THESIS FOR THE DEGREE OF LICENTIATE OF ENGINEERING

Understanding and Evaluating Chatbot Interactions in Software Engineering

RANIM KHOJAH

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden, 2025

Understanding and Evaluating Chatbot Interactions in Software Engineering

RANIM KHOJAH

Department of Computer Science and Engineering
Division of Interaction Design and Software Engineering
Internet Computing and Emerging Technologies lab (ICET-lab)
Chalmers University of Technology | University of Gothenburg
SE-412 96 Göteborg,
Sweden

Printed by Chalmers Digitaltryck,
Gothenburg, Sweden 2025.

"The art of knowing is knowing what to ignore."
- Rumi

Abstract

Chatbots have been used in software engineering for a long time. Initially, they were based on basic commands. Then, Artificial Intelligence introduced many components such as Natural Language Understanding (NLU) and made the architecture of the chatbot slightly more complex to be able to automate simple tasks such as closing issues on GitHub and to retrieve information and documentation. However, the emergence of Large Language Models (LLMs) unlocked many other possibilities for chatbots, which allowed them to have extensive knowledge and be context-aware while being able to perform complex tasks and make decisions during the software development process. Consequently, chatbots could assist in requirement elicitation, code generation, and even analyzing monitoring logs of the software. This enabled software engineers to explore more possibilities, in particular, focusing on automating complex tasks using LLM chatbots and interacting with them as traditional chatbots. However, this created new challenges that need to be addressed, for example, hallucinating requirements or providing vulnerable code. Consequently, human factors such as trust began to fade slowly. In this thesis, I argue that to better use chatbots for the right use cases, we need to understand the interactions with them, including the usage and conversational flow. In addition, the evaluation of chatbots (both NLU and LLM based) should go beyond their performance and focus on the value that they bring to software engineers through their interactions. Using empirical methods in four observational and experimental studies, I present an analysis of the characteristics of interactions with NLU and LLM chatbots in comparison with those with human developers. NLU chatbots are used as tools where reliability is an evaluation criterion that complements performance. However, interactions with LLM chatbots are more complex and are impacted by many factors that I introduce in a personal experience framework. In addition, I show how different dimensions of productivity are affected based on whether the chatbot is used to provide guidance, manipulate artifacts, or learn new concepts. Moreover, since prompt programming is commonly used to enhance the outcome of the interactions, I show how certain prompt techniques improve code generation, but their overall impact remains limited. Therefore, this thesis guides chatbot designers in enhancing chatbots' ability to communicate to improve the user's personal experience. It also urges practitioners to adapt their use of chatbots to focus on collaborating with them rather than using them as automation tools. This also encourages researchers to investigate effective ways to implement collaboration with chatbots at different stages of the software development lifecycle.

Keywords

Chatbots, Software Engineering, Natural Language Understanding, Large Language Models, Interactions

Acknowledgment

First and foremost, all praise and thanks belong to God Almighty for giving me the courage to pursue this journey.

I want to express my gratitude to my supervisors Francisco Gomes de Oliveira Neto and Philipp Leitner, for their patience, guidance, and unwavering support. I have learned a lot from you. I want to also thank my examiner Robert Feldt, for his valuable feedback and thoughtful advice.

I extend my sincere thanks to my friends Mazen and Sabina for being there when I needed help, always offering encouragement and a listening ear during difficult times. To my dear friends Krishna, Wardah, Habib, Bea, Malsha, Cristy, Amna, Çağrı, Tayssir, Linda, Babu, Sushant, Hamdy, Razan, Ricardo, Teodor and all my colleagues in the Cabo group and IDSE division. You have been a great support in helping me navigate many challenges and celebrate my achievements along the way.

I am deeply thankful to my parents, Fatima Alzhra Mufti and Imad Khojah, for their prayers and for believing in me even when I doubted myself. My siblings Laith, Hamza, Osama, Lin, and Osaid, you kept bringing me joy and laughter during stressful times, I am very grateful. Thank you, Lujain, Abdelrahman and Kerim, for being there to lift my spirits.

Finally, I want to note that my work has been supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

List of Publications

Appended publications

This thesis is based on the following publications:

[**Paper A**] **Khojah, R.**, de Oliveira Neto, F. G., Leitner, P., *From Human-to-Human to Human-to-Bot Conversations in Software Engineering.*
Proceedings of the 1st ACM International Conference on AI-Powered Software (AIware) (2024, July), 38-44.

[**Paper B**] **Khojah, R.**, Mohamad, M., Leitner, P., de Oliveira Neto, F. G., *Beyond Code Generation: An Observational Study of ChatGPT Usage in Software Engineering Practice.*
Proceedings of the ACM on Software Engineering (FSE) (2024, July), 1819-1840.

[**Paper C**] **Khojah, R.**, Berman, A., Larsson, S., *Evaluating N-Best Calibration of Natural Language Understanding for Dialogue Systems.*
In Proceedings of the 23rd Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDial) (2022, September), 582-594.

[**Paper D**] **Khojah, R.**, de Oliveira Neto, F. G., Mohamad, M., Leitner, P., *The Impact of Prompt Programming on Function-Level Code Generation.*
Submitted, under review. preprint arXiv:2412.20545.

Other publications

The following publications were published during my PhD studies, or are currently in submission/under revision. However, they are not appended to this thesis, due to contents overlapping that of appended publications or contents not related to the thesis.

[**I**] **Khojah, R.**, Chao, C. H., de Oliveira Neto, F. G., *Evaluating the Trade-offs of Text-based Diversity in Test Prioritisation.*
In 2023 IEEE/ACM International Conference on Automation of Software Test (AST) (2023, May), 168-178.

[**II**] Martinez Montes, C., **Khojah, R.**, *Emotional Strain and Frustration in LLM Interactions in Software Engineering.*
Submitted, under review.

[**III**] Gultekin, F. M., Lilja, O., **Khojah, R.**, Wohlrab, R., Damschen, M., Mohamad, M., *Leveraging Large Language Models for Cybersecurity Risk Assessment in Cyber-Physical Systems: A Case Study in Forestry Automation.*
Submitted, under review.

Research Contribution

Table 1 summarizes my contribution to each of the appended papers using the CRediT (Contribution Roles Taxonomy) model[1].

Table 1: The author's contribution to the appended papers of this thesis

Role / Paper	**A**	**B**	**C**	**D**
Conceptualization	✓	✓	✓	✓
Methodology	✓	✓	✓	✓
Software	✓		✓	✓
Validation	✓	✓	✓	✓
Formal analysis	✓	✓	✓	✓
Investigation	✓	✓	✓	✓
Resources	✓			✓
Data Curation	✓	✓	✓	✓
Writing - Original Draft	✓	✓	✓	✓
Writing - Review & Editing				
Visualization	✓	✓	✓	✓
Supervision				
Project administration				
Funding acquisition				

[1] https://credit.niso.org

Contents

Chapter 1

Introduction

Chatbots have long been valuable in software engineering, from chatbots that allow developers to "chat" with their repositories, to chatbots that use generative AI to generate software artifacts [1], [2]. Traditionally, chatbots relied on Natural Language Understanding (NLU) and were used mainly as automation tools for repetitive tasks [3]. However, the emergence of AI chatbots, particularly those powered by Large Language Models (LLMs), has introduced new interaction possibilities. These include more natural human-like conversations, support in many software-related activities, and relevant responses that adapt to their context at a project or organization level. This made it possible for such chatbots to help in eliciting requirements, software testing, code generation, among others [4]–[6]. This led researchers to explore fully automating tasks with chatbots [7], but it began to reveal new challenges, such as introducing software vulnerabilities, bugs, or even requiring high resources [8], [9].

These challenges require a shift of focus to *collaborating* with chatbots and leveraging their conversational capabilities rather than limiting them to automation tools. To enable better collaboration between software practitioners and chatbots, it is crucial to understand how developers interact with chatbots, considering factors such as context, intent, and chatbot characteristics [10]. Without such an understanding, we fail to capture how to assess chatbots in terms of the value they bring to software engineers. This makes it difficult to measure the chatbots' true impact on the productivity, trust, and decision making of software engineers.

1.1 Research goal

This thesis addresses challenges related to the interactions of chatbots for different purposes by achieving two main goals in Figure 1.1, that is, (G1) understanding why and how software practitioners interact with chatbots, as well as (G2) evaluating these interactions using broader criteria beyond performance, e.g., accuracy by including factors such as reliability and user experience. To achieve these goals, I formulate the following research questions:

RQ1. What are the characteristics of the interactions with chatbots?

To answer this, I analyzed the differences in conversations with chatbots and with human colleagues in terms of their conversational properties, such as purpose, understanding, and trustworthiness. Then I further investigate the conversational flows and purpose of using LLM-based chatbots in software engineering by analyzing 180 dialogues (580 prompts) between software practitioners and ChatGPT.

RQ2. How can interactions with chatbots be evaluated?

Going beyond the performance of NLU and LLM chatbots, I define criteria on what aspects beyond performance are important to evaluate in chatbots. As a result, I perform an evaluation of the reliability of NLU chatbots by looking at how well-calibrated NLU models are, as well as the trust and productivity that practitioners perceive after interacting with an LLM chatbot.

RQ3. How does prompt programming impact the interaction's outcome?

I conducted a full factorial experiment to evaluate the impact of five common prompt programming techniques and their combinations in a prompt on the code generation of three popular LLMs. As a result, I introduced a dataset of 7072 prompts to evaluate the impact of prompt programming on the correctness, quality and readability of the generated code.

The answers to the research questions are presented in four papers that form the foundation of this thesis. Figure 1.1 illustrates how each paper and its corresponding research questions contribute to achieving the two main goals of the thesis.

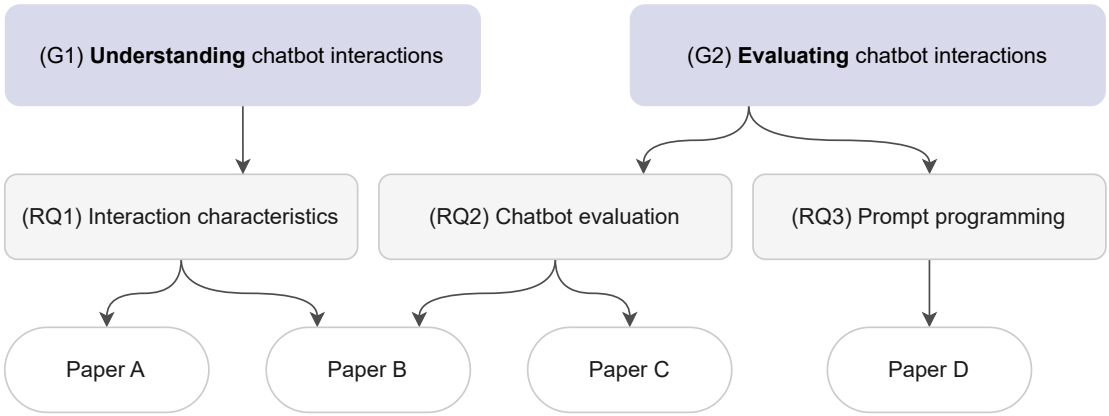

Figure 1.1: Mapping between the research goals, questions, and appended papers.

1.2 Background and related work

Chatbots in software engineering have become common tools that play an important role in software teams [11]. Traditionally, chatbots relied on Natural Language Understanding (NLU) models, provided by services like IBM Watson[1] (closed-source)

[1] https://www.ibm.com/watson

and Rasa NLU[2] (open-source). These NLU models perform intent classification on a user prompt and predict several hypotheses of the intention of the user with the corresponding confidence scores. Then, based on the top intention and its confidence estimate, a pre-defined response is returned to the user [12]. An illustration of this process is in Figure 1.2.

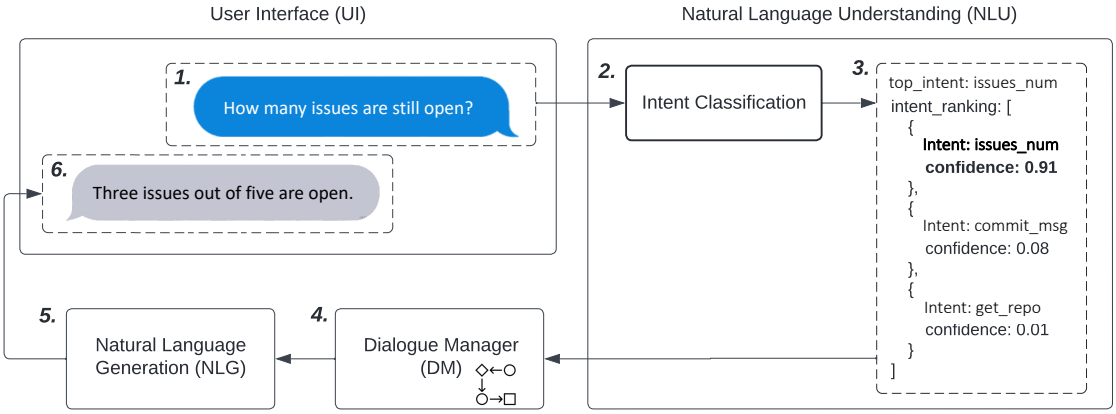

Figure 1.2: An explanatory example that demonstrates the role of NLU in a traditional chatbot.

Previous research has focused on the role of traditional chatbots in software engineering. Erlenhov et al. [10] categorize different types of bot, including chatbots, based on the tasks they can perform in software engineering. Hefny et al. [13] leverage the automation capabilities of NLU chatbots to handle management-related tasks within software teams. Abdellatif et al. [14] focus on information retrieval and introduce MSRbot, which allows developers to interact with their GitHub repositories through chat. As a result, the evaluation of NLU chatbots has primarily focused on their performance, e.g., accuracy [1].

However, the emergence of large language models (LLMs) shifted the focus to studying the abilities of LLMs such as Codex [15] and GPT models [16] that power GitHub CoPilot and ChatGPT, respectively. LLMs stood out due to their ability to generate complex and context-specific responses, including software artifacts such as test suites and documentation [17]. Therefore, researchers began exploring the potential of such LLM chatbots in software-related tasks [18] such as requirement elicitation [4], program repair [19], and most commonly, code generation [6].

Unlike NLU chatbots, evaluating LLM chatbots is more complex and cannot be captured by a single measure, such as performance. Current evaluations depend on the generated output, which is either qualitatively assessed by human experts or compared to human-written baselines [20]. For some generated outputs, metrics such as BLEU [21] for text or CodeBLEU for code [22] are appropriate for measuring human-likeness or similarity to a human-written baseline. Specifically for generated code, the most common metric is Pass@k, which measures the rate at which LLM-generated functions pass a test suite within k attempts [15], [23].

Beyond evaluating the outcome, recent work has looked at the practical impact of LLM chatbots on the efficiency and productivity of software engineers in practice

[2] https://rasa.com

[24], [25]. Evaluating other human factors, such as trust, is challenging, as it can be subjective and influenced by individual user experiences and expectations [26], [27].

To improve the generated outcome, prompt programming (or prompt engineering) has been recommended by researchers and LLM providers such as OpenAI [28]. White et al. [29] proposed different prompting techniques for different types of software-related tasks.

For code-related tasks specifically, Wang et al. showed that prompt techniques have a positive effect on code generation in the domain of education [30]. Based on this, researchers have proposed various prompt strategies that incorporate contextual information to enhance code-related tasks, such as using data flow information to improve code summarization [31], [32]. Common prompt techniques are Few-shot learning [33] where shots (or examples) are provided within the prompt to help the LLM generalize to new tasks without fine-tuning [33]. Automatic Chain-of-Thought (CoT) guides the LLM to solve problems step by step [34]. Finally, persona techniques enable the LLM to adopt specific roles to generate responses from particular perspectives [35].

This is a moving field, where both the technology i.e., chatbots and the nature of interactions with them have been changing as advancement in state of the art and practice surface. In this thesis, I investigate these evolving interactions through an experiment on prompt programming, demonstrating how different prompt formulations can influence various aspects of the generated output. To further contextualize these findings, I include the viewpoints of practitioners in different roles to complement previous research on the potential of LLM-based chatbots [6], [36].

1.3 Research approach

This thesis presents empirical studies that were conducted to understand the interactions between software engineers and chatbots, as well as to evaluate different aspects of them through observations and experiments, respectively. A summary of the different data collection and analysis elements that shaped our research approach is given in Figure 1.3.

The observational approach was appropriate to explore the current usage of chatbots in software engineering practice and the nature of interactions. For this, I recruited 24 software practitioners of different roles (e.g., requirement engineers and software testers) who worked across 10 software organizations of varying sizes and domains. This methodology included gathering interaction logs (conversations with an LLM chatbot) of software engineers over a period of one week and covering reflections about different aspects of their personal experiences during interactions using an exit questionnaire. Data analysis was mainly qualitative, including content analysis to describe conversational flow, the type of conversation, and the purpose of use. The analysis also included interpretative phenomenological analysis (IPA), which is a qualitative research approach aimed at understanding how individuals perceive their personal experiences in a specific context [37]. For this research, I used IPA specifically to capture the personal experience of the study participants when interacting with an LLM chatbot for a duration of one week.

Based on the different observations about interactions with both NLU- and LLM-

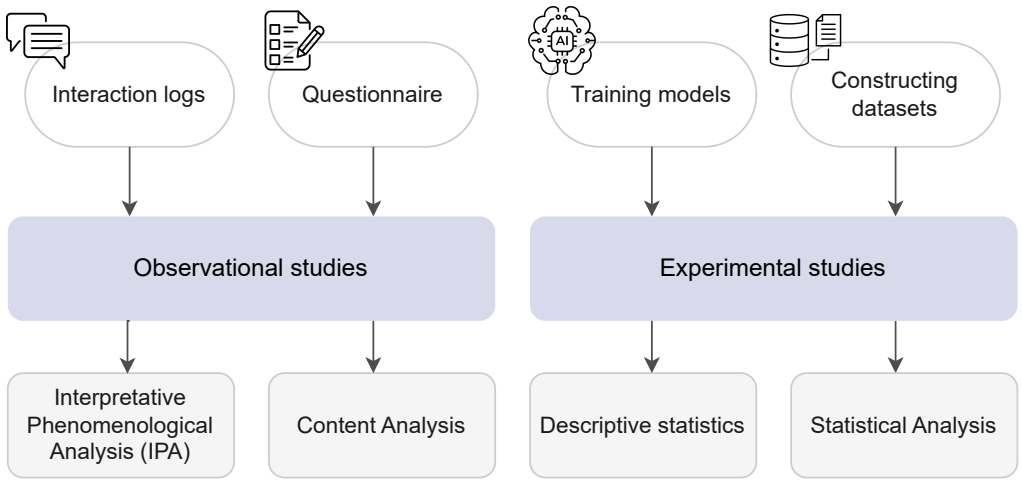

Figure 1.3: The type of studies that were conducted to reach the research goals along with the data collection and analysis elements.

based chatbots, I instrumented two experiments to evaluate: the reliability of NLU chatbots that are mostly used for simple task automation, as well as the outcome of the interactions with LLM chatbots specifically when used for code generation. These experiments required constructing datasets to train and/or test the two types of chatbot.

As a result of the observational and experimental studies, I was able to introduce (i) an analysis of the conversational flow of conversations within a software team, (ii) a theoretical framework of the factors that influence the personal experience of interactions with LLM chatbots, and (iii) CodePromptEval[3], which is a dataset consisting of 7072 prompts used to evaluate the impact of (combinations) of prompt programming techniques on code generation. I detail those contributions in the next sections.

1.4 Nature of chatbot interactions

The nature of interactions within a software team depends on the communication partner — whether they are human colleagues or chatbots. These interactions can vary in terms of purpose, style, and expressions involved, among others.

In this section, I discuss the characteristics of conversations between software engineers and chatbots of two types (NLU-based and LLM-based) in comparison to conversations with human colleagues in Section 1.4.1. Then, in Section 1.4.2, I further investigate the conversational flow, purpose, and personal experience of software engineers interacting with LLM chatbots specifically.

The findings in the following sections are based on the observational study illustrated in Figure 1.4. The observational study involved 24 participants from 10 different organizations who used an LLM chatbot (ChatGPT) for one week and then completed an exit survey to reflect on different aspects of their interaction experience. I then qualitatively and quantitatively analyze their chat logs and survey responses.

[3] https://github.com/icetlab/CodePromptEval

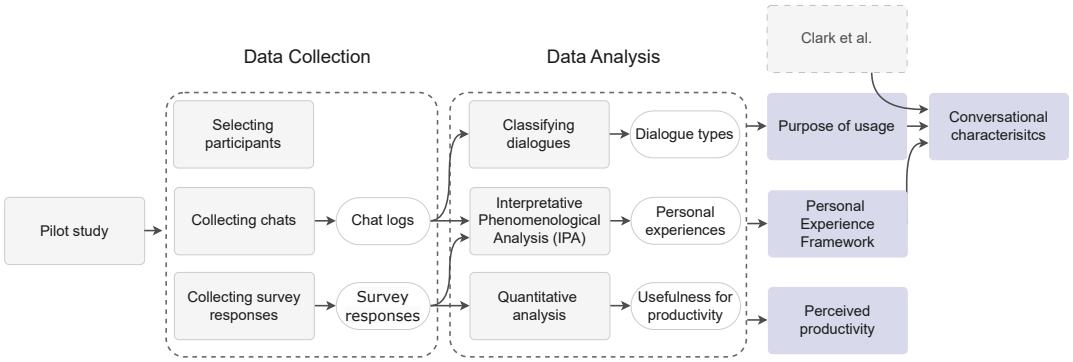

Figure 1.4: The process followed in Paper A [27] to understand the nature of interactions with chatbots. Paper B [24] uses the same methodology, but also includes the insights from Clark et al. [38] to analyze the conversational characteristics.

1.4.1 Characteristics of interactions in software teams

I followed the characteristics of conversation defined by Clark et al. [38] along with a detailed comparison between conversations with traditional (NLU-based) chatbots and humans. These characteristics were the most suitable because they are based on aspects that people value in conversations and can be applied to interactions with humans and chatbots. After adopting the comparison to development chatbots and human software developers specifically, I include another element in the comparison, that is, LLM chatbots. An overview of the characteristics of the different conversational styles is depicted in Table 1.1.

Table 1.1: Summary of the conversation attributes between software developers and (i) other developers, (ii) NLU-based chatbots, and (iii) LLM-based chatbots

	Human Developers	**NLU-based Chatbots**	**LLM-based Chatbots**
Purpose	Social, general guidance, training	Basic information retrieval, simple automation	General guidance, training, artifact generation and manipulation
Understanding of Scope	Mutual understanding	Fixed customization	Dynamic customization
Listening	Body language and knowledge	Acknowledgment and intent classification	Query summary and knowledge
Trustworthiness	Shared experiences and previous interactions	Performance and efficiency	Meeting expectations and transparency
Use of Humour	Common	Not applicable	On-demand

The comparisons revealed that traditional NLU-based chatbots, e.g., GitHub bot[4] are treated by developers more as tools that are configured to understand a specific context, e.g., a repository. Communication (listening) relies on intent classification and acknowledgments such as “Commit pushed.”. Conversational aspects such as trust

[4] `https://slack.github.com`

are determined by pre-defined performance metrics and configuration settings, rather than evolving through dynamic interactions.

On the other hand, LLMs offer various use cases and conversational possibilities that are inherited from their "bot" nature along with their language capabilities. Such capabilities include their ability to comprehend a wide range of topics and adapt responses based on the context of the conversation. This makes them more flexible than traditional chatbots, which are typically used to automate simple tasks (e.g., retrieving documentation or track issues on GitHub). LLMs are also more adaptable than humans, for whom providing software artifacts and considering multiple contexts simultaneously may be challenging to achieve in a conversation. Regarding other conversational characteristics, the ability to build social bonds is still limited in LLM chatbots compared to human developers, other aspects like trust and the development of shared understanding can gradually be built through several interactions and experiences. The acknowledgment in LLM chatbots is illustrated in the summary that the LLM provides of its own understanding before it gives an answer. Humor is still an aspect that is more acceptable in interactions with humans than chatbots.

These different conversational styles are not mutually exclusive, but rather complementary — the limitation of one conversational style, e.g., social bonds, can be mitigated in another style. Therefore, combining different conversational styles can help minimizing the limitations of individual ones and amplify their advantages.

Characteristics of interactions (RQ1): LLM chatbot conversations are flexible, enabling human-like interactions with bot-like efficiency. However, they cannot replace human conversations due to social aspects, nor can they replace NLU chatbots, which prioritize outcomes over dialogue.

1.4.2 Purpose of interactions with LLM chatbots

The purpose of interaction is on of the main conversational properties that I introduced in Table 1.1. To better understand the purpose and how software engineers in different roles use LLM chatbots and how their conversations are structured, I performed a qualitative analysis and classification on 180 dialogues. These dialogues were collected from interactions between practitioners from 10 software organizations and ChatGPT consisting of a total of 580 prompts. I present an overview of the participants in Table 1.2 including their roles, responsibilities, and domains. The organization size classification follows the categories recommended by the European Commission [39].

There were three main types of dialogue. The first is **artifact manipulation**, where the conversation focuses on creating or modifying an artifact, such as test cases. The second is **expert consultation**, in which the chatbot is treated as an expert and consulted for recommendations or guidance on specific topics. The third type is **training**, characterized by longer conversations with follow-up questions and requests for examples, which aims to help the engineer learn or deepen their understanding on a specific topic or technology.

Although LLM chatbots have generative capabilities that distinguish their interactions from those with human engineers or traditional chatbots — allowing them to manipulate software artifacts — practitioners mostly used ChatGPT for guidance (62%

Table 1.2: Demographic information about the participants. IDs refer to different participants and their corresponding organisations. The sizes used are Startup, Small and Medium enterprises (SME), and Large enterprises.

ID	Role	Responsibilities	Org. ID	Org. Size	Domain
P1	Software Tester	UX inspections, usability and testing	A	SME	Testing
P2	Test Engineer	Test case execution	A	SME	Testing
P3	Test Engineer	Test planning, design, and execution	A	SME	Testing
P4	Software Engineer	Development and maintenance	B	SME	E-learning
P5	Software Engineer	Architecture and platform development	B	SME	E-learning
P6	Full-stack Developer	Development of web app's new features	B	SME	E-learning
P7	Product Manager	Managing Frontend Digital Dep.	C	Startup	Medical
P8	Cloud Architect	Architect applications and infrastructure	C	Startup	Medical
P9	Software Engineer	Development and maintenance	C	Startup	Medical
P10	DevOps Engineer	Manage and maintain the pipeline	C	Startup	Medical
P11	Software Developer	Front-end development	D	Startup	Gaming
P12	Game Developer	Programming game logic	E	Startup	Gaming
P13	Software Developer	Development, code review and testing	F	Large	E-commerce
P14	Group Manager	Managing software development teams	G	Large	Automotive
P15	Software Engineer	Mobile app feature development	G	Large	Automotive
P16	Android Developer	Development and maintenance	G	Large	Automotive
P17	System Leader	System Design	G	Large	Automotive
P18	Sub-portfolio Manager	Creation of roadmaps	G	Large	Automotive
P19	Product Manager	Creation of roadmaps	G	Large	Automotive
P20	Android Developer	Mobile app feature development	G	Large	Automotive
P21	Software Engineer	Development	H	Large	Consultancy
P22	Head of Operations	Define and establish processes	I	SME	Consultancy
P23	Software Developer	Development and requirement analysis	I	SME	Consultancy
P24	Software Engineer	Development and requirement analysis	J	Large	Automotive

of dialogues) rather than directly for generating software artifacts (32% of dialogues) or learning (6% of dialogues), and even found it more useful for this purpose (see Section 1.5.2). After a further analysis of the dialogues, many use cases for ChatGPT in software engineering practice were revealed beyond only generating code artifacts. Many participants described ChatGPT as creative, smart and attentive like humans. Therefore, they used it to support them in decision-making, problem solving, and brainstorming. Others limited their usage to bot-like tasks such as information retrieval and performing simple side tasks such as renaming files. A summary of the different use cases can be found in Figure 1.5.

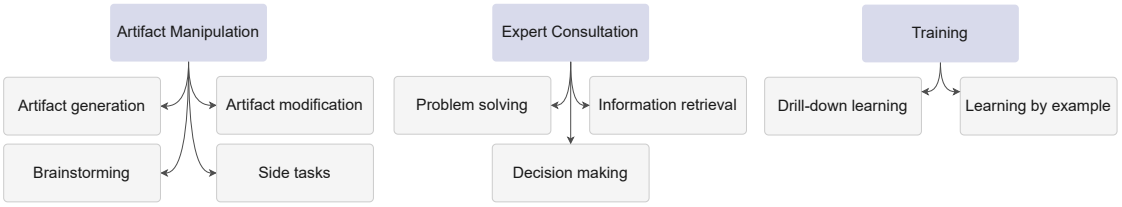

Figure 1.5: Taxonomy of purposes for the usage of ChatGPT in software engineering.

However, after performing an interpretative phenomenological analysis on the different scenarios that the participants described, I noticed that these interactions and the overall personal experience were influenced by many factors that the participants described in their survey responses. Based on the dialogue purposes and these factors,

I construct a framework of factors that impact the personal experience of interactions with ChatGPT in Figure 1.6.

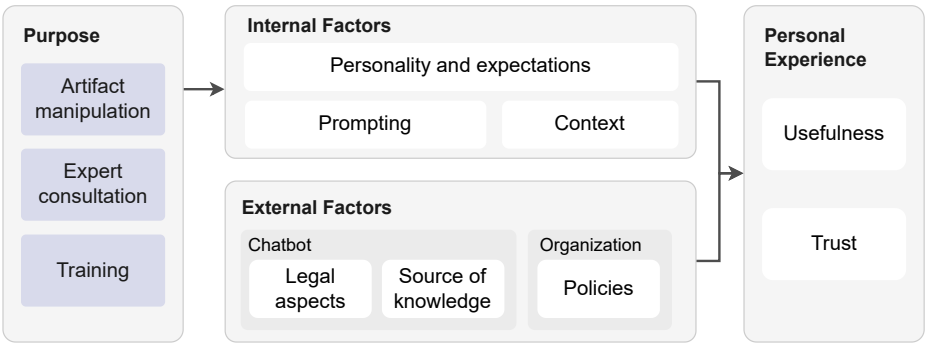

Figure 1.6: A theoretical framework of the factors that influence the personal experience of interactions with ChatGPT in industrial software engineering.

For instance, let us take the example of a test engineer who is skeptical about the use of AI in software engineering, and always expects accurate outcomes from any tool (personality and expectations). She works in a company with a strict policy about the use of generative AI (organization-related external factor), and has also heard about how the conversations in ChatGPT are used for retraining the base LLM (chatbot-related external factor).

> "I can't really ask ChatGPT to help me analyze requirements since I am not allowed to share that information outside my company." (Test engineer).

When notices an ambiguous requirement, and wants to use ChatGPT to modify it (purpose), although she phrases the prompt clearly (prompting), she needs to make sure not to provide sensitive information regarding the original requirements (context). All of these factors contribute to the degree to which this interaction is useful and helps building trust towards the chatbot. As a result, the engineer reported little usefulness in reducing repetitive tasks and low trust in the chatbot.

> "I would also not put too much trust in the answers to such complex questions" (Test engineer).

The personal experience framework helps researchers analyze how various factors influence the usefulness and trust in LLM chatbot interactions. By offering a structured approach, it enables academic discussions on LLM chatbot adoption in software engineering and serves as a foundation for future LLM empirical research in this field.

Characteristics of interactions (RQ1): The purpose of LLM chatbot interactions is to manipulate artifacts, get guidance, and learn new concepts. The overall personal experience that affects productivity and trust is impacted by factors such as the purpose, personality, and the company's policy.

1.5 Evaluation of chatbots

Based on the purpose of interactions and the expectations of practitioners, evaluating chatbots based on their accuracy (or correctness of the prediction or outcome) can be limiting, as it does not capture the personal experience and human factors such as trust. Chatbots should be evaluated based on the criteria that are important to users, bring them value, and align with their goals. For instance, NLU chatbots that are used for automation, such as resolving issues on GitHub, should be accurate but also reliable and consistent. In LLM chatbots, practitioners employ them for context-specific and general-purpose tasks in the expectation that it can help them to be more efficient and productive. Next, I present an evaluation of NLU chatbots in terms of their calibration and reliability in Section 1.5.1, as well as an evaluation of an LLM chatbot based on the perceived productivity by software engineers in Section 1.5.2.

1.5.1 Reliability of NLU chatbots

NLU chatbots are powered by NLU models that control the flow and the outcome of the interaction. The NLU models predict the user's intention and state the confidence level for the prediction. Based on the confidence estimation, the chatbot can be assigned a threshold for the confidence to decide what output to return to the users. For instance, if the NLU model predicts that the practitioner wants to merge a pull request with a confidence estimate of 60%, then it should ask a clarification question to ensure that the intention is to merge a pull request before performing an action. Therefore, I evaluate the confidence calibration of five popular NLU services. Confidence calibration is the extent to which a model can provide a confidence estimation that reflects the true likelihood of the respective intent prediction [40]. For instance, in a reliable and well-calibrated NLU model, a prediction with a confidence estimate of 0.7 is correct in 70% of the cases.

The five NLU services are IBM Watson Assistant, Microsoft's Language Understanding Intelligent Service (LUIS), which are closed-source as well as the open-source NLUs Snips.ai, and Rasa (with two pipelines Rasa-Sklearn and Rasa-DIET). I found that while more popular NLUs such as Watson have higher performance and accuracy, the corresponding confidence estimates for their predictions are not reliable and do not reflect the accuracy or the likelihood of that prediction. In Figure 1.7, I present the reliability diagram showing how Rasa Sklearn, an open source NLU, shows the best calibration (aligns with the diagonal line).

The diagram shows that all NLUs have a generally monotonic relationship between confidence and accuracy. In particular, Rasa-Sklearn is the closest to the gold standard, and is thus the best calibrated NLU according to this analysis. There is also a sign of under-confidence in Snips which estimates a confidence lower than the true likelihood of predictions, while LUIS is over-confident.

Given that Watson is the best performing NLU and Rasa-NLU has lower accuracy levels, there is a trade-off between reliability and performance. This trade-off can influence the decision on which NLU chatbot to use for different purposes, depending on which criteria bring more value to the user.

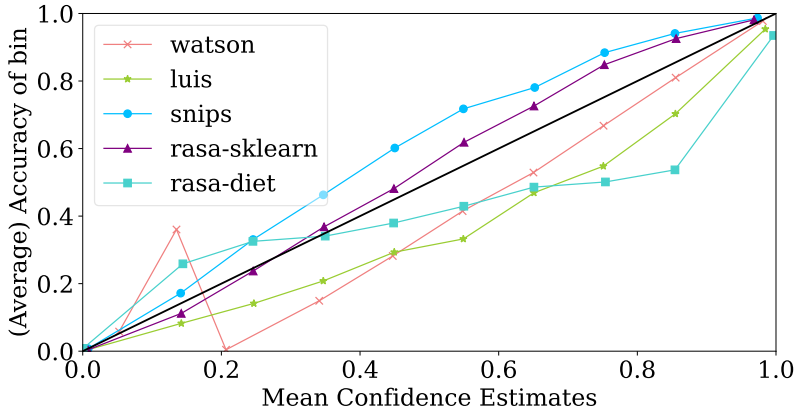

Figure 1.7: Reliability diagram. The x-axis shows the confidence estimates, while the y-axis shows the likelihood in terms of accuracy. The diagonal line represents a gold standard of a perfectly-calibrated model [41].

Chatbot Evaluation (RQ2): NLUs are mainly evaluated on performance, but calibration provides insights into reliability, which is a key factor for automation. However, there was a trade-off between the NLU's performance and reliability.

1.5.2 Productivity of LLM chatbots

When it comes to LLM chatbots and their usage, it is important to understand their ability to *efficiently* assist with complex coding tasks and problem solving. Therefore, using the methodology in Figure 1.4, I used a chatbot productivity framework by Storey et al. [42] to design a Likert scale in the survey to assess the perceived usefulness of different productivity dimensions such as reducing repetitive tasks, initiating discussions with the team, and maintaining focus. In Figure 1.8, I present the results of practitioners rating the usefulness of these productivity dimensions in Figure 1.8 when interacting with ChatGPT, as well as their trust in its answers.

ChatGPT was perceived as useful for learning new concepts and making better decisions, but can hurt team discussions and split focus, which are important dimensions of productivity. When it comes to reducing repetitive tasks, the usefulness depended on the purpose of usage. In addition, most of the participants trusted ChatGPT, and those who had little trust due to lack of transparency were still willing to use ChatGPT.

Overall, this reinforces the point that such chatbot evaluations help identify weaknesses in aspects that users find important in software engineering activities. Such input provides chatbot designers with valuable insights to improve the usability and user trust in the chatbots they build.

Chatbot Evaluation (RQ2): LLM chatbots were useful in some productivity dimensions, like making better decisions, but over-reliance on chatbot responses and excessive prompt tweaking can reduce focus and team communication.

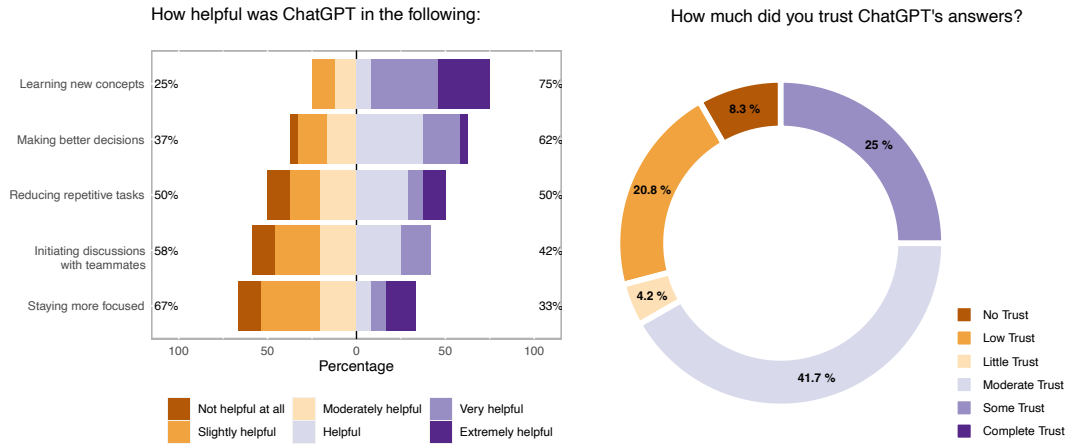

Figure 1.8: Plots showing how the 24 participants reported ChatGPT's usefulness for productivity aspects (left) and trust in its answer (right).

1.6 Impact of prompt programming

Improving the outcome of an interaction with chatbots can occur on multiple levels. For instance, in NLU models, performance is improved by fine-tuning the model itself, adjusting parameters and configurations, or improving intent recognition [43]. In LLM chatbots, the improvement can be performed on an interaction level, that is, using the prompts. Researchers call this concept *prompt programming* (or prompt engineering). It involves incorporating prompt techniques and providing relevant contextual information in order to minimize the limitations of the model and trigger the LLM to produce a more desirable response [28]. This encouraged researchers and LLM providers to come up with prompt techniques that can improve, inter alia, the relevance, correctness and quality of the outcome (see OpenAI strategies[5] and White et al. [29]).

Therefore, I wanted to investigate whether these prompt techniques are actually effective in software engineering, particularly in code generation. I evaluated prompts with all possible combinations of 5 main prompt techniques which sum up to 32 unique combinations of prompt techniques for each generation task. The prompt techniques are few-shot learning, persona, chain-of-thought, function signature, and list of packages.

To perform this evaluation, I follow the process in Figure 1.9, which started by constructing a dataset called CodePromptEval that consists of 7072 datapoints based on CoderEval's [44] 221 generation tasks. Then, I evaluate the generated code by three popular LLMs, that is, GPT-4o (200B parameters), Llama3 (70B parameters), and Mistral (22B parameters) in terms of the code correctness, quality and similarity to human-written code (ground truth).

I found that the inclusion of more prompt techniques in a prompt does not have an impact that is more significant than including one specific prompt technique. For instance, I found that providing few-shot examples or the signature of the function yields more correct functions (i.e., pass assigned tests).

[5] `https://platform.openai.com/docs/guides/prompt-engineering`

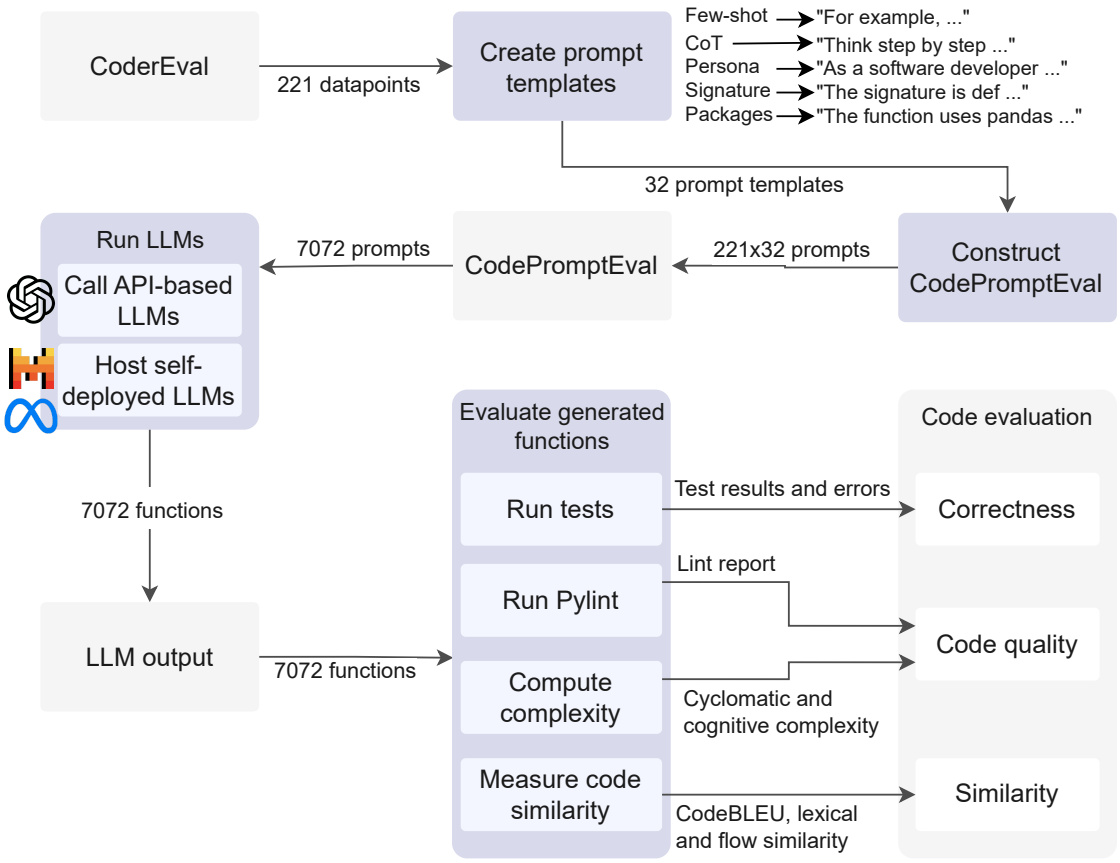

Figure 1.9: The process followed in Paper D [45] to evaluate the code generated using different prompts by different LLMs.

The results of the multi-linear regression analysis in Figures 1.10 and 1.11 show the effect of five prompt techniques and their interactions on the test results (pass/fail) and similarity (using CodeBLEU), respectively. For instance, "CoT:Persona" describes if the impact on the test results comes from the interaction of CoT (Chain of Thought) and persona in a prompt, regardless of whether that prompt includes other prompt techniques. Similarly, "Sig." (signature) refers to all prompts that include at least the signature and is not limited to prompts that only specify the signature.

I found that the signature of the function seemed to have the highest impact on the correctness of the code and its similarity to the ground truth. Few-shot also had a positive impact but with lower significance levels and only for selected LLMs. While persona and CoT do not have a significant impact on correctness (coefficient estimate below zero), they seem to have a significant positive impact on the similarity of the function to the ground truth regardless of whether the function is correct or not. In general, I note that the difference between prompt techniques with positive and negative impact is surprisingly low (the difference between the most negative and most positive is less than 0.04).

When looking at the quality-related results, there was a trade-off between correctness and quality. In other words, the prompts that led to generating more correct functions also produced functions with low quality, e.g., with many code smells. In Figure 1.12, I highlight the different percentages of code smells (represented by IDs

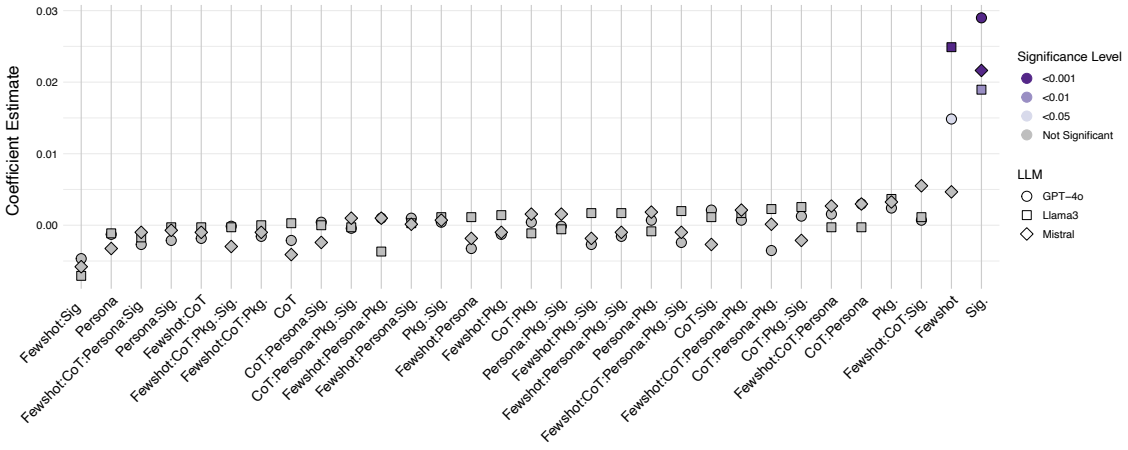

Figure 1.10: Multi-linear regression results for the test results (pass or fail). Each point visualizes the coefficient estimate for the corresponding combination. Coefficients close to zero have little effect on the number of tests passed. The darker colors represent more conservative significance levels (α), hence higher effects.

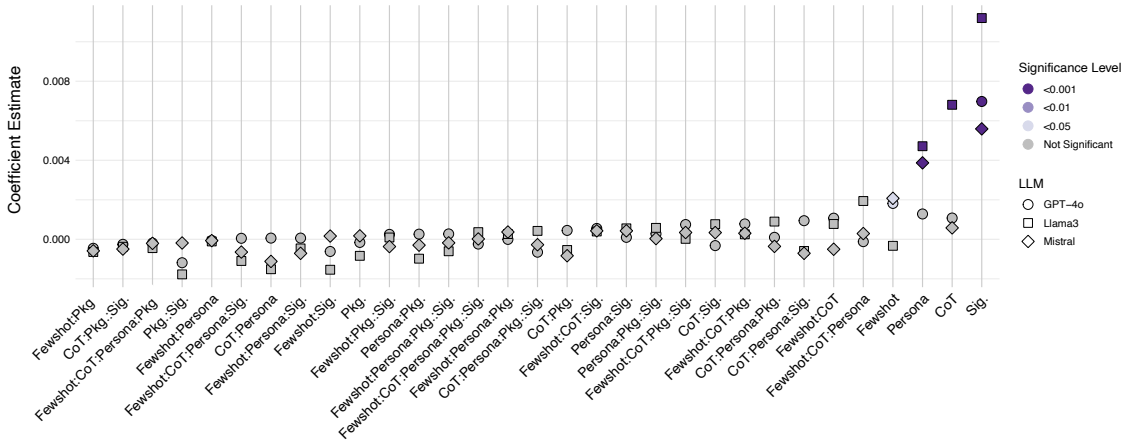

Figure 1.11: The coefficient estimates from the multi-linear regression of prompt technique combinations that significantly impact the CodeBLEU. Each point visualizes the coefficient estimate for the corresponding combination. Coefficients close to zero have little effect on the number of tests passed. The darker colors represent more conservative significance levels (α), hence higher effects.

defined by Pylint[6]) for functions generated by each of the combinations of prompt techniques (the complete table of code smells can be found in Paper D [45]).

The prompts that include personas (e.g., "As a developer who cares about clean code, and ..."), CoT ("Think step by step ..."), and sometimes specify the packages return functions with the lowest rate of code smells. Few-shot examples and signatures cause the code smell rate to increase. An example of a code smell is C0116, which is related to conventions (hence the C in the ID) and means that the function lacks a descriptive docstring. The heatmap shows that 72% of the functions generated using the few-shot technique contain C0116 code smells, in contrast to only 17% of the functions generated by CoT combined with persona and package information.

[6] `https://pylint.readthedocs.io/en/stable/user_guide/messages`

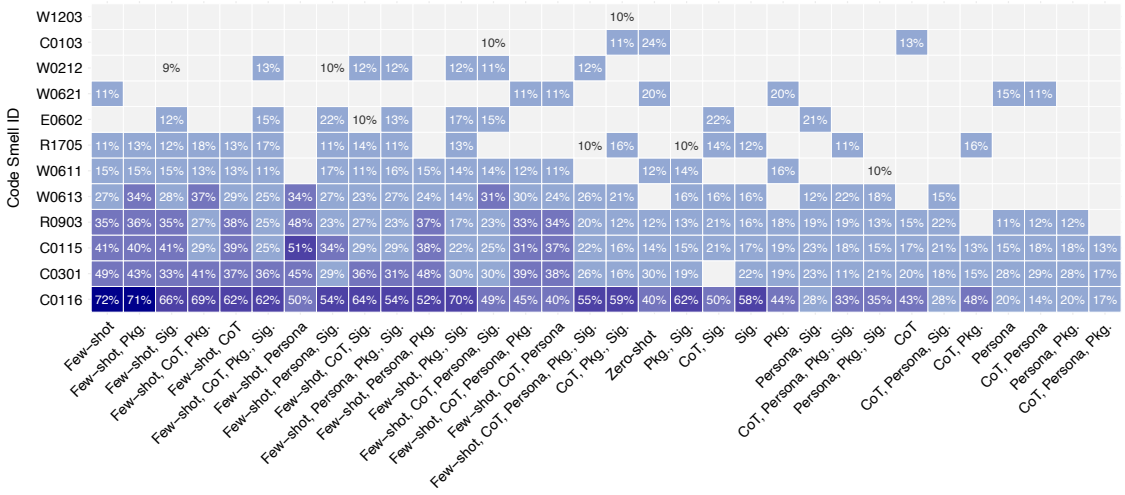

Figure 1.12: Percentages of functions generated by GPT-4o that have different code smells. Empty fields indicate that no smell of this type is found. Note that functions can have instances of multiple types of smells.

Impact of prompt programming (RQ3): Prompt programming generally has a low impact on code generation. However, few-shot learning and providing a signature in a prompt can improve the correctness of the code. Chain-of-thought and providing a persona improves the code quality.

1.7 Reflections

Using LLMs to power chatbots Based on the characteristics of conversations within a software team in Section 1.4.1, I argue that although LLM chatbots offer a wide range of advanced capabilities, they cannot substitute NLU chatbots or human developers. There should be an understanding that NLU chatbots still have features that have not been achieved by LLM chatbots, such as transparency (by displaying a confidence score), consistency in responses, and low resource requirements. However, the current shift of interest towards LLMs made chatbots users and designers overlook the advantages of NLU chatbots. Even some NLU providers, such as Rasa, updated their model configurations to also include LLMs to perform intent classification[7] although LLMs have been shown to perform poorly in classification tasks [46], [47]. This shows that there is a tendency to prioritize hype and public interest over practical efficiency.

Less human intervention when using LLMs Recent research has been investigating the idea of using LLMs to completely automate software-related activities [19], [48], or building multi-agent systems, where a system is composed of multiple LLM-based components that communicate with each other in order to perform a complex task. This means that there are very limited interactions with humans [49]. However, this

[7] https://rasa.com/docs/rasa/next/llms/large-language-models/

approach exposed new challenges that need to be addressed. For example, using LLMs to automate function-level code generation requires dealing with vulnerable and buggy code, hallucinations, or code that does not align with organizational standards [9], [48]. This caused extensive research on prompt programming that aims to improve the outcome of a response to a prompt by introducing new prompt techniques [33]–[35]. However, in Section 3.4.2.1, I show how these prompt techniques can have very little impact on complex tasks, such as code generation. In addition, the absence of human oversight in some activities or reliance on LLMs for decision making decreased trust and confidence in the software product. Also, it limits initiating discussions with the rest of the team, which hurts productivity (See Section 1.5.2).

Towards collaboration with chatbots Despite the challenges that emerge when using LLMs to automate complex tasks, LLMs should not be avoided. It should encourage chatbot users to focus on **collaboration** with LLM chatbots and to make use of the conversational advantages they offer. In Section 1.5.2, I show how practitioners who engaged with the full conversation — using the chatbot for guidance and learning new concepts — found the interaction more useful and productive.

From a chatbot designer perspective, there should be a focus on improving certain aspects of LLM chatbot communication that humans and NLU chatbots overcome. Wu et al. [50] introduced HumanEvalComm, a benchmark that evaluates conversations with LLM chatbots in terms of inconsistency, ambiguity, and incompleteness. Improving these conversational aspects not only improved the measures related to the quality of the conversation, but also improved the correctness of the generated code. Zhang et al. [51] also showed how their developed approach that supports Human-LLM teamwork can significantly improve the performance and relevance of the outcome of the interaction.

Evaluating chatbot interactions and prompt programming Based on the previous reflections, I argue that we should rethink the evaluation of chatbots of different kinds, where the criterion that should be prioritized is about the value that the chatbot brings to practitioners rather than performance. The definition and scope of value differ across chatbots and use cases. In this thesis, we saw how reliability of NLU chatbots brought more insights about their interactions [41], as well as the productivity of LLM chatbots, which was connected not only to the outcome of the chatbot, but also to the interaction as a whole.

This also applies to prompt programming, where selected prompt techniques can improve certain aspects of the generated code (See Section 3.4.2.1). Similarly, in other code-related tasks, prompt techniques and incorporating certain context information have shown potential to improve the result [30]–[32]. However, I found that the overall impact is low, which raises an important question: should prompt techniques be designed to improve the *outcome* of a single prompt, or should they aim to enhance the *entire interaction* in terms of the value they bring to the user, for instance, by ensuring a productive and seamless user experience?

Final remarks To sum up, understanding the nature of interactions with chatbots (first goal of the thesis) highlights the need to adapt our interaction model for better collaboration. In addition, evaluating chatbots based on reliability and productivity (the second goal) reveals their impact on workflow efficiency and decision making, which emphasizes the need to balance performance with user experience.

1.8 Next steps

To move forward, I want to explore how chatbots can become collaborators within a software team. Consequently, I want evaluate the dynamics and the outcome of the team as a whole when chatbots are involved. In particular, in a follow-up study, I will analyze different use cases in a development process that require decision making (which can be handeled by the practitioner), and the tasks for which NLU and LLM chatbots can be useful (e.g., automating simple tasks and provide guidance respectively). When I achieve the understanding of how to effectively collaborate with chatbots among software engineers, trust can increase, which can encourage software organizations to start integrating chatbots into their process and workflow.

However, providing empirical evidence to software organizations is not sufficient, since other factors can be involved e.g., policies and relationship with customers. Therefore, I am currently conducting a study to understand concerns of integrating chatbots on an organizational levels. In addition, I look deeper into their policies and what would need to be added or changed to allow a safe and efficient integration of chatbots in their development process.

Moreover, I argue that researchers should rethink the appropriate use cases for prompt programming and clarify the value these techniques would bring to software engineers. In the future, I want to revisit the goal of prompt programming, whether it is for improving understandability and the communication with the chatbots or for improving the outcome. Then, I can introduce new prompt techniques that would focus on improving the interaction on a dialogue level rather than prompt level. The introduction of a new prompt technique should come information about its impact on different use cases and interactions within software engineering.