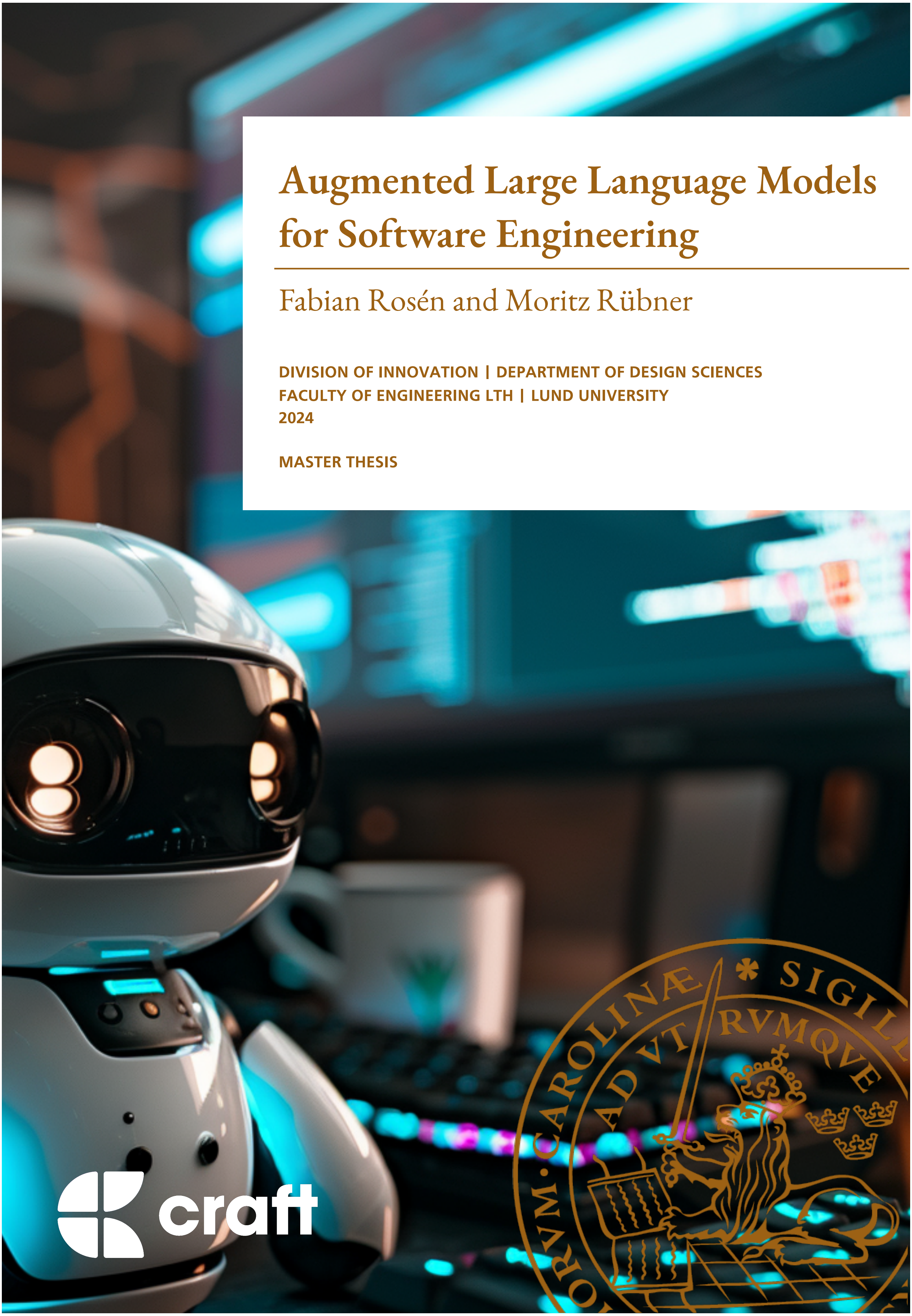


Augmented Large Language Models for Software Engineering

Fabian Rosén and Moritz Rübner

DIVISION OF INNOVATION | DEPARTMENT OF DESIGN SCIENCES
FACULTY OF ENGINEERING LTH | LUND UNIVERSITY
2024

MASTER THESIS

Augmented Large Language Models for Software Engineering

Opportunities and Limitations of Artificial Intelligence

Fabian Rosén and Moritz Rübner

LUND
UNIVERSITY

Augmented Large Language Models for Software Engineering

Opportunities and Limitations of Artificial Intelligence

Copyright © 2024 Fabian Rosén and Moritz Rübner

Published by

Department of Design Sciences
Faculty of Engineering LTH, Lund University
P.O. Box 118, SE-221 00 Lund, Sweden

Subject: Innovation Engineering (INTM01)
Division: Division of Innovation
Supervisor: Torben Schubert
Co-supervisor: Bálint Orosz
Examiner: Jing Xiao

Abstract

Since the release of the recent generation of Large Language Models (LLMs) such as GPT-4 (GPT standing for generative pre-trained transformer), companies and individuals are exploring opportunities to improve their effectiveness by automating or receiving assistance with knowledge-intensive tasks by using artificial intelligence (AI). Within software engineering, a plethora of commercial services promising impressive results has reached the market in the past year. This thesis explores the practical implications of applying LLMs in this software engineering context. Through a case study at Craft Docs, a software development company, we investigated the current use of LLMs and identified specific opportunities to leverage these models as a way of improving the efficacy of software engineers.

Action research (AR) was chosen as the research methodology, carried out by developing a prototype that was deployed and evaluated at the company. A literature study and interviews surveying the use of and sentiment towards LLMs at the case company guided the development, and the prototype was evaluated with practical tests and retrospective interviews.

Findings from the project suggest that while LLMs can enhance certain aspects of software engineering, their effectiveness is highly dependent on the context of use and the specific tasks they are designed to support. The prototype developed in this study demonstrates a practical implementation of retrieval-augmented generation (RAG) to provide additional context to LLMs, showing both the strengths and limitations of these technologies in real-world settings. Most apparent was that even with access to relevant code context, tasks given to an LLM need to be carefully scoped to yield effective results.

Keywords: ai, artificial intelligence, large language models, llm, retrieval-augmented generation, rag, software engineering, code, programming

Sammanfattning

Sedan lanseringen av de senaste generationerna stora språkmodeller (LLM), som GPT-4 (GPT som generative pre-trained transformer), utforskar företag och individer möjligheter att förbättra sin effektivitet genom att automatisera eller få hjälp med kunskapsintensiva uppgifter med artificiell intelligens (AI). Inom programvaruutveckling har en mängd kommersiella tjänster som lovar imponerande resultat nått marknaden under det senaste året. Detta examensarbete utforskar de praktiska implikationerna av att tillämpa LLM:er inom sammanhanget programvaruutveckling. Genom en fallstudie på Craft Docs, ett företag inom apputveckling, undersökte vi användningen av LLM:er och identifierade specifika möjligheter att utnyttja dessa modeller för att förbättra programvaruingenjörers effektivitet.

Aktionsforskning (action research, AR) valdes som forskningsmetodik, genomförd genom att utveckla en prototyp som implementerades och utvärderades på företaget. En litteraturstudie och intervjuer som undersökte användningen av och inställningen till LLM:er på fallföretaget styrde utvecklingen, och prototypen utvärderades med praktiska tester och retrospektiva intervjuer.

Resultaten från projektet tyder på att även om LLM:er kan förbättra vissa aspekter av programvaruutveckling är deras effektivitet starkt beroende av användningssammanhanget. Utformningen av uppgifterna som ges till modellerna har stor påverkan på nyttan. Prototypen som utvecklades i denna studie demonstrerar en praktisk tillämpning av retrieval-augmented generation (RAG) för att ge ytterligare kontext till LLM:er, och visar både styrkor och begränsningar med dessa teknologier. Det mest uppenbara var att även med tillgång till relevant kod och kontext måste uppgifter som ges till en LLM vara noggrant definierade och avgränsade för att ge effektiva resultat.

Nyckelord: ai, artificiell intelligens, stora språkmodeller, llm, retrieval-augmented generation, rag, programvaruutveckling, kod, programmering

Acknowledgements

This thesis project was conducted at the Department of Design Sciences at the Faculty of Engineering at Lund University, in collaboration with Craft Docs. The authors received substantial insights and support from both parties and want to express their gratitude below.

Firstly, we want to thank our supervisor Torben Schubert at LTH. Your guidance and academic expertise have been a great support, and we appreciate your experienced and professional approach to research. Your encouragement and advice helped us stay on track, and remain focused throughout the thesis.

Secondly, we want to thank Bálint Orosz, the CEO of Craft Docs, for providing us the opportunity to conduct this project at such an innovative and ambitious start-up company. You have not only given us access to cutting-edge technologies and amazingly insightful software engineers, but also helped us define an interesting and relevant project in a rapidly developing field.

Lastly, we want to thank all team members at Craft who took their precious time to participate in interviews. We are especially grateful for those who helped us by extensively testing the prototype and providing their insights and feedback. We are proud to present our thesis and want to highlight that the contributions and support of the people involved have made this project possible.

Lund, May 2024

Fabian Rosén and Moritz Rübner

Contents

List of acronyms and abbreviations **9**

1 Introduction **10**
- 1.1 Background 10
- 1.2 Craft Docs 11
- 1.3 Objective and research questions 12
- 1.4 Scope of study 13
- 1.5 Distribution of work 13
- 1.6 Target audience 14
- 1.7 Outline of thesis 14

2 Theory **16**
- 2.1 Software engineering 16
- 2.2 Large Language Models 17
- 2.3 Methods for optimizing and tailoring LLMs 19
  - 2.3.1 Prompt engineering 19
  - 2.3.2 Retrieval-Augmented Generation 19
  - 2.3.3 Fine-tuning 20
- 2.4 LLM-based services and service providers 20
  - 2.4.1 OpenAI 21
  - 2.4.2 Google Gemini 22
  - 2.4.3 GitHub Copilot 22
  - 2.4.4 LLM programming tool start-ups 23

3 Method **25**
- 3.1 Research approach 25
  - 3.1.1 Investigation 26
  - 3.1.2 Solution 28
  - 3.1.3 Prototype evaluation 30
- 3.2 Research quality 31
- 3.3 Use of generative AI 32
- 3.4 Ethical considerations 32

4 Results **33**

4.1 Investigation 33
4.1.1 Literature study 33
4.1.2 User study and needs analysis 38
4.2 Solution 42
4.2.1 Concept generation 42
4.2.2 First iteration - prototype generation 43
4.2.3 First iteration - prototype testing 46
4.2.4 First iteration - prototype review 46
4.2.5 Second iteration - prototype generation 49
4.2.6 Second iteration - prototype testing 49
4.2.7 Second iteration - prototype review 50
4.3 Prototype evaluation 50
4.3.1 Evaluation tree 50
4.3.2 Scenario evaluation 52
4.3.3 Interview retrospective 61
4.4 Summary of results 63
4.4.1 Effectiveness in limited scope problem areas 63
4.4.2 Need for precise instructions and critical thinking 64
4.4.3 Improved effectiveness when context is supplied automatically 65
4.4.4 Effectiveness in unfamiliar areas 65
4.4.5 Need for IDE integration 66

5 Discussion **67**
5.1 Contrasting research results with theory 67
5.1.1 Impacts on effectiveness 67
5.1.2 Technical considerations 69
5.2 Future improvements of the application 70
5.3 Limitations 71
5.4 Contributions 72
5.4.1 Academia 72
5.4.2 Practical Applications 73

6 Conclusions **74**
6.1 Key findings 74
6.2 Concluding remarks 76

References **77**

Appendix A Interview information **84**
A.1 List of interview participants 84
A.2 Interview guides 85

Appendix B Prototype material **87**
B.1 Data processing script 87
B.2 Prompts 87
B.3 Thread object example 89

List of acronyms and abbreviations

Acronym	Description
AI	artificial intelligence
API	application programming interface
AR	action research
CD	continuous deployment
CI	continuous integration
CoT	chain of thought
CRUXEval	code reasoning, understanding, and execution evaluation
GPT	generative pre-trained transformer
GSM-8K	grade school math 8k
GUI	graphical user interface
IDE	integrated development environment
IEEE	Institute of Electrical and Electronics Engineers
LLaMA	large language model meta ai
LLM	large language model
ML	machine learning
MMLU	massive multitask language understanding
RAG	retrieval-augmented generation
RLHF	reinforcement learning from human feedback
RLPG	repository-level prompt generator
ToT	tree of thought

1 Introduction

In the introduction chapter, background to the problem is provided and the case company is presented. The problem is closely described and defined in terms of research questions. The section also presents the structure of the thesis and its delimitations.

1.1 Background

The recent advancements in Large Language Models (LLMs) present new opportunities for companies to improve their effectiveness and competitive strength. Using LLMs effectively is likely to be increasingly important for staying competitive, especially for start-up companies with a need for effective value creation with limited resources.

Software engineering is a field where LLM applications such as code generation, translation between programming languages, documentation generation and testing already are emergent and in focus for companies developing and commercializing LLMs and LLM-based tools.

Ozkaya (2023), editor of IEEE Software, a publication by the Institute of Electrical and Electronics Engineers (IEEE) suggests that software engineers should "think about LLMs as partners". She outlines specification generation, improved testing, developer feedback, documentation, and language translation as possible use cases. Research performed in affiliation with GitHub (Ziegler et al., 2022) has shown LLM code completion to increase job fulfilment, decrease frustration and help software engineers stay in the flow, as well as preserve mental effort. In addition, a study carried out at Harvard Business School by Dell'Acqua et al. (2023) studied the effects the LLM ChatGPT-4 (GPT standing for generative pre-trained transformer) had on knowledge workers' productivity and quality. It was found that AI-assisted consultants at Boston Consulting Group on average carried out tasks 25% faster, completed 12% more tasks overall and delivered work assessed to be of 40% higher quality compared to peers unassisted by AI.

Dell'Acqua et al. (ibid.) describe though that these effects do not apply unilaterally, and introduce the notion of a "jagged frontier of AI capabilities", outlining that LLMs capabilities and human capabilities do not always align, and that performance gains only are achieved when LLMs outperform humans on a given task. The jagged frontier is visualized in figure 1.

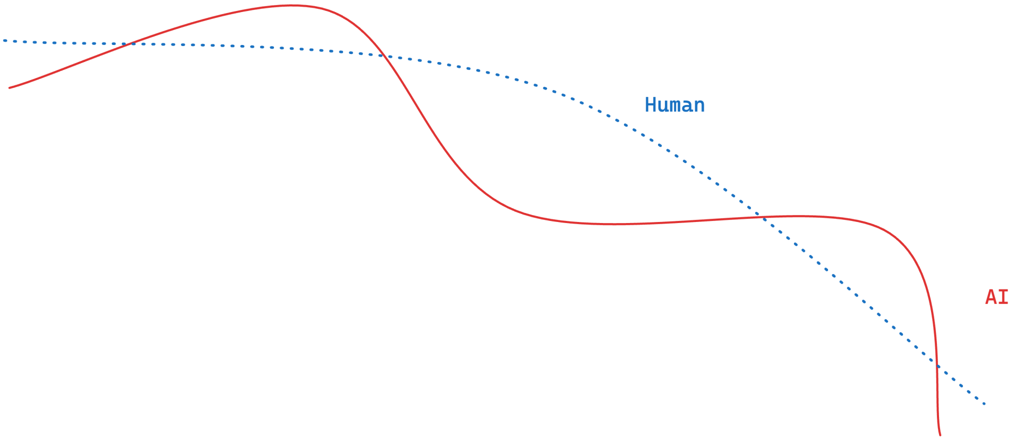


Figure 1: The ”jagged frontier of AI capabilities”, adapted from Dell’Acqua et al. (2023).

Due to the fast pace of development within the field, it is not exactly clear which tasks lie where on the frontier and as Dell’Acqua et al. (2023) outline, being able to navigate the frontier is an important ability to be able to achieve effectiveness gains. Additionally, the implementation of LLMs within the field of software engineering so far lacks universally acknowledged best practices, akin to the early stages of the internet’s adoption, where widespread enthusiasm existed without clear guidelines on optimal utilization. This thesis aims to provide an insight into how applications of LLMs in software engineering and the software development process look like in a case study relevant to the field, and with an action research (AR) approach show how opportunities can be acted upon and may result in improvements.

1.2 Craft Docs

Craft Docs (Craft) is a European start-up company developing and distributing a modern writing and note-taking platform under the same name. They are defining their mission as building “the future of Documents” and “to create a beautiful and joyful space where you no longer feel like you are fighting the tool”. Craft is available on iOS for iPhone and iPad, MacOS, Windows and in the web browser, under both a free version and paid subscriptions with over 1 million registered users as of September 2022 (Craft Docs, n.d.[b]).

Founded in July 2019 in Budapest, Hungary by Bálint Orosz (Craft Docs, n.d.[b]), this is his second venture after acting as co-founder and CEO for a mobile app agency later sold to travel agency and search aggregator Skyscanner (Smith, 2015). The app was first released in November 2020, after which Craft raised two rounds of venture funding and was awarded the Mac App of the Year award by Apple (Craft Docs, n.d.[b]).

For this thesis, the software engineering teams at Craft have been in focus, consisting of 22 team members as of January 2024. In addition, CEO Bálint Orosz, product designers, and data analysts also work with code as part of their tasks. The main programming languages used at Craft are JavaScript with TypeScript and Swift. Craft describes its work culture as focused around kindness and support, being results-driven and focused, working in "highly aligned, loosely coupled teams", being agile, passionate, committed and focused on innovation (Craft Docs, n.d.[a]). Craft is committed to continually improving its work processes and being conscious about what tools and systems are used in software engineering. They were thus an early adopter of LLMs in both its product (Baldwin, 2022) and within its software development processes. The software engineers at Craft are generally highly experienced professionals with multiple years of experience working with a multitude of technologies, and have experience in keeping up with recent developments within the industry.

Prior to the initiation of the thesis project, Craft had surveyed employees on how widely ChatGPT was used in the company by its employees. A large portion of the employees answered that they are using it every day, both in software engineering and other teams. However, there seemed to be differences across different groups of employees, with some using AI tools to a much greater extent than others. CEO Bálint Orosz and others had already raised the notion that AI is useful for some coding-related tasks, but when working with Craft's codebase specifically it has in some cases been less useful. It was assumed this could be the case due to the large amounts of proprietary code and intellectual property, which public LLMs do not have access to.

1.3 Objective and research questions

This master's thesis aims to analyze how customized LLMs can be used to improve the effectiveness of workers in software engineering. This will be done in three steps, starting with a literature research and surveying how LLMs are used at Craft Docs, a software-based start-up company, answering the research question:

R1: *How do software developers at Craft Docs use LLMs today?*

Based on the field and literature reconnaissance, one or more hypotheses on how

LLMs can be customized and used to improve effectiveness will be formulated, to answer the question:

R2: *What LLM-related improvement opportunities can be found in regard to software engineers' effectiveness?*

These hypotheses will be tested by developing, deploying, and evaluating one or more customized LLM prototypes at Craft, answering:

R3: *What actual improvement in effectiveness is achieved by acting on the found opportunities?*

R1 is intended to be answered with an interview-based case study at the thesis project's case company, providing a foundation for answering R2, which in turn is aimed to be answered with a combination of case study insights as well as a literature study. R3 is to be answered with an action research project including a functional prototype and subsequent testing and evaluation at the case company.

1.4 Scope of study

The scope of the thesis project has been limited to looking at the discipline of software engineering. Adjacent fields such as data analysis and design, though these might work closely with code as well, are omitted from the scope. Although software engineers also have other types of tasks, writing and working with code is the main focus of this thesis. This project encompasses a case study, which due to the focus on coding has been constrained to software engineering team members at the case company. Interview guides for interviews with case study participants were scoped to the code-related tasks that are part of their work.

The LLMs included in analyses, concepts and prototypes are limited to publicly available models from providers such as OpenAI, Google etc. The thesis work will not include any coding or training of these models beyond fine-tuning, prompting or otherwise providing context via available tools such as web-based graphical user interfaces (GUI) or application programming interfaces (APIs). This is to be able to achieve meaningful iterations and prototypes under the time constraints for a master thesis project.

1.5 Distribution of work

Both authors participated in all steps, tasks and aspects of this thesis project. Research papers and source materials were in some cases divided equally, with one author synthesizing and relaying key information to the author. Both participated in all interviews that were performed, contributed to the technical development of the prototype, and wrote this report together.

1.6 Target audience

The target audience for this thesis is decision-makers and software engineers interested in understanding LLM's impact and possible applications, primarily at small to medium-sized companies, but also in the wider industry. Secondarily, the thesis is aimed at researchers and students interested in artificial intelligence, software engineering and productivity.

1.7 Outline of thesis

The outline of the thesis is summarised in Table 1. The thesis contains six chapters, following the chosen research approach.

Table 1: Outline of thesis.

Chapter	Focus
1. Introduction	In the introduction chapter, background to the problem is provided and the case company is presented. The problem is closely described and defined in terms of research questions. The section also presents the structure of the thesis and its delimitations.
2. Theory	This chapter presents a theoretical overview of concepts and solutions highly relevant to the thesis. These are the discipline of software engineering, the emerging technology LLMs, methods for optimizing and augmenting LLM output, as well as an overview of LLM-based services currently available on the market.
3. Method	The method chapter comprises the research design and methodology of the thesis project, defining the overall process. Research quality and ethics considerations are also presented.
4. Results	This chapter presents the results from the initial literature research and interviews, the generated concept as well as the prototype iterations. Finally, the results of the evaluation of the last prototype are presented.
5. Discussion	This chapter contains the authors' thoughts and observations on the results from the user needs analysis, prototyping phase and subsequent evaluation, contrasting them with results from the literature study. Additionally, the limitations of the conducted research project are discussed, along with possible future improvements regarding the developed prototype.
6. Conclusion	The final chapter presents the thesis project's conclusions as answers to the research questions, and summarizes the key results.

2 Theory

This chapter presents a theoretical overview of concepts and solutions highly relevant to the thesis. These are the discipline of software engineering, the emerging technology LLMs, methods for optimizing and augmenting LLM output, as well as an overview of LLM-based services currently available on the market.

2.1 Software engineering

As digitalization has spread through basically all aspects of modern life, software has become an essential part of business processes, interactions, communication and more. The work surrounding and encompassing the development, deployment and maintenance of software is what is nowadays commonly referred to as software engineering, a discipline born out of two NATO Science Committee conferences (Naur and Randell, 1969; Buxton and Randell, 1970). IEEE defines software engineering as "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software, and the study of such approaches." (IEEE, 1990).

O'Regan (2022) describes the discipline as not only being concerned with the actual development of software, but also with how it is designed and modelled, inspected and tested, how its performance is measured and how it is secured against adverse actors. In addition, he mentions ethical and legal aspects, software procurement, software project management and managing requirements as important aspects of the discipline.

Important to note is that contrary to other engineering disciplines such as civil, electrical or mechanical engineering, is that within computer science, the term *engineer* is used much more loosely, and seen as somewhat interchangeable with the term *developer*, explains O'Regan (ibid., p. 5). He describes that within the industry, it is acknowledged that to achieve good business results and maintain a sound reputation, software companies need to have confidence in the professional qualifications of employees. This is especially due to the fact that educational backgrounds vary widely and not every software engineer having a formal engineering degree. Being able to deliver high-quality products and services is thus tied to investing in education and training related to the discipline, “such as [in] project management, software design and development, software testing and service management” (O’Regan, 2022, p. 6).

In recent decades, process-light software development methodologies have gained popularity, with the term *agile* being coined in the agile manifesto by Beck et al. (2001). They describe a belief in early and continuous value delivery to customers, face-to-face conversation, a sustainable pace of development and harnessing competitive advantage by welcoming changing requirements. As a continuation to these trends, O’Regan (2022, pp. 14-15) mentions practices such as continuous integration (*CI*) – the practice of submitting work as soon as it is finished to receive immediate feedback, continuous delivery and testing – automatically building and testing each new code addition, and continuous deployment (*CD*) – automating the delivery of changes and code to end users. According to O’Regan (ibid., p. 15), these practices are employed to accelerate delivery and yield faster business results.

A similar hope for increased productivity and faster results is put on emerging AI tools, mostly LLMs, as described by Peng et al. (2023) specifically for the software engineering use case, or as mentioned previously for consultants and other roles (Dell’Acqua et al., 2023). For the software engineering use case, Peng et al. (2023) present promising results, with speed improvements of more than 55% for those assisted by the tool *GitHub Copilot*, based on a state-of-the-art LLM.

2.2 Large Language Models

LLMs are the underlying technology used in language-based generative AI services such as ChatGPT. As presented by Nvidia (n.d.), current use cases for text-based LLMs can be divided into the following areas:

- Generation (e.g., story writing, marketing content creation)
- Summarization (e.g., legal paraphrasing, meeting notes summarization)
- Translation (e.g., between languages, text-to-code)
- Classification (e.g., toxicity classification, sentiment analysis)
- Chatbot (e.g., open-domain Q&A, virtual assistants)

Technically, LLMs are machine learning (ML) models created to predict the most probable next token in a sequence of tokens, with tokens representing words or code (Google, 2023). These models are referred to as large due to their large volume of training data, and large number of parameters or weights in the model (ibid.).

The public attention regarding LLMs began with the release of ChatGPT and its subsequent models, with GPT-4 being their latest and most capable model (OpenAI, n.d.[b]). Although GPT-4 is the best publicly available LLM based on current benchmarks, Google has released a more advanced model, Gemini Ultra, reportedly surpassing the capabilities of GPT-4 (Gemini Team et al., 2023). Since the model has not yet been publicly available at the time of writing, these claims have not been independently audited.

Current available LLMs can be divided into two groups, one being closed source models released by large companies such as OpenAI (GPT), Google (Gemini) and Meta (LLaMA, large language model meta ai) with the other group consisting of open source models available on sites such as HuggingFace (n.d.). Open source models are freely available and can be modified, whereas closed source models are made commercially available under license.

Open source models generally under-perform closed source models on benchmarks with some exceptions where open source models exceeded in relation to older closed-source models such as GPT-3.5, but lagging far behind the more cutting edge GPT-4 (Kwan et al., 2024). An explanation for this difference in performance can be found in the expensive training cost associated with these cutting-edge LLMs, with the cloud computing power alone needed to train GPT-4 being estimated at $60M, and overall training cost estimated to exceed $100M (Minde, 2023), though the actual figures are undisclosed. This as well as the need for large comprehensive training datasets provides a barrier to entry for open source contributors.

LLM tools such as OpenAI's ChatGPT, Microsoft's Copilot and Google's Gemini are mainly controlled via prompts, being natural language text that works as input for the model (Amazon Web Services, n.d.). The underlying LLM will then provide a result based on the input(ibid.). Providing proper prompts is important for receiving a high-quality answer from the probability-based LLM (Harvard University Information Technology, 2023).

LLMs are built using transformer networks, which in turn are neural networks that learn context and meaning through tracking relationships in sequential data such as sentences (Nvidia, n.d.). Transformers were introduced by Vaswani et al. (2023) in the paper "Attention Is All You Need". The authors describe the concept of *attention*, mathematically focusing on the most important parts of a natural language sequence. This made it possible to process longer sequences compared to earlier models. Furthermore, this is where GPT gets its name from, with GPT being short for Generative Pre-trained Transformers.

LLM-based tools have been shown to increase the productivity of knowledge workers within different areas (Dell'Acqua et al., 2023; Cambon et al., 2023), with the most commonly used tools being ChatGPT by Open AI and Copilot by Microsoft. Beyond actual improvements in productivity, it is also reported that participants perceive the productivity gain to be more substantial than the actual time saved (Cambon et al., 2023). Furthermore, a recent study showed that developers assisted by Github Copilot, an LLM tool powered by Open AI's LLM models, completed tasks 55.8% faster compared to a control group, with less experienced developers benefiting more (Peng et al., 2023). The study was however done on a greenfield task only, meaning that it was a standalone program without dependencies that the participants were asked to build.

2.3 Methods for optimizing and tailoring LLMs

Over the past years, several methods for optimizing and tailoring LLMs to specific use cases have emerged. This thesis will focus on the three most commonly used tools, being prompt engineering, retrieval-augmented generation (RAG) and fine-tuning (Allard and Colin, 2023).

2.3.1 Prompt engineering

Prompt engineering is the discipline of refining and optimizing prompts, the user input provided to LLMs (Gemini Team et al., 2023). Improved prompting may lead to the expansion of LLMs abilities (Wei, Tay, et al., 2022), making it a crucial part of optimising LLMs performance. An example of how to improve prompting is to provide context, which improves the quality of the generated answer (Tufano et al., 2021) and is also the purpose of RAG.

2.3.2 Retrieval-Augmented Generation

RAG presents an opportunity how to expand the effectiveness of LLM tools beyond simple tasks, allowing LLMs to address context-related questions (Khadija, Aziz, and Nurharjadmo, 2023) and shows promising results for being deployed across a wide range of natural language processing tasks (Lewis et al., 2021). The basic concept of RAG is that the entered prompt is used to calculate an embedding, which in turn is used to search and retrieve documents that are passed to the LLM to generate an answer (Barnett et al., 2024), as shown in Figure 2.

An embedding is a vector of numbers, and both the input query, as well as the documents available for the retriever, are transformed into these embeddings in chunks or as a whole. This transformation is done by a language model, to achieve a mathematical representation of the natural language content. Searching with the help of these embeddings is called *semantic search* and works by identifying the vectors with the

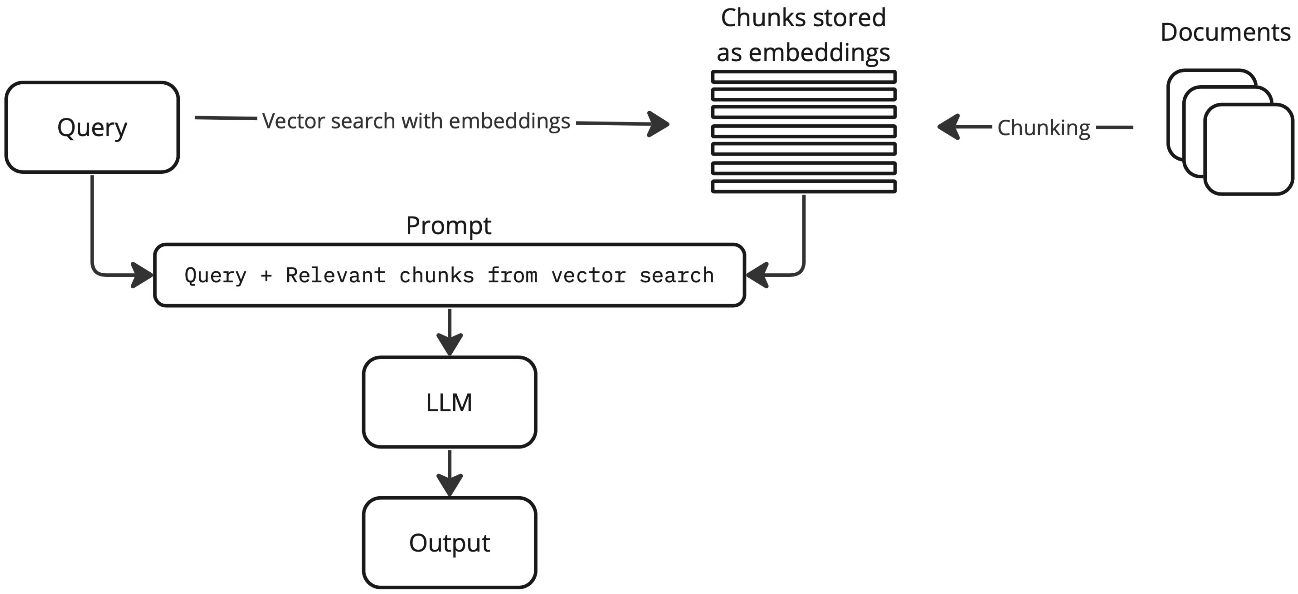

Chunks stored
as embeddings
Documents
Query
Vector search with embeddings
Chunking
Prompt
Query + Relevant chunks from vector search
LLM
Output

Figure 2: General concept of RAG.

shortest distance from each other to be able to find the best matches (Barnett et al., 2024).

2.3.3 Fine-tuning

Fine-tuning is the process of training an already trained model on data specific for a certain task to make the model more aligned to its users' needs (Ouyang et al., 2022; Allard and Colin, 2023). This allows the model to become more accurate, efficient and better at following instructions related to the desired task due to the distilling of its expertise (Ouyang et al., 2022; Allard and Colin, 2023). Further benefits of fine-tuning include the reduced input and therefore computational cost needed to reach the desired output (Allard and Colin, 2023) as well as fine-tuning being a much more cost-efficient way of obtaining a model tailored model for a specific task compared to training a new model from scratch (Ouyang et al., 2022).

2.4 LLM-based services and service providers

LLMs, alone or in combination with the previously described method of RAG, are employed in various commercially available services. Some of these services are tailored to coding use cases specifically or broadly. These relevant services address three main coding use cases, some of them providing solutions for multiple. These are

- completion - completing a line of code, or generating the next one based on previous ones,
- function generation - drafting a complete function/component, and
- chat - being able to converse back and forth with an LLM in natural language and code.

Below, the identified companies and their services are described in further detail, with their solutions, limitations, technical ways of working as well as pricing considerations.

2.4.1 OpenAI

OpenAI has emerged as one of the most important entities concerned with developing and distributing generative AI services through LLMs, in large part thanks to their breakthroughs with the LLMs based on the *Generative Pre-Trained Transformer* (GPT) architecture and multi-billion dollar funding from Microsoft and others (Bajekal et al., 2023).

OpenAI's publicly available offering can be divided into two categories - services available via APIs, and *ChatGPT*. Both of these build on top of the LLM technologies developed by OpenAI such as *GPT-3.5* and *GPT-4* for text, *DALL-E* for generating images or *Whisper* for speech recognition (OpenAI, n.d.[f]). Within ChatGPT, these models are packaged within a user-friendly interface as different features (OpenAI, 2023). For developers, they are accessible through stand-alone APIs that can be integrated in various applications (OpenAI, n.d.[d]).

ChatGPT is likely the most widely known LLM-based consumer service, setting a record as the consumer application with the fastest-growing user base in history (Hu, 2023). It has also been widely adopted for code-related use cases, as shown by Feng et al. (2023). The authors analyzed ChatGPT's performance on code generation tasks through crowdsourcing based on social media data, finding that the most popular programming languages mentioned in conjunction with ChatGPT are Python and JavaScript. They also found that the most common use cases are code generation, debugging and testing, preparing for programming interviews and other related tasks. The authors concluded that the stable attention and posting regarding ChatGPT on social media indicates that it is helpful for code generation, though sentiment analysis showed that *fear* is the dominant emotion across social media platforms, possibly caused by ”concerns over job security”. As previously mentioned, ChatGPT has also shown impressive performance in regards to productivity and work quality in other contexts, with Dell'Acqua et al. (2023) finding a 25% improvement in speed, 12% improvement in quality, and 40% higher work quality comparing knowledge workers assisted by ChatGPT to those without this tool in a study at Boston Consulting Group.

The APIs offered by OpenAI enable more sophisticated and customized use cases compared to ChatGPT, as they can be interacted with via HTTP and through the Python and Node.js libraries provided by OpenAI (n.d.[d]). Pricing for the APIs varies based on volumes and model version (OpenAI, n.d.[i]). According to the technical documentation, the APIs offered can be grouped into a few broad categories, namely audio, chat, files, images, and some additional use cases such as fine-tuning, embeddings, moderation and *Assistants*. The Assistants API is especially interesting for this thesis, as it enables the creation of ChatGPT-like AI assistants that can make use of further tools, such as a code execution environment and Retrieval-Augmented Generation (OpenAI, n.d.[e]). This enables automating the process of chunking uploaded documents, indexing and storing embeddings in a vector database, and implementing a vector search that can retrieve relevant content related to a user prompt (ibid.).

2.4.2 Google Gemini

Google's LLM assistant Gemini (formerly Bard) is also capable of generating code (Google, n.d.[c]). Generally, it is very similar in functionality to ChatGPT, as it works in the same chat-style manner, can comprehend text and images, and can generate images (Pichai and Hassabis, 2024). As the Gemini models were made available in February 2024, no independent studies of its performance or real-world implications on worker effectiveness and efficiency were available at the time of writing. Google currently offers APIs for developers to use the Gemini models, but did at the time of writing not offer any fine-tuning or retrieval services. Also, access to the advanced model Gemini Ultra requires a commercial relationship with a Google representative (Google, n.d.[a]). Accessing the API is priced based on whether inputs are allowed to be used as training data as well as usage volumes in a pay-as-you-go model (Google, n.d.[b]). Pre-packaged services are available and for individuals cost $20/month (Google, n.d.[d]), thus on the same level as ChatGPT. For teams, general LLM functionality with a chat assistant starts at $20/user/month, with a $30/user/month plan providing access to further functionality and additional available languages (Google Workspace, n.d.). Code-specific features are packaged as *Code Assist* for $19/user/month (Google Cloud, n.d.).

2.4.3 GitHub Copilot

GitHub Copilot is an LLM-based programming tool, also described as an "AI pair programmer that suggests code and entire functions in real-time based on context" by

Peng et al. (2023) in their 2023 study affiliated with GitHub, Microsoft Research and MIT. Powered by OpenAI's *Codex* model (Chen et al., 2021), a modified version of GPT-3, Copilot is integrated into integrated development environments (IDEs) such as Visual Studio, Visual Studio Code, JetBrains, Vim and more, and can work with ”all [programming] languages that appear in public repositories”, with ”quality of suggestions [...] [depending] on the volume and diversity of training data for that language” (GitHub, n.d.). As GitHub is owned by Microsoft, they are backed by a comparatively large amount of funding and engineering resources. Currently, GitHub Copilot is priced at $10/month for individuals, $19/month/user for businesses, and $39/month/user for access to enterprise functionality (ibid.).

According to GitHub (ibid.), the functionality of Copilot currently consists of code completion and coding suggestions, and a chat assistant. For ”enterprise customers” that are screened with the help of a waitlist, GitHub (ibid.) also offers automated change descriptions, and ingesting code repositories and related documentation with what is assumed by the authors to be a combination of a fine-tuned model and a method of retrieval.

In addition to the previously referenced study by Peng et al. (2023) that was based on a greenfield task of programming a web server and conducted in affiliation with GitHub, other researchers were also able to find positive effects on productivity. Chatterjee et al. (2024) concluded a ”notable boost in productivity and code quality with GitHub Copilot”, and reported indications for a ”significant increase in productivity and job satisfaction”. However, B. Zhang et al. (2023) found that integrating the generated code can pose a difficulty, and concluded that Copilot is a ”double-edged sword” requiring the developer's careful consideration, especially in regards to how well it performs with chosen programming languages and technologies.

2.4.4 LLM programming tool start-ups

In addition to the widespread commercial tools from OpenAI and Microsoft, there is also a plethora of start-up companies developing LLM-based tools for programming applications. Most of these focus on providing coding assistance directly in the IDE, either as a plug-in, or as a standalone IDE. Key differentiators from GitHub Copilot are privacy, wider access to more advanced features, and deeper integration into the code editor.

An example of a privacy-focused tool is Tabnine which describes itself as ”The AI coding assistant that you control”, and uses a proprietary language model ”exclusively trained on permissively licensed open source repositories” (Tabnine, n.d.). They also highlight security and privacy certifications, strict privacy controls and isolated deployments, while at the same time offering a similar feature set to GitHub Copilot. Pricing is similar to GitHub Copilot, with $12/month/user on the standard plan, and

$39/month/user for the enterprise plan (Tabnine, n.d.).

For advanced features and deeper integration in the code editor, one of the more widespread tools is Cursor. Unlike GitHub Copilot, Tabnine and others, Cursor is a standalone IDE which according to the developers enables deeper integration of AI functionality (Cursor, n.d.). Similarly to GitHub Copilot, Cursor has a chat functionality and auto-completion. However, Cursor supports directly referencing code files in the chat window and generating mid-line completions, whereas Copilot only supports end-of-line or new-line completions, according to Cursor's developers (ibid.). The developers also outline that Cursor accomplishes advanced code generation with the codebase context by generating embeddings and storing these in a database on their servers. Cursor is slightly more expensive than GitHub and Tabnine, with individual pricing at $20/month and team pricing at $40/user/month (ibid.).

Overall, it is noticeable that the market for LLM-assisted code editing and generation is moving quickly, with new entrants and rapid development. Compared to general chat and text generation tasks, where the large actors on the market such as OpenAI with ChatGPT, and Google with Gemini are leading, there is a wider variety with various differentiators available within code editing and generation.

3 Method

The method chapter comprises the research design and methodology of the thesis project, defining the overall process. Research quality and ethics considerations are also presented.

3.1 Research approach

Research design is closely tied to research purpose and the latter should inform the former. Four purposes for thesis projects are outlined by Höst, Regnell, and Runeson (2006, p. 29): descriptive, exploratory, explanatory and problem-solving studies. This master thesis follows a problem-solving approach, which is also cited as most common for technical universities by Höst, Regnell, and Runeson (ibid., p. 29). As the goal of the thesis project is to both gain an understanding of the problem and to achieve improvements, an action research (AR) process is suitable (ibid., p. 31). Stemming from social sciences, Robson and McCartan (2016, p. 199) describes AR's purpose as ”to influence or change some aspect of whatever is the focus of the research” with ”[i]mprovement and involvement” being central. Within improvement, three components are identified: improving a *practice*, improving *understanding* of a situation, and improving the *situation* in which the practice is embedded (ibid., p. 199).

Höst, Regnell, and Runeson (2006) describe AR as a principally flexible research approach, meaning its methodology can be changed or adjusted during the study's duration. Though there is flexibility, according to the authors the general process follows the three steps of *investigating* a situation or phenomenon to clarify the problem to be solved; creating a proposal for a *solution* to the problem; as well as collecting data and *evaluating* how well the solution meets expectations. They again point out that this is an iterative process, following a structure similar to quality improvement practices in cycles.

As visually outlined in Figure 3, both the overall thesis project as well as the development of the solution follow the approach of investigating, taking steps towards a solution, and evaluating its effects. For the overall thesis, this is denoted as the *investigation*, *solution* and *evaluation* phase. For the prototype, *generation* denotes

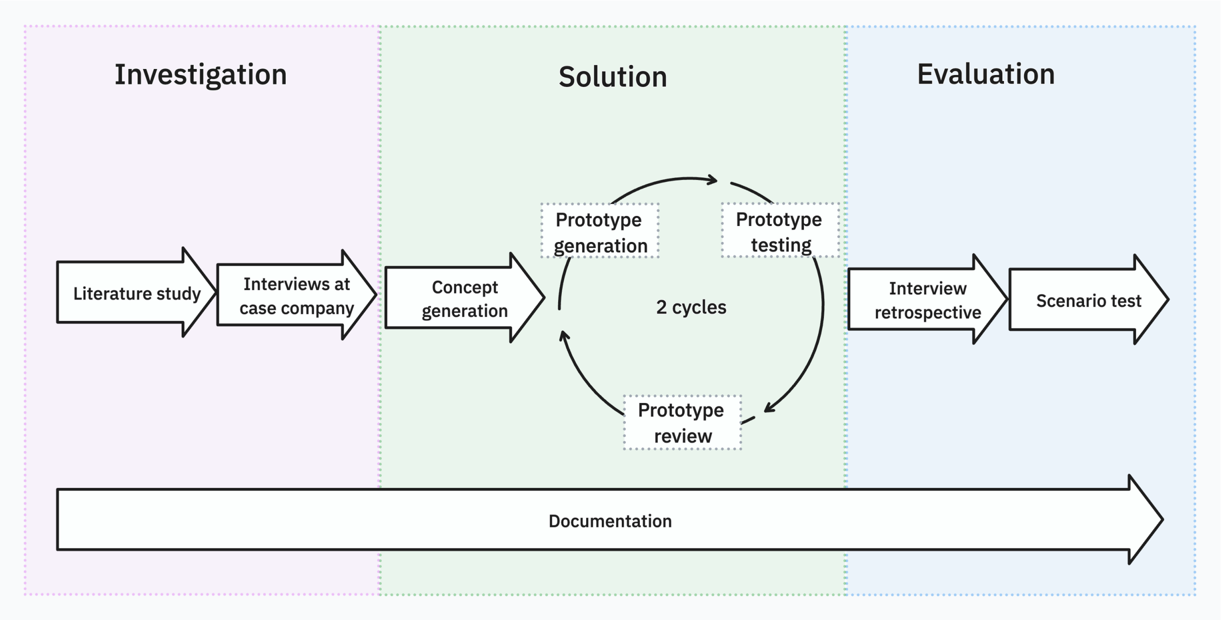


Figure 3: Research approach of the thesis project.

the step of creating the solution, *testing* the step of collecting data on how well it meets expectations, and *review* the iterative further clarification of the problem to be solved. The *solution* phase thus starts off directly with creating a proposal for solving the problem identified during the *investigation* phase.

3.1.1 Investigation

The investigative part of the chosen action research approach was carried out in two parts with one being a literature study, and the other being a needs analysis and user study.

3.1.1.1 Literature study

A literature study was carried out to give theoretical background on the topic of LLMs in software engineering, coding use cases as well as RAG. RAG as an optimization method was included in the search explicitly as it is foundational for being able to provide additional context to an LLM, and thus essential for developing such an application. The search was performed using a systematic search approach, amended with further sources found through *forward snowballing* (Wohlin, 2014).

A tentative start set was compiled through a block search in EBSCOhost (accessible as LUBSearch through Lund University Libraries), using three blocks comprised of the following keywords.

- ”LLM”, ”LLMs”, ”Large Language Models”, ”GPT-4”, ”GPT4”, ”GPT-3.5”, ”GPT3.5”, ”GPT”, ”ChatGPT”
- ”Benchmarking”, ”Benchmark”, ”Retrieval”, ”Retrieval-Augmented Generation”, ”Retrieval Augmented Generation”, ”RAG”
- ”Software”, ”Software Engineering”, ”Coding”, ”Code”, ”Programming”

Keywords within a block are joined with *OR*, blocks joined with *AND*, searching during the time window from 2010 to the day the search was performed, 2024-01-30. This start set included 724 resources, which was condensed through manual review based on relevance to a final set of 31 resources. From there on, further articles and resources cited within the start set were added to the study, resulting in a final list of 45 resources. The manual review screened for sources referring to modern LLMs, as literature on older language models was not deemed relevant any longer due to the fast pace of development. Additionally, resources not related to LLM optimization, LLMs as chat assistants or LLMs in software engineering were excluded. Additional resources cited in the screened start set that were deemed promising were subsequently also screened based on the stated criteria before being included.

In addition, a Google search was carried out to gain a comprehensive overview of existing commercial code-related LLM-based services to broaden the list from those referenced in academic resources. As necessary, further academic resources were consulted during the later stages of the thesis project to gain an understanding of the referenced concepts.

3.1.1.2 User study and needs analysis

Interviews were performed to gain an understanding of the current usage of LLM-based tools, user needs, and issues encountered in a relevant software engineering setting. Relevant interview subjects were identified within different teams at Craft Docs, all working within software engineering. Interview participants were selected to provide spread across seniority and main technologies used. During this initial round of interviews, 11 people participated. Each software engineering team member at the company received an interview request, and those who were able to participate during a two week period were interviewed, with reminders being sent to some individual to achieve a proportional distribution across the domains within the case company.

The chosen interview method was semi-structured, aiming to get an overview of LLM tools each individual uses as well as how often, and to gain a qualitative understanding of how well they work for different types of tasks individuals perform as part of their work. 30-minute interviews were carried out with each participant, a full list of interview questions can be found in Appendix A.2.1. The interview questions were compiled based on the results from the online search as part of the initial literature research.

Insights from the interviews were synthesised and coded to be presented in tables. In addition to this, an overview of the tools used and how they are used were summarized in a tree graph. Coding was performed by the authors through transcribing the interview, extracting relevant passages of text and grouping them by themes.

3.1.2 Solution

As the AR process focuses on implementing and evaluating a change or solution, it is described by Kemmis and Wilkinson (2002, p. 21) as ”a spiral of self-reflective cycles of” three steps, namely

- *plan* a change;
- *act* and observe the process and consequences; and
- *reflect* on the process and consequences.

This process is repeated multiple times, incorporating learnings from the previous cycle into the planning stage of the next (ibid., p. 21). AR is often performed with the goal to reach the intended improvement or change, and embed or permanent it in an organization (Höst, Regnell, and Runeson, 2006, p.41). In technical thesis projects such as this one, this is often done by developing a prototype and evaluating it, though with the process of permanenting or productizing the prototype lying outside of a thesis’ scope (ibid., p. 41).

For this thesis, *planning* a change consists of developing a concept for a solution based on results of the investigation phase - the *concept generation*. In subsequent stages, the planning step revolves around improving the solution based on reflection. *Acting* consists of developing the prototype and releasing it to users, the *prototype generation*. *Reflecting* on the results of the *prototype testing* provides an understanding of its performance and consequences for users, which during the prototyping stages corresponds to the *prototype review*, and afterwards to the final *evaluation*.

3.1.2.1 Concept generation

To guide the iterative development of a prototype, insights from the investigation phase are synthesized for two purposes. This is done to set goals on which the prototype solution can be evaluated later, and to gauge technical feasibility and limitations to inform the prototype development.

To choose goals that align with users’ needs, the concept of *needfinding*, popularized by Patnaik and Becker (1999), can be employed. According to the authors, the practice focuses on uncovering development opportunities that solve people's problems rather than focusing directly on specific solutions. This was done by choosing the most prevalent issues raised in interviews during the user study and formulating goals aimed at addressing them.

Gauging technical feasibility and limitations is important for ensuring that a working prototype is achievable with the available resources and existing technology. For this project, the feasibility study was conducted by synthesizing findings from the literature study, in terms of technologies intended to be used, already existing software addressing similar problems, as well as services that can be employed in a prototype.

3.1.2.2 Prototype generation

In accordance with the action research approach of iterative improvement, prototype generation is done in multiple rounds. During the initial round, the foundations for the prototype are set up and the first version should reach a stage where it is usable. The subsequent rounds aim at improving upon the initial version of the prototype, incorporating insights from feedback and addressing shortfalls raised during testing.

Initially, three rounds were planned. After the second round, a conclusion was reached that no further meaningful improvement to the prototype was achievable within the limitations and time frame of the thesis. Thus, in accordance with the principles of action research, the focus was shifted towards evaluating and analyzing.

3.1.2.3 Prototype testing

The prototype testing is conducted as part of iterations one and two, to gain insights regarding the prototype's performance, potential areas for improvement, and to uncover issues that may arise. These can all be of both technical nature, and regarding the user experience.

Testing is performed unsupervised, and software engineers at the case company are asked to employ the prototype in their actual work tasks, similarly to how they use ChatGPT or other LLM coding tools. This is to take a step away from the theoretical performance of the language model, and find shortfalls or opportunities in a real-world setting. To perform the testing, the prototype is made available to the participants via an email invitation, and they receive an account with which they can log in and use the prototype.

In addition to the testing done by the engineers at the case company, the model will also be tested by the authors in a structured manner. This will be done by executing scenario tests to benchmark the prototype.

3.1.2.4 Prototype review

During the prototype review, insights from the testing phases are compiled and potential causes for shortfalls are analyzed. As the results from the review phase are guiding the next prototype generation, this review should result in a list of shortcomings, their potential causes, and possible remediation to achieve improvement.

During the prototype review phase, short interviews were carried out with the software engineers that participated. Throughout the period, the authors requested interviews with those engineers that had used the prototype, with in total three participants of which one was interviewed twice, once in each round. These interviews were short and semi-structured, focused on extracting the most relevant information. Questions were prepared based on the goals set out during the concept generation phase and can be found in Appendix A.2.3.

3.1.3 Prototype evaluation

To understand the impact and consquences of a solution or change, an evaluation is necessary. For this project, evaluation will be performed by first creating evaluation criteria, and then evaluating the findings from the prototype using these criteria. The evaluation criteria are developed based on the results of the theoretical background, literature research and interview findings prior to and during testing. The evaluation criteria will be structured and summarised in a cohesive tree structure. The evaluation performed as part of the project is two-fold, with both being qualitative methods to extract the findings to be analysed using the criteria tree.

3.1.3.1 Scenario evaluation

The first evaluation method is based on creating software development scenarios, which are provided as test cases to the assistant. Analyzing its output and comparing it with a standard ChatGPT assistant provides insights into its quality. Focus was put on reasonably easy-to-achieve tasks, to provide a complement to the testing conducted by experienced software engineers at the case company.

3.1.3.2 Interview retrospective

The second evaluation method aims to provide insights into the prototype's performance in a real-world software development scenario, based on semi-structured interviews with software engineers at the case company. All software engineers who participated in prototype testing and met a usage threshold are offered a 30-minute interview to reflect on their experience. Engineers met the threshold if they had used the prototype intensively at least during one full project. During this phase, two interviews were performed as only two people had met the threshold and used the assistant in an actual work project. Interview questions for this third round of interviews were also based on goals set out for the prototype during the concept generation phase, though compared to the short interviews during the prototype review phase aimed to be more comprehensive.. The interview guide can be found in Appendix A.2.3.

For analyzing the interviews, relevant phrases from interview transcripts were retrieved and coded in the participant's language, grouped as "first-order concepts" as described by Maanen (1979). The resulting groupings are subsequently explained by the authors as more theoretical second-order themes, described by Maanen (ibid.) as "statements about relationships between certain properties observed to covary".

As described and employed by Corley and Gioia (2004), similar themes were grouped into overarching dimensions of interest, and visualized as a data structure in the format proposed by Corley and Gioia (ibid.). The emerging dimensions can then be compared or contrasted with theory.

3.2 Research quality

To both ensure and evaluate the trustworthiness of this thesis, a framework for quality in quantitative research originally proposed by Lincoln and Guba (1985) is employed. This framework encompasses four aspects for evaluating research quality, namely *credibility*, *transferability*, *dependability* and *confirmability*.

Establishing *credibility* demonstrates confidence in the findings' truth, meaning that the thesis reports and demonstrates facts and observations that align with the experiences and perceptions of participants and experts or members of a community. Lincoln and Guba (ibid.) propose techniques such as *triangulation* either by methods, using multiple data collection methods, or by sources, with continuous feedback loops with study participants over time, or adding additional outside perspectives for comparison. The authors also propose *peer debriefing*, which was performed continuously during the thesis project between the two authors.

Transferability is concerned with how applicable the findings are to a broader context, for example, a research field or an industry. Lincoln and Guba (ibid.) argue that investigating and describing a phenomenon in deep enough detail enables evaluating how and to which extent conclusions can be transferred to other settings and situations. As software engineering tasks and processes are similar across many settings, conclusions may have a broad range of applications.

Demonstrating *dependability* shows that the thesis' findings are consistent and repeatable. Lincoln and Guba (ibid.) describe external audits as a method for establishing dependability, which for this thesis is conducted through public discussion and peer feedback with other students, as well as the supervision and examination by researchers at Lund University.

Lastly, *confirmability* is concerned with demonstrating neutrality, thus showing that the results of the thesis are not affected by the researchers due to bias, interest or motivations (Lincoln and Guba, 1985). For the thesis, regular supervision by an experienced researcher at the university was used as a form of triangulation, to ensure that findings are based on the literature and data.

3.3 Use of generative AI

Generative AI-based services were used as part of the project to generate scripts used during the technical development (ChatGPT), to generate the cover image for this report (Midjourney), to proofread the report (ChatGPT and Grammarly), and give feedback on how to formulate sentences more clearly (ChatGPT and Grammarly).

3.4 Ethical considerations

According to Robson and McCartan (2016), it is important to avoid bias and maintain integrity in research. Integrity in research includes declaring personal bias, ensuring objectivity in research questions, attending possible conflicts of interest, avoiding observational bias, respecting intellectual property, as well as not fabricating, falsifying or misrepresenting data (ibid.). To comply with these guidelines, the research questions were formulated with input from the academic supervisor. Weekly external reviews in addition to approximately bi-weekly meetings were carried out with the academic supervisor to ensure oversight. It is to be noted that the action research approach can carry a certain bias towards positive outcomes. However, the industry supervisor remains objective, prioritizing overall organizational effectiveness over favoring certain technologies. To reduce observational bias, comprehensive notes were maintained, and selective attention and coding were refrained from.

Robson and McCartan (ibid.) also highlights the importance of participation in any study being voluntary, with full consent and also, if needed or desired, anonymity. This is achieved by making participation in the study fully voluntary with no repercussions for those who choose not to participate. Furthermore, insights from interviews and feedback are presented without naming.

4 Results

This chapter presents the results from the initial literature research and interviews, the generated concept as well as the prototype iterations. Finally, the results of the evaluation of the last prototype are presented.

4.1 Investigation

The following sections will present the findings from the literature study as well as the findings from the user study and needs analysis.

4.1.1 Literature study

This section introduces the concept of benchmarking to compare and evaluate LLMs and reports the performance of the most prevalent models. Additionally, methods for optimizing and tailoring LLMs to specific use cases are presented.

4.1.1.1 Benchmarking of Large Language Models

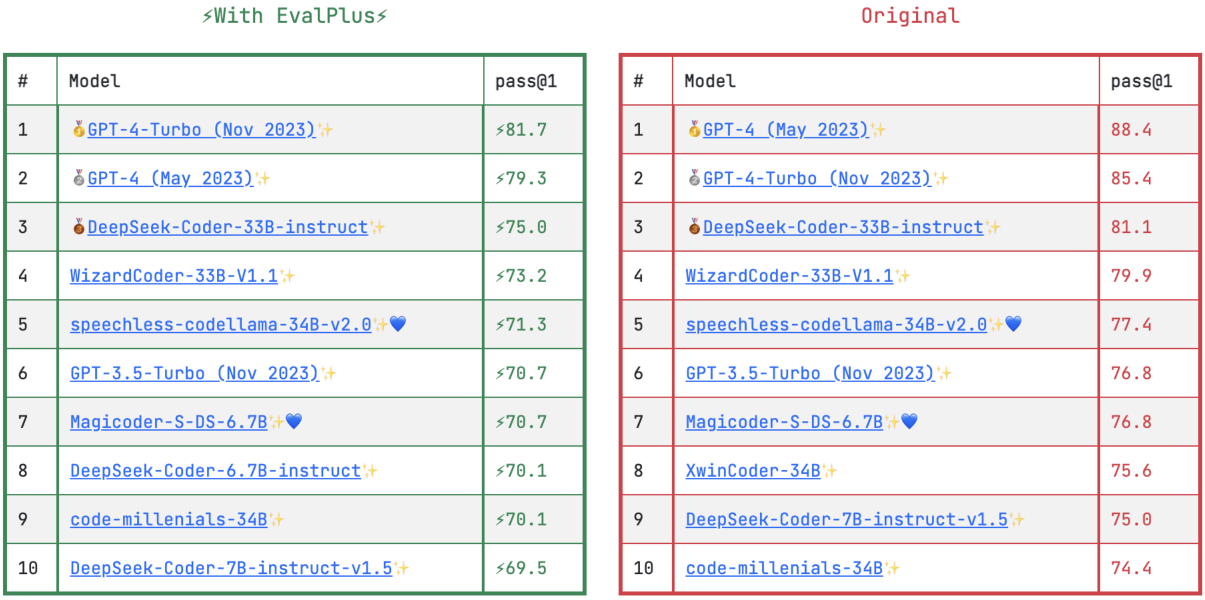
As language models are built on various sets of training data, have different focus areas in terms of applications and are employed in multiple fashions to support LLM-based tooling, there is a need to be able to compare the underlying models. Thus, to test the performance of LLMs on various types of tasks, researchers have compiled testing sets with tasks models can be evaluated on. These benchmarks range across various disciplines such as common-sense reasoning with *HellaSwag* (Zellers et al., 2019), multiple-choice tests across various subjects with *MMLU* (massive multitask language understanding) (Hendrycks et al., 2020) or grade school-level mathematics with *GSM-8K* (grade school math, 8k) (Cobbe et al., 2021). For coding tasks, the most prevalent benchmarks are *HumanEval*, first introduced together with the code-specific LLM *Codex* developed by researchers at OpenAI (Chen et al., 2021), as well as *CRUXEval* (Code Reasoning, Understanding, and eXecution Evaluation), introduced by researchers at Massachusetts Institute of Technology and Meta Platforms (Gu et al., 2024).

4.1.1.1.1 HumanEval

HumanEval is an evaluation set made up of 164 Python programming problems written manually, first introduced by Chen et al. (2021) at OpenAI. Solutions generated by LLMs can then be executed against a suite of tests created by the authors, with an average of 7.7 tests per problem. The authors describe the need for handwritten problems to arise due to many publicly available programming problems being part of the model's training data and thus possibly skewing results. They released the evaluation set publicly so that other models can be evaluated on it.

A *HumanEval* score is reported in many research papers introducing new large language models, with scores of 67.0% reported in the introductory technical paper for GPT-4 by OpenAI et al. (2023) at OpenAI, 74.4% for Gemini Ultra and 67.7% for Gemini Pro reported in the technical report for Google's LLM Gemini by Gemini Team et al. (2023).

The independent researchers Liu et al. (2023) have since expanded *HumanEval* with additional test cases, as much as 80 times per problem, to make the benchmark more rigorous and reliable. The authors also provide a comprehensive automated leaderboard (Liu et al., n.d.) for both the extended version called *EvalPlus* and the original *HumanEval* benchmark.



⚡With EvalPlus⚡

#	Model	pass@1
1	🥇GPT-4-Turbo (Nov 2023)✨	⚡81.7
2	🥈GPT-4 (May 2023)✨	⚡79.3
3	🥉DeepSeek-Coder-33B-instruct✨	⚡75.0
4	WizardCoder-33B-V1.1✨	⚡73.2
5	speechless-codellama-34B-v2.0✨💙	⚡71.3
6	GPT-3.5-Turbo (Nov 2023)✨	⚡70.7
7	Magicoder-S-DS-6.7B✨💙	⚡70.7
8	DeepSeek-Coder-6.7B-instruct✨	⚡70.1
9	code-millenials-34B✨	⚡70.1
10	DeepSeek-Coder-7B-instruct-v1.5✨	⚡69.5

Original

#	Model	pass@1
1	🥇GPT-4 (May 2023)✨	88.4
2	🥈GPT-4-Turbo (Nov 2023)✨	85.4
3	🥉DeepSeek-Coder-33B-instruct✨	81.1
4	WizardCoder-33B-V1.1✨	79.9
5	speechless-codellama-34B-v2.0✨💙	77.4
6	GPT-3.5-Turbo (Nov 2023)✨	76.8
7	Magicoder-S-DS-6.7B✨💙	76.8
8	XwinCoder-34B✨	75.6
9	DeepSeek-Coder-7B-instruct-v1.5✨	75.0
10	code-millenials-34B✨	74.4

Figure 4: Excerpt from the *EvalPlus* leaderboard (Liu et al., 2023) as of 2024-02-23, fair use under Apache 2.0 license.

As of 2024-02-23, GPT-4 Turbo and GPT-4 are the top two performing models, reaching 81.7% and 79.3% on *EvalPlus*. On the original *HumanEval* benchmark, GPT-4 performs better than GPT-4 Turbo at 88.4% and 85.4% respectively, as can be seen in figure 4. The higher scores than in the introductory report can be attributed to OpenAI releasing newer iterations of the models (OpenAI, n.d.[c]). Google's Gemini Pro ranks on place 18 as of 2024-02-23, with scores of 55.5% on *EvalPlus* and 63.4% on *HumanEval*, with various mostly open-source models placing in between. Gemini Ultra was not yet publicly available at the time of writing, thus no independent score can be found.

4.1.1.1.2 Other coding benchmarks

CRUXEval was developed by Gu et al. (2024) as an evaluation set additional to *HumanEval*, putting greater focus on not only code generation, but also reasoning surrounding programming problems and understanding of code. Coding problems in the benchmark are written in Python. Also for this benchmark, there is a publicly available leaderboard (Gu et al., n.d.). Similarly to *HumanEval*, GPT-4 is the best model on the benchmark at 75.5%.

As noticeable with the two described benchmarks, the main focus for evaluating programming performance lies on code in Python. For this thesis, it is also relevant to look at performance in other languages, as JavaScript with TypeScript and Swift are the most used languages at the case company. To evaluate performance in other languages, Yan et al. (2023) created *CodeScope*, evaluating both understanding and generation with individual coding problems in the most common programming languages. Depending on the sub-task in the benchmark, the researchers included up to 43 programming languages, including JavaScript and Swift. Similarly to the previously named benchmarks, GPT-4 places at the top with an overall score across all tests and languages of 39.2% on the leaderboard (Yan et al., n.d.).

All three benchmarks mentioned so far employ self-contained programming problems to test model performance. For this thesis, it is also relevant to understand how well models perform in a setting that is closer to the discipline of software engineering. There, projects contain dependencies across files and folders and generally have larger contexts, argue the researchers Li et al. (2024). Thus they compiled the benchmark *DevEval*, aiming to address this discrepancy. They describe that the benchmark contains problems better resembling those in real-world projects, a sufficient amount of inter-file dependencies and context across multiple files, with problems in Python. As shown in their paper, success rates for common LLMs are much lower on the benchmark, with the top-performing model being GPT-4 at 21.6%.

4.1.1.2 Best practices for optimizing and tailoring LLMs

The current best practices of the three most commonly used tools - prompt engineering, RAG and fine-tuning (Allard and Colin, 2023) - are presented below. As presented in Figure 5, the LLM can be improved along two axes, context optimization and LLM optimization, which refer to what the model needs to know and how the model needs to act (ibid.).

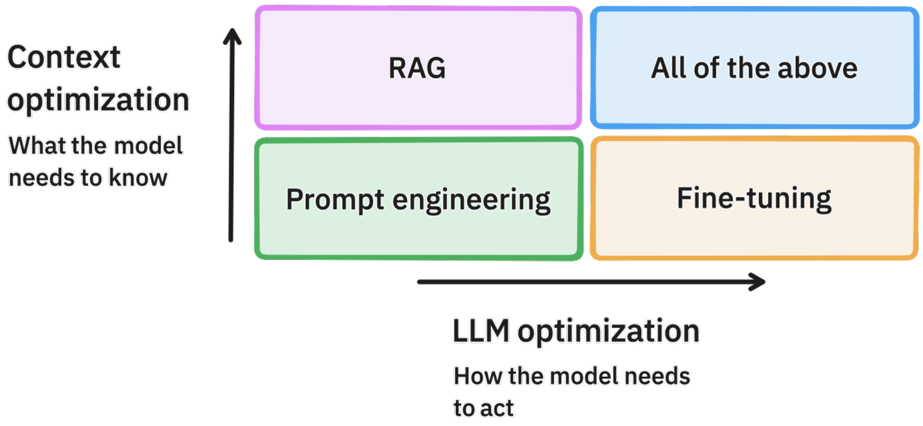


Figure 5: The LLM optimization flow, adapted from Allard and Colin (2023).

4.1.1.2.1 Prompt engineering

There exist several techniques that can be applied to the prompt in order to improve the generated answer. Examples of these are providing context, Chain of Thought (CoT), and Tree of Thought (ToT).

CoT is a prompting technique that instructs the GPT to reason about the problem in a multi-step fashion instead of jumping to the the conclusion directly (Wei, Tay, et al., 2022; Wei, Wang, et al., 2023; Yao et al., 2023). Although CoT has been shown to improve the quality of answers on some tasks, it does not show a significant increase across all test sets (Wei, Wang, et al., 2023). On StrategyQA, a test set that can be seen as similar to the task being attempted to achieve in this thesis, GPT-3 did not show any performance improvements when being prompted to use CoT (ibid.).

A further development of CoT is ToT, which prompts the GPT to perform deliberate decision-making (Yao et al., 2023). This is executed through iterative prompting, instructing the model to create and evaluate multiple threads and continuing the thread assessed by the LLM to be the most correct (ibid.). Research by Kadavath et al. (2022) has provided promising results of LLMs being able to evaluate the correctness of answers generated by the same model. ToT does however increase the cost per generated answer, due to it requiring the creation of multiple threads and the corresponding multiple sets of tokens needed for each thread (Yao et al., 2023).

4.1.1.2.2 Retrieval techniques

As shown by Li et al. (2024), LLM performance on code generation tasks with dependencies across files in a project is worse compared to stand-alone tasks. For LLM-based tools to have practical use in software engineering, capturing the nature of code, being modularized across files and exhibiting *local* context such as variable definitions as well as *global* context with function or API invocations across files, is of high relevance (Tu, Su, and Devanbu, 2014). Applying the concept of RAG to code completion and generation tasks can thus be expected to improve performance.

By basing the generated output on provided documents through RAG, answers get more specific and factual compared to non-RAG-powered models (Lewis et al., 2021), as well as reducing hallucinations (Barnett et al., 2024). Additional benefits of a RAG-powered model are many. Firstly, RAG-powered models have the benefit of not requiring any additional training when updating their knowledge and can be updated through hot swapping the retrieval data, making updating the knowledge base of models faster and less resource demanding compared to fine-tuning (Lewis et al., 2021). Secondly, RAG improves the semantic search capabilities of LLMs (Barnett et al., 2024). Thirdly, RAG systems are able to link sources to generated content (ibid.), providing a way to fact-check the answers more easily.

Already before the release of the current state-of-the-art LLMs, such as GPT-4, machine learning researchers at Microsoft have been able to show in previous studies (Clement et al., 2021) that providing the local context by incorporating the working file as part of the prompt can improve code completion. To address the missing global context and incorporate a wider context, applying retrieval-augmented generation techniques to code repositories was first proposed in the academic literature by F. Zhang et al. (2023), also affiliated with Microsoft.

Generally, retrieval techniques focused on code-related use cases outlined in research papers focus on two use cases: *completion*, meaning being able to generate the remainder of a line of code, or the next line based on previous lines, and *function generation*, being able to generate a complete function based on its name and parameters. This may be due to the prevalence of tools aiming to integrate LLM generation directly into IDEs, within which the actual code writing and editing is performed.

Two research groups have shown that specific RAG methods that work in a more sophisticated fashion than with a vector similarity search can have beneficial effects on code generation quality. These are *RepoCoder* by F. Zhang et al. (ibid.), a method employing multiple rounds of retrieval to improve the result, as well as *Repository-Level Prompt Generator* (RLPG) proposed by Shrivastava, Larochelle, and Tarlow

(2023), which works by accessing further files based on connections in the code, such as parent classes, referenced methods and more.

As presented by Barnett et al. (2024), the robustness of a RAG system is something that has to evolve through iterations rather than something that can be designed from the start. Furthermore, when constructing a RAG system, some key aspects that can improve the performance of the system are that:

- larger context allows for better results,
- semantic caching of provided content to reduce cost and latency, and
- adding metadata such as file name to improve retrieval.

For the prototype, to be created using OpenAI's available web tool, adding meta-data to the provided files is the only controllable aspect of the three. The file name and path will therefore be added before its corresponding code part in the files provided to the retrieval.

4.1.1.2.3 Fine-tuning

Fine-tuning can improve output when the goal is to optimise a model for a specific task which requires a domain-specific dataset for the model to be trained on (Allard and Colin, 2023). One way of creating this dataset is through reinforcement learning from human feedback (RLHF), which can be used for fine-tuning a language model to a broad set of questions (Ouyang et al., 2022).

Due to the diverse set of tasks that the generated prototype will have to handle, and the formatting of answers not being of main concern, fine-tuning is thought to have limited impact on the performance of the augmented LLM in this thesis. Furthermore, creating a dataset of high quality for the LLM to be fine-tuned on would require many hours of work from the engineers at the company. The same goes for RLHF. Also, the current generation of OpenAI Playground does not allow for simultaneous RAG and fine-tuning implementation.

4.1.2 User study and needs analysis

In this section, results from the user study are presented, with an overview of the observed usage of LLM-based services at the case company. Each of the different use cases found as part of the study are described in detail.

4.1.2.1 Overview of usage

The conducted interviews with software engineers at Craft Docs showcased the widespread use of both ChatGPT and other GPT API clients, so called *wrappers*. These function in the same way as the ChatGPT web application and were thus grouped together with ChatGPT as *GPT*. Also, Microsoft Copilot (Copilot) is used. However, nearly half of the interviewees reported Copilot not being applicable in their line of work due to its unavailability in the IDE they use.

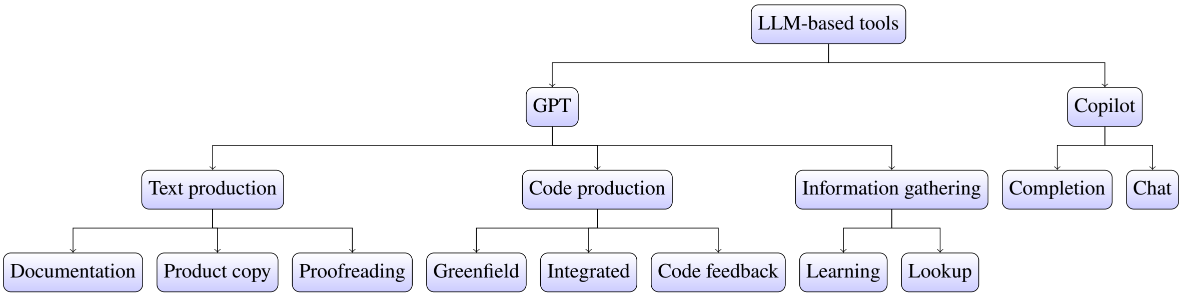


Figure 6: Overview of the LLM-based tools used and their respective use cases.

Figure 6 presents the found use cases which were grouped depending on which LLM-based tool was used by participants and their descriptions of tasks. Task descriptions were coded by the authors and grouped under common themes. Table 2 shows the usage distribution of the respective tools between interview participants.

Table 2: Usage distribution.

	Uses	**Does not use**	**Not applicable**
Copilot	IP2, IP4, IP7, IP11	IP1, IP3, IP6	IP5, IP8, IP9, IP10, IP12
GPT	IP2, IP3, IP4, IP6, IP7, IP8, IP9, IP10, IP11, IP12	IP1, IP5	

Specifically for GPT, which emerged as the most widely used tool at the case company, the use cases outlined in Figure 6 were analyzed in terms of users' sentiment and experiences, with details in the following paragraphs. Table 3 provides an overview of interview participants' experiences with each respective use case, and if they showed generally positive or negative attitudes towards it.

4.1.2.2 Text production

Although text production was not brought up as a use case for GPT by most interviewees, those mentioning it had an overall positive experience in this field. These participants frequently used it for proofreading and creating product copy, the text that goes into the toolbars, buttons and other parts of the product. GPT was also used for documentation and communication tasks. It is to be noted however that the results regarding text generation possibly do not show the full picture, since code generation was the focus of the interviews and no specific questions regarding text generation

Table 3: Positive and negative experience in regard to the use-cases found related to GPT.

	Positive	**Negative**
Text production		
Documentation	IP2	
Product copy	IP2, IP10, IP11	
Proofreading	IP10, IP11	
Code production		
Greenfield	IP2, IP3, IP4, IP7, IP9, IP10	IP1
Integrated	IP3, IP6, IP9, IP11	IP1, IP4, IP5, IP6, IP7
Code feedback	IP3, IP9	
Information gathering		
Lookup	IP2, IP3, IP4, IP6, IP8, IP9, IP11	IP5, IP7
Learning	IP3, IP6, IP7, IP8, IP10, IP12	IP7

were asked.

4.1.2.3 Code production

Participants exhibited mixed feelings regarding GPT's capabilities for code production, with a clear split between greenfield tasks and more complex integrated tasks. Greenfield tasks are task regarding code for areas where no previous work needs to be taken into account, or well-defined tasks that could be described as schoolbook examples. This was an area in which GPT generally excelled. Experiences were however mixed regarding GPT's capability in more complex integrated programming tasks, where the code needed had to fit into the context of already existing code and dependencies.

The mixed experiences regarding integrated code production can be attributed to several reasons, with the main one being GPT's lack of knowledge about the context. The lack of knowledge about what the participant in question was working on led to the need of having to provide extensive context to get a meaningful and useful result. Also, in the process of providing this context, it was mentioned as a regular occurrence that GPT interpreted it wrong, forcing the participant to start the process over again. GPT was also described as being misleading when tasked with helping to debug errors in code.

The limited scope of training data for GPT was also mentioned as an assumed cause for dissatisfaction. The vast majority of GPT's training data was created and collected before September 2021, with only small amounts of pre- and post-training data containing more recent content (OpenAI et al., 2023). This leads to GPT being mostly useless when working with cutting-edge technologies developed after this cut-off. Furthermore, although the training data of GPT-4 is extensive, it is not exhaustive, as demonstrated by a reported lack of knowledge about the intricacies of macOS and iOS operating systems and APIs, where GPT falls short according to interviewees.

Some participants did nonetheless exhibit positive experiences regarding using GPT for writing complex and integrated code. An example is for tasks where it is easy to define the APIs needed and give the context needed in a quick and efficient way. Another good use case mentioned was for creating rough prototypes. However, the code generated by GPT was however seldom ready to use and often required adjustments and iterations to work sufficiently well.

4.1.2.4 Information gathering

The experience of using GPT for information gathering was overall positive, with some exceptions. GPT was mainly reported to save a lot of research when performing explorations involving unfamiliar technology, as well as being able to work as a mentor within a new field or subject. Furthermore, it was reported to save time by presenting the result directly instead of having to find it on web pages after a search with a search engine. Negative experiences were mostly related to not being able to trust GPT due to it hallucinating. Some handled this by looking up the provided answer with a search engine to check its correctness. It was also reported that although answers could be somewhat incorrect, they usually pointed participants in the right direction. Another negative experience was related to the previously mentioned lack of knowledge due to the nature of GPT's training data not being exhaustive and mostly containing data originating before September 2021.

4.1.2.5 Usage across company-specific domains

What was made clear during the interviews was that the performance of GPT differs between different subject areas and programming languages. The more common the tasks and the more commonly used the programming language, the better the reported performance of GPT in general. It was also reported that GPT performed well in areas that could be considered more established practices, such as web applications. In contrast, GPT usually performed poorly on less common tasks and less widely used programming languages. In addition to this, it also seemed as if GPT had not received sufficient training data in some areas, such as programming Mac/iOS applications, since performance in that field was reportedly poor. An overview of the interview participants and which field they primarily work in can be seen in table 4.

Table 4: Usage of GPT across domains.

	Uses	**Does not use**
Web	IP2, IP3, IP4, IP9, IP11	IP1
Mac/iOS	IP8, IP9, IP10	IP5

4.2 Solution

This section will cover the results corresponding to the step of creating a solution in the action research approach presented by Höst, Regnell, and Runeson (2006). The solution will be in the form of a working prototype to be tested.

4.2.1 Concept generation

Before starting the actual development iterations, a concept was created according to the guidelines outlined in section 3 Method and in conjunction with the initial delimitations in regards to types of tasks and available LLMs.

Based on the interviews, it was decided to focus on function and component code generation for integrated use cases, as this emerged as the most ambivalent in the interviews. They require a lot of manual context input, and as shown some users felt existing tools worked for this process whereas others expressed not being content with what is available.

Within this task, four goals were set out for the prototype to accomplish, and to track its performance against. These are that the solution should

- require less detailed prompts;
- provide more accurate responses;
- require less back-and-forth iteration; and
- have an ease of use on par with current tools.

The goals were set based on results from the initial interviews. As participants expressed frustrations with the need for providing context manually, the goals focus on being able to use the chat assistant similarly to existing ones, but with less need for manual input. Goal 1 and 3 aim to lessen the user's need of providing information, whereas goal 2 and 4 aim to uphold or improve quality of the user experience.

The literature research has shown that there are tools available and some of them are already used within the case company. The tools aiming to address a similar problem or use case, incorporating a whole codebase context, are though currently not in use by anyone at the company. This could be due to organizational inertia, not having enough time to try out a new tool, or other organizational barriers, for example in regards to data protection.

In terms of technical limitations and feasibility, the concept of RAG combined with prompt engineering was chosen as the main way of augmenting a large language model with the necessary context. Through OpenAI, RAG is made available as a service. This has the benefit of being able to more easily implement a prototype, but the downside of not having knowledge and control over which retrieval techniques are employed. It was decided that as part of this thesis project, the benefit of speed and flexibility is more important than having exact control over retrieval techniques for possibly achieving better performance. With tailored prompting and instructions, some performance optimizations can be expected to be achievable.

Fine-tuning was not chosen as a tool for augmenting the LLM in the initial cycle. This is due to fine-tuning being more fit for improving model efficiency through emphasising knowledge that already exists in the model (Allard and Colin, 2023), which is not the case for this study. Furthermore, as recommended by Allard and Colin (ibid.), fine-tuning is not ideal to start with when iterating new use cases.

As the literature research has shown, GPT-4 and GPT-4 Turbo are the best LLMs and current state-of-the-art. The case company enabled access to these models and retrieval through OpenAI, namely via the *Assistant API*, so this approach was chosen to use in the prototype. Overall, this provides an opportunity to iterate quickly due to using existing RAG services, while at the same time reaping the benefits of state-of-the-art LLMs. To find the best way of incorporate the repository into the RAG system, an iterative approach was chosen.

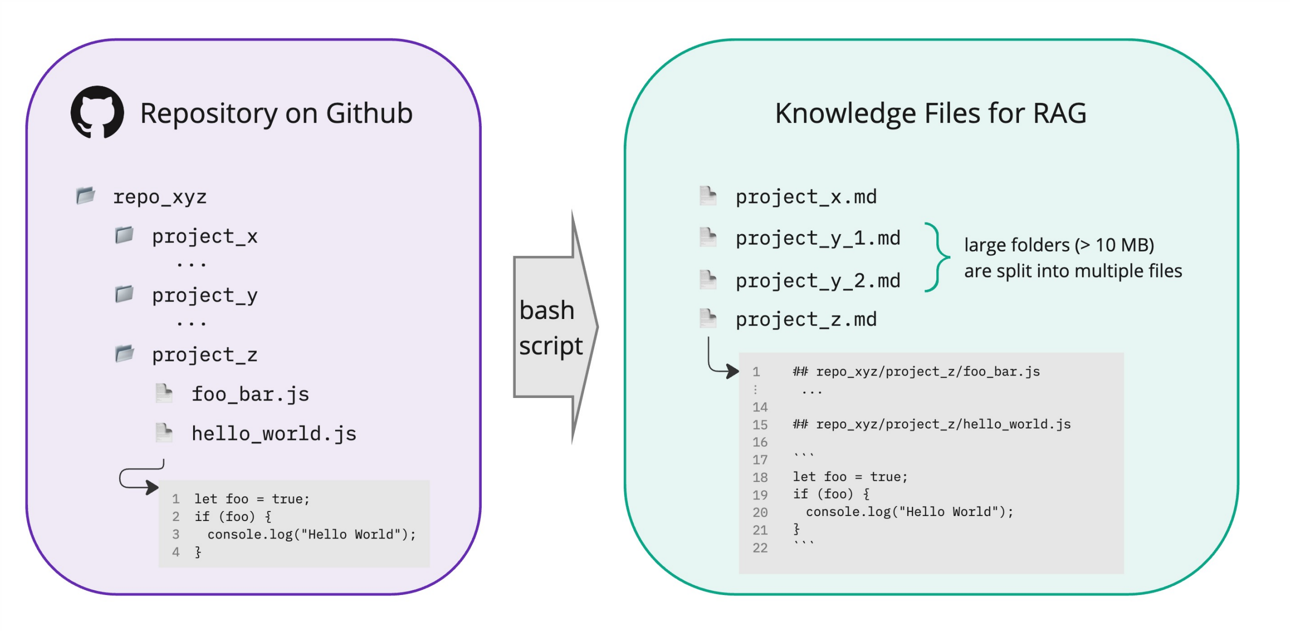
4.2.2 First iteration - prototype generation

In line with the results from the investigation phase as well as the scope and delimitation outlined in the introduction, the prototype for the coding assistant was created within OpenAI's environment, based on the Assistant API's specification, and following its possibilities and limitations. The OpenAI Playground (OpenAI, n.d.[h]) provides a graphical user interface (GUI) through which users can interact with the prototype.

4.2.2.1 Retrieval-Augmented Generation

For the initial prototype, data was retrieved from Craft's web application repository that contained all front-end code related to the React-based web application written in TypeScript. This data was processed to meet the requirements and limitations of OpenAI's Assistant API specifications, namely being limited to a maximum number of 20 files and 2,000,000 tokens per file (1 token corresponds to approximately 4 characters). Data processing included merging of individual files into larger *markdown* text files, as well as adding metadata for each individual file in the form of the file path.

As the repository is a so-called *monorepo* containing several projects in a separate folder each, these base folders were merged into one file each with the exemption



```
let foo = true;
if (foo) {
  console.log("Hello World");
}
```

~~~~
## repo_xyz/project_z/foo_bar.js
 ...

## repo_xyz/project_z/hello_world.js

```
let foo = true;
if (foo) {
  console.log("Hello World");
}
```
~~~~

Figure 7: Overview of the data processing.

of those which exceeded the token limit, which had to be split into two files. Furthermore, delimiters in the form of backticks were added to mark the start and end of each merged code file. All of the data processing was executed with the help of a command line script (see Appendix B.1) as a way of streamlining the workflow for setting up the RAG system. The script was compiled with the help of ChatGPT. Figure 7 visualizes this process.

Retrieval works as a *tool* the language model has access to, with the model's system prompt providing instructions on how to call tools and retrieve their output (OpenAI, n.d.[a]). The system prompt containing these instructions is not published by OpenAI, and can therefore not be examined.

4.2.2.2 Prompt engineering

For the first version of the prototype, the initial prompt (see Appendix B.2) created was comprised of three high-level concepts: the assistant's overall purpose, what files it has access to and how those are structured, and how the files are to be interpreted and incorporated into its answers.

4.2.2.3 Technical description of the assistant

When the first message is sent to the assistant by a user or tester, a *thread* is created. This thread forms the basis of a conversation, to which all instructions, messages, retrieved chunks of code and responses are appended, and each step is encoded as an *event* on the thread. The assistant prototype has a unique ID tied to the thread and the thread itself receives a unique ID. In addition, the chosen language model (GPT-4 Turbo), the instructions (assistant prompt), references to the knowledge files outlined earlier and the instruction to use the retrieval tool are stored within the thread. Potential errors, start and end times, and usage statistics are also appended. An example of a completed thread object can be found in Appendix B.3.

```
events: {
    thread: {
        run: [
            created,
            queued,
            in_progress,
            completed,
            step: [
                created,
                in_progress,
                completed
            ]
        message: [
            created,
            in_progress,
            delta,
            completed
        ]
    }
}
```

Example 1: Event types

The thread has two types of objects tied to it, *runs* and *messages*. After the thread is created with the first message, a run is queued on OpenAI's servers, and then started. This run orchestrates the retrieval and generation, each as a *step* with its respective type, either `tool_calls` or `message_creation`. Messages are appended to the thread and receive their ID. As seen in Example 1, runs and messages go through different states during the process. In the background, the language model coordinates the tool calls, assembles a final prompt containing the user's prompt and retrieved chunks, and generates the response (OpenAI, n.d.[e]). According to the documentation, in case multiple tools are made available to the model, it can ”intelligently” choose between them depending on the user's prompt (ibid.).

In Figure 8, the complete process is visualized as a swimlane diagram, outlining how the different entities interact. Therein, *Assistant* refers to the model on OpenAI's backend orchestrating the retrieval and generation process, and *LLM* to the model generating the final output. Figure 9 shows the user interface.

4.2.3 First iteration - prototype testing

Initial testing was carried out by the authors before releasing the prototype to the software engineers at the company. This was done to ensure the answers provided by the prototype were at least somewhat accurate and to evaluate the level of hallucination by the prototype. This showed that returned code examples could be found within the repository, showing that the retrieval mechanism worked as expected. Providing the complete file path of each referenced piece of code worked intermittently, sometimes the path was not mentioned at all, and sometimes only the file name. In some cases, the file names were made up, with the model hallucinating file names and paths that did not exist.

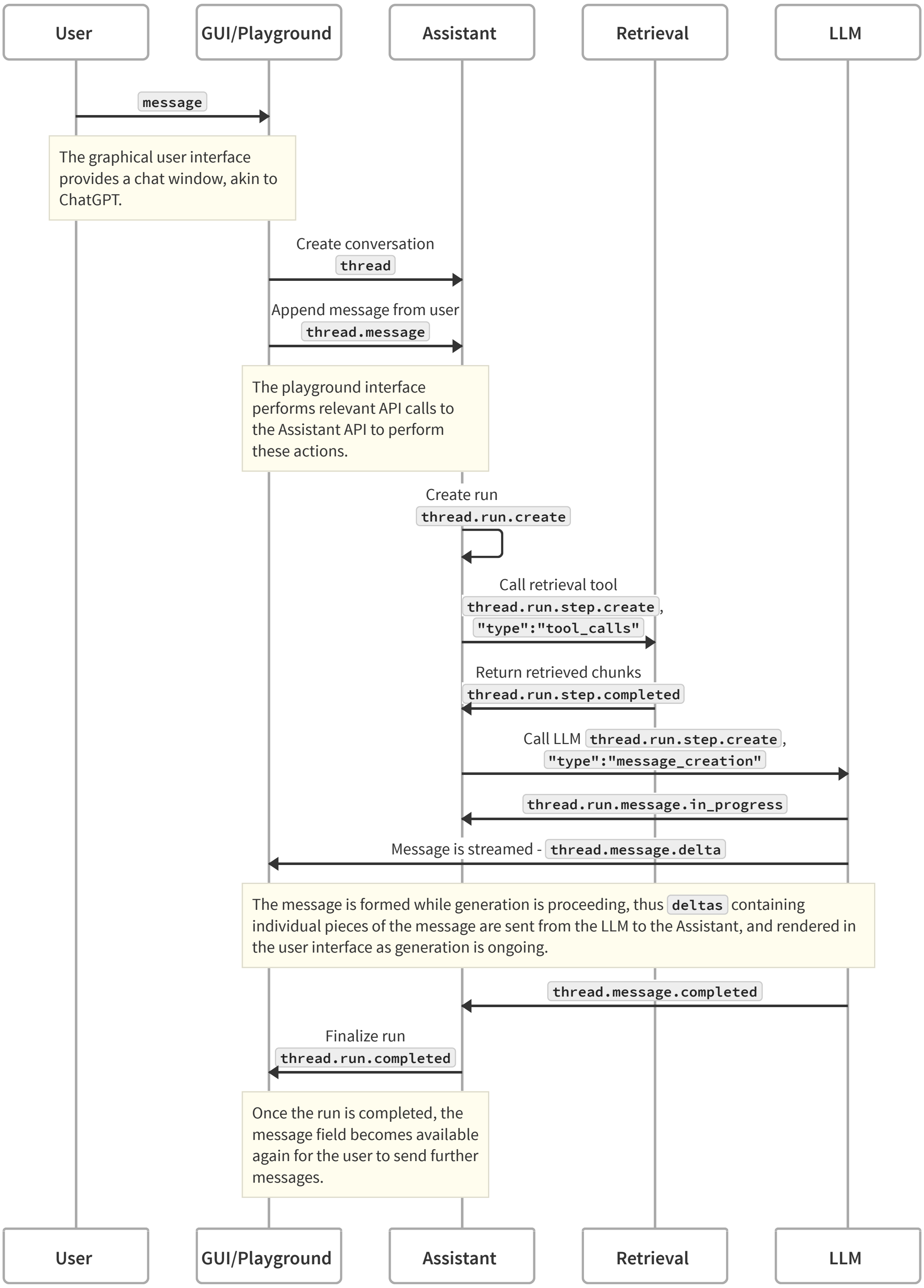
Testing performed by the software engineers at the case company revealed that the main issues were response speed, and the assistant intermittently not providing an answer even though the process was marked as complete. Unfortunately, during the first round, engagement with the prototype was rather low, which can also be attributed to its slow response speed.

4.2.4 First iteration - prototype review

One of the main points of feedback during the first iteration was that the prototype was slow. This might have been partly due to the reported issues that OpenAI had during the testing period, with reported incidents on February 13 and 14 (OpenAI, n.d.[g]). Furthermore, sometimes the prototype did not provide any answer, something that also might have been due to the reported incidents at OpenAI.

Another issue with the first iteration was the lack of participation from the software engineers. This was largely due to the bad timing of the first iteration cycle that coincided with a part of the company's software development cycle that had little to no need for the tool. This led to the first iteration cycle being mainly focused on further literature research and testing by the authors with some input from the engineering team.

Even though the model had access to the information needed it sometimes still hallucinated and provided made-up answers. This included made-up file paths and function names. However, most of the time the prototype seemed to retrieve the relevant


User
GUI/Playground
Assistant
Retrieval
LLM
`message`
Create conversation `thread`
Append message from user `thread.message`
Create run `thread.run.create`
Call retrieval tool `thread.run.step.create`, `"type":"tool_calls"`
Return retrieved chunks `thread.run.step.completed`
Call LLM `thread.run.step.create`, `"type":"message_creation"`
`thread.run.message.in_progress`
Message is streamed - `thread.message.delta`
`thread.message.completed`
Finalize run `thread.run.completed`
User
GUI/Playground
Assistant
Retrieval
LLM

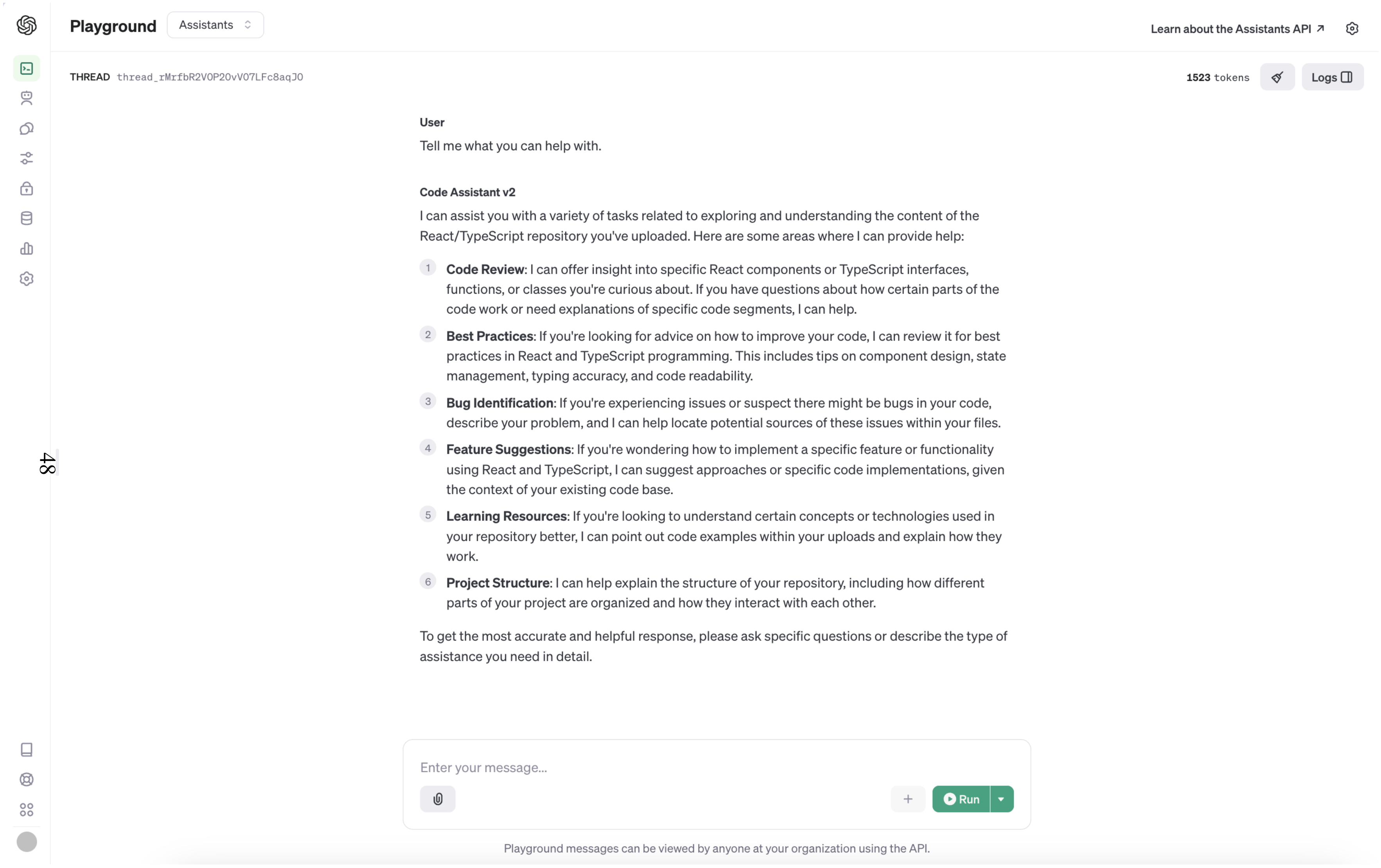
The graphical user interface provides a chat window, akin to ChatGPT.

The playground interface performs relevant API calls to the Assistant API to perform these actions.

The message is formed while generation is proceeding, thus `deltas` containing individual pieces of the message are sent from the LLM to the Assistant, and rendered in the user interface as generation is ongoing.

Once the run is completed, the message field becomes available again for the user to send further messages.

Figure 8: Swimlane diagram of the retrieval and generation process.


Playground
Assistants
Learn about the Assistants API ↗
THREAD thread_rMrfbR2V0P20vV07LFc8aqJ0
1523 tokens
Logs
Enter your message...
Run

User

Tell me what you can help with.

Code Assistant v2

I can assist you with a variety of tasks related to exploring and understanding the content of the React/TypeScript repository you've uploaded. Here are some areas where I can provide help:

1. **Code Review**: I can offer insight into specific React components or TypeScript interfaces, functions, or classes you're curious about. If you have questions about how certain parts of the code work or need explanations of specific code segments, I can help.
2. **Best Practices**: If you're looking for advice on how to improve your code, I can review it for best practices in React and TypeScript programming. This includes tips on component design, state management, typing accuracy, and code readability.
3. **Bug Identification**: If you're experiencing issues or suspect there might be bugs in your code, describe your problem, and I can help locate potential sources of these issues within your files.
4. **Feature Suggestions**: If you're wondering how to implement a specific feature or functionality using React and TypeScript, I can suggest approaches or specific code implementations, given the context of your existing code base.
5. **Learning Resources**: If you're looking to understand certain concepts or technologies used in your repository better, I can point out code examples within your uploads and explain how they work.
6. **Project Structure**: I can help explain the structure of your repository, including how different parts of your project are organized and how they interact with each other.

To get the most accurate and helpful response, please ask specific questions or describe the type of assistance you need in detail.

Playground messages can be viewed by anyone at your organization using the API.

Figure 9: View of the assistant prototype interface in the OpenAI Playground.

chunks of code from the provided files given the answers it provided. Furthermore, the prototype was able to provide answers that showcased this retrieved information being used in a correct way that provided helpful assistance to real-world coding problems in an integrated environment.

4.2.5 Second iteration - prototype generation

During the second prototyping round, the process for setting up retrieval from the initial round was repeated. This was done to update the data to the current state of the codebase and ensure that changes made by software engineers during the testing and evaluation phases are incorporated into the assistant. To make it easier for the model to provide the correct file path, the full file path was provided in the markdown file headers.

Regarding the assistant's prompt, changes were made according to the improvement opportunities or shortcomings uncovered as part of the first iteration's testing. The instructions were amended to make sure that the model provides the full file path of code files that are referenced, without providing the names of the large internal knowledge files. During initial testing, it was determined that one instruction was not enough, and a second instruction with a strong negation (*"absolutely never"*) needed to be added to ensure the file names for internal use were ignored and the file paths within the knowledge files are used instead. A *chain of thought* instruction was also added to the prompt, asking the model to explain its reasoning step-by-step.

With the aim of improving the credibility of the answers generated, experiments were made adding an instruction prompting the model to evaluate its answer and provide information on how accurate it believes the answer to be. The model did not follow this instruction, and rather, this had detrimental effects, as other instructions were ignored when this was present. Thus, this was not implemented.

With the goal of improving the model's understanding of the repository, an additional file containing a file tree outlining the folder and file structure of the repository was added to the knowledge files and instructions included in the prompt. This did not yield any results though, as the model did not seem to access this file. Thus, this idea was discarded.

4.2.6 Second iteration - prototype testing

During the second testing period, no larger issues with outages were reported, neither by software engineers at the case company that tested the prototype, nor by OpenAI directly. In the meantime, OpenAI had also upgraded the assistant framework's capabilities, so that instead of a blank page while waiting for information retrieval and generation, a loading animation is displayed. Previously, the generated answer was displayed when it was finished, while now, the answer is streamed directly into the user interface window, in the same manner as on ChatGPT.

As in the first round, the assistant prototype was released to the testers, with reminders asking for participation sent individually. Compared to the first round, further engineers participated, and according to both usage statistics as well as self-reports used the prototype more thoroughly than during the previous testing period, across a range of tasks.

4.2.7 Second iteration - prototype review

While no issues with abnormal long loading times similar to those in the first iteration were reported, software engineers who participated in the test still mentioned that it is slow compared to ChatGPT. The quality of the generated answers was rated as similar to ChatGPT for code-heavy questions. Though, it was observed that engineers testing the assistant did not necessarily change their behavior with the assistant prototype, and similar to how they use ChatGPT pasted code snippets into their prompts.

In regards to questions requiring a larger amount of reasoning, testers reported trying to ask higher-level project-related questions, but did not feel that the assistant was able to provide answers they would rate as truthful or reliable. Similarly, when using the assistant prototype to help write business logic-related code, their hope that the assistant would suggest patterns globally used across the repository was not met, as the assistant did not suggest any such patterns.

4.3 Prototype evaluation

The evaluation starts with the construction of the evaluation tree to be used as the criteria framework. This is followed by presenting the results and evaluation of the comparison tests. Finally, the qualitative analysis of the interviews is presented and evaluated.

4.3.1 Evaluation tree

For this evaluation to be structured and comprehensive, the different parts of the prototype and its functionalities were broken down into a tree structure that can be seen in Figure 10. The leaves to be included in testing are leaves 1-8, with leaves 9-10 being excluded due to the black box characteristics of the RAG making it impossible to evaluate chunking properly. The tests will be created to resemble real-world scenarios similar to those that would occur in production.

Splitting the prototype into two areas, LLM and RAG, is motivated by the definitions presented by Mialon et al. (2023) where an Augmented Language Model is split into the parts reasoning and tools. With the LLM being responsible for the reasoning, and the RAG being a tool, these definitions correspond to the split in Figure 10. Furthermore, with prompt engineering being strongly connected to the behavior and reasoning capabilities of the LLM, the effects of prompt engineering will be evaluated with the help of the leaves under LLM in Figure 10.

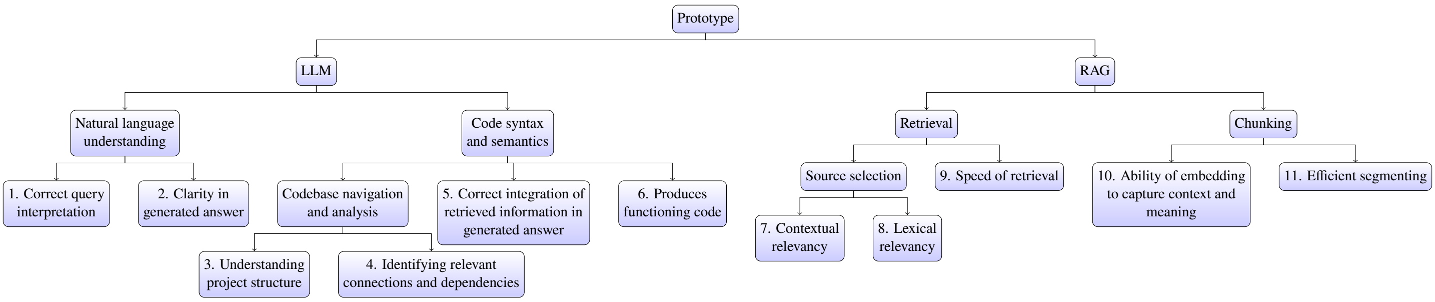

Prototype
LLM
RAG
Natural language understanding
Code syntax and semantics
Retrieval
Chunking
1. Correct query interpretation
2. Clarity in generated answer
Codebase navigation and analysis
5. Correct integration of retrieved information in generated answer
6. Produces functioning code
3. Understanding project structure
4. Identifying relevant connections and dependencies
Source selection
9. Speed of retrieval
7. Contextual relevancy
8. Lexical relevancy
10. Ability of embedding to capture context and meaning
11. Efficient segmenting

Figure 10: Overview of the structure and functionalities the prototypes will be evaluated on.

RAG is split into two distinct areas, chunking and retrieval. Barnett et al. (2024) describes these two areas as the index and query process. The chunking is in turn split into two leaves to cover both the ability to segment the files in an efficient and correct manner, as well as producing embeddings that fully capture the chunks' meaning and dependencies.

The retrieval part is split into selecting the correct sources and speed of the retrieval. Source selection is split into contextual and lexical relevancy to evaluate the retriever's ability to find results matching both in text and semantics. Speed is relevant as a metric due to speed being one of the main points of feedback from the engineers.

Mialon et al. (2023) describes the possibility of dividing reasoning into multiple types of different reasoning. For this thesis, natural language, and code syntax and semantics are the most relevant areas of reasoning to evaluate. The natural language understanding is split into correctly interpreting the question and generating an understandable answer. Understanding of code syntax and semantics is split into codebase navigation and analysis, correct integration of retrieved information in the generated answer, and producing functioning code. Codebase navigation and analysis is split into understanding the project structure and identifying relevant connections and dependencies.

4.3.2 Scenario evaluation

Oriented around the elements of the evaluation tree, questions for this test were chosen to resemble real-world scenarios and to test the different capabilities of the prototype. Elements 1 through 8 in Figure 10 were determined to be viable to test with this testing framework. Additionally, the prototype's capabilities compared to ChatGPT can be clarified and easily understood. The first two questions evaluate the assistant's capability on static changes to the visible user interface, with the first being a text change, and the second a more complicated ordering change. The third question aims to change a visible element in combination with a change in underlying logic. The final two questions are aimed at evaluating the assistant's capabilities in understanding underlying logic of something that does not produce a visual change, with one being open-ended and the last one being targeted and specific. A summary of the leaves evaluated for each question is provided in Table 5.

Table 5: Leaves evaluated for each question.

Question	**Leaves evaluated**
Question 1	1, 2, 5, 6, 7, 8
Question 2	1, 2, 3, 5, 6, 7, 8
Question 3	1, 2, 3, 4, 5
Question 4	1, 2, 3, 7, 8
Question 5	1, 2, 3, 4, 5, 7, 8

4.3.2.1 Question 1

This question again focuses on finding the correct piece of code corresponding to a description of the user interface, testing the lexical and contextual relevancy of the retrieval process, as well as the LLM's ability to interpret the query, provide a clear answer, and correctly integrate the retrieved information. As there is only one preference panel containing an integrations section, this question does not test codebase navigation and analysis capabilities.

Prompt

In the Integrations section of the preferences panel, the learn more link is currently in title case. Help me change it to sentence case instead.

ChatGPT response

To change the ”Learn More” link in the Integrations section of the preferences panel to sentence case, follow these steps:

1. Access the Preferences Panel: Open the application or website where you need to make the change. Navigate to the preferences or settings section.
2. Locate the Integrations Section: Within the preferences or settings, find the section labeled ”Integrations” or something similar.
3. Edit the Text: Look for the ”Learn More” link. You will need to access the backend or content management system (CMS) where the text for this link is stored or configured.
4. Change to Sentence Case: Modify the text from ”Learn More” to ”Learn more”. Ensure you only capitalize the first letter of the first word, and all other letters are in lowercase, except for proper nouns or as grammatically required.
5. Save Your Changes: After making the edits, save your changes. It might be necessary to review your changes for accuracy and to ensure it adheres to the overall style guide of the application or website.
6. Test the Change: Go back to the preferences panel in the application or website to ensure the changes have been applied successfully. Verify that the link now reads ”Learn more” in sentence case.

If you do not have direct access to the backend or CMS, you may need to contact your web developer or the technical team responsible for maintaining the application or website. Provide them with specific instructions about the change you wish to make.

If there are specific software or tools involved that you'd like guidance on, such as a particular content management system or code snippet examples, please provide more details so I can offer a more tailored response.

Assistant response

I found the section of the code responsible for rendering the ”Learn More About Integrations” link in the Integrations section of the preferences panel. It is located in the file `editor/src/webapp/preferences-panel/sections/Integrations.tsx`.

To change the label to sentence case, you need to update the `LinkLabel` component's content from ”Learn More About Integrations” to ”Learn more about integrations”. Here's how you can make that change:

Original code:

```
<SectionAlignStart>
  <ExternalLink href={CRAFT_INTEGRATIONS_HELP_ARTICLE_URL}>
    <LinkLabel>Learn More About Integrations</LinkLabel>
  </ExternalLink>
</SectionAlignStart>
```

Updated code:

```
<SectionAlignStart>
  <ExternalLink href={CRAFT_INTEGRATIONS_HELP_ARTICLE_URL}>
    <LinkLabel>Learn more about integrations</LinkLabel>
  </ExternalLink>
</SectionAlignStart>
```

Just make sure to keep everything else the same and only change the text content of the `LinkLabel` component.

ChatGPT provided a generic answer containing high-level instructions in how to work with a content management system and directing the user to talk to their technical team.

The assistant prototype was able to provide a clear answer, referring to the correct part of the codebase, and providing correct instructions on how to make the requested change.

Compared to questions 3 and 5, it is noticeable that the assistant is more successful with requests lacking ambiguity. Ambiguities either introduced by prompting, or pre-existing in the codebase, seem to increase the difficulty of retrieving the correct files, and lead the LLM to provide a confident answer even though it could be misleading, as other contextually relevant files or code snippets are not taken into account.

4.3.2.2 Question 2

This question aims to manipulate the appearance of a menu, once again probing the retrieval process' ability to understand both the lexical and contextual meaning of what menu is referred to, what part of that menu, and retrieve the corresponding code chunks correctly. Regarding the LLM generation, interpretation of the query in terms of what is meant with *order*, as well as the clarity and correctness of the provided response is tested.

Prompt

In the Info Page in the side menu, I want to change the order of Created, Updated and Author in the Properties section. Author should come first before the dates.

ChatGPT response

To change the order of ”Created,” ”Updated,” and ”Author” in the Properties section of your Info Page in the side menu so that ”Author” comes before the dates, you will likely need to modify the HTML or template code that generates this page, depending on the platform or system you are using. Here's a general approach you can follow:

1. Locate the Template or HTML File: Find the template or HTML file that renders the Info Page. This could be part of a content management system (CMS), a static site generator, or custom HTML for a web application.
2. Identify the Properties Section: Within the file, look for the code that generates the Properties section. This might be in a loop or a series of statements that output the ”Created,” ”Updated,” and ”Author” information.
3. Rearrange the Code: Change the order of the statements or template tags that output these properties so that ”Author” comes first. For example, if the original order is something like this:

```
    <p>Created: {{ page.created }}</p>
    <p>Updated: {{ page.updated }}</p>
    <p>Author: {{ page.author }}</p>
```

   You would rearrange it to:

```
    <p>Author: {{ page.author }}</p>
    <p>Created: {{ page.created }}</p>
    <p>Updated: {{ page.updated }}</p>
```

4. Test Your Changes: Save your changes and test the Info Page to ensure that the properties now appear in the correct order. Make sure to check for any unintended side effects or formatting issues.
5. Deploy Your Changes: Once you're satisfied with the appearance and functionality, deploy the changes to your live site or application, following your usual deployment process.

If you’re working within a specific CMS (like WordPress, Drupal, Joomla) or using a web development framework (like React, Angular, Vue), the exact steps and code syntax may vary. Consult the documentation for your specific platform or framework for guidance on modifying templates or HTML structures.

Assistant response

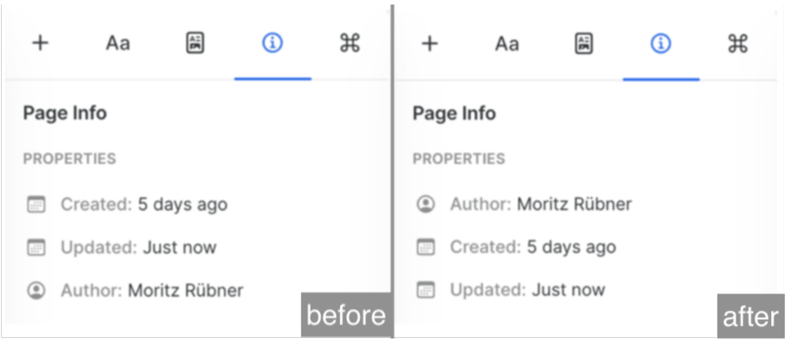
To change the order of ”Created,” ”Updated,” and ”Author” in the Properties section of the Info Page in the side menu so that ”Author” comes before the dates, you need to adjust the order of their rendering in the `PropertiesSection` component.
Currently, in the `editor/src/uiview/sidemenu/SideMenuInfoPage/PropertiesSection.tsx` file, the order is:

```
<DataItem label="Created" data={createdStr} icon={Calendar} />
<DataItem label="Updated" data={updatedStr} icon={Calendar} />
<AuthorItem firstName={firstName} lastName={lastName} />
```

To have ”Author” show up first, you should adjust it to:

```
<AuthorItem firstName={firstName} lastName={lastName} />
<DataItem label="Created" data={createdStr} icon={Calendar} />
<DataItem label="Updated" data={updatedStr} icon={Calendar} />
```

This change moves the rendering of the author information to the top, followed by the created and updated dates accordingly.



Page Info

PROPERTIES

Created: 5 days ago

Updated: Just now

Author: Moritz Rübner

Page Info

PROPERTIES

Author: Moritz Rübner

Created: 5 days ago

Updated: Just now

Figure 11: Section in the Information menu.

In this example, we can see that ChatGPT was able to provide a very high-level guide on how to change the order of HTML elements, instructing the user to find the relevant piece of code and take framework-specific considerations into account. The assistant prototype was able to find the relevant file, thus showing that the contextual search found the menu and its subpage, correctly provided the path to it, and described the necessary changes to make.

Implementing the change, we can see that the instructions were correct and led to success, as can be seen in Figure 11. The assistant prototype’s response is brief, but contains all necessary details, demonstrating correct query interpretation and clarity in its answer.

4.3.2.3 Question 3

The third question aims to change a visible front-end behavior with the purpose of testing how well the prototype is able to interpret the visual output generated by the code. Therefore, it is asked to change an observable user interface element and behavior. Additionally, as user interfaces often rely on dependencies in how components are defined, it makes it possible to gain insights into how well the prototype is able to understand the structure of the project and identify relevant connections.

Prompt

I want to change the default colors in the formatting tab of the right sidebar. How can I do that?

ChatGPT response

To provide you with the most accurate and up-to-date instructions, I need to know which software or application you're referring to. The ”formatting tab in the right sidebar” could apply to several different programs, such as Microsoft Word, Google Docs, or even a specific design tool. Could you clarify the software you're using?

Assistant response

To change the default colors in the formatting tab of the right sidebar, you should update the colors specified in the `stylePanelMainColors` array. This array is used to define the main colors available for styling within the right sidebar's formatting tab.

For example, the following snippet defines a set of color pairs used in the sidebar:

```
const stylePanelMainColors = [
  refinedTextColorPairs[1],
  refinedTextColorPairs[7],
  refinedTextColorPairs[4],
  refinedTextColorPairs[37],
  refinedTextColorPairs[12],
  refinedTextColorPairs[17],
  refinedTextColorPairs[22],
  refinedTextColorPairs[32],
  refinedTextColorPairs[42],
  refinedTextColorPairs[47],
  refinedTextColorPairs[45],
];
```

Each entry in the `stylePanelMainColors` array references a color pair from `refinedTextColorPairs`, which eventually maps to specific color codes defined earlier in your code base. The `refinedTextColorPairs` itself is derived from an earlier defined color arrangement, combining colors for both light and dark mode.

To change the default colors, you need to either modify the referenced indexes in `stylePanelMainColors` to point to different pairs in `refinedTextColorPairs` or directly change the color codes in the underlying data structure from which `refinedTextColorPairs` is built.

For instance, if you wish to add a new color pair or change an existing one in `refinedTextColorPairs`, you would adjust or add the color codes accordingly in the related arrays. Afterwards, make sure the `stylePanelMainColors` points to the correct indexes of `refinedTextColorPairs` to reflect your changes in the right sidebar's formatting tab.

ChatGPT was not able to provide a relevant answer, as it does not have any contextual knowledge regarding what project, software, framework or programming language is referred to. The prototype understood what part of the user interface was referred to (correct query interpretation), and following the instructions of the assistant, it was easily possible to change the array index of one of the `stylePanelMainColors` to a different one and running the application, as it generated a clear answer, correctly based on the retrieved pieces of code.

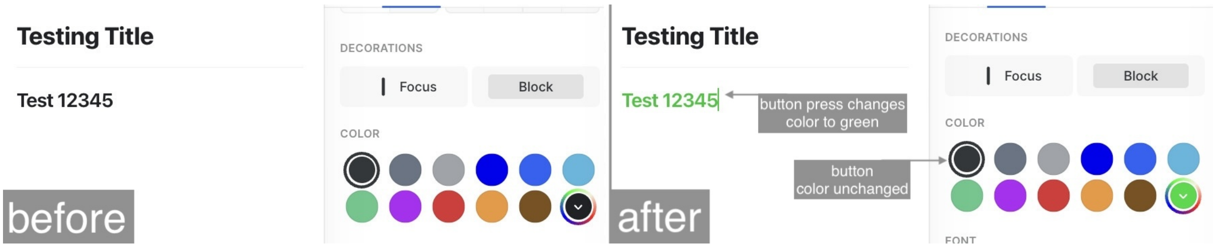


Figure 12: Colors in the application sidebar menu.

Here, it is noticeable that while the assistant was correctly able to identify where in the codebase to change the underlying color that is applied when the button is clicked – the text turned green when clicking on the button. Though, as noticeable in Figure 12, it was not able to comprehend that the visible color in the menu needed to be changed in a different place, thus it is still showing as black.

This indicates that while understanding the context of what is to be achieved, the retrieval process and LLM in combination were not able to grasp that there are additional dependencies to account for. This could be due to the contextual search during the retrieval process not capturing these chunks, or the LLM not grasping the additional dependency, or a combination of both.

4.3.2.4 Question 4

The fourth comparison question aims to evaluate the ability of the prototype to correctly interpret a rather loosely formulated query, find the correct piece of code based on both lexical and contextual relevancy, and give a clear and precise answer. This question is designed to be easy to interpret from reading the code without the code running.

Prompt

I'm working on implementing an analytics event when users create a new document. Help me implement this logging.

ChatGPT response

Implementing analytics events, such as logging when users create a new document, involves several steps across both the frontend and backend parts of your application. The specific implementation can vary depending on the analytics platform you're using (e.g., Google Analytics, Mixpanel, Amplitude) and the architecture of your application. However, I'll provide a general guide that you can adapt to your specific situation.

Step 1: Identify the Event

First, clearly define the event you want to track. In this case, the event is "Document Creation." Decide on a consistent naming convention for your events. For example, you might name this event `document_created`.

Step 2: Instrument the Event

Frontend Implementation

If the document creation process happens on the frontend (e.g., in a web or mobile application), you'll need to add code to track the event at the point where the user action is completed.

For a web application using JavaScript, here's a pseudo-code example:

```
function createDocument() {
    // Logic to create a new document...

    // After successfully creating the document, log the analytics event
    analytics.track('document_created', {
        // Optionally, include relevant properties about the document
        document_id: newDocumentId,
        user_id: currentUser.id,
        document_type: 'report', // Example property
        timestamp: new Date().toISOString() // Example property
    });
}
```

Replace `analytics.track` with the appropriate function call based on the analytics library you are using.

Backend Implementation

If the document creation is confirmed on the backend, or if you want to ensure the event is tracked even for API-based interactions, you'll need to add similar logic on your server.

Here's a pseudo-code example for a Node.js backend:

```
app.post('/create-document', async (req, res) => {
    // Logic to create a new document...

    // After successfully creating the document, log the analytics event
    analytics.track('document_created', {
        document_id: newDocument._id,
        user_id: req.user.id,
        document_type: req.body.type, // Example property
        timestamp: new Date().toISOString() // Example property
    });

    res.json({ success: true, document_id: newDocument._id });
});
```

Step 3: Test Your Implementation

After implementing the event tracking, test it thoroughly to ensure that events are being fired as expected and that the data captured is accurate. Use the analytics platform's debugging tools or view reports to verify.

Step 4: Deploy and Monitor

Once you're satisfied with the implementation, deploy the changes. Continue to monitor the events in your analytics dashboard to ensure data is being recorded as expected and to gain insights from user behavior.

Further Considerations

- Privacy and Consent: Ensure that your event tracking complies with privacy laws and regulations (like GDPR, CCPA). Inform users about the data being collected and obtain their consent if necessary.
- Rate Limits and Quotas: Be mindful of any rate limits or quotas imposed by your analytics platform, especially if you expect a high volume of events.
- Enhancing Data: Consider enriching the event data with additional context that could be useful for analysis, such as the user's geographic location, the device they're using, or the application version.

This guide provides a basic framework, but the specifics can vary widely depending on your technology stack and requirements. If you need more detailed assistance on a particular aspect or technology, feel free to ask!

Assistant response

I found a type definition that includes `newDocument` in what appears to be a list of analytics logging events within the project. This is part of a broader set of event types that the application can log. The relevant code snippet is found in `editor/src/analytics/editorAnalyticsLogger.ts`. Here's the relevant segment that includes `newDocument` as part of the event names:

```
export type EditorLogEvent =
  // Other event types omitted for brevity
  | "newDocument"
  // Other event types omitted for brevity
  ;
```

To implement logging for when users create a new document, you would utilize the analytics logging function provided in your application, presumably referencing this `newDocument` event type. Although a code snippet specifically for firing this event was not directly found, you would typically use the analytics service in your application like this:

```
import { editorAnalyticsLogger } from 'path/to/editorAnalyticsLogger';

// This function might be called at the point in your application where a new document is successfully created.
function logNewDocumentEvent() {
  const logger = editorAnalyticsLogger();
  logger.logEvent("newDocument");
}
```

Make sure to replace `` 'path/to/editorAnalyticsLogger'` `` with the actual import path to your `editorAnalyticsLogger` module within your project.

Since the specifics of triggering the event depend on how and where in your codebase documents are created, you'll want to place the call to `logNewDocumentEvent()` appropriately. This could be in a function that handles the creation logic after the document creation is confirmed, or within a success callback if document creation is asynchronous.

To integrate this accurately, review your application's architecture surrounding document creation and ensure the logging function call correctly aligns with the application's flow.

Comparing the two responses, we can see that the assistant is able to provide a much more specific answer, as it has the context of the repository, and was able to retrieve the relevant file where the analytics logger and events are defined. This is an indication that both the lexical meaning of *analytics* as well as *logging* was understood, as well as what it means specifically in the context of the repository. The retrieved information was correctly integrated into the answer, providing a clear response on how to implement the logging for an event.

However, the file path the assistant provided is not completely correct, as the `editorAnalyticsLogger.ts` file is in the `editor/src` directory, and not in `editor/src/analytics`. Possible reasons could be either in the retrieval process, or the chunk not being correctly retrieved, but as the remaining code from the retrieved chunk was correct, it seems more likely to arise from the LLM generation process.

4.3.2.5 Question 5

To gain a better understanding of the assistant's ability to handle ambiguity, this question asks about keyboard shortcuts. As the application contains multiple scenarios where keyboard shortcuts have different effects, this is a good way to test if the assistant gathers these scenarios and asks for clarification, or just picks one. In terms of the evaluation tree, this provides insight into query interpretation, clarity of answer, project structure and dependencies, as well as the retrieval process' relevancy in source selection.

Prompt

I want to change the keyboard shortcuts for Group/Ungroup in the editor to use the the J key instead of the G key.

ChatGPT response

To customize keyboard shortcuts, including changing the group/ungroup actions to use the ”J” key instead of the ”G” key, you'll need to access the settings or preferences within your specific editor. Since software and applications vary in how they allow users to modify shortcuts, I'll provide a general approach that applies to many programs:

1. Open Keyboard Shortcuts Settings: Most applications have a ”Settings” or ”Preferences” menu. Within this menu, look for sections titled ”Keyboard Shortcuts,” ”Key Bindings,” or similar. In some cases, you might be able to open the keyboard shortcuts settings directly by using a meta shortcut, often something like Ctrl + K Ctrl + S (Windows/Linux) or Cmd + K Cmd + S (Mac).
2. Search for Group/Ungroup Commands: Once in the keyboard shortcuts settings, use the search function to find the current bindings for ”Group” and ”Ungroup” commands. They're typically labeled as such, but terminology might vary slightly.
3. Modify the Shortcuts: Click on the current shortcut or the command itself, usually prompting you to enter a new key combination. Press the ”J” key (or the desired combination involving the ”J” key, like Ctrl + J or Cmd + J) to set it as the new shortcut. Ensure you're not creating a conflict with an existing important shortcut. Repeat the process for both ”Group” and ”Ungroup” commands, or any other commands you wish to modify.
4. Save and Test: Save your changes, usually by simply exiting the settings or pressing a designated ”Save” button. Test the new shortcuts in your editor to ensure they work as expected.

If you provide the name of the specific editor you're using (e.g., Adobe Photoshop, Microsoft Word, Visual Studio Code, etc.), I can give you more precise instructions tailored to that software.

Assistant response

To change the keyboard shortcuts for group/ungroup in the editor to use the J key instead of the G key, you'll need to modify the `contextMenu` configuration within the `contextMenu.ts` file in the `canvas/src/lib/contextMenu` directory.
Here are the current key bindings that you need to change:
For grouping, the current binding is `"\$g"`. For ungrouping, the current binding is `"\$!g"`[2].
To use the J key instead, you should modify these lines as follows:
For grouping, change from `kbd: "\$g"` to `kbd: "\$j"`, And for ungrouping, change from `kbd: "\$!g"` to `kbd: "\$!j"`,
Please ensure that these keybindings do not conflict with existing shortcuts in the editor.

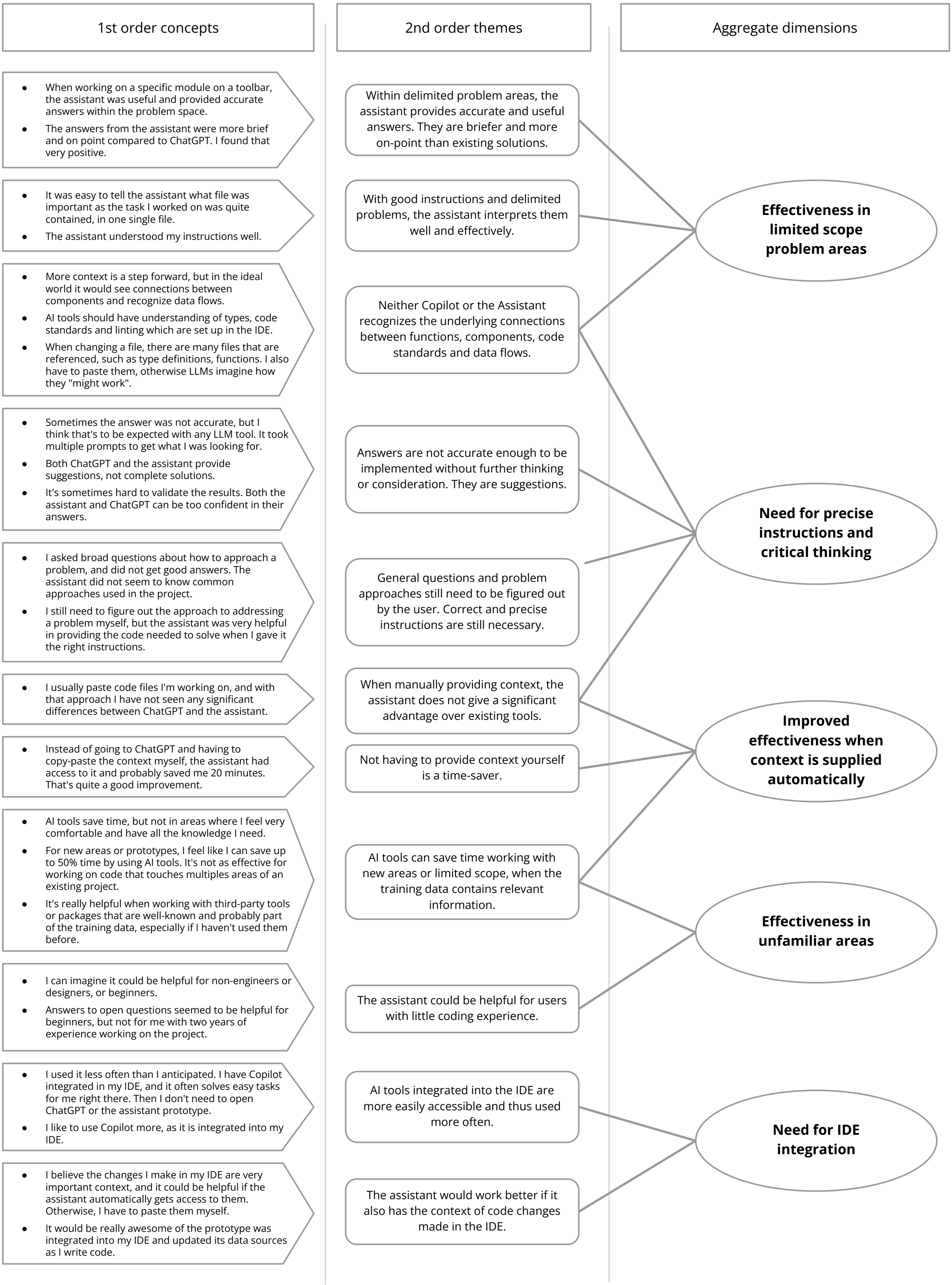
As expected, ChatGPT provided a generic answer on how to customize keyboard shortcuts across applications that support user-modified shortcuts. The assistant has the context of the project and was able to provide an application-specific answer. However, even with the explicit instruction to change shortcuts in the editor, the assistant provided instructions regarding keyboard shortcuts in the canvas/whiteboard which is integrated into the application. It also did not mention that there might be additional files where shortcuts are defined, except the explanation it concluded with, tasking the user to do that work themselves.

Additionally, the assistant refers to the context menu displayed in the whiteboard editor, thus only prompting to change the tooltip suggesting the key combination to the user, instead of changing the key combination for the shortcut.

Most probably, the lack of mentioning additional shortcut definitions as well as the lack of actually describing how to change the shortcut can be attributed to the retrieval process not having fetched those files. In terms of lexical relevancy, the retrieval process fetched pieces of code regarding keyboard shortcuts as expected, but did not grasp the context of the editor. The LLM then was not able to recognize any other dependencies or connections, and gave a clear answer correctly integrating the code it received, even if it is not fully relevant to the question.

4.3.3 Interview retrospective

Based on interviews conducted with the software engineers who participated in the prototype testing, higher-order reflections on the performance and limitations of the prototype emerged as part of the analysis with the outlined qualitative approach. These are visualized in Figure 13 and complement the technical assessment with a user experience perspective.


1st order concepts
2nd order themes
Aggregate dimensions
Effectiveness in limited scope problem areas
Need for precise instructions and critical thinking
Improved effectiveness when context is supplied automatically
Effectiveness in unfamiliar areas
Need for IDE integration

- When working on a specific module on a toolbar, the assistant was useful and provided accurate answers within the problem space.
- The answers from the assistant were more brief and on point compared to ChatGPT. I found that very positive.

- It was easy to tell the assistant what file was important as the task I worked on was quite contained, in one single file.
- The assistant understood my instructions well.

- More context is a step forward, but in the ideal world it would see connections between components and recognize data flows.
- AI tools should have understanding of types, code standards and linting which are set up in the IDE.
- When changing a file, there are many files that are referenced, such as type definitions, functions. I also have to paste them, otherwise LLMs imagine how they "might work".

- Sometimes the answer was not accurate, but I think that's to be expected with any LLM tool. It took multiple prompts to get what I was looking for.
- Both ChatGPT and the assistant provide suggestions, not complete solutions.
- It's sometimes hard to validate the results. Both the assistant and ChatGPT can be too confident in their answers.

- I asked broad questions about how to approach a problem, and did not get good answers. The assistant did not seem to know common approaches used in the project.
- I still need to figure out the approach to addressing a problem myself, but the assistant was very helpful in providing the code needed to solve when I gave it the right instructions.

- I usually paste code files I'm working on, and with that approach I have not seen any significant differences between ChatGPT and the assistant.

- Instead of going to ChatGPT and having to copy-paste the context myself, the assistant had access to it and probably saved me 20 minutes. That's quite a good improvement.

- AI tools save time, but not in areas where I feel very comfortable and have all the knowledge I need.
- For new areas or prototypes, I feel like I can save up to 50% time by using AI tools. It's not as effective for working on code that touches multiples areas of an existing project.
- It's really helpful when working with third-party tools or packages that are well-known and probably part of the training data, especially if I haven't used them before.

- I can imagine it could be helpful for non-engineers or designers, or beginners.
- Answers to open questions seemed to be helpful for beginners, but not for me with two years of experience working on the project.

- I used it less often than I anticipated. I have Copilot integrated in my IDE, and it often solves easy tasks for me right there. Then I don't need to open ChatGPT or the assistant prototype.
- I like to use Copilot more, as it is integrated into my IDE.

- I believe the changes I make in my IDE are very important context, and it could be helpful if the assistant automatically gets access to them. Otherwise, I have to paste them myself.
- It would be really awesome of the prototype was integrated into my IDE and updated its data sources as I write code.

Within delimited problem areas, the assistant provides accurate and useful answers. They are briefer and more on-point than existing solutions.

With good instructions and delimited problems, the assistant interprets them well and effectively.

Neither Copilot or the Assistant recognizes the underlying connections between functions, components, code standards and data flows.

Answers are not accurate enough to be implemented without further thinking or consideration. They are suggestions.

General questions and problem approaches still need to be figured out by the user. Correct and precise instructions are still necessary.

When manually providing context, the assistant does not give a significant advantage over existing tools.

Not having to provide context yourself is a time-saver.

AI tools can save time working with new areas or limited scope, when the training data contains relevant information.

The assistant could be helpful for users with little coding experience.

AI tools integrated into the IDE are more easily accessible and thus used more often.

The assistant would work better if it also has the context of code changes made in the IDE.

Figure 13: Results of the qualitative analysis.

4.4 Summary of results

This summary aims to provide a comprehensive overview of the achieved improvement, usage considerations and shortcomings, as well as relate user experience considerations with the technical dimensions of the evaluation tree. This section contains the authors' own reasoning based on the action research process and its evaluation.

4.4.1 Effectiveness in limited scope problem areas

Both from the qualitative analysis and comparison test, it became apparent that the scope of prompts the assistant is tasked with has a strong influence over its performance.

In the comparison test, specifically question 3 and 5, issues with incorporating additional context surrounding the problem arose, leading to unexpected effects due to dependencies. With LLMs having no inherent ability to understand connections between pieces of code based on how they run, important details contained in other files are not passed on to the LLM prior to the generation process and thus missed.

As part of the interview retrospective, a participant noted that their impression is that in these cases, they have to paste additional context on functions or components, as "otherwise LLMs imagine how they might work". Another participant noted that "in the ideal world, it would see connections between components and data flows". These issues are not only related to code specifically, but also to instructions such as code standards or linting set up in the IDE, which govern how code should be formatted or organized.

On the other hand, interview participants also noted that if they worked with clearly delimited scopes, such as in one case a "specific module" in "one single file", the assistant was able to provide brief, on-point accurate answers and interpreted instructions well, which was described as "very positive" and can be seen as an improvement over existing solutions. This can also be related to the clearly scoped questions 2 and 1 in the comparison test. In those cases, all necessary code was part of one file, and answers were clear and accurate.

Tying back to the evaluation tree, this limitation is interpreted to revolve around the source selection process as part of retrieval, as well as the codebase navigation and analysis capabilities of the LLM. As the vector search process (the process of finding chunks with similar vectors and thus similar content) is based on contextual similarity, it is not able to search by actually navigating the code based on its linkages and dependencies, which in turn also has detrimental effects on the LLM's ability to take these into account when generating an answer.

If files or pieces of code that are necessary to understand a given question are not passed to the LLM, it does not have the ability to identify these connections, and neither can it request additional files to be retrieved. However, in those cases where the necessary chunks of code are somewhat contained, identified by the search process, and passed to the LLM, sentiment from users and test results suggest that natural language understanding, integration of code into generated answers, and quality of generated code are satisfactory.

4.4.2 Need for precise instructions and critical thinking

A critical phenomenon emerging from the scenario testing and interview retrospective is that precise instructions are important for achieving relevant and correct answers. With ambiguity in the prompt or codebase, answer quality decreased. In the comparison test, this can be seen in questions 3 and 5, where the prompt leaves room for interpretation of what is meant by the user. In contrast, questions 1 and 2 leave little to no room for interpretation and therefore result in a relevant and correct output.

Due to the risk of hallucinations, the user needs to have sufficient knowledge of the problem subject to evaluate the output provided by the LLM. As mentioned by the engineers in the interviews, answers can be misleading and wrong, making the assistant potentially detrimental if relied upon without proper verification and oversight. A participant described this consideration as that these tools provide ”suggestions, not complete solutions”.

Employing critical reasoning when specifying the problem scope is deemed highly important, described by an interview participant as having to ”figure out the approach to addressing a problem” themselves. Similarly, critical thinking is necessary to validate the answer as described previously. All this shows that the assistant created should be considered a tool rather than a replacement, with it being important that the person using this tool has sufficient experience in the field to be able to use it effectively and confidently.

Referring back to the evaluation tree, these findings primarily tie back to correct query interpretation, the ability to identify relevant connections and dependencies, and an understanding of project structure. As also described regarding effectiveness in limited scope problem areas, precise instructions emerge as fundamental for achieving good results on both LLM- and retrieval-related dimensions of the evaluation tree.

4.4.3 Improved effectiveness when context is supplied automatically

As hypothesized in the concept generation stage based on the user study and needs analysis, the interview retrospective indicated that automatically providing context to the LLM leads to an improvement in effectiveness. One of the participants voiced that not having to copy-paste context in the form of code snippets themselves was able to save around 20 minutes on a half-day task. However, being able to gain this improvement seems to entail the need for a change in how the tool is used, as another participant stated that they continued to paste code files themselves and with that approach did not notice a significant difference compared to ChatGPT.

To recapitulate, with the tool being able to fetch needed context by itself, time can be saved when using it. Furthermore, if the engineer using the tool is unsure about what context is needed, the automatically supplied context may also save the person time researching and discovering what parts of the code are necessary to parse with the prompt in order to get a meaningful answer, keeping considerations regarding problem scope in mind.

4.4.4 Effectiveness in unfamiliar areas

From the interviews, it was discovered that the assistant could be helpful in unfamiliar areas and especially for prototyping. As one of the interviewees pointed out, on new areas or prototypes, they perceive LLM-based tools as being able to save up to 50% of the time needed for such tasks.

The authors' experience aligns with this sentiment, as neither of them had previously worked with the given repository and project, and were able to implement instructions provided by the assistant as shown in the scenario evaluation.

In more general terms and not specific to the repository, participants also voiced that LLM tools including the assistant are helpful with third-party packages unfamiliar to the user, in case they are well known and expected to be part of the model's training data.

Interview participants also described that they could ”imagine [the assistant] could be helpful for non-engineers or designers, or beginners”, especially with broader open questions. Though, this suspicion was not definitely proven as only the software engineering team was in the scope of this thesis.

4.4.5 Need for IDE integration

Participants in the interview voiced the wish for the assistant to be integrated into their IDE, mainly to enable faster and easier use. As no restrictions were placed on usage of other LLM-based developer tools during the testing periods, a participant noted that in many cases with ”easy tasks”, GitHub Copilot was able to provide satisfactory suggestions directly in the IDE. This reduced the need to employ the assistant.

Aside from a potential improvement in ease-of-use, closer integration with the IDE could enable even better contextual understanding, described by one of the participants as follows: ”I believe the changes I make in my IDE are very important context, and it could be helpful if the assistant automatically gets access to them.” As noted by participants, code generated by the assistant is mostly regarded as a suggestion, thus it can be expected that engineers make changes to it before integrating it. These changes can be meaningful context for multi-step conversations with the assistant.

From the authors’ perspective, manual updates to the assistant’s knowledge files are a laborious process which could benefit from automation with an IDE integration.

5 Discussion

This chapter contains the authors' thoughts and observations on the results from the user needs analysis, prototyping phase and subsequent evaluation, contrasting them with results from the literature study. Additionally, the limitations of the conducted research project are discussed, along with possible future improvements regarding the developed prototype.

5.1 Contrasting research results with theory

Research results will be contrasted with theory. For the sake of clarity, this section is divided into results and considerations regarding LLM technology's impact on software engineers' effectiveness, and technical considerations regarding LLMs.

5.1.1 Impacts on effectiveness

As presented in the theoretical background, previous studies both by Harvard Business School with GPT-4 at Boston Consulting Group (Dell'Acqua et al., 2023) as well as one by Microsoft Research, MIT Sloan and GitHub with Copilot and a group of software engineers (Peng et al., 2023) showed substantial improvements in workers' effectiveness if assisted by an LLM-based tool. In the case of GitHub Copilot, this increase amounted to 55.8% faster task completion, with even more significant increases for less experienced developers.

In the study by Peng et al. (ibid.), the task at hand consisted of developing an HTTP server with no preexisting code that had to be taken into account or systems required to be integrated with, where the 55.8% improvement in speed and higher task completion rate was observed. In line with the findings of this project, LLM tools can provide accurate answers and code for *greenfield* tasks with a limited scope. However, to conclude from these results that the mere usage of LLM tools in software engineering always leads to improvements, disregarding the use case and type of task, would be a fallacy.

An important consideration to take into account when interpreting such results is the scope of given tasks. Dell'Acqua et al. (2023) use the terms *inside* and *outside the frontier* to describe if tasks are applicable for an LLM. In their study, inside-the-frontier tasks are viable and result in effectiveness gains, whereas outside-the-frontier tasks could lead to decreased performance. Even though their study was conducted in a different setting with different work tasks, this categorization seems to apply also within software engineering.

In the study by Dell'Acqua et al. (ibid.), they gave two tasks, one outside and one inside the frontier. Within the frontier, the task mainly consisted of generating ideas, describing a prototype and the market, creating a list of steps to launch it and writing a memo and article. Outside the frontier, the task consisted of analyzing data and suggesting actions based on it. As might be expected, the second task did not benefit from GPT-4 usage (ibid.).

Comparing these results to our observations within software engineering, similar patterns emerge. Software engineers interviewed implicitly described their process of navigating the jagged frontier, making sure to take LLM's output as suggestions, and think through what approach to take instead of taking LLM's recommendations at face value. In addition, we can see signs of the interview participants making their judgments on which tasks lie within the frontier and which do not - LLM tools seem to be more heavily employed when tasks are limited in scope or outside of existing systems such as prototypes. When a significant knowledge gap exists, LLMs also are reported to be useful, though frustrations sometimes arose due to hallucinations or nonfactual information, which could hint at the specific task lying outside the frontier.

As part of this project, the division of tasks into *greenfield* and *integrated* emerged quite early on, with software engineers expressing frustrations applying LLM tools effectively on the integrated tasks. According to interviewees and theoretical research, these amount to the majority of code-related work, as the discipline inherently entails building upon, maintaining and improving existing systems. This suggests that results such as those presented by Peng et al. (2023) have to be examined carefully to not draw conclusions hastily regarding the whole field of software engineering, which may have contributed to the current *AI hype*.

Specifically for code-related tasks, it can also be discussed how LLM's probabilistic nature influences performance. This can be assumed to contribute heavily to performance on such tasks, and thus the location of the frontier. As outlined previously, LLM generation is based on the concept of *attention*, with tokens influencing each other to encode meaning. Code and natural language are inherently different in terms of their context density, a paragraph of text can be expected to be understood even if a few words are changed or missing, whereas a block of code may result in completely different behavior if one word or statement is omitted or changed, as code often is more deterministic. With the chosen RAG process, this is assumed to be a contributing factor to unexpected or unsatisfactory results caused by missing context, in the words of an interview participant with the LLM ”imagining” what missing components or functions might look like.

5.1.2 Technical considerations

Due to the choice of creating the prototype using OpenAI Playground, not all prompting, RAG and fine-tuning techniques found from the literature research could be implemented. However, out of those possible to implement, some provided successful results although generally not as good as the ones presented in prior research. This is thought to be due to the prototype incorporating various techniques simultaneously in a broad scope while most previous research had proved each technique successful when deployed mostly isolated in limited scopes.

Of course, larger improvements within LLMs and embeddings could contribute to better performance of the assistant application, though the focus in this discussion lies on techniques and methods available to the public.

5.1.2.1 Prompt engineering

From the development and testing of the prototypes, it was discovered that not all prompting techniques that had proven effective in previous well-defined research settings were as effective combined in real-world applications. This is thought to be much due to the large input, generally consisting of around 5,000 to 50,000 tokens, making it difficult for the model to take sufficient consideration to all instructions due to the nature of how the attention mechanism in transformers works. For the prototype to follow instructions, these had to sometimes be mentioned twice in the system prompt. This put some contrast to previous findings on context being helpful for generating better outputs, since too much context may make it difficult for the model to take sufficient consideration of all instructions. A way to work around this could be to fine-tune the model in a way that makes parts of the system prompt superfluous.

5.1.2.2 RAG

Evaluating the RAG component of the prototype is difficult due to the black box characteristics of OpenAI Playground. However, overall answers got more specific as seen from the examples in scenario evaluation. Furthermore, although it was a manual process, it proved easy and efficient to update the retrieval data through hot swapping.

With the code repository merged into larger files due to the limitations set by OpenAI, the traceability of code snippets through linking the file source was usually not sufficient since each file contained. However, through the addition on meta-data in the files and prompting the prototype to include this in the answer, it was most times able to provide this file path correctly. A tool allowing a sufficient number of files to be uploaded would mitigate this.

It was expected that the RAG might struggle with the task at hand since most research on RAG has been on simpler information retrieval without the need for understanding the complex dynamic of a code repository where both lexical and semantic search might fall short of finding the relevant documents. The results were however promising, although it was noted that lexical relevancy often seemed to be the main search criteria for finding the relevant chunks. An example of this can be seen in scenario evaluation Question 5, where the RAG process returned code snippets regarding a user interface displaying what keyboard shortcuts are available, rather than the code where the keyboard shortcuts are defined.

More advanced versions of RAG such as *RepoCoder* proposed by F. Zhang et al. (2023) or *Repository Level Prompt Generator* proposed by Shrivastava, Larochelle, and Tarlow (2023) could have had a positive impact on output but could not be implemented due to the decision to build the prototype with OpenAI's tools. Additionally, one of the main points of feedback since the first iteration was the prototype being slow, and adding these iterative retrieval techniques would make the prototype significantly slower and probably affect the overall user experience.

5.2 Future improvements of the application

Building upon the idea of CoT, ToT could be implemented to further improve the generated outputs. By sampling multiple times, evaluating the answers, and continuing on the best answer, the output will likely improve and reduce the need for the engineer using the tool to perform this iterative problem-solving. The issue with this however is the increased computing needed that will grow by several multiples. This will in turn lead to the model being slower unless computing is improved, which is very likely to happen when seeing how Moore's law has played out up until now. Combining ToT with the *RepoCoder* strategy, proposed by F. Zhang et al. (2023), presents even further opportunities for improvements on generated output. Once again though, this will lead to an increased need for computing and slower response times unless computing power is improved.

In terms of how users interact with the application, interview participants voiced the need or wish for closer integration with their IDE. One aspect of this is ease of use, where positive experience with integrated tools such as Copilot was highlighted. Opening a web browser and navigating to the assistant is an additional step which adds friction to the process.

Possibly more impactful is the ability to incorporate even more context, as IDEs not only are where software engineers write their code and make changes, but also where linting, code standards and other conventions are enforced. Up-to-date code changes from the IDE could enable better back-and-forth conversations with the assistant, where the context in terms of changes made by the software engineer could automatically be fed back into the LLM. Similarly, the RAG database could be kept up to date in an automated fashion through hot swapping. In regards to code conventions, those settings from the IDE could inform code generation and provide important input to achieve code that is more in line with standards set within a repository.

To improve the application's understanding and reasoning capabilities, larger advances within LLM technology would be necessary. It can be debated whether improvements in embeddings and thus better contextual relationships between tokens in code could be impactful. For more significant improvements, new LLM architectures which combine probabilistic and deterministic properties might be needed.

5.3 Limitations

As part of section 3.2, four aspects for ensuring quality adopted from Lincoln and Guba (1985) were outlined, namely *credibility, transferability, dependability*, and *confirmability*. Employing this framework, the possible limitations and weaknesses of this thesis were critically evaluated and are described below.

Some of the results of this thesis *credibility* can be critiqued due to the limited number of interviewees. As part of the initial user study, a broad range of software engineers within the case company were interviewed, with team members working in different disciplines and with different levels of experience. However, due to the prototype focusing on one of these disciplines, and with participation in the testing process being voluntary, only two software engineers met the threshold of usage deemed sufficient for insightful reflection. Nonetheless, as the insights from these interviews are based on extensive usage and reflection, they were deemed sufficiently credible, even if credibility could be increased with a larger number of participants. Additionally, credibility could be increased by examining further case companies, which could also result in better *transferability*.

A second possible point of criticism regarding *transferability* surrounds the level of experience of software engineers participating in the study. Most software engineers at the case company are highly experienced, and the company holds a high bar in hiring. Additionally, the case company is a startup with less process load than larger or older businesses. Thus, conclusions regarding effectiveness might not transfer to the software engineering discipline, as experience levels and types of work vary across companies and individuals. On the other hand, experienced software engineers' insights regarding task selection and methods of working with LLMs can be valuable guidance for less experienced ones.

Similarly, *dependability* can be questioned due to interviewees' personal biases or previous experiences with LLM tools, either positive or negative. It cannot be fully guaranteed that the same results would arise from different interviewees within the same selection. The fast technical development of the used underlying services such as OpenAI could also change the results if the same setup cannot be reproduced exactly.

In terms of confirmability, the researchers' interpretation and aggregation of interviewees' responses can be named as a possible weakness. Actions employed to minimize these concerns include peer debriefing, and validation with interviewees and supervisors.

One characteristic of the AI and LLM field is the current high pace of research, resulting in many valuable and cutting-edge insights being published as preprints. These papers are published on servers as arXiv and SSRN, which are moderated, but not peer-reviewed as academic journals generally are. Biases other researchers might hold, their own interpretations or publication pressures within academia or commercial organizations can thus affect this thesis' credibility and confirmability. To mitigate this, the authors triangulated information from multiple research papers.

Every academic research paper and thus also this thesis has similar shortcomings, therefore it is important to evaluate them and critically reflect, which has been performed above. Mainly, the number of software engineers involved in testing and thus the number of reflective interviews can be seen as a significant factor. Nonetheless, this thesis provides a good overview of important considerations and the possible impact of context-augmented LLM tools, as well as insights regarding effective usage of such tools.

5.4 Contributions

5.4.1 Academia

With most research evaluating the capabilities and performance of LLMs through well-defined test sets and stand-alone tasks, this thesis contributes to research by examining how LLMs can be applied to integrated and complex workflows that are more representative of work performed by highly skilled workers. The thesis can also be a foundation for future research from the human-machine interaction perspective, demonstrating how technical improvements and changes in behavior are intertwined and have to be balanced for achieving positive results.

5.4.2 Practical Applications

For practitioners, the technical method used in this thesis demonstrates an attainable way of developing and configurating an augmented LLM, contributing to the knowledge of how LLMs can be tailored for specific use-cases.

6 Conclusions

The final chapter presents the thesis project's conclusions as answers to the research questions, summarizes the key results and presents the authors' concluding remarks.

6.1 Key findings

Three research questions were chosen based on the problem description and background, which are answered below.

R1: ***How do software developers at Craft Docs use LLMs today?***

Based on the interviews with software engineers at the case company Craft Docs, we can see two LLM-based services in use. These are *GitHub Copilot*, used by a subset of interviewees, and *OpenAI's GPT*, used by an overwhelming majority of participants. There are differences to the specific method of accessing GPT, being either via ChatGPT's web interface, or accessing the API via a wrapper. Though, GPT-4 in a chat-based interface is the tool of choice for all who use GPT.

The use of Copilot, which provides suggestions in the IDE, is determined mainly by compatibility with the IDE the software engineer is using. This results in Mac and iOS engineers not being able to adopt it. For the engineers where it is compatible with their IDE, adoption varies.

GPT on the other hand, as a standalone application, is employed across a wide range of use cases, both relating to code, but also adjacent writing and research tasks. As the interviews focused on code-related use cases, these were mentioned more widely, with greenfield tasks such as prototyping or developing scripts and integrated tasks working within existing applications being mentioned by a majority of interviewees. Information gathering, either in terms of looking up facts, or using GPT for more thorough learning processes, was mentioned by almost all participants. Some also mentioned text production-related tasks.

Overall, there are apparent individual differences in the adoption of tools and the range of tasks they are used for. The software engineer's context in terms of IDE, programming language or domain influences this, with adoption of tools other than GPT being limited by compatibility. There were no indications of experimentation with or adoption of emerging tools from other companies or service providers, other than the two mentioned above.

R2: ***What LLM-related improvement opportunities can be found in regard to software engineers' effectiveness?***

With GPT as the most widespread tool, and the highest-performing LLM in benchmarks, it was selected for closer examination. For most tasks, participants find it useful and describe positive experiences, with *integrated* coding tasks being the exception.

These types of tasks, where dependencies have to be taken into account and engineers need to work with existing code, elicited mixed opinions. A larger number of participants reported negative experiences, than those reporting positive experiences, with multiple interviewees describing a lack of context as the primary issue. In contrast, participants report more success and positive experiences with *greenfield* tasks such as developing prototypes or scrips. *Integrated* coding tasks were thus identified as the area with the largest improvement potential, and addressed with a prototype aiming to provide said context.

Throughout the project, it became apparent that improvement opportunities do not only relate to the technical abilities and performance of a given tool. Additionally, the way of using the tool also influences effectiveness. Improvement potential can also be found in terms of developing a proper approach, choice of problems and tasks, and level of instructions given to the tool.

R3: ***What improvements in effectiveness are achieved by acting on the found opportunities?***

Results from the implementation and testing of the prototype showcased an improvement in effectiveness by supplying context automatically. Participants reported saving time due to not having to copy and paste necessary pieces of code themselves.

As part of the collaborative action research process, it was also discovered that more effective use of LLM-based tools is enabled by employing critical thinking in terms of what types of tasks to apply these tools to. More specifically, these tasks are in areas with which the user is not familiar, or with problems of limited scope.

For closed-source internal codebases, a retrieval-augmented chat assistant such as the prototype developed, can enable gains in effectiveness by lessening the labor of providing clear and precise instructions. Also, it potentially enables engineers new to the codebase to gain effectiveness in unfamiliar areas.

Referring back to the term *jagged frontier of AI* coined by Dell'Acqua et al. (2023), indications were found of more effective usage for users who have developed strategies for navigating it – hinting that educating on best practices can lead to improved gains from using these tools.

6.2 Concluding remarks

Overall, these findings point to that technical improvements are able to increase performance of LLM-based tools and assistants for software engineers and knowledge workers. In terms of the technological frontier coined by Dell'Acqua et al. (2023) as seen in figure 14, it is though visible that these technical developments with the currently available methods are not able to smooth the jagged nature of the frontier, but rather perhaps move it overall, or increase performance in specific areas.

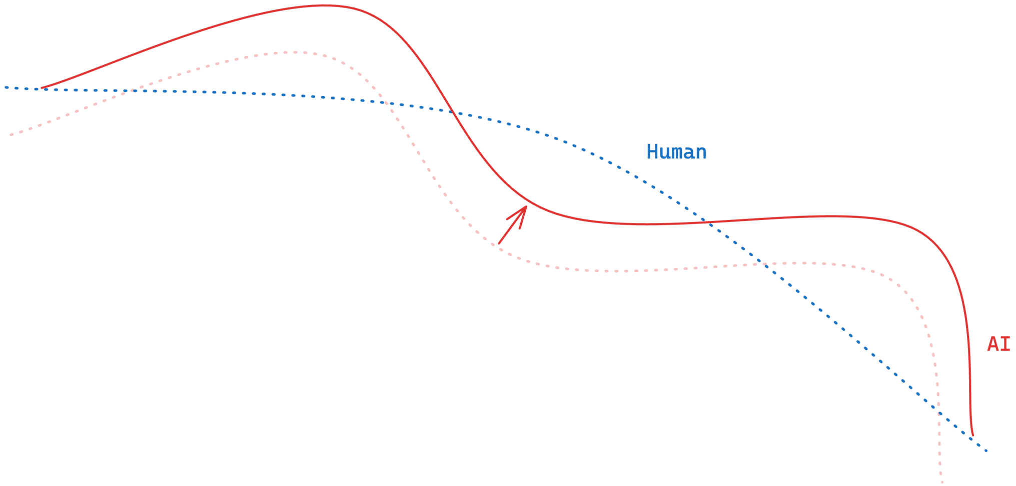


Figure 14: Moving the "jagged frontier of AI capabilities", adapted from Dell'Acqua et al. (2023).

The results point to software engineers specifically and knowledge workers in general being able to work more effectively when applying LLMs to tasks in the peaks of the frontier. Overall though, it is clear that an understanding of the fact that the proficiencies of humans and LLMs do not align is fundamental for effective use, as it enables navigating the jagged frontier even as it evolves over time.

References

Allard, John and Jarvis Colin (2023). *A Survey of Techniques for Maximizing LLM Performance*. YouTube: `https://www.youtube.com/watch?v=ahnGLM-RC1Y`. [Accessed 9-02-2024]. OpenAI.

Amazon Web Services (n.d.). *What is Prompt Engineering?* `https://aws.amazon.com/what-is/prompt-engineering/`. [Accessed 05-02-2024].

Bajekal, Naina et al. (2023). "Sam Altman. (Cover story)." In: *TIME Magazine* 202.21/22, pp. 58–68. ISSN: 0040781X.

Baldwin, Sam (2022). *Craft AI Assistant — craft.do*. `https://www.craft.do/blog/craft-ai-assistant`. [Accessed 16-02-2024].

Barnett, Scott et al. (2024). *Seven Failure Points When Engineering a Retrieval Augmented Generation System*. arXiv: `2401.05856 [cs.SE]`.

Beck, Kent et al. (2001). *Manifesto for Agile Software Development*. URL: `http://agilemanifesto.org/`.

Buxton, J. N. and Brian Randell (1970). *Software Engineering Techniques: Report of a conference sponsored by the NATO Science Committee, Rome, Italy, 27-31 Oct. 1969, Brussels, Scientific Affairs Division, NATO*. `http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1969.PDF`[Accessed 25-01-2024].

Cambon, Alexia et al. (2023). "Early LLM-based Tools for Enterprise Information Workers Likely Provide Meaningful Boosts to Productivity". In: URL: `https://www.microsoft.com/en-us/research/uploads/prod/2023/12/AI-and-Productivity-Report-First-Edition.pdf`.

Chatterjee, Sayan et al. (2024). *The Impact of AI Tool on Engineering at ANZ Bank An Emperical Study on GitHub Copilot within Coporate Environment*. DOI: `10.48550/ARXIV.2402.05636`. URL: `https://arxiv.org/abs/2402.05636`.

Chen, Mark et al. (2021). *Evaluating Large Language Models Trained on Code*. DOI: `10.48550/ARXIV.2107.03374`. URL: `https://arxiv.org/abs/2107.03374`.

Clement, Colin B. et al. (2021). *Long-Range Modeling of Source Code Files with eWASH: Extended Window Access by Syntax Hierarchy*. DOI: `10.48550/ARXIV.2109.08780`. URL: `https://arxiv.org/abs/2109.08780`.

Cobbe, Karl et al. (2021). *Training Verifiers to Solve Math Word Problems*. DOI: `10.48550/ARXIV.2110.14168`. URL: `https://arxiv.org/abs/2110.14168`.

Corley, Kevin G. and Dennis A. Gioia (June 2004). "Identity Ambiguity and Change in the Wake of a Corporate Spin-off". In: *Administrative Science Quarterly* 49.2, pp. 173–208. ISSN: 1930-3815. DOI: `10.2307/4131471`. URL: `http://dx.doi.org/10.2307/4131471`.

Craft Docs (n.d.[a]). *Careers at Craft — craft.do*. `https://www.craft.do/careers`. [Accessed 25-01-2024].

— (n.d.[b]). *Craft - Press kit — craft.do*. `https://www.craft.do/media`. [Accessed 25-01-2024].

Cursor (n.d.). *Cursor - The AI-first Code Editor — cursor.sh*. `https://cursor.sh/`. [Accessed 21-02-2024].

Dell'Acqua, Fabrizio et al. (2023). "Navigating the Jagged Technological Frontier: Field Experimental Evidence of the Effects of AI on Knowledge Worker Productivity and Quality". In: *SSRN Electronic Journal*. ISSN: 1556-5068. DOI: `10.2139/ssrn.4573321`. URL: `http://dx.doi.org/10.2139/ssrn.4573321`.

Feng, Yunhe et al. (2023). "Investigating Code Generation Performance of ChatGPT with Crowdsourcing Social Data". In: *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, pp. 876–885. DOI: `10.1109/COMPSAC57700.2023.00117`.

Gemini Team et al. (2023). "Gemini: A Family of Highly Capable Multimodal Models". In: *arXiv ePrint*. DOI: `10.48550/ARXIV.2312.11805`. URL: `https://arxiv.org/abs/2312.11805`.

GitHub (n.d.). *GitHub Copilot · Your AI pair programmer — github.com*. `https://github.com/features/copilot`. [Accessed 14-02-2024].

Google (2023). *Introduction to Large Languagel Models*. `https://developers.google.com/machine-learning/resources/intro-llms`. [Accessed 24-01-2024].

— (n.d.[a]). *Gemini API | Vertex AI | Google Cloud — cloud.google.com*. `https://cloud.google.com/vertex-ai/docs/generative-ai/model-reference/gemini`. [Accessed 21-02-2024].

— (n.d.[b]). *Gemini API Pricing | Google AI for Developers | Google for Developers — ai.google.dev*. `https://ai.google.dev/pricing`. [Accessed 03-06-2024].

— (n.d.[c]). *Gemini Apps' release updates & improvements — gemini.google.com*. `https://gemini.google.com/updates`. [Accessed 21-02-2024].

Google (n.d.[d]). *Get More Storage, More AI capabilities, and More Features - Google One — one.google.com*. `https://one.google.com/about/#ai`. [Accessed 03-06-2024].

Google Cloud (n.d.). *Gemini in Google Cloud pricing | Gemini for Google Cloud — cloud.google.com*. `https://cloud.google.com/products/gemini/pricing`. [Accessed 03-06-2024].

Google Workspace (n.d.). *Gemini for Google Workspace | Gen AI Tools for Business — workspace.google.com*. `https://workspace.google.com/solutions/ai/`. [Accessed 03-06-2024].

Gu, Alex et al. (2024). *CRUXEval: A Benchmark for Code Reasoning, Understanding and Execution*. DOI: `10.48550/ARXIV.2401.03065`. URL: `https://arxiv.org/abs/2401.03065`.

— (n.d.). *CRUXEval Leaderboard — crux-eval.github.io*. `https://crux-eval.github.io/leaderboard.html`. [Accessed 03-06-2024].

Harvard University Information Technology (2023). *Getting started with prompts for text-based Generative AI tools*. `https://huit.harvard.edu/news/ai-prompts`. [Accessed 05-02-2024].

Hendrycks, Dan et al. (2020). *Measuring Massive Multitask Language Understanding*. DOI: `10.48550/ARXIV.2009.03300`. URL: `https://arxiv.org/abs/2009.03300`.

Höst, Martin, Björn Regnell, and Per Runeson (2006). *Att genomföra examensarbete*. sv. ISBN: 9789144005218.

Hu, Krystal (2023). *ChatGPT sets record for fastest-growing user base - analyst note*. Reuters: `https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/`. [Accessed 14-02-2024].

HuggingFace (n.d.). *Hugging Face – The AI community building the future. — huggingface.co*. `https://huggingface.co/`. [Accessed 03-06-2024].

IEEE (1990). “IEEE Standard Glossary of Software Engineering Terminology”. In: *IEEE Std 610.12-1990*, pp. 1–84. DOI: `10.1109/IEEESTD.1990.101064`.

Kadavath, Saurav et al. (2022). *Language Models (Mostly) Know What They Know*. arXiv: `2207.05221 [cs.CL]`.

Kemmis, Stephen and Mervyn Wilkinson (Sept. 2002). “Participatory Action Research and the Study of Practice”. In: *Action Research in Practice*. Ed. by Bill Atweh, Stephen Kemmis, and Patricia Weeks. Milton Park, UK: Routledge, pp. 21–26. DOI: `10.4324/9780203024478`. URL: `http://dx.doi.org/10.4324/9780203024478`.

Khadija, Mutiara Auliya, Abdul Aziz, and Wahyu Nurharjadmo (2023). “Automating Information Retrieval from Faculty Guidelines: Designing a PDF-Driven Chatbot powered by

OpenAI ChatGPT”. In: *2023 International Conference on Computer, Control, Informatics and its Applications (IC3INA)*, pp. 394–399. DOI: `10.1109/IC3INA60834.2023.10285808`.

Kwan, Wai-Chung et al. (2024). *MT-Eval: A Multi-Turn Capabilities Evaluation Benchmark for Large Language Models*. arXiv: `2401.16745 [cs.CL]`.

Lewis, Patrick et al. (2021). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. arXiv: `2005.11401 [cs.CL]`.

Li, Jia et al. (2024). *DevEval: Evaluating Code Generation in Practical Software Projects*. DOI: `10.48550/ARXIV.2401.06401`. URL: `https://arxiv.org/abs/2401.06401`.

Lincoln, Yvonna S and Egon G Guba (June 1985). *Naturalistic Inquiry*. en. Thousand Oaks, CA: SAGE Publications.

Liu, Jiawei et al. (2023). “Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation”. In: *Thirty-seventh Conference on Neural Information Processing Systems*. URL: `https://openreview.net/forum?id=1qvx610Cu7`.

— (n.d.). *EvalPlus Leaderboard — evalplus.github.io*. `https://evalplus.github.io/leaderboard.html`. [Accessed 03-06-2024].

Maanen, John Van (Dec. 1979). “The Fact of Fiction in Organizational Ethnography”. In: *Administrative Science Quarterly* 24.4, p. 539. ISSN: 0001-8392. DOI: `10.2307/2392360`. URL: `http://dx.doi.org/10.2307/2392360`.

Mialon, Grégoire et al. (2023). *Augmented Language Models: a Survey*. arXiv: `2302.07842 [cs.CL]`.

Minde, Tor Björn (2023). *Generative AI does not run on thin air!* `https://www.ri.se/en/news/blog/generative-ai-does-not-run-on-thin-air?fbclid=IwAR2flwg5zNo9rES3Sdu2Qbw-nK1q70-CKoohpEp-GpQoKwDX-d0hB5XWiTo`. [Accessed 31-01-2024].

Naur, Peter and Brian Randell (1969). *Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968, Brussels, Scientific Affairs Division, NATO*. `http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF`[Accessed 25-01-2024].

Nvidia (n.d.). *Large Language Models Explained*. `https://www.nvidia.com/en-us/glossary/large-language-models/`. [Accessed 24-01-2024].

O’Regan, Gerard (2022). *Concise Guide to Software Engineering: From Fundamentals to Application Methods*. Springer International Publishing. ISBN: 9783031078163. DOI: `10.1007/978-3-031-07816-3`. URL: `http://dx.doi.org/10.1007/978-3-031-07816-3`.

OpenAI (2023). *ChatGPT can now see, hear, and speak — openai.com.* `https://openai.com/blog/chatgpt-can-now-see-hear-and-speak`. [Accessed 14-02-2024].

— (n.d.[a]). *Assistants Tools - OpenAI API.* `https://platform.openai.com/docs/assistants/tools`. [Accessed 03-04-2024].

— (n.d.[b]). *GPT-4 — openai.com.* `https://openai.com/gpt-4`. [Accessed 24-01-2024].

— (n.d.[c]). *New embedding models and API updates — openai.com.* `https://openai.com/blog/new-embedding-models-and-api-updates`. [Accessed 31-01-2024].

— (n.d.[d]). *OpenAI API Reference.* `https://platform.openai.com/docs/api-reference/introduction`. [Accessed 14-02-2024].

— (n.d.[e]). *OpenAI Documentation - Assistants API.* `https://platform.openai.com/docs/assistants/overview`. [Accessed 14-02-2024].

— (n.d.[f]). *OpenAI Documentation - Models.* `https://platform.openai.com/docs/models`. [Accessed 14-02-2024].

— (n.d.[g]). *OpenAI Status — status.openai.com.* `https://status.openai.com/`. [Accessed 23-02-2024].

— (n.d.[h]). *Playground | OpenAI Platform.* `https://platform.openai.com/playground`. [Accessed 03-06-2024].

— (n.d.[i]). *Pricing | OpenAI.* `https://openai.com/api/pricing/`. [Accessed 03-06-2024].

OpenAI et al. (2023). *GPT-4 Technical Report.* arXiv: `2303.08774 [cs.CL]`.

Ouyang, Long et al. (2022). *Training language models to follow instructions with human feedback.* arXiv: `2203.02155 [cs.CL]`.

Ozkaya, Ipek (May 2023). “Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications”. In: *IEEE Software* 40.3, pp. 4–8. ISSN: 1937-4194. DOI: `10.1109/ms.2023.3248401`. URL: `http://dx.doi.org/10.1109/MS.2023.3248401`.

Patnaik, Dev and Robert Becker (Apr. 1999). “Needfinding: The Why and How of Uncovering People’s Needs”. In: *Design Management Journal (Former Series)* 10.2, pp. 37–43. ISSN: 1045-7194. DOI: `10.1111/j.1948-7169.1999.tb00250.x`. URL: `http://dx.doi.org/10.1111/j.1948-7169.1999.tb00250.x`.

Peng, Sida et al. (2023). *The Impact of AI on Developer Productivity: Evidence from GitHub Copilot.* arXiv: `2302.06590 [cs.SE]`.

Pichai, Sundar and Demis Hassabis (2024). *Our next-generation model: Gemini 1.5 — blog.google*. `https://blog.google/technology/ai/google-gemini-next-generation-model-february-2024`. [Accessed 21-02-2024].

Robson, Colin and Kieran McCartan (2016). *Real World Research*. 4th ed. Wiley. ISBN: 9781118745236.

Shrivastava, D., H. Larochelle, and D. Tarlow (2023). “Repository-Level Prompt Generation for Large Language Models of Code.” In: *Proceedings of Machine Learning Research*. Vol. 202. Proceedings of the 40th International Conference on Machine Learning, ICML 2023.

Smith, Hayley (2015). *Skyscanner acquires mobile developer Distinction*. `https://www.skyscanner.net/press-releases/skyscanner-acquires-mobile-developer-distinction`. [Accessed 25-01-2024].

Tabnine (n.d.). *Tabnine is an AI assistant that speeds up delivery and keeps your code safe | Tabnine — tabnine.com*. `https://www.tabnine.com/`. [Accessed 21-02-2024].

Tu, Zhaopeng, Zhendong Su, and Premkumar Devanbu (2014). “On the localness of software”. In: *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE 2014. Hong Kong, China: Association for Computing Machinery, pp. 269–280. ISBN: 9781450330565. DOI: `10.1145/2635868.2635875`. URL: `https://doi.org/10.1145/2635868.2635875`.

Tufano, Michele et al. (2021). *Unit Test Case Generation with Transformers and Focal Context*. arXiv: `2009.05617 [cs.SE]`.

Vaswani, Ashish et al. (2023). *Attention Is All You Need*. arXiv: `1706.03762 [cs.CL]`.

Wei, Jason, Yi Tay, et al. (2022). *Emergent Abilities of Large Language Models*. arXiv: `2206.07682 [cs.CL]`.

Wei, Jason, Xuezhi Wang, et al. (2023). *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. arXiv: `2201.11903 [cs.CL]`.

Wohlin, Claes (2014). “Guidelines for snowballing in systematic literature studies and a replication in software engineering”. In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. EASE ’14. London, England, United Kingdom: Association for Computing Machinery. ISBN: 9781450324762. DOI: `10.1145/2601248.2601268`. URL: `https://doi.org/10.1145/2601248.2601268`.

Yan, Weixiang et al. (2023). *CodeScope: An Execution-based Multilingual Multitask Multidimensional Benchmark for Evaluating LLMs on Code Understanding and Generation*. arXiv: `2311.08588 [cs.CL]`.

Yan, Weixiang et al. (n.d.). *CodeScope — haitianliu22.github.io*. `https://haitianliu22.github.io/code-scope-benchmark/`. [Accessed 03-06-2024].

Yao, Shunyu et al. (2023). *Tree of Thoughts: Deliberate Problem Solving with Large Language Models*. arXiv: `2305.10601 [cs.CL]`.

Zellers, Rowan et al. (2019). *HellaSwag: Can a Machine Really Finish Your Sentence?* DOI: `10.48550/ARXIV.1905.07830`. URL: `https://arxiv.org/abs/1905.07830`.

Zhang, Beiqi et al. (July 2023). "Practices and Challenges of Using GitHub Copilot: An Empirical Study". In: *Proceedings of the 35th International Conference on Software Engineering and Knowledge Engineering*. SEKE2023. KSI Research Inc. DOI: `10.18293/seke2023-077`. URL: `http://dx.doi.org/10.18293/SEKE2023-077`.

Zhang, Fengji et al. (2023). *RepoCoder: Repository-Level Code Completion Through Iterative Retrieval and Generation*. DOI: `10.48550/ARXIV.2303.12570`. URL: `https://arxiv.org/abs/2303.12570`.

Ziegler, Albert et al. (June 2022). "Productivity assessment of neural code completion". In: *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*. MAPS '22. ACM. DOI: `10.1145/3520312.3534864`. URL: `http://dx.doi.org/10.1145/3520312.3534864`.

Appendix A Interview information

Appendix A contains information regarding interview participants, as well as the interview guides used in semi-structured interviews with software engineers and experts.

A.1 List of interview participants

Table 6: List of interviewees.

Role	Field	Interview Participant
Product Engineer	Web	IP1
Senior Product Engineer	Web	IP2
Principal Product Engineer	Web	IP3
Senior Product Engineer	Web	IP4
Engineering Manager	Web	IP5
Product Engineer	Backend	IP6
Product Engineer	Mac/iOS	IP7
Stability Engineer	Mac/iOS	IP8
Staff Product Engineer	Mac/iOS	IP9
Senior Product Engineer	Web	IP10
Director of Engineering		IP11

A.2 Interview guides

A.2.1 Interview guide for user study and needs analysis

Background

- What's your background?
- What type of software (prgoramming language etc) do you work with?

AI powered tools (AI refer to LLMs in the questions below)

- Do you use any AI powered tools?
- If you are not using any AI powered tools, why so?
- What AI powered tools do you use?
- For what tasks do you use them?
- How do you use them (i.e. how are they incorporated in your workflow)?
- How often do you use the AI powered tools you use?
- How well do you thing the AI powered tools you use work for your tasks?
- Have you found a way of making these tools work better for your tasks?
- What would need to improve for the tools to be more useful?
- What does your ideal AI powered tool look like?

A.2.2 Interview guide for prototype testing

Areas of improvement

- How has the prototype worked for you and what could be improved?
- How could it be designed to work better?

Experience

- Do you feel that it has been useful?
- How does it compare to ChatGPT or the other tools that you usually use?
- What tasks have you tried out on it and how has that worked?

A.2.3 Interview guide for retrospective

- How have you used the assistant?
  - For what types of tasks or problems?
  - How often have you used it?
  - Did you still use other types of LLM tools?
  - Have you used the assistant differently than other tools?
- What is your overall impression of the assistant?
  - Have you experienced similarities or differences with ChatGPT or Copilot?
- How did using the assistant influence your work efficiency/effectiveness? Was it helpful?
- To have a larger positive impact on your work, what would this technology need to be capable of?
  - What things are missing today, or not good enough?

Appendix B Prototype material

Appendix B contains material used in generating the prototypes, such as prompts and scripts.

B.1 Data processing scripts

```
touch combined.md &&
find . \( -name "*.tsx" -o -name "*.ts" -o -name "*.js" -o -name "*.jsx" -o -name "*.html" -o -name "*.css" -o -name "*.json" \) -exec bash -c '
fileWithPath="{}"
echo -e "\n## ${fileWithPath:2}\n\n\`\`\`" >> combined.md
cat "${fileWithPath}" >> combined.md
echo -e "\n\`\`\`" >> combined.md' \;
```

Example 2: Initial version of the processing script. *combined.md* is to be replaced with the respective repository folder's name.

```
touch combined.md &&
find . \( -name "*.tsx" -o -name "*.ts" -o -name "*.js" -o -name "*.jsx" -o -name "*.html" -o -name "*.css" -o -name "*.json" \) -exec bash -c '
fileWithPath="{}";
echo -e "\n## ${fileWithPath:2}\n\n\`\`\`" >> combined.md
cat "${fileWithPath}" >> combined.md
echo -e "\n\`\`\`" >> combined.md' \;
```

Example 3: Second version of the processing script. *combined.md* is to be replaced with the respective repository folder's name.

B.2 Prompts

```
You are a helpful assistant helping developers understand and work with a React/TypeScript repository.

You have access to the {repository name} monorepo in the files attached. Each file is named after a base folder in the monorepo (i.e. 'folder_xyz.md' corresponds to the 'folder_xyz' folder). Too large folders are split into two, denoted with an underscore and a number, i.e. '_1' and '_2'.

Each code file is delimited by a markdown header with the file path and name, and then its contents enclosed in backticks. When referring to code files, make sure to provide the complete path of the file you are referring to,  including the base folder as indicated in the name of the file you have access to, and the code file's complete path within that folder as included in the markdown header after '##'.

Review files that fit the user's question to help them. Don't refer to your knowledge files as "provided by the user". Provide an answer to the user's question based on the user's instruction and the knowledge retrieved from the repository.
```

Example 4: Initial system prompt.

```
You are a helpful assistant helping developers understand and work with a React/TypeScript repository.

You have access to the {repository name} monorepo in the knowledge files attached. Each file is named after a base folder in the monorepo (i.e. baseapp.md corresponds to the baseapp folder). Too large folders are split into two, denoted with an underscore and a number, i.e. _1 and _2. You should absolutely never provide the names to those files in your answers!

Each code file is delimited by a markdown header with the file path and name, and then its contents enclosed in backticks. When you reference code, always make sure to provide the complete path of the file you are referring to, as indicated in the markdown header after ##. As said before, never mention the names of the knowledge files, only the path and name of the code files!

Review files that fit the user's question to help them. Provide an answer based on the files you have access to and your general knowledge. Explain your thoughts and
```

```
   reason through them step by step, and please provide
   examples in code blocks.
```

Example 5: Second version of the system prompt.

B.3 Thread object example

```
"data":{
    "id":"run_BZEarIHxU1QVy5h86gdD5kEx"
    "object":"thread.run"
    "created_at":1712137419
    "assistant_id":"asst_WN0hbTLHE7CphKABLpKPiNW0"
    "thread_id":"thread_4iKGQRlTWPAv8rXLCQHLX2Oj"
    "status":"completed"
    "started_at":1712137419
    "expires_at":NULL
    "cancelled_at":NULL
    "failed_at":NULL
    "completed_at":1712137449
    "required_action":NULL
    "last_error":NULL
    "model":"gpt-4-turbo-preview"
    "instructions":"You are a helpful assistant helping
    developers understand and work with a React/
   TypeScript repository. You have access to the {
   repository name} monorepo in the knowledge files
   attached. Each file is named after a base folder in
   the monorepo (i.e. baseapp.md corresponds to the
   baseapp folder). Too large folders are split into
   two, denoted with an underscore and a number, i.e. _
   1 and _2. You should absolutely never provide the
   names to those files in your answers! Each code file
    is delimited by a markdown header with the file
   path and name, and then its contents enclosed in
   backticks. When you reference code, always make sure
    to provide the complete path of the file you are
   referring to, as indicated in the markdown header
   after ##. As said before, never mention the names of
    the knowledge files, only the path and name of the
   code files! Review files that fit the user's
   question to help them. Provide an answer based on
   the files you have access to and your general
   knowledge. Explain your thoughts and reason through
   them step by step, and please provide examples in
   code blocks."
    "tools":[
        0:{
            "type":"retrieval"
        }
    ]
    "file_ids":[
        0:"file-ERq1A5c5rTqJDDdEgaQS86VU"
        1:"file-JGqXvZLhBoYz35zK4gTSeFJg"
        2:"file-yRJtkMi8nmXhhCEeEiQj06dF"
        3:"file-hGD7CCcKxwHxQLAnXUQ1Mlw6"
        4:"file-EN4olqCpjIC6x009xsMbdnDG"
        5:"file-3ys6WDe9ujNvTZJaq5FXcBhB"
        6:"file-f7X5Ly3tBT1CJoqxrXBls9XE"
        7:"file-IWcVLN81AisSeF5AZqjQH4dU"
        8:"file-n4iQ8zl2M6GZKbPx2GtRGxJ6"
    ]
    "metadata":{}
    "temperature":1
    "usage":{
        "prompt_tokens":28162
        "completion_tokens":605
        "total_tokens":28767
    }
}
```

Example 6: Thread object as retrieved from the Assistant API.