in the analysis section 5.6.

The key elements in this figure are Code Block A and Code Block B, which are interdependent. Code Block A defines an object that is called by Code Block B, meaning that any changes made to Code Block A will directly affect the logic of Code Block B. In traditional localized editing of code, tools like GitHub Copilot and similar IDE extensions focus on assisting the user with specific code segments in isolation. While this is useful, it falls short when dealing with repository-level complexities where code blocks across different parts of the project are interconnected.

The sections marked with C and D in the figure represent the current way of interacting with LLMs through your IDE via extensions. My conceptual design aims to improve this process without altering the user experience. The interface remains familiar, but the functionality is vastly enhanced. By integrating repository-level editing capabilities, the LLM can automatically recognize the interdependencies between Code Block A and Code Block B. This is made possible by the RAG solution built on top, which has the capability to search through the entire repository and identify affected code blocks. As a result, any changes made are propagated throughout the relevant parts of the codebase, maintaining consistency and preventing logic errors.



File Edit Selection View Go Run Terminal Help
Edit c:\Users\mads_\Crypto exchanger via prompt engi using GPT.py:1-22 with Cody
Enter edit instructions (type @ to include code, ⏎ to submit)
Range
Selection
Model
Claude 3.5 Sonnet
Document Code
Generate Tests
edit options
edit commands
D
A
B
C

```python
import requests
import json

def fetch_top_10_cryptocurrencies(api_key):
    """
    Fetches the top 10 cryptocurrencies by market capitalization.

    Parameters:
    api_key (str): Your CoinMarketCap API key.

    Returns:
    list: A list of dictionaries containing cryptocurrency data.
    """
    url = 'https://pro-api.coinmarketcap.com/v1/cryptocurrency/listings/latest'
    headers = {
        'Accepts': 'application/json',
        'X-CMC_PRO_API_KEY': api_key,
    }
    parameters = {
        'start': '1',
        'limit': '10',
        'convert': 'USD'
    }

    try:
        response = requests.get(url, headers=headers, params=parameters)
        response.raise_for_status()  # Raise HTTPError for bad responses (4xx and 5xx)
        data = response.json()

        if 'data' not in data:
            raise ValueError("Invalid response format: 'data' field not found")

        return data['data']

    except requests.exceptions.RequestException as e:
        print(f"Network error occurred: {e}")
        return None
    except ValueError as e:
        print(f"Data error: {e}")
        return None  Alt+K to Edit, Alt+L to Chat

def display_cryptocurrencies(cryptocurrencies):
    """
    Displays the top 10 cryptocurrencies in a formatted manner.
```

```
PS C:\Users\mads_>
```

f what the code aims to accomplish without getting too technical. Format the explanation in coherent
n and grammar. Write the explanation assuming no prior context about the code is known. Do not make
tions not shown in the shared code. Start the answer with the name of the code that is being explained.

The get_top_cryptocurrencies() function is designed to fetch and display information about the top 10 cryptocurrencies by market capitalization. Here's a simple explanation of what it does:

1. Purpose: The function's main purpose is to retrieve data about the most valuable cryptocurrencies and present it in a readable format to the user.
2. Inputs: This function doesn't take any direct inputs from the user. However, it uses a pre-defined API URL and headers (including an API key) to make a request to an external cryptocurrency data service.
3. Outputs: The function prints information about the top 10 cryptocurrencies to the console. For each cryptocurrency, it displays the name, symbol, current price in USD, and market capitalization.
4. How it works: The function sends a request to the CoinMarketCap API to get cryptocurrency data. It then processes this data and extracts the information for the top 10 cryptocurrencies. Finally, it formats and prints this information.
5. Logic flow: The function follows these steps:
   - It makes an API request using the requests library.
   - It checks if the request was successful.
   - If successful, it parses the JSON response.
   - It extracts data for the top 10 cryptocurrencies.
   - For each cryptocurrency, it pulls out the name, symbol, price, and market cap.
   - It then prints this information in a formatted string.

The function also includes error handling. If there's a network issue, it prints an error message about a network problem. If the API response is not in the expected format, it prints a message about an invalid response. For any other unexpected errors, it prints a general error message.

This code is useful for anyone who wants to quickly see the current state of the top cryptocurrencies without having to manually look up this information from various sources. It provides a snapshot of the most valuable digital currencies in the market at the time the function is run.

Mads "Tzubaki" B

Ln 40, Col 20 (574 selected) Spaces: 4 UTF-8 CRLF Python 3.11.9 64-bit (Microsoft Store)

Figure 6.1: Illustration of the current usage of code assistance tools in IDEs, highlighting the interdependent code issue. This figure is sourced from the code generation experiment I conducted as part of my analysis 5.6

With my conceptual design, the end user will not need to adapt to new methods or tools. Instead, they will benefit from the ability to have the LLM operate on a larger scale, managing the entire repository rather than just localized code blocks. The RAG solution's ability to search and understand the interdependencies within the repository ensures that all affected parts of the code are taken into consideration when applying changes. This leads to more robust, error-free code and a more efficient development process overall.

6.2 Dynamic Vector Database and Modification Capabilities

In this design, the vector database will house the entire codebase of one or more services. Using prompt engineering, new features can be added to an existing codebase by generating a new repository that mirrors the one in the vector database. The model will need to understand the requirements for the new feature and integrate the feature without disrupting the existing logic or functionality on all interdependent code. It will continuously update and modify the mirrored repository to ensure the codebase evolves in line with project needs, and run tests to evaluate the impact of modifications on the new repository. Finally, the software developer can review and merge changes from the mirrored repository into the ´master´ or ´main´ repository similar to branching in. The newly updated repository can be added for re-indexing for continuous development.

This capability makes the solution even more dynamic by creating an intuitive flow between building, coding, implementation, testing, deployment, and maintenance/operation.

6.3 Use Case 1: Adding New Code to Existing Codebases

Imagine a scenario where a software engineer needs to patch and add a few lines of code to a codebase already in production. This addition could potentially create conflicts or errors, affecting a microservice called by an API or causing non-syntax-related issues due to interactions with existing code. These errors can happen in large codebases where code is interdependent in different parts of the repository.

In this dynamic RAG solution, the model can be instructed to add a new feature, comprehend its functionality, and integrate data from other code blocks. The model will integrate the new feature seamlessly, ensure the overall integrity of the codebase, and run tests in the mirrored repository to check for conflicts and errors. The solution will provide documentation detailing the changes made, their impact on other code blocks, and how conflicts were prevented, allowing the software developer to review and merge the changes into the primary vector database.

6.4 Use Case 2: Generating New Services Based on Existing Codebases

Before my research question was generated, I had a preliminary interview with Senior Developer Regnar Vedsted from Lego, as mentioned at the beginning of the methodology chapter 2. His idea or request heavily inspired this conceptual design. Lego has countless services, websites, games, platforms, and webshops within their fleet of services. There is significant overlap in terms of fonts, layout, transitions, user-experience interactions, images of Lego blocks, and common use of HTML, CSS, and JavaScript code.

Regnar's idea was to have all these services available to an LLM, either in a RAG constellation or by training an LLM, capable of creating brand-new services based on all their previously created services. The model would learn from all the codebases and repositories created by Lego and generate new services that adhere to their organizational standards together with code documentation and requirement specifications of the new service they envisioned. This approach would enable Lego to create new services built in the exact same way as their existing ones, significantly reducing the time and effort required to maintain the "Lego Vibe" across all their platforms.

For example, if Lego has 18 entire codebases/repositories of their product line within a RAG, it should be able to generate the draft of number 19 seamlessly. This would greatly ease the workload for Regnar and his team, saving a substantial amount of money and hours spent on repetitive building as part of their DevOps environment.

6.5 Process Description

The process outlined here provides a dynamic and intelligent approach for integrating new features into an existing software repository. Use Case 1 is described and illustrated in the accompanying figure 6.2. Unlike traditional code-generation tools like GitHub Copilot, which operate on a more localized level, this solution operates at the repository level, ensuring all interdependent parts of the code are updated in a cohesive manner. This ensures that changes are made without compromising the overall functionality and logic of the software.

The architecture of this solution is structured into two layers, each requiring user interaction. The first layer focuses on setting up the RAG system and includes the first two steps of the process - (labeled 'A' on figure 6.2). The second layer involves interacting with the system during the remaining steps of the process, starting with the query stage where the user performs prompt engineering - (labeled 'B' on figure 6.2).

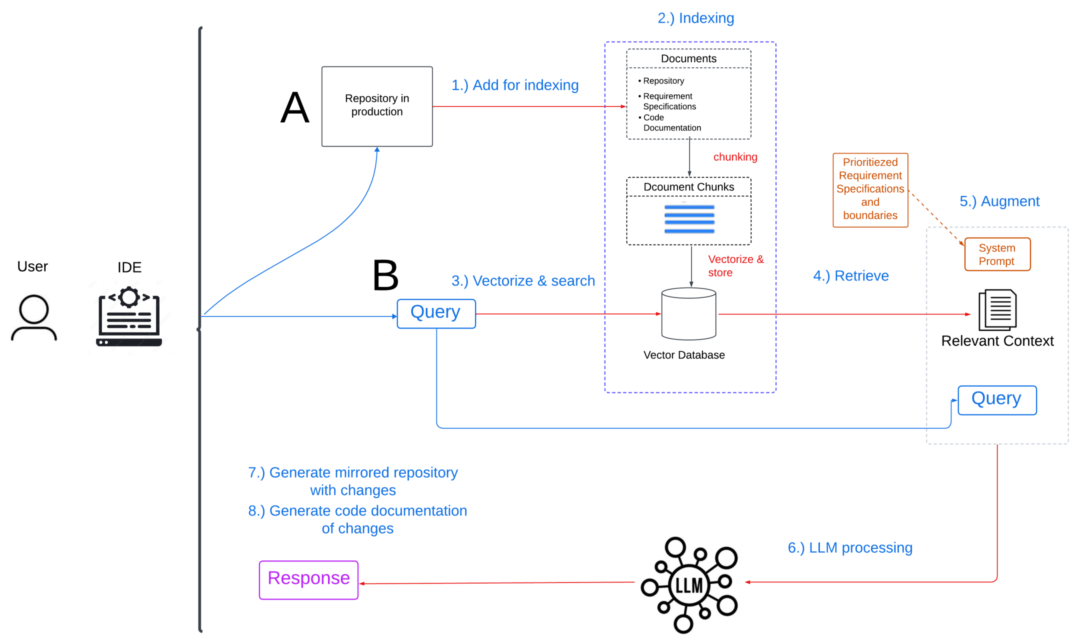


Figure 6.2: Conceptual draft of the dynamic RAG architecture - created using Lucidchart [40]

6.5.1 Step 1: Add for Indexing

In the first step, the user begins by adding the target repository, which is an existing software project in production, to the system for indexing (labeled 'A' on figure 6.2). This repository serves as the foundation on which new features will be built. The purpose of this step is to prepare the repository for subsequent processing by the RAG system, ensuring that all existing code is available for analysis and modification.

6.5.2 Step 2: Indexing

Once the repository is added, it undergoes an indexing process. This process is divided into three sub-steps:

- **Chunking:** The repository's content is divided into smaller, manageable chunks. This allows the system to process and analyze the code more effectively.
- **Vectorizing:** Each chunk of code is transformed into vector representations. This conversion enables efficient semantic searches and comparisons later in the process.
- **Storing:** The vectorized chunks are stored within a vector database, forming the core data that the RAG system will reference throughout the process.

6.5.3 Step 3: Query, Vectorize, and Search

In this step, the user inputs a descriptive requirement specification for the new feature they wish to implement (labeled 'B' on figure 6.2). Alongside this, the user can provide necessary code documentation related to the new feature. The system takes this user input and vectorizes it, allowing for a semantic search through the indexed database. This search identifies code that is relevant to the new feature, ensuring that all necessary dependencies are considered.

6.5.4 Step 4: Retrieve

The system retrieves the semantically relevant context from the vector database. This context includes code snippets, functions, classes, or any other components that are related to the new feature. Simultaneously, the user's prompt is prepared for the next stage in the process, setting the stage for augmentation.

6.5.5 Step 5: Augment

In this step, the system assembles a comprehensive prompt for the LLM. This prompt includes:

- **System prompt:** This ensures that the generated code adheres to specific requirement specifications and boundaries.
- **Relevant context:** Retrieved in the previous step, this ensures that the LLM has all the necessary information about the existing codebase.
- **User query:** The original input provided by the user is included to guide the LLM in generating the desired feature.

The combined prompt is then prepared for processing by the LLM.

6.5.6 Step 6: LLM Processing

The prepared prompt is sent to the LLM for processing. The system allows for flexibility in choosing the LLM, with integration options similar to the Codey Extension used in earlier experiments. The LLM processes the prompt, generating code that reflects the new feature while considering the existing codebase.

6.5.7 Step 7: Generate Mirrored Repository with Changes

In this step, a new mirrored repository is generated. This process is similar to creating a new branch from the main branch. The generated repository incorporates the new feature and reflects all the necessary changes across interdependent parts of the code. Unlike localized code generation, this approach ensures that the entire codebase is updated to maintain functionality and logic. The user can run tests, debugging, and other post-processing tasks using standard IDE tools.

6.5.8 Step 8: Generate Code Documentation of Changes

After generating the new code, the system automatically creates comprehensive documentation detailing all the changes made. This documentation is written in natural language and sent back to the user for review. It serves as a record of what was changed and why, aiding in future maintenance and understanding of the code.

6.5.9 Step 9: Merge with the Repository

The final step involves merging the newly generated repository with the original repository. This process is similar to a "branching in" operation. The user reviews the changes and, once satisfied, merges them into the main branch. If the user plans to iterate on the changes further, they can re-index the updated repository, allowing for multiple iterations of the process. This ensures safe and controlled integration of new features before the software is released back into production.

This process description provides a detailed overview of each step in the Dynamic

RAG solution, highlighting how it enables repository-wide code generation and integration, thereby supporting complex software development tasks with greater accuracy and efficiency.

6.6 Efficiency, Optimization, and Reversion

The RAG solution will also feature a "Modification" element that allows it to edit the mirrored repository content and revert those changes immediately if the software engineer is not satisfied. This feature ensures flexibility and maintains code integrity while allowing rapid adjustments. The model should be highly efficient in code management. It should assist software engineers in:

- Adding new features more effectively than current methods.
  (rationale of Use Case 1 6.3 and Experiment Three 5.4.3)
- Entire repository-level editing.
  (rationale of Use case 2 6.4 and Experiment Four 5.4.4)
- Revoking changes

By optimizing these practices, the advanced RAG solution can significantly reduce the time and effort engineers spend on these tasks, enhancing productivity and efficiency across the DevOps pipeline.

6.7 Current Limitations and Future Improvements

Achieving this conceptual design with current technology is challenging due to several limitations. One major constraint is the token limit of existing models. With limits of 8k, 16k, or even 32k tokens, these models are not capable of handling entire codebases as context for modifications. For instance, a codebase consisting of 1000 lines of code, along with necessary standards and organizational requirements, would exceed the token capacity of current models. This limitation prevents the model from taking all 1000 lines of code and relevant standards as a reference when making modifications, significantly hindering its ability to provide comprehensive and context-aware updates. To manage within these limits, the model might perform error-handling maneuvers such as truncating the output to fit within the token limit, which means cutting off part of the text. This truncation results in incomplete information. 5.11.

Additionally, issues like hallucination and generic answers pose significant risks when generating code. Even minor errors can create vulnerability gaps within the code, potentially leading to malicious attacks on models and AI solutions that have not been seen before. Models should strive to avoid providing generic information and instead make use of plugins and other methods to obtain "fresh data." An example is Microsoft Semantic Kernel, which extracts real-time data by executing plugins externally and retrieves this information to base its answers on. This approach can help avoid hallucinations and prevent the model from filling in the blanks with generic answers.

Furthermore, I believe we will see fewer very large generic LLMs similar to GPT and Gemini, which attempt to cover as many use cases as possible. Instead, we will likely see smaller and more specific LLMs trained for particular tasks or processes. Consulting engineering companies might develop smaller models and solutions tailored to sensitive organizational data provided by their customers. Currently, models are trained on freely available data. For example, if we want a model extremely proficient in COBOL, we could greatly benefit from organizational data that IBM could provide from their mainframe services. However, IBM has no interest in sharing this data with an LLM due to rules, regulations, and competitive reasons. This is exactly what Tom from UFST demands of an LLM: the ability to train it on his own organizational data without the risk of sharing this data with OpenAI or Google. He envisions being able to modify the models in a manner similar to ordering a website from an engineer but then inserting his own data after the basic structure has been created. The risk of sharing data with large companies and models is too high and violates several GDPR-related standards.

To realize this conceptual design, future iterations of language models need to support much larger token limits and ensure accurate, fresh data retrieval. By overcoming these challenges, future RAG solutions could fully leverage the potential to dynamically interact with and modify codebases, providing a powerful tool for software development and maintenance.

Chapter 7

Discussion

This chapter will discuss and explore the implications and results of the information gathered throughout the project, providing a comprehensive analysis of the findings and their potential impact.

7.1 Reflections

As I reflect on the journey of completing this thesis, I feel a deep sense of satisfaction with the knowledge I've gained and the insights I've developed. The project has been a fulfilling intellectual experience, but it also highlighted a recurring challenge that I face in my work: my tendency to broaden the scope as new information and ideas emerge. While this often enriches the exploration, it can also lead to a loss of focus. Initially, my research was concentrated on optimizing the coding stage of the DevOps pipeline. However, as the project progressed, and through my interactions with experienced IT professionals from leading organizations, I was inspired to expand my focus. I began to look at the entire DevOps process with an eye toward optimization, rather than limiting my research to just the coding stage.

The collaboration with my assisting supervisor, Peter Anglov, and the industry professionals he introduced me to, including Jan Cordtz & Rasmus from Microsoft, and Tom Willy Nielsen from UFST, proved to be invaluable. These interactions provided me with practical insights into the challenges and processes faced by some of the largest IT organizations. This experience was particularly meaningful to me, as someone who has worked as an Operations Engineer at Netcompany for nearly two years. The opportunity to engage with these professionals gave me a preview of the challenges and opportunities I will encounter in my future career.

Regarding my inclination to expand the scope of my projects, I've come to view this tendency not as a flaw but as a potential strength. Rather than trying to suppress this inclination, I'm learning to embrace it and see it as something that could be valuable in my professional career. In a field as dynamic as IT, the ability to adapt and explore new avenues is crucial.

One of the most significant lessons I learned during this process was the importance of trust and authenticity in content generated by generative AI. This revelation came during a conversation with Rasmus from Microsoft, who challenged the common perception of AI as "intelligent." He explained that AI-driven technologies, including LLMs, lack true understanding. They don't "know" things in the human sense; instead, they process data, recognize patterns, and produce outputs based on their training. For example, when an AI is asked to calculate that 2 plus 2 equals 4, it doesn't truly understand the arithmetic; it simply identifies that 4 is the correct response based on prior data. In practice, an LLM might initiate a Python script to perform the calculation or rely on a plugin to obtain the answer, but it cannot calculate in the intuitive way that a human does.

This conversation profoundly influenced my perspective. It led me to place greater value on AI architectures that prioritize handling fresh data over those that rely heavily on the general knowledge base of LLMs. I began to see LLMs less as autonomous sources of knowledge and more as tools—gateways to optimize processes, similar to plugins or extensions. This shift in perspective significantly impacted my approach to the research.

Reflecting on my initial experiments, I now realize that while they demonstrated the potential for LLMs to assist in code generation, they also introduced new challenges. For instance, instead of having a software engineer spend time understanding requirement specifications and building a codebase manually, they now need to craft precise prompts and verify the accuracy of the generated code. While this approach may be faster and more convenient, the broader implications and potential consequences are still unclear. In essence, we've replaced one work process with another, placing a higher value on prompt engineering and validation.

With the knowledge I have now, I would have approached my research question differently. I would have placed more emphasis on the issues of trust and authenticity in AI-generated content, particularly in reducing reliance on the general knowledge base of LLMs. This realization led me to draft a conceptual design that shifts away from using LLMs solely for their general knowledge. Instead, I would focus on a RAG solution, where the LLM generates code based on specific data provided to it by the user. This approach offers better control over the accuracy and validity of the generated content.

In hindsight, this experience has taught me the importance of continuously questioning and refining my approach, especially in a field as rapidly evolving as AI. While I am pleased with the outcomes of my thesis, I now see areas where I could have delved deeper and perhaps uncovered even more valuable insights. Moving forward, I aim to carry these lessons into my professional career, where I hope to continue exploring the intersection of AI and DevOps with a more refined and focused approach.

7.2 The EU AI Act, Explainable AI, and Quality Assurance

The European Union's Artificial Intelligence Act (EU AI Act), introduced on July 12, 2024, marks the first comprehensive regulatory framework for AI systems across the EU [56]. This legislation will be fully implemented across all EU Member States starting on August 1, 2024, with most provisions taking effect on August 2, 2026. It follows a risk-based approach similar to the GDPR. It focuses on different phases of the AI lifecycle, imposing specific rules and obligations on organizations and stakeholders responsible for developing, deploying, and providing AI services.

The EU AI Act primarily impacts providers, deployers, and distributors of AI systems, rather than end users. For instance, considering Microsoft Copilot, the Act directly affects OpenAI as the provider of AI models. Microsoft, which utilizes these services and models provided by OpenAI to create their Copilot tools, is impacted as the second link in the chain. Organizations that subscribe to Microsoft's services and create RAG solutions within the Azure Cloud, similar to what I did in experiment two, are also affected 5.7. However, if OpenAI and Microsoft take appropriate precautions in relation to the AI Act, the impact on the remaining parts of the supply chain diminishes progressively.

Given that the Act was released relatively close to the submission of my master's thesis, I did not fully consider its implications for the first experiment I conducted on AI-driven code generation. However, the concept of Explainable AI (XAI) might play a significant role in this context.

Although the Act does not explicitly mention XAI as a distinct concept, its principles are embedded within the framework, particularly regarding transparency, accountability, and fairness in AI systems. The Act mandates that users should have access to information about the inner workings of an AI system, including the logic behind its decisions. XAI is designed to make AI decisions understandable and interpretable for users. This can be particularly valuable in QA for the content generated by LLMs. While XAI does not completely eliminate the "black box" phenomenon, it offers users better explanations of why outputs appear as they do.

A critical takeaway from the AI Act is its emphasis on enforcement at the earliest stages of AI model development, including the idea and training phases [25]. The requirement for explainability allows human operators to gain more insight into the neural network's inner workings and intervene in decision-making processes if necessary. This could be instrumental in preventing AI models from making unfair or discriminatory decisions. For example, there were instances where Google Bard generated racist content, including deliberately generating images exclusively of people from Black, Hispanic, and Asian backgrounds while avoiding creating images of white individuals [35]. This bias was likely the result of an attempt to prevent bias, which backfired and instead introduced bias in the opposite direction. Such incidents underscore the importance of human intervention during AI model training to prevent biases but also highlight the risks of inadvertently skewing the model's judgments of what is fair and unbiased.

As the first official regulatory framework of its kind, the AI Act is likely to undergo many changes and iterations in the coming years. From an end-user perspective, controlling the inner workings of AI models can be best managed through prompt engineering, which helps reduce ambiguity and provides better control over the outputs produced within the neural network's "black box."

In terms of QA and QC, it will be crucial to educate the next generation of software engineers to approach AI-generated content with caution. They should not rely solely on the outputs of LLMs without verification. New QA and QC methodologies and tools specifically addressing the challenges posed by AI-generated content will be essential, particularly in niche fields like code generation, similar to the conceptual design I have discussed.

LLMs have the capability to generate responses based on a vast general knowledge base, but the accuracy and reliability of this information are not always guaranteed. This makes fact-checking a vital, albeit time-consuming, part of the process. While LLMs have made information retrieval more convenient, they introduce a trade-off in trust and authenticity, necessitating extensive verification to ensure outputs are accurate.

RAG solutions, on the other hand, leverage the capabilities of LLMs to work with "fresh" data that users provide, ensuring a higher degree of accuracy. This approach significantly reduces the need for extensive fact-checking, as the model generates responses based on trusted, up-to-date data. However, it does not completely eliminate the need for verification but does make the process more reliable and efficient.

The reliability and trustworthiness of outputs tend to decrease as queries become more specific and delve into deeper, narrower areas of knowledge. As the model moves away from general information and into specialized domains, the risk of inaccuracies increases. This reinforces the ongoing need for rigorous fact-checking, accuracy, and verification to maintain trust in the information provided.

Jan Cordtz introduced me to Microsoft Semantic Kernel, a tool that plays a crucial role in helping organizations fetch "fresh data" from various Office365 services as part of the generative process. For example, if a user needs to create a weekly presentation with updated financial figures, Microsoft Semantic Kernel would handle the process of fetching this fresh data from an Excel spreadsheet, dramatically reducing the complexity and time required to prepare the presentation. The Semantic Kernel operates by making API calls across the Office365 platform on behalf of the user, ensuring that the information retrieved is accurate and up-to-date. According to Jan, one of the key challenges lies in enabling the AI model to determine when to make an API call for fresh data and when to rely on general information.

This sophisticated approach to integrating AI with fresh data sources exemplifies the direction in which QA and control must evolve. Ensuring that AI-generated content is both accurate and reliable will be an ongoing challenge, particularly as regulations like the EU AI Act become more entrenched and potentially more stringent.

7.2.1 Model Feedback Loop

One major concern I have related to the training of new AI models is the potential feedback loop created by feeding AI-generated content back into the training datasets of newer models. Since the release of GPT-3 in November 2022, there has been a surge in AI-generated content, raising questions about the long-term impact on future AI training. When AI-generated content is used to train new models, which then produce more content, I believe a feedback loop can emerge with potentially detrimental effects. This is similar to the feedback loop experienced when a microphone is placed too close to a speaker, resulting in an unpleasant and often disruptive squeal. My concern is that this could lead to a situation where it becomes increasingly difficult to distinguish between human-made content and AI-generated content, thus complicating quality assurance efforts on a macro level.

To mitigate this risk, I believe it is essential to maintain strict, supervised datasets of human-created content, similar to the foundational training sets used for GPT-3 before its release. My hypothesis is that the ideal time to create such models was before the widespread proliferation of AI-generated content. At that point, quality control over distinguishing computer-generated content from genuinely human-authored material was more manageable. As the volume of AI-generated content continues to grow, ensuring the authenticity and reliability of future AI models will become an even greater challenge.

In my conceptual design, where entire repositories are stored within a data registry, I can foresee potential conflicts when generating new services. For example, if a specific method is commonly used across multiple repositories, the AI might arbitrarily incorporate it into new software. This could happen not because it's the most suitable for the new requirements, but because its frequent usage influences the model's outputs. As a result, new software might rely on methods, libraries, and APIs that aren't ideal for their intended tasks. When past outputs heavily influence the AI's decisions for new projects, a feedback loop could develop. This would compromise the integrity and functionality of the resulting codebase. These risks highlight the critical need for supervision, the application of XAI, and strict QA to ensure that outputs align accurately with current and specific development objectives.

7.3 The Social Impact of Integrating LLMs into DevOps

The reliance on LLMs and AI technologies is not only transforming the technical tasks that IT professionals perform but also altering the industry's view of what skills are considered valuable. Jan Cordtz supports this assertion by predicting a rise in machine-generated code. This evolving landscape suggests a future where the emphasis on skills may shift towards prompt engineering potentially becoming a more sought-after competency than traditional coding skills. Researching this shift is crucial for several reasons: it informs curriculum development for educational institutions preparing the next generation of IT professionals. Incorporating prompt engineering into engineering university curriculums would not only align educational offerings with industry needs but also equip future professionals with the skills necessary to thrive in an AI-integrated workforce. Given the projected importance of prompt engineering, as highlighted by both conducted experiments, embedding this skill set into the academic journey of engineering students is crucial 5.6 5.7. This approach would ensure they are well-prepared for the complexities of working with AI technologies. This proactive approach in education would facilitate a smoother transition for graduates entering the IT industry, where the ability to effectively utilize LLMs and other AI tools is becoming increasingly important.

To illustrate the shifting skill requirements in software engineering, I reflect on my own academic experience. During my initial two semesters at university, the assessment in our object-oriented programming course involved demonstrating coding proficiency by manually writing code on a blackboard. A particular challenge I encountered was to develop a method for a nested for loop, a task designed to test our grasp of basic programming concepts. At the time, this method assessed fundamental skills effectively. However, in the context of today's advanced IDEs, such manual coding exercises can appear somewhat antiquated. IDEs often feature autocomplete functions that simplify coding tasks significantly. Furthermore, during a recent experiment involving code generation with an LLM like the Llama model, I found that these advanced AI tools could generate the required code based solely on a description in human language. This experience underscores the evolving nature of what constitutes essential skills in software engineering. It also highlights the growing need for educational programs to adapt, teaching students how to leverage AI technologies like prompt engineering effectively, thereby ensuring they remain competitive and proficient in a rapidly transforming industry.

Chapter 8

Conclusion

This chapter will conclude the research by summarizing how the research question was addressed, outlining the key findings, and reflecting on the information gathered throughout the study.

This thesis set out to explore the optimization of the DevOps pipeline, specifically focusing on the coding stage, by leveraging Prompt Engineering and Large Language Models to generate code from descriptive documentation. The research was driven by an increasing industry trend of integrating AI into coding processes, as evidenced by insights gathered from preliminary interviews with leading IT companies, including Microsoft. These discussions underscored the growing importance of prompt engineering and precise documentation as essential skills for software developers in the evolving landscape of AI-assisted coding. Based on these industry insights, the following research question was formulated:

How can generative AI and prompt engineering improve code development and maintenance in a DevOps environment?

To help answer this primary question, three underlying questions were developed and explored:

1. **How is the growing use of LLMs in IT expected to impact the importance of prompt engineering over traditional coding skills?**

   The growing integration of LLMs in software development is transforming the role of developers, with prompt engineering becoming increasingly essential. As Jan Cordtz estimates, the proportion of machine-generated code is set to increase from 60% to 80%, highlighting the need for developers to shift focus from manual coding to crafting precise and effective prompts. This shift was evident in experiments with LLMs like GPT and Llama3, where the quality of generated code relied significantly on the clarity and structure of the prompts. While traditional coding skills remain valuable, they may increasingly be applied in new ways, such as performing quality control on the outputs of machine-generated code.

2. **How can we develop software explicitly using natural language and prompt engineering in low-code and no-code environments? (Follow-up: How can we verify the quality of code generated by machines?)**

   This question was investigated through the first experiment, where LLMs were tasked with generating software code from natural language prompts. Both GPT and Llama3 were able to produce functional code, but their effectiveness was limited to specific, localized sections of the codebase. This revealed a key limitation: while natural language prompts are powerful, current models struggle to manage interdependencies across larger repositories. One possible factor is the output capacity of the models.

   As for the follow-up question regarding code verification, the quality of machine-generated code will rely primarily on human supervision and code review. Developers will play a critical role in examining the generated code to ensure it follows best practices, is maintainable, and meets the necessary quality standards. In the future, we can expect a rise in advanced code review tools that will assist developers by automating parts of the review process, such as code analysis and testing. In addition to human oversight, machine-generated code should undergo rigorous automated testing, including unit and integration tests, to ensure it meets functional and performance requirements.

3. **How can a Retrieval-Augmented Generation (RAG) solution be developed to streamline the comprehension and analysis of large technical documents in a DevOps environment?**

   This question was directly addressed by the second experiment, which was conducted in collaboration with Microsoft and Udviklings- og Forenklingsstyrelsen. By developing a RAG solution in Azure, the project efficiently indexed large, complex documents, significantly improving search and retrieval processes. This experiment demonstrated how AI can enhance the comprehension and analysis of interconnected information, drastically reducing the time required for manual searches and improving overall workflow efficiency within the DevOps pipeline.

Building on the insights gained from these experiments, a conceptual design was developed to further optimize the coding stage in DevOps. The design combines the strengths of RAG with the code-generation capabilities of LLMs to create a repository-level editing tool, addressing the need for LLMs to understand and edit entire codebases rather than just isolated segments. However, a critical challenge remains: current models are limited by their token capacity, typically ranging between 8,000 and 16,000 tokens. A full repository often exceeds this token limit, making it difficult for the model to process all necessary context at once, thereby limiting its ability to handle large, complex codebases in a single iteration.

By leveraging the RAG component, the conceptual design aims to provide better context for the model to understand which code blocks will be affected by a given change due to the semantic relation between the change and the impacted parts. This additional context would ideally improve the model's efficiency in applying changes across all affected areas of the repository, rather than addressing one isolated segment at a time. While this may not be the perfect solution, it represents a step in the right direction toward achieving full repository coding assistance

In conclusion, this thesis proposes a promising conceptual design for enhancing the coding stage in DevOps by integrating generative AI and prompt engineering. While the research has identified both potential benefits and limitations of current AI technologies, it also emphasizes the importance of continued innovation to fully realize this potential. The evolving role of AI in software development will require a careful balance between automation and human expertise, ultimately positioning AI as not just an assisting tool, but a future colleague of software developers. This research has taken important steps toward answering both the primary research question and the three underlying questions, demonstrating how AI can contribute to code development and maintenance in DevOps, while also highlighting areas where further progress is necessary.

Bibliography

[1] *About Google DeepMind*. Google DeepMind, 2015. URL: https://deepmind.google/about/ (visited on 04/05/2024).

[2] Reed Albergotti. *OpenAI has received just a fraction of Microsoft's $10 billion investment | Semafor*. Semafor.com, Nov. 2023. URL: https://www.semafor.com/article/11/18/2023/openai-has-received-just-a-fraction-of-microsofts-10-billion-investment (visited on 02/06/2024).

[3] Microsoft Corporate Blogs. *Microsoft and OpenAI extend partnership - The Official Microsoft Blog*. The Official Microsoft Blog, Jan. 2023. URL: https://blogs.microsoft.com/blog/2023/01/23/microsoftandopenaiextendpartnership/ (visited on 02/06/2024).

[4] Alexander Bogner, Beate Littig, and Wolfgang Menz. "Introduction: Expert Interviews — An Introduction to a New Methodological Debate". In: *Palgrave Macmillan UK eBooks* (Jan. 2009), pp. 1–13. DOI: 10.1057/9780230244276_1. URL: https://link.springer.com/chapter/10.1057/9780230244276_1 (visited on 03/14/2024).

[5] Rodrigo Campos. "Experimental methodology". In: *Fat crystal networks*. CRC Press, 2004, pp. 284–365.

[6] Giuseppe Carleo et al. "Machine learning and the physical sciences". In: *Reviews of Modern Physics* 91 (Dec. 2019). DOI: 10.1103/revmodphys.91.045002. URL: https://journals.aps.org/rmp/abstract/10.1103/RevModPhys.91.045002 (visited on 02/20/2024).

[7] *ChatGPT — Release Notes | OpenAI Help Center*. Openai.com, 2024. URL: https://help.openai.com/en/articles/6825453-chatgpt-release-notes (visited on 02/06/2024).

[8] European Commission. *Commission Implementing Regulation (EU) 2022/1636*. 2022. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32022R1636&qid=1675868137077.

[9] European Commission. *Commission Implementing Regulation (EU) 2022/1637*. 2022. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32022R1637&qid=1675868165905.

[10] European Commission. *Council Directive (EU) 2020/262*. 2022. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex%3A32020L0262.

[11] European Commission. *Excise Movement and Control System*. 2022. URL: https://taxation-customs.ec.europa.eu/taxation/excise-duties/excise-movement-control-system_en.

[12] Octopus Deploy. *Continuous delivery & deployment automation tool for DevOps teams | Octopus Deploy*. Octopus Deploy, 2019. URL: https://octopus.com/ (visited on 04/16/2024).

[13] Presley DevAnnand. *LinkedIn*. Linkedin.com, Mar. 2023. URL: https://www.linkedin.com/pulse/why-prompt-engineering-jobs-next-big-thing-presley-devannand/ (visited on 02/12/2024).

[14] Christof Ebert et al. "DevOps". In: *IEEE Software* 33 (May 2016), pp. 94–100. DOI: 10.1109/ms.2016.68. URL: https://ieeexplore.ieee.org/abstract/document/7458761?casa_token=UmMP8-Vw3rwAAAAA:cGwMIjx5r6RIa_ZJlsmowWYyh57iDrA_1qsOinOaxoibBAWpQuVR7QCl8lRW_ja3mHPVKAM (visited on 02/19/2024).

[15] *Explainable AI | Google Cloud*. Google Cloud, 2024. URL: https://cloud.google.com/explainable-ai (visited on 02/15/2024).

[16] Chaim Gartenberg. *What is a long context window?* Google, Feb. 2024. URL: https://blog.google/technology/ai/long-context-window-ai-models/ (visited on 04/05/2024).

[17] *Gemini - Google DeepMind*. @googledeepmind, 2024. URL: https://deepmind.google/technologies/gemini/#introduction (visited on 04/05/2024).

[18] *Gemini models*. Google AI for Developers, 2024. URL: https://ai.google.dev/models/gemini (visited on 04/16/2024).

[19] *Get started with Gemini Nano on Android (on-device)*. Google AI for Developers, 2024. URL: https://ai.google.dev/tutorials/android_aicore (visited on 04/05/2024).

[20] *Get started with GitLab CI/CD | GitLab*. Gitlab.com, 2024. URL: https://docs.gitlab.com/ee/ci/#the-gitlab-ciyml-file (visited on 04/16/2024).

[21] *Git*. Git-scm.com, 2024. URL: https://git-scm.com/ (visited on 04/12/2024).

[22] *GitHub Copilot · Your AI pair programmer*. GitHub, 2024. URL: https://github.com/features/copilot (visited on 03/05/2024).

[23] *Google AI Foundation models – Google AI*. Google AI, 2024. URL: https://ai.google/discover/foundation-models/ (visited on 04/05/2024).

[24] *Grafana: The open observability platform | Grafana Labs*. Grafana Labs, 2024. URL: https://grafana.com/ (visited on 04/16/2024).

[25] *High-level summary of the AI Act | EU Artificial Intelligence Act*. Artificialintelligenceact.eu, 2024. URL: https://artificialintelligenceact.eu/high-level-summary/ (visited on 08/19/2024).

[26] Michael Hüttermann. *DevOps for Developers*. Google Books, 2012. URL: https://books.google.dk/books?hl=da&lr=&id=JfUAkB8AA7EC&oi=fnd&pg=PR3&dq=DevOps&ots=wpola5znjN&sig=P0i67_L8xi9L59N6j0DNH3HVC60&redir_esc=y#v=onepage&q=DevOps&f=false (visited on 04/12/2024).

[27] *Introducing GPTs*. Openai.com, 2023. URL: https://openai.com/blog/introducing-gpts (visited on 03/25/2024).

[28] *Introduction to the 3270 terminal*. Ibm.com, June 2023. URL: https://www.ibm.com/docs/en/zos-basic-skills?topic=enhanced-introduction-3270-terminal (visited on 08/17/2024).

[29] Kazuki Irie et al. "Language modeling with deep transformers". In: *arXiv preprint arXiv:1905.04226* (2019).

[30] Ramtin Jabbari et al. "What is DevOps?" In: *Proceedings of the Scientific Workshop Proceedings of XP2016 on - XP '16 Workshops* (2016). DOI: 10.1145/2962695.2962707.

[31] *Jenkins*. Jenkins, 2024. URL: https://www.jenkins.io/ (visited on 04/12/2024).

[32] Florian June. *A Brief Introduction to Retrieval Augmented Generation(RAG)*. Medium, Jan. 2024. URL: https://ai.plainenglish.io/a-brief-introduction-to-retrieval-augmented-generation-rag-b7eb70982891 (visited on 07/15/2024).

[33] *JUnit 5*. Junit.org, 2019. URL: https://junit.org/junit5/ (visited on 04/12/2024).

[34] Joe El khoury. *Explained Methodologies and frameworks in Prompt Engineering*. Medium, Oct. 2023. URL: https://medium.com/@jelkhoury880/some-methodologies-in-prompt-engineering-fa1a0e1a9edb (visited on 04/16/2024).

[35] Zoe Kleinman. *Why Google's 'woke' AI problem won't be an easy fix*. Bbc.com, Feb. 2024. URL: https://www.bbc.com/news/technology-68412620 (visited on 08/19/2024).

[36] Julia Km. *What is Azure Pipelines? - Azure Pipelines*. Microsoft.com, Feb. 2024. URL: https://learn.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops (visited on 04/12/2024).

[37] Chuangji Li, Shizhuo Li, and Alan Wang. *Retrieval-Augmented Multi-hop Code Generation with CodeLlama and Unlimiformer*. URL: https://www.shizhuo-profile.com/docs/Retrieval-Augmented%20Multi-hop%20Code%20Generation%20with%20CodeLlama%20and%20Unlimiformer.pdf (visited on 08/21/2024).

[38] Wei Li et al. “Improving Natural Language Capability of Code Large Language Model”. In: *arXiv preprint arXiv:2401.14242* (2024).

[39] Yu Liang et al. “Explaining the black-box model: A survey of local interpretation methods for deep neural networks”. In: *Neurocomputing* 419 (2021), pp. 168–182.

[40] *Lucidchart tool*. Lucidchart.com, 2015. URL: `https://www.lucidchart.com/pages/` (visited on 08/19/2024).

[41] Microsoft. *Din daglige AI-ledsager | Microsoft Copilot*. Microsoft.com, 2024. URL: `https://www.microsoft.com/da-dk/microsoft-copilot/learn?form=MG0AUO&OCID=MG0AUO#faq` (visited on 07/30/2024).

[42] Microsoft. *Microsoft Copilot: Din daglige ledsager med kunstig intelligens*. Microsoft Copilot: Din daglige ledsager med kunstig intelligens, 2024. URL: `https://copilot.microsoft.com/` (visited on 07/30/2024).

[43] Microsoft. *Visual Studio Code*. Visualstudio.com, Nov. 2021. URL: `https://code.visualstudio.com/docs/copilot/overview` (visited on 03/05/2024).

[44] Prakash M Nadkarni, Lucila Ohno-Machado, and Wendy W Chapman. “Natural language processing: an introduction”. In: *Journal of the American Medical Informatics Association* 18.5 (2011), pp. 544–551.

[45] Zhaoyang Niu, Guoqiang Zhong, and Hui Yu. “A review on the attention mechanism of deep learning”. In: *Neurocomputing* 452 (2021), pp. 48–62.

[46] *Observability and IT Management Platform | SolarWinds*. Solarwinds.com, 2023. URL: `https://www.solarwinds.com/` (visited on 04/16/2024).

[47] *Ollama*. ollama, 2024. URL: `https://www.ollama.com/library` (visited on 09/04/2024).

[48] *OpenAI Platform*. Openai.com, 2024. URL: `https://platform.openai.com/docs/assistants/overview?context=with-streaming` (visited on 03/15/2024).

[49] *OpenAI Platform*. Openai.com, 2024. URL: `https://platform.openai.com/tokenizer` (visited on 03/27/2024).

[50] *OpenAI Platform*. Openai.com, 2024. URL: `https://platform.openai.com/docs/models` (visited on 04/02/2024).

[51] Helen Orvaschel. “Structured and semistructured interviews”. In: *Clinician's handbook of child behavioral assessment*. Elsevier, 2006, pp. 159–179.

[52] Athena Ozanich. *The DevOps Pipeline: How It Works and How to Build One*. Hubspot.com, Oct. 2023. URL: `https://blog.hubspot.com/website/devops-pipeline` (visited on 06/13/2024).

[53] Samson Oˇslakov. “Generative language models, AI chatbots, Retrieval augmented generation”. In: (Dec. 2023).

[54] Sophia Lagerkrans Pandey. *Introduction to Semantic Kernel*. Microsoft.com, June 2024. URL: https://learn.microsoft.com/en-us/semantic-kernel/overview/ (visited on 08/17/2024).

[55] Patrick Parker. "Prompt Engineering for identity security professionals". In: (May 2023).

[56] European Parliament. *EU AI Act: first regulation on artificial intelligence*. Europa.eu, June 2023. URL: https://www.europarl.europa.eu/topics/en/article/20230601STO93804/eu-ai-act-first-regulation-on-artificial-intelligence (visited on 02/15/2024).

[57] Mohammed Karimkhan Pathan. *LinkedIn*. Linkedin.com, 2024. URL: https://www.linkedin.com/pulse/zero-shot-one-few-learning-prompt-engineering-pathan/ (visited on 04/16/2024).

[58] Sundar Pichai. *An important next step on our AI journey*. Google, Feb. 2023. URL: https://blog.google/technology/ai/bard-google-ai-search-updates/ (visited on 04/05/2024).

[59] Sundar Pichai. *Introducing Gemini: our largest and most capable AI model*. Google, Dec. 2023. URL: https://blog.google/technology/ai/google-gemini-ai/ (visited on 04/16/2024).

[60] Sundar Pichai. *Our next-generation model: Gemini 1.5*. Google, Feb. 2024. URL: https://blog.google/technology/ai/google-gemini-next-generation-model-february-2024/#sundar-note (visited on 04/16/2024).

[61] Christopher CR Powers David MW Turk. *Machine learning of natural language*. Springer Science & Business Media, 2012.

[62] Sunil Ramlochan. *System Prompts in Large Language Models*. Prompt Engineering, Mar. 2024. URL: https://promptengineering.org/system-prompts-in-large-language-models/ (visited on 04/16/2024).

[63] Michele Salvagno, Fabio Silvio Taccone, and Alberto Giovanni Gerli. "Artificial intelligence hallucinations". In: *Critical Care* 27.1 (2023), p. 180.

[64] Christiane Schmidt. "The analysis of semi-structured interviews". In: *A companion to qualitative research* 253.41 (2004), p. 258.

[65] Selenium. *Selenium*. Selenium, 2024. URL: https://www.selenium.dev/ (visited on 04/12/2024).

[66] M. Shen. *Anomaly Detection Tumours Lecture in ICTE MSc Program*. Presentation in ICTE MSc Program, Aalborg University. Lecture conducted at Aalborg University, Denmark, March 31. 2023.

[67] M. Shen. *Linear Regression and Gradient Descent Lecture in ICTE MSc Program*. Presentation in ICTE MSc Program, Aalborg University. Lecture conducted at Aalborg University, Denmark, February 10. 2023.

[68] M. Shen. *Unsupervised learning and machine learning implementation Lecture in ICTE MSc Program*. Presentation in ICTE MSc Program, Aalborg University. Lecture conducted at Aalborg University, Denmark, April 21. 2023.

[69] M. Shen. *What is Machine Learning? Lecture in ICTE MSc Program*. Presentation in ICTE MSc Program, Aalborg University. Lecture conducted at Aalborg University, Denmark, February 3. 2023.

[70] Tarry Singh. *LinkedIn*. Linkedin.com, Mar. 2023. URL: `https://www.linkedin.com/pulse/impact-large-language-models-future-jobs-tarry-singh/` (visited on 02/06/2024).

[71] Christian von Soest. "Why Do We Speak to Experts? Reviving the Strength of the Expert Interview Method". In: *Perspectives on Politics* 21 (June 2022), pp. 277–287. DOI: `10.1017/s1537592722001116`. URL: `https://www.cambridge.org/core/journals/perspectives-on-politics/article/why-do-we-speak-to-experts-reviving-the-strength-of-the-expert-interview-method/45E710F27CEC6E739B015E10A161E140` (visited on 03/14/2024).

[72] *Sora: Creating video from text*. Openai.com, 2015. URL: `https://openai.com/sora` (visited on 04/05/2024).

[73] *The ATM | IBM*. Ibm.com, 2024. URL: `https://www.ibm.com/history/atm` (visited on 08/17/2024).

[74] *The Size and Quality of a Data Sets in Machine Learning*. Google for Developers, 2022. URL: `https://developers.google.com/machine-learning/data-prep/construct/collect/data-size-quality` (visited on 03/14/2024).

[75] Gary A Troia. "Phonological awareness intervention research: A critical review of the experimental methodology". In: *Reading Research Quarterly* 34.1 (1999), pp. 28–52.

[76] Ashish Vaswani et al. *Attention Is All You Need*. arXiv.org, 2017. URL: `https://arxiv.org/abs/1706.03762` (visited on 03/25/2024).

[77] Chenguang Wang, Mu Li, and Alexander J Smola. "Language models with transformers". In: *arXiv preprint arXiv:1904.09408* (2019).

[78] Shangwen Wang et al. "Natural Language to Code: How Far Are We?" In: *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2023, pp. 375–387.

[79] Tian Wang and Kyunghyun Cho. "Larger-context language modelling". In: *arXiv preprint arXiv:1511.03729* (2015).

[80] Jonathan J Webster and Chunyu Kit. "Tokenization as the initial phase in NLP". In: *COLING 1992 volume 4: The 14th international conference on computational linguistics*. 1992.

[81] *What is explainable AI? | IBM*. Ibm.com, 2024. URL: `https://www.ibm.com/topics/explainable-ai` (visited on 02/15/2024).

[82] Qinyuan Ye et al. *Prompt Engineering a Prompt Engineer*. arXiv.org, 2023. URL: `https://arxiv.org/abs/2311.05661` (visited on 04/16/2024).

[83] Wojciech Zaremba and Greg Brockman. *OpenAI Codex*. Openai.com, 2021. URL: `https://openai.com/blog/openai-codex` (visited on 02/06/2024).

[84] *Zero-Shot Prompting – Nextra*. Promptingguide.ai, 2022. URL: `https://www.promptingguide.ai/techniques/zeroshot` (visited on 04/16/2024).

[85] Yongchao Zhou et al. "Large language models are human-level prompt engineers". In: *arXiv preprint arXiv:2211.01910* (2022).

Appendix A

Appendix A name

A.1 Pictures generated using DALL·E 3

Illustrating how the model can be confused and treats ambiguity of words in a semantic relation to the sentence. Click here to get back to the text 3.2.

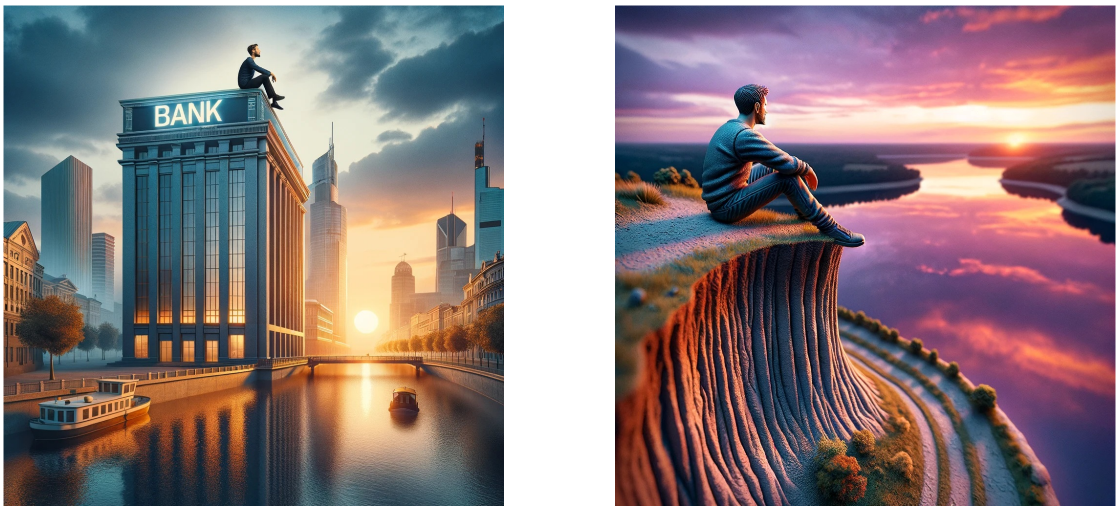


Figure A.1: Illustrating how models can be confused about the ambiguity of words - A financial institution (bank) is different from a riverbank. The semantic relation between the words in a sentence guides the model to distinguish ambiguity.

A.2 Tagged prompt engineering conversation

The link to the conversation I had when creating the example for section 3.5.1: https://chat.openai.com/sha
a144-4b52-8c59-d6274afe849a

A.3 Interviews

A.3.1 Mads Brodt - Dwarf

Front-End Lead | Helping front-end engineers level up their skills and land jobs | JavaScript, React, Vue, TailwindCSS

LinkedIn: https://www.linkedin.com/in/madsbrodt/

Front-end lead employed at Dwarf - Works on large consultant projects. Lego as main customer, working across teams beyond front-end.

Mængde af møder - Møde referat - Hvor mange deltager ift. hvor mange der er AKTIVE til mødet. Tid og pengebesparelse.

A.3.2 Johan Fenn Bagger Nærby - Connected Cars

Software Engineer by trade, entrepreneur by heart

LinkedIn: https://www.linkedin.com/in/johanfbn/

A.3.3 Andreas Dahl Pedersen - ASIMUT Software

Media developer at ASIMUT software ApS

LinkedIn: https://www.linkedin.com/in/andreas-dahl-pedersen-1049ba28b/

A.3.4 Regnar Vedsted - Lego

Software Developer

LinkedIn: https://www.linkedin.com/in/regnar-vedsted-9180a517a/

A.3.5 Mathias Vilbrad - Netcompany

Tech enthusiast working within Data Analytics | Software Development | Digital Marketing | Artificial Intelligence
LinkedIn: https://www.linkedin.com/in/mathias-vildbrad-72b654155/