SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Informatics

An LLM-based Method to Generate Instructions for Users How to Navigate Web-based User Interfaces

Doruk Gerçel

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Informatics

An LLM-based Method to Generate Instructions for Users How to Navigate Web-based User Interfaces

Eine LLM-basierte Methode zum Generieren von Benutzeranweisungen zur Navigation in webbasierten Benutzeroberflächen

Author:	Doruk Gerçel
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	M.Sc. Nektarios Machner
Submission Date:	01.03.2025

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 01.03.2025 Doruk Gerçel

Location, Submission Date Author

AI Assistant Usage Disclosure

Introduction

Performing work or conducting research at the Chair of Software Engineering for Business Information Systems (sebis) at TUM often entails dynamic and multi-faceted tasks. At sebis, we promote the responsible use of *AI Assistants* in the effective and efficient completion of such work. However, in the spirit of ethical and transparent research, we require all student researchers working with sebis to disclose their usage of such assistants.

For examples of correct and incorrect AI Assistant usage, please refer to the original, unabridged version of this form, located at this link.

Use of *AI Assistants* for Research Purposes

I have used AI Assistant(s) for the purposes of my research as part of this thesis.

Yes ◉ No ○

Explanation:

- Translating Thesis Title from English to German.
- Translating Abstract from English to German.
- Correcting grammar mistakes, refining the language.
- Fixing LaTeX code errors.
- Fixing Biber errors and refining bibliography entry formats.
- Fixing Python Matplotlib code errors.

I confirm in signing below, that I have reported all usage of AI Assistants for my research, and that the report is truthful and complete.

Munich, 01.03.2025	Doruk Gerçel
Location, Date	Author

Acknowledgments

I would like to express my gratitude to all those who played a role in bringing this thesis to completion. To begin with, I want to thank Nektarios Machner for his guidance throughout the thesis. I would also like to extend my appreciation to Prof. Dr. Florian Matthes for giving me the opportunity to conduct my thesis under the Chair of Software Engineering for Business Information Systems.

To continue with, I want to thank Norman Schleicher and Ai Ping Chew for their supervision and guidance throughout the project. I would also like to thank Laura Haller for providing insightful feedback throughout the project. Furthermore, I wish to acknowledge and thank Jürgen Basedahl, Hatem OdetAlla, as well as all my other team members for all their support.

In closing, I want to express my sincerest appreciation to my family, who provided me with constant love, encouragement, and support every step of the way, and my dearest, Özge Naz Kocabıyık, who always motivated me to strive for even greater achievements.

Abstract

The popularity of Large Language Models (LLMs) and the development of business applications based on LLMs are increasing rapidly. Different types of businesses and companies integrate LLMs into their businesses to develop innovative solutions. While much research is being conducted, there is a lack of LLM-based methods to generate instructions for users and guide them on navigating web-based user interfaces. This thesis addresses this research gap by proposing a grammar to textually represent an industrial web user interface (UI) application and generating an instruction and prompt format to receive UI instructions from the LLM. Using these methodologies and foundations, a chatbot application is designed and implemented to provide user guidance. This study advances research on developing a methodology that leverages the intelligence of LLMs to provide user guidance in web UI applications and develop a state-of-the-art chatbot application.

Keywords:: Large Language Models (LLMs), LLM-based Chatbots, Web User Interface (UI) Applications, Textual Representation of Web UI Applications, Conversational Interfaces, Prompt Engineering, Human-Computer Interaction, User Experience

Kurzfassung

Die Popularität von LLMs und die Entwicklung von Geschäftsanwendungen auf Basis von LLMs nehmen rapide zu. Verschiedene Arten von Unternehmen und Firmen integrieren LLMs in ihre Geschäfte, um innovative Lösungen zu entwickeln. Während viele Forschungen durchgeführt werden, besteht ein Mangel an LLM-basierten Methoden, um Anweisungen für Benutzer zu generieren und sie bei der Navigation von webbasierten Benutzeroberflächen zu unterstützen. Diese Arbeit schließt diese Forschungslücke, indem sie eine Grammatik zur textuellen Darstellung einer industriellen Web-Benutzeroberfläche vorschlägt und ein Anweisungs- und Prompt-Format zur Empfangnahme von Benutzeroberflächen-Anweisungen vom LLM generiert. Unter Verwendung dieser Methoden und Grundlagen wird eine Chatbot-Anwendung entwickelt und implementiert, um Benutzerunterstützung zu bieten. Diese Studie fördert die Forschung auf dem Gebiet der Entwicklung einer Methodik, die die Intelligenz von LLMs für die Bereitstellung von Benutzerunterstützung in Web-Benutzeroberflächen nutzt und entwickelt eine state-of-the-art-Chatbot-Anwendung.

Schlüsselwörter: Große Sprachmodelle (LLMs), LLM-basierte Chatbots, Web-Benutzeroberflächen (UI)-Anwendungen, Textuelle Darstellung von Web-Benutzeroberflächen, Konversationsinterfaces, Prompt-Engineering, Mensch-Computer-Interaktion, Benutzererfahrung

Contents

Acknowledgments **iv**

Abstract **v**

Kurzfassung **vi**

1. Introduction **1**
1.1. Motivation . . . 1
1.2. Background . . . 1
1.3. Problem Statement . . . 2
1.4. Research Questions . . . 2
1.5. Outline . . . 3

2. Background Knowledge **5**
2.1. Large Language Models . . . 5
2.2. Methods for Representing Web UI Applications and Grammar . . . 6
2.3. Programmatic Interaction with Web Interfaces . . . 6

3. Related Work **8**
3.1. Textual Representations of Web UI Applications . . . 8
3.1.1. Model Driven Approach . . . 8
3.1.2. Event Driven Approach . . . 8
3.1.3. YAML vs JSON: Efficiency for LLMs . . . 9
3.2. Dynamically Manipulating Browser-based Web Applications . . . 9
3.3. Developing Business Applications based on LLM . . . 9

4. Methodology **11**
4.1. Methodological Framework . . . 11
4.2. Requirements of the Chatbot Application . . . 12
4.3. Prototype Proposal . . . 13
4.4. Grammar Definition for UI Components . . . 17
4.4.1. Compare and Contrast Model-driven and Event-driven Approaches . . . 17
4.4.2. Proposed Grammar Objectives and Attributes . . . 17
4.4.3. Grammar Fields . . . 18
4.5. Generating UI Instructions from the LLM . . . 22
4.5.1. Textual Format for UI Instructions . . . 22
4.5.2. Prompt LLM with the Proposed Textual Format . . . 23

4.5.3. Extracting UI Instructions from LLM's Textual Output . . . 23
4.6. Interaction with UI Components in a Web-based Application . . . 24
4.7. Architecture Overview . . . 25
4.8. Algorithm for User Guidance . . . 27
4.9. User Interaction and Application Flow . . . 28
4.10. Implementation of the Prototype . . . 30
4.10.1. Design and Implementation of Algorithms . . . 31
4.10.2. Special Details and Considerations for Implementation . . . 35
4.11. Real-Life Prototype and User's Journey . . . 39
4.12. Evaluation Methods . . . 48
4.12.1. Measuring the Understandability of the Grammar . . . 48
4.12.2. Measuring the Generation of UI Instructions from the LLM . . . 49
4.12.3. Conducting Interviews about the Chatbot Application . . . 50

5. Results **53**
5.1. Understandability of the Grammar . . . 53
5.2. Generation of UI Instructions from the LLM . . . 56
5.3. Results of the Interviews . . . 57
5.3.1. Design . . . 58
5.3.2. Navigation and Flow . . . 59
5.3.3. Feedback and Improvement . . . 60
5.3.4. Software Packages and Integration . . . 61

6. Discussion **63**
6.1. Understandability of the Grammar . . . 63
6.2. Generation of UI Instructions from the LLM . . . 64
6.3. Usability of the Chatbot Application . . . 65
6.4. Software Packages and Integration . . . 65
6.5. Limitations and Future Work . . . 66

7. Conclusion **68**
7.1. Summary . . . 68
7.2. Findings . . . 68

A. Appendix **70**
A.1. Measurement for Generation of UI Instructions from the LLM . . . 70
A.2. Question Examples for Understandability of Grammar . . . 71

List of Figures **72**

List of Tables **74**

Acronyms **75**

Bibliography **76**

1. Introduction

1.1. Motivation

LLMs have gained enormous popularity in recent years and are used in many fields today [1] [2]. This popularity has significantly increased with deploying these models on companies' cloud instances or on-premise servers [3]. Companies can fine-tune and feed these base models according to their business needs and domain-specific data. Also, this increases the innovation capacity of the companies as they do not need to train their own LLM, which consumes significant time and resources [4]. This approach allows the use of LLMs inside the company for internal usage and enables them to develop new applications backed by LLMs for customer usage. Therefore, companies do not need to design complex application backends and can shift the intelligence to the LLM [5] [6] [7].

There are different ways to develop and integrate LLM-based applications, and one of these applications is chatbots [8]. Most organizations and companies integrate LLM-based chatbots into their industrial applications [9] [10]. These organizations and companies feed LLM with domain-specific data and connect the integrated chatbot application to the LLM [11]. In this way, users can type their prompts and questions to the chatbot and receive answers that the LLM backs with the fed domain-specific knowledge.

These systems have an extensive natural language understanding and perform context-aware interactions. The benefits of these applications have a wide range from customer services to educational tools [12]. Most of the use cases encapsulate asking questions directly to the chatbot to provide automated customer service or searching certain functionalities in the application [13]. Users do not need to read the documentation and forums and conveniently perform their desired operations. This significantly increases the customer's journey and the user experience [14] [15]. Positive customer feedback enabled the development of new types of chatbots with different use cases [16] [17] [18].

1.2. Background

CMX500, offered by Rohde & Schwarz, is a high-performance vector signal generator that performs real-time signal processing [19]. Rohde & Schwarz has a web-based UI application, CMSquares, accessible from a standard browser. CMSquares is necessary to configure and monitor the status and performance of CMX500 [20]. CMSquares contains many components, options, and configurations, and customers can save, load, and manage configurations with this application. Also, customers can monitor the status and performance of CMX500 in real time using CMSquares. Although CMSquares provides context-sensitive help and

documentation for users, it is usually hard for users to perform the necessary configurations and monitor the desired functionalities because of the large number of components and menus.

1.3. Problem Statement

Rohde & Schwarz want to develop a state-of-the-art solution to make the web UI setup easier with virtual assistance and guide the user in the UI according to the user's prompts. The main objective of this study is to find an LLM-based method to generate instructions for users to navigate web-based UI. The study aims to develop a solution that can be easily integrated into the web application to guide users in the application conveniently by considering the user experience and enhancing the user journey. Also, utilizing the merits of the LLM is a critical point of the study, and its intelligence shall be leveraged to develop an innovative solution. To achieve this goal, efficient approaches to interact with the LLM, describe the web application, and generate the required guidance steps are formulated and tested to evaluate whether they meet the requirements. Afterward, these approaches are combined in harmony, and an LLM-based chatbot is designed, partially implemented for specific use cases, and integrated into the CMSquares application.

1.4. Research Questions

This study investigates the following research questions:

RQ1) How to model UI components to feed LLM to generate the required steps to perform UI actions?

The main objective of this research question is to describe the web UI components in the web application with a textual data model that is interpretable by the LLM. The prior related studies and works describing UI components will be searched, and different approaches will be compared and contrasted. Additionally, resources on data representation formats will be researched to assess their efficiency for the LLMs. After finding the efficient data representation format and collecting the attributes derived from the researched approaches, the main objectives and attributes of the grammar will be defined, and the grammar will be constructed with the required fields. This proposed grammar will uniquely and semantically identify the UI components and describe the UI components' attributes and properties.

Additionally, several parts of the web application will be defined using the proposed grammar. This definition will be prompted to the LLM, and a set of questions will be prompted to test and measure the understandability of the grammar by the LLM.

RQ2) How to generate UI instructions from the textual output generated from the LLM?

The existing technological automated tools that interact with the web UI using the scripts, and their script languages alongside the UI instruction formats will be investigated. After the research, a textual UI instruction format containing end-user instructions and identifiers for the web components in the web page will be constructed. Then, a prompt containing and defining the specifications of this format will be prepared. Moreover, a method will be developed to extract both the UI instructions and their subpatterns from the textual output generated by the LLM. Furthermore, in a web UI application, methods to interact with the web components using their identifiers will be searched. Subsequently, a user guidance algorithm will be developed to interpret the visibility of the UI components in the viewed web page and guide the user step-by-step until the last step according to the received list of steps of instructions with the proposed UI instructions' format.

In addition, prompts containing the web application's textual definition, tested current page, the specifications of the UI instruction format, and tested use-case will be prepared. They will be prompted to the LLM to evaluate the generation of UI instructions from the LLM's output and discuss whether the user guidance algorithm can function properly with the output.

RQ3) How to integrate this chatbot into the product by following and considering software engineering principles?

The primary focus is to develop a chatbot application that provides user guidance to the end-users according to their prompt by highlighting the web components and displaying instructions. This application will be designed and implemented using software engineering principles, such as source code structure, code integration, usability, and user experience.

In the beginning, the requirements of the chatbot application will be specified, and a mock prototype will be designed based on the design elements of the web application. Afterward, all necessary components will be designed by defining their functionalities and interfaces. Then, the routines and algorithms for these components will be developed and implemented, and the chatbot application will be integrated into the main web application.

Furthermore, interviews with several target groups will be conducted, and participants will be asked questions regarding the chatbot application. These results will be used to qualitatively evaluate the chatbot application's usability, user experience, source code structure, and code integration.

1.5. Outline

The following describes how the rest of the thesis is structured. In Chapter 2, an overview of the nature and attributes of the LLMs, the methods of feeding them, different methods of hosting them, and errors based on their nature are given. Furthermore, the methods of representing web UI applications and programmatic interaction with web interfaces are stated. In Chapter 3, related researches that have been conducted about textual representations of web UI applications, works regarding dynamically manipulating web UI applications,

and developing business applications based on LLM are discussed. In Chapter 4, the methodological works that are conducted regarding proposing a grammar to textually define web UI applications to the LLM, generating UI instructions from the LLM, and developing a chatbot application are briefly explained. Moreover, the methods that are designed and followed to evaluate them are described. In Chapter 5, results of these measurements and evaluation are presented. In Chapter 6, the interpretations and discussions regarding the obtained results are highlighted, and limitations of the study and the potential future work are addressed. Conclusively, in Chapter 7, the significant contributions of the research are reviewed.

2. Background Knowledge

2.1. Large Language Models

An LLM is an artificial intelligence model that is mainly designed to process and generate human-like text [21]. LLMs are mainly developed using a deep learning architecture called transformers. Transformers enable LLMs to understand and generate text with contextual awareness [22]. LLMs are trained with large amounts of textual datasets from various sources, and LLMs gain reasoning abilities by learning grammar and semantics [23] [24]. First, LLMs are trained on these large datasets unsupervised with a process called training. Afterward, they are refined for specific tasks through a process called fine-tuning. Therefore, they can be adapted to particular domains and use cases [25]. LLMs are probabilistic as they are developed with deep learning techniques [26] [27].

There are different methods to prepare an LLM for specific tasks. The main methods are training, fine-tuning, and feeding. These methods are also referred to as methods for feeding LLMs.

Training an LLM refers to adjusting the LLM's parameters from scratch via learning algorithms and a vast amount of textual data [28]. LLM learns the general language patterns via training. The training process requires significant computational power and large datasets [29].

Fine-tuning an LLM involves taking a pre-trained model and further training it with a domain or task-specific dataset [30]. Therefore, the model adapts to domains and tasks and produces a model optimized for specific tasks [31]. Compared to the training, fine-tuning requires less data and computational resources [32].

Prompting an LLM is providing input to the LLM to generate responses based on the provided input [33]. Prompting an LLM is much more lightweight than training and fine-tuning as it does not involve changing the model's parameters. Still, it leverages the existing capabilities of the model for specific tasks [34]. Prompting an LLM consists of several stages: crafting input, submitting the prompt, tokenization, model processing, output generation, decoding, and response delivery [35]. There are different types of prompting, such as instructional, contextual, role-playing, etc. Instructional prompts give explicit instructions to the LLM on how to respond, and contextual prompts provide background information to guide the model's response [36]. Prompting an LLM has many application areas, from content generation to chatbots [37].

The LLMs can be publicly or privately hosted, exposing application programming interface (API) endpoints for software integration and application development based on the LLMs. For example, OpenAI hosts its models on its own servers for public access. Also, these models can be hosted by Microsoft Azure Cloud so that enterprises can have private access.

The OpenAI API is used to access public models, while the Azure OpenAI API provides access to private ones. OpenAI API pricing is based on API usage and token consumption, whereas Azure OpenAI API incurs additional costs for Azure resources. Despite the pricing differences, Azure OpenAI API offers lower latency due to Microsoft's data center locations and enhanced customizable security features. As a result, models deployed on Microsoft Azure Cloud are more suitable for enterprise use [38] [39] [40].

The probabilistic nature of the LLMs makes them prone to particular types of errors. There are a variety of LLM-based errors, but the errors mentioned in the following chapters are polysemy confusion and semantic drift.

- **Polysemy Confusion:** LLM may struggle to correctly interpret the context and meaning of words with multiple meanings in different contexts. This leads to an error if the LLM selects the incorrect meaning [41].
- **Semantic Drift:** In a regular conversation, when the context shifts, the meaning of certain words drifts away from their original intent. This may lead to misinterpretations, and LLM may misinterpret the correct meaning of certain words [42].

2.2. Methods for Representing Web UI Applications and Grammar

Different web UI development technologies exist, and within a company, different teams, such as software developers, user experience researchers, and product managers, work on web UI development together. Therefore, bridging the gap between web UI components and the technical details is essential. To achieve that, methods for representing web UI applications have been proposed. Two popular methodologies are model-driven and event-driven approaches. The model-driven approach represents the static structure of a web UI by using domain-specific languages (DSL) [43] [44]. This approach represents the web components, their attributes, and layouts. Unlike the prior approach, the event-driven approach reflects the dynamic behavior of the UI by defining the user interactions and the corresponding events [45]. Using these approaches together is also possible, providing a blueprint and reflecting the interactive flow of the web UI.

Grammar is a pre-defined set of rules to describe the structure of components and the relationships of these components with each other. Several concepts, such as attributes and relationships, are essential to model the components. The grammar can be constructed with a formal approach or human-readable format [46]. Constructing and using grammar is helpful in different fields of computer science, like software engineering, system design, and documentation.

2.3. Programmatic Interaction with Web Interfaces

The **Document Object Model (DOM)** is a standardized interface that interacts with and manipulates web documents. Web pages are represented as a tree where nodes correspond to

the parts of the document, such as elements, attributes, or text. When the page loads, the browser parses the HyperText Markup Language (HTML) and constructs the corresponding DOM tree. Components can be created, modified, and removed via dynamic manipulation of the DOM [47]. Web development libraries and frameworks like jQuery, Angular, and React enable this feature.

The **Robotic Process Automation (RPA)** is an innovative technology that automates rule-based and repetitive tasks. Most of the time, it uses software bots to mimic human-computer interactions. Scripts are developed to define the actions and rules bots must follow [48]. Robot Framework is an open-source RPA framework that supports writing automation scripts in a simple and human-readable format, and enables the RPA and web UI interaction testing. It comes with its own script language, Robot Framework Language, that is not a programming language, but a plain language that can define automation scripts. The necessary UI interaction instructions are defined in a simple format that contains the action, the element, identifier, and if relevant input data in order, such as `Click #elementID` and `Input Text #elementID ##textData`. For web applications, these identifiers can be Cascading Style Sheets (CSS) selectors like in the example: `Click div.main-navigation .auto` [49] [50].

3. Related Work

This chapter reviews the related work and relevant literature across two key areas: textual representations of web UI applications and developing business applications based on LLM. Usages of both of these topics and their extended usages are explained in detail in the later chapters.

3.1. Textual Representations of Web UI Applications

3.1.1. Model Driven Approach

- Brambilla et al. [51] propose the Interaction Flow Modeling Language (IFML). This introduced language is a model-driven language that describes the web application UI. This work proposes a formal way to represent the static and dynamic aspects of UI. IFML is a modeling language that can represent the structure of UI components, the interaction flows, and navigation patterns. IFML provides a standardized diagrammatic notation. Although it can be serialized to a textual format, it does not offer a purely textual representation. However, this thesis tries to represent the web application to the LLM with prompts. Therefore, this thesis attempts to develop a textual representation that is promptable and differs from this work.

- Karu et al. [52] introduce a textual domain-specific language (DSL) for UI modeling. This introduced approach is model-driven. This method defines a high-level, formal syntax that defines the UI's structural elements and dynamic behaviors. The structural elements consist of layout, visual components, and data bindings. Meanwhile, dynamic behaviors encompass user interactions and event handling. This proposed syntax ensures modularity and reduces the inconsistencies across different UI components. This contribution is beneficial in bridging the gap between abstract design and concrete implementation. This study proposes a textual representation to model the web UI components. Still, it is essential to use a modeling language that is easily recognizable by LLM, leverages the intelligence of the LLM, and reflects a sequence of actions. This thesis tries to develop a textual representation that tackles these objectives.

3.1.2. Event Driven Approach

- Silva et al. [53] designed a domain-specific language to specify interaction scenarios for a web-based graphical UI. Their approach focuses on capturing the sequence of user actions and system responses formed with these interactions. Therefore, this approach is

event-driven. Also, it incorporates model-driven principles but is considered primarily event-driven. This work is essential in reflecting the sequence of user interactions and corresponding actions and encompassing several UI component structures. Although this study combines an event-driven approach with a model-driven approach to a certain extent, for this thesis, it is necessary to develop a modeling language that clearly defines web component attributes and is easily recognizable by the LLM.

3.1.3. YAML vs JSON: Efficiency for LLMs

- Livshitz [54] researches to compare whether YAML and JSON are more efficient for LLMs. According to the research, when YAML is used to describe the components within a prompt, the response time is approximately 5,200 ms, whereas JSON results in a response time of 7,266 ms for the same semantic descriptions. According to the research, the reason for this difference is that YAML reduces the number of tokens and characters required in the prompt. In contrast, JSON's use of curly brackets increases token density. Additionally, YAML is more suitable for representing natural language structures and preferable for LLM prompting. While both YAML and JSON yield semantically similar responses, YAML is more advantageous, ensuring correctness while significantly reducing response time. Therefore, YAML is the more efficient format for structuring prompts in LLM interactions.

3.2. Dynamically Manipulating Browser-based Web Applications

- Franklin et al. [55] work on dynamic content manipulation in browser-based web applications. This work uses jQuery and CSS selectors for this objective. This work examines the mechanisms and methods of jQuery for DOM manipulation, and CSS selectors are used to target and dynamically style web elements. These methods enable efficient content updates and animations to significantly improve user experience. In this thesis, web elements shall be highlighted to provide user guidance. Therefore, dynamic manipulation of web elements is required, and the proposed methods examined in this study can be used and extended.

3.3. Developing Business Applications based on LLM

- Kale et al. [56] build intelligent systems with OpenAI API and try to examine practical methodologies to leverage the technology. Moreover, this work highlights the best practices in prompt engineering and API integration and addresses the corresponding challenges. This study is a valuable framework for developers and researchers who work on generative artificial intelligence and develop applications based on it. This thesis develops a chatbot application based on LLM, and this study provides guidelines and examples that benefit this thesis.

- Hansmann et al. [57] work on integrating LLM-based chatbots into the design thinking process and collaborative decision-making. In this study, the OpenAI API enables the interaction between the application and the LLM. The developed chatbot application can produce contextually relevant suggestions and interactive guidance to improve the overall design process using OpenAI API. In this thesis, the objective is to create an LLM-based chatbot as well, but both the domain and the objectives of the chatbot application are different.

4. Methodology

This chapter first introduces the methodological framework and explains each step of methodology that has been performed. After defining the framework, each research and work is explained in detail. Then, the chapter focuses on the methodologies used to obtain quantitative and qualitative results and the evaluation methods used.

4.1. Methodological Framework

The main product this study is trying to develop is an LLM-based chatbot application that provides virtual assistance and guidance to users. This section briefly explains all the steps performed throughout the study chronologically to give an overview of each step and explain how these steps led to the development of the main objective.

Just like in the development of every software application, the first requirement is to define the objectives and requirements of the application. Therefore, the project's requirements are discussed with the project supervisors from Rohde & Schwarz. After defining both the requirement and objective, a prototype proposal is prepared to mock the view and the use case of the chatbot application. This proposal will be presented and explained in the following section.

Following the completion of the prototype proposal, work on the theoretical foundations of the chatbot application has started. First, a methodology to define the web UI application in a textual representation has been investigated, and a grammar that describes all the web components and is understandable by the LLM is proposed. After defining certain parts of the web application with the proposed grammar, the understandability of the grammar is evaluated by conducting a relevant experiment.

Subsequently, a format that defines UI instructions is prepared. Special consideration is taken for this format to encapsulate the necessary user instruction and the identifier of the web component to be highlighted in the web application's UI. Afterward, a method to systematically extract these instructions and capture the relevant groups of this format is implemented. A prompt format for the LLM is prepared that encapsulates the web application definition with the proposed grammar, the special format for UI instructions, the current page information, and the user's input prompt. Then, the generation of UI instructions from the textual LLM output is measured by conducting a relevant test.

Thereafter, as the chatbot application must understand whether a web component is visible on the current web page, a method to check the visibility is searched. Also, a technique to dynamically change the border color of a web application is investigated.

Following the research and developed methods, the architectural aspect of the chatbot

application is designed. All the necessary components required for the chatbot application, their functionalities, their interactions with each other, and how they function with each other are defined. Moreover, an algorithm for user guidance, according to the list of steps of instructions received and the user's interactions with the web application, is designed.

With the completion of all research and design phases, the chatbot application is developed by considering the software design principles and patterns. Then, it is integrated into the main web application. The usability of the chatbot application is assessed by conducting interviews with defined target groups from Rohde & Schwarz.

4.2. Requirements of the Chatbot Application

The defined requirements and objectives of the chatbot application are:

- Implementing a chatbot application with a UI like a messaging application.
- The chatbot must be interacting with the LLM.
- Implementing a dual mode of 'Chat Mode' and 'User Guidance Mode.' In the 'Chat Mode,' the chatbot accepts textual user prompts, and in the 'User Guidance Mode,' the chatbot guides the user according to the prompts.
- In the 'Chat Mode,' after the chatbot receives the necessary steps from the LLM, all the required steps shall be displayed to the user as a list of instructions.
- After viewing all the necessary steps, the user may switch to the 'User Guidance Mode' by pressing a button in the chatbot application.
- In the 'User Guidance Mode,' the current step of instruction shall be displayed to the user like a message in the chatbot. Also, the current web component of the current step must be highlighted in the web application to draw the user's attention.
- The chatbot application must guide the user until the last instruction or until the user turns off the guidance mode.
- The chatbot application must continue to guide the user even if the user does not follow the instructions as expected or makes an error.
- The design elements of the chatbot application shall be compatible with the other design elements of the web application.
- The OpenAI's GPT-4o mini shall be used as the LLM. This model is deployed on the Microsoft Azure instances that are owned by Rohde & Schwarz, and its API is configured for LLM-based application development. Also the default temperature value, 1.0, shall be used to balance the randomness of the application.

4.3. Prototype Proposal

The UI and user experience are among the main concerns of this thesis. Therefore, before the implementation of the real prototype, a proposal for the design has been prepared to meet the requirements of the user experience. This section explains the prototype proposal by giving details about the requirements and explaining them visually through the images from the proposal.

As a general requirement, it is essential for the chatbot application to provide user guidance conveniently. The user shall understand the instructions clearly through the written user instructions. Also, the choice of color is essential for highlighting the elements related to the current step. The color of choice shall draw the attention of the user, and it shall be distinguishable from the different component colors of the web application.

Moreover, this chatbot application has two phases for the customer: 'Chat Mode' and 'User Guidance Mode.' The first phase is the interaction of the user with the chatbot, where the user enters his/her prompts to the chatbot, and the chatbot returns the list of steps to be achieved from the current web page. The second phase consists of highlighting the element in the web page and displaying the current user instruction textually to the user. This separation of phases is especially considered in the design because the list of steps of instructions may be clear enough for the user to perform the prompted actions and enables the user to just view the necessary steps or be guided throughout the execution.

The example flow of the application is described below, including the actions performed and images displaying the prototype proposal. Changing the application's background color serves as an example use case.

In Figure 4.1, the user types the prompt to be performed.

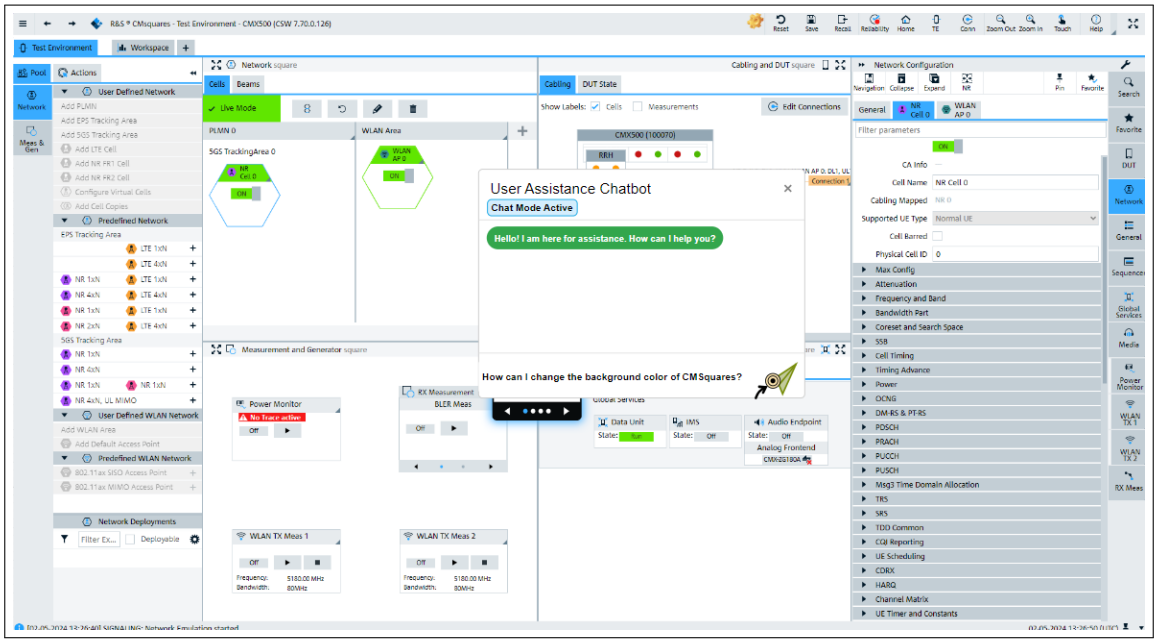

Figure 4.1.: Prototype Proposal: Chat Mode Send Prompt

In Figure 4.1, the user presses the send button to send the prompt. In Figure 4.2, the chatbot displays all required steps. The user shall activate user guidance mode if instructions are not clear enough.

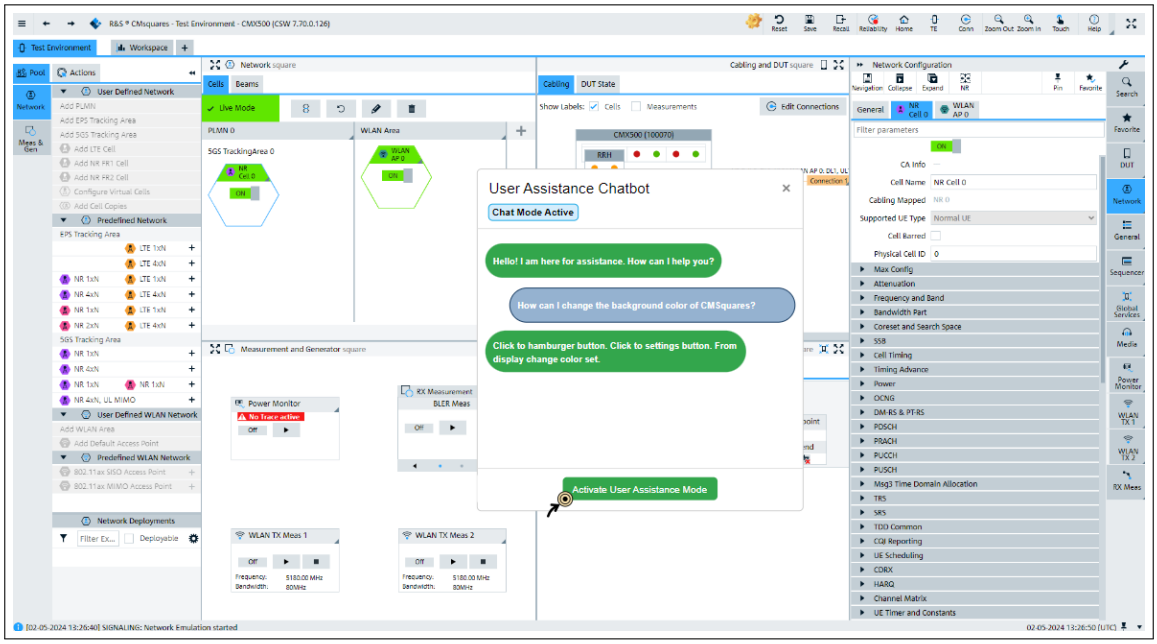

Figure 4.2.: Prototype Proposal: Display List of Instructions

In Figure 4.2, the user clicks the 'Activate User Assistance Mode' button. After the activation, in Figure 4.3, the chatbot displays the required step and highlights the element, which is the hamburger button on the top left.

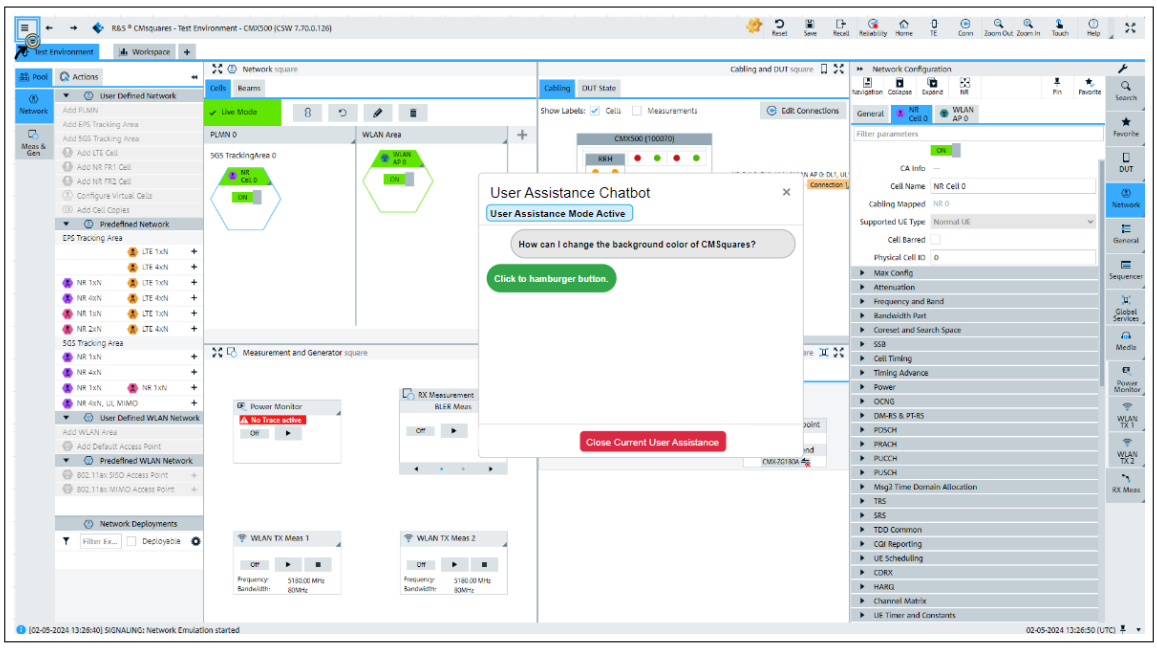

Figure 4.3.: Prototype Proposal: Highlight Hamburger Button

In Figure 4.3, the user clicks the hamburger button, and the menu is expanded. In Figure 4.4, the chatbot displays the next required step and highlights the relevant element, which is the 'Settings Button.'

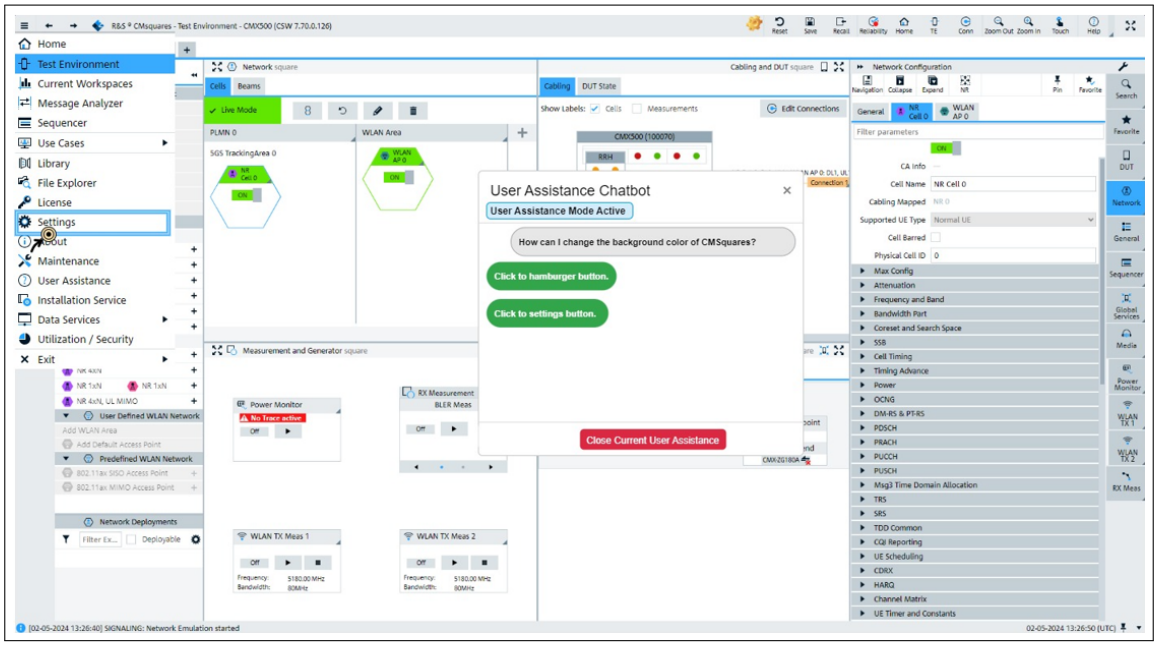

Figure 4.4.: Prototype Proposal: Highlight Settings Button

In Figure 4.4, the user clicks the 'Settings Button,' and the 'Settings' page is opened. In Figure 4.5, the chatbot displays the next required step and highlights the relevant element, which is the 'Color Set Dropdown,' and notifies the user that all steps of instructions are completed.

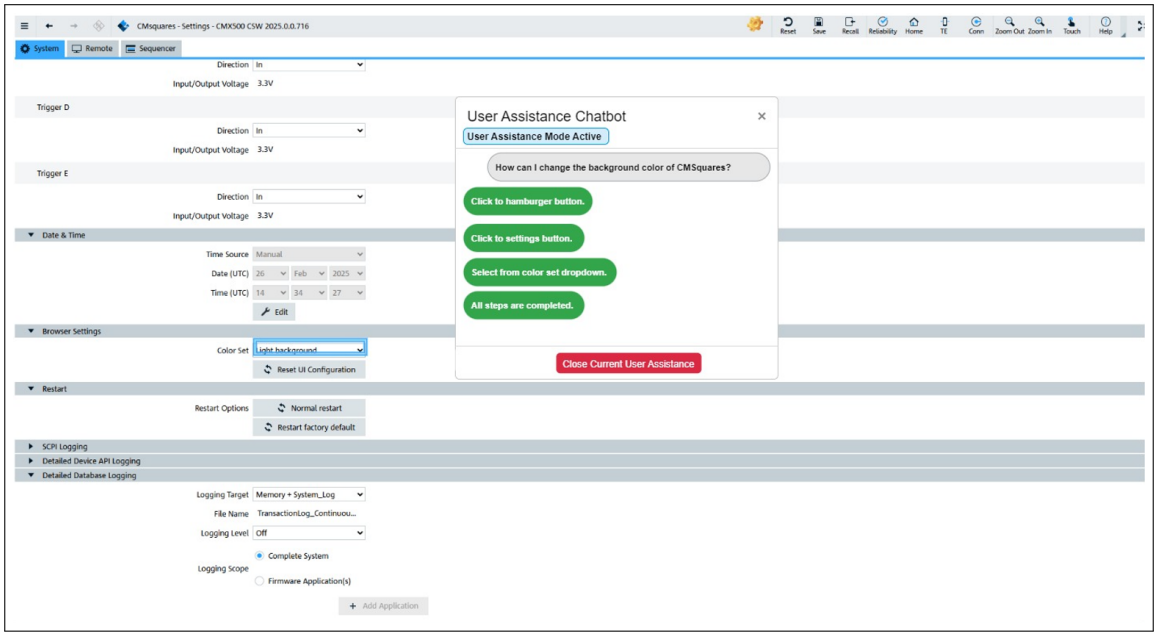

Figure 4.5.: Prototype Proposal: Display that All Steps are Completed

4.4. Grammar Definition for UI Components

This section investigates an approach to describe the web UI application in a textual representation. To achieve that, a grammar that defines the UI components is proposed, and the attributes of the grammar are described. While describing the proposed grammar, it also compares and contrasts the 'Model-driven' and 'Event-driven' based web UI representations and grammar definition examples and tries to gather their desired properties while constructing the newly proposed grammar.

4.4.1. Compare and Contrast Model-driven and Event-driven Approaches

The 'Model-driven' approach encapsulates the individual use cases and the web components that are interacted with. Based on the investigated examples, the use cases are explicitly defined, and web components' do not contain any field or attribute related to their use case. Instead, the use case serves as a container that encompasses the associated web components. Moreover, navigation and connections between the related web components are explicitly defined. In this approach, use cases are state machines, and the components associated with them are the individual states.

The 'Event-driven' approach tries to capture user interactions and corresponding events. Therefore, based on the researched examples, each web component is described with initial states, forwarded states, and relevant transitions. Navigation between the web components is not directly defined like a connection, instead is defined as state transitions, so it is more challenging to represent. In this approach, each component is a state machine on its own, and these state machines are connected to each other.

The main objective of both of these approaches is to facilitate internal communication within a company, such as engineering, product management, and user experience research, between the different teams. These approaches were proposed long before the development of LLMs. Therefore, these approaches highlight certain use cases or web components, and they do not have any objective regarding leveraging the intelligence of the LLMs or making their descriptions interpretable.

4.4.2. Proposed Grammar Objectives and Attributes

Several main objectives are defined for the proposed grammar, and specific attributes are added to meet these objectives. Some objectives and attributes are derived from the approaches that are explained in Subsection 4.4.1, and some are added to leverage the intelligence of the LLM.

The grammar shall include the attributes of the relevant web UI components. These attributes can be the type of the component, the children components the component contains, the action performed by the component when the user interacts with it, and many more. Also, navigation from and between the components shall be explicitly defined. This property is essential to determine the transitions between the components. Moreover, the grammar shall

be able to identify each component with a unique identifier to distinguish each component from the others.

This thesis aims to describe the web UI to the LLM, so it is critical to make the grammar and definitions of the web UI understandable by the LLM. Also, as LLMs can understand many relations or hidden patterns in the prompt, explicitly defining every property of the components is not necessary. It is beneficial to leverage the intelligence of the LLM to reduce redundant definitions. Therefore, the time needed to define the web application may be significantly reduced.

The web application's UI will be prompted to the LLM by using the proposed grammar, so it must be represented in a structured textual format, and it is crucial to select a format that is easily processable by the LLM. Two structured data formats that are considered are YAML and JSON, as both of these data structures can cope with the objectives and attributes of the grammar, and both of them are able to textually represent web UI layout and components. Previous research [54] highlights that when YAML is used to describe the components within a prompt, the response time is less than when JSON is used, while both of them yield semantically similar responses. YAML is more advantageous as it ensures correctness while significantly reducing response time, and is a more efficient format for structuring prompts. Therefore, YAML is chosen to represent the proposed grammar.

4.4.3. Grammar Fields

The grammar intends to represent components' attributes according to the objectives in Subsection 4.4.2. Therefore, several mandatory and optional fields are defined to represent these attributes. In Figure 4.6, the proposed grammar is given with mandatory and optional fields. The mandatory fields are given in Table 4.1.

Table 4.1.: Mandatory Fields and Definitions

Field	Description
ID	The distinct identifier that uniquely defines the component.
ComponentUserName	The name that specifies the component to the client.
ComponentSelector	The component's selector to identify the element on the web UI's HTML page.
Path	The URL path where the component resides.
ComponentType	The type of the component, such as button, text, dropdown, etc.

Several optional fields are proposed to encapsulate details regarding the web components. These fields are marked as nullable fields because they are not required to be added for every component. Several of these fields may or may not be null according to the type of the defined component. Several notable fields that are optional are given in Table 4.2.

Table 4.2.: Optional Fields and Definitions

Field	Description
Label	The visible text that is included in the component.
Description	The reference to a user assistance or help documentation if a document is provided.
UserAssistanceID	The component's selector to identify the element on the web UI's HTML page.
IsGlobalView	The boolean value that highlights whether the component is visible from every web application page.
IsExpandable	The boolean value that emphasizes whether the component may be shrunk or expanded.
CanBeDisabled	The boolean value that reflects that the component may be disabled or enabled.
Attributes	The specific details about the component according to the component type.

Several base components, such as button, div, dropdown, etc., are defined to reduce replication. They include a field that defines the user's action to interact with these components. If a subcomponent's type is one of these base components, the relevant user action is automatically inherited from the definition, so this field is reused for each newly defined subcomponent. The defined base components are given in Figure 4.7.

```
ComponentName:
  ID: #id
  ComponentUserName: #component-username
  ComponentSelector: #css-selector
  Label: #label-str | null
  Path: #path
  Description: #usage-str | null
  UserAssistanceID: #id | null
  ComponentType: #component-key | null
  IsGlobalView: #true | false | null
  IsExpandable: #true | false | null
  CanBeDisabled: #true | false | null
  Attributes:
    Opens: #component-key | null
    ChangesTo: #component-key | null
    Enables:
      - #component1
      - #component2
    Disables:
      - #component1
      - #component2
    NavigatesTo: #component-key | null
    Contains:
      - #comp1
      - #comp2
    Value: #text | number
    HasValues:
      - #value1
      - #value2
    HasDynamicValues: #true | false | null
    UserInput:
      Type: #text | number
      Placeholder: #value | null
```

Figure 4.6.: Proposed Grammar Definition

```
BaseComponents:
  Button:
    ComponentUserAction: Press
  Section:
    ComponentUserAction: View
  Div:
    ComponentUserAction: View
  Text:
    ComponentUserAction: View
  Dropdown:
    ComponentUserAction: Select From
  DragAndDrop:
    ComponentUserAction: Drag And Drop
```

Figure 4.7.: Base Components' Grammar Definition

This proposed grammar's understandability by the LLM is evaluated in the following chapter. After the evaluations, the web application is defined by using this proposed grammar

and is integrated to the chatbot application codebase. The example for definitions of several web components with the proposed grammar are demonstrated in Figure 4.8 and Figure 4.9.

```
WebApp:
  HamburgerBtn:
    ComponentUserName: Hamburger Button
    ComponentSelector: bh-burger-menu bh-btn
    ComponentType: Button
    IsGlobalView: true
    Attributes:
      Opens: MenuSection

  MenuSection:
    ComponentUserName: Menu Section
    ComponentSelector: bh-burger-menu div.list
    ComponentType: Section
    Attributes:
      Contains:
        - Home Button
        - Current Workspaces Button
        - Message Analyzer Button
        - Sequencer Button
        - Use Cases Button
        - Library Button
        - File Explorer Button
        - License Button
        - Settings Button
        - About Button
        - Maintenance Button
        - User Assistance Button
        - Installation Service Button
        - Data Services Button
        - Utilization/Security Button
```

Figure 4.8.: Definition of the Hamburger Button and the Menu Section with the Grammar

```yaml
ColorSetDiv:
  ComponentUserName: Color Set Div
  ComponentSelector: bh-display-settings div.contentWrp
  ComponentType: Div
  Path: /settings/system
  Attributes:
    Contains:
      - Color Set Text
      - Color Set Dropdown


ColorSetText:
  ComponentUserName: Color Set Text
  ComponentSelector:  bh-display-settings div.contentWrp bh-settings-line .settingsLineTitle
  ComponentType: Text
  Path: /settings/system
  Attributes:
    Value: Color Set

ColorSetDropdown:
  ComponentUserName: Color Set Dropdown
  ComponentSelector:  bh-display-settings div.contentWrp bh-settings-line .settingsLineContent bh-drop-down
  ComponentType: Dropdown
  Path: /settings/system
  Attributes:
    HasValues:
      - Light background
      - Dark background
```

Figure 4.9.: Definition of the Color Set Components with the Grammar

4.5. Generating UI Instructions from the LLM

Section 4.4 describes the proposed grammar that describes the web application and is necessary to feed the LLM. This section switches focus to generating UI instructions according to the prompt from the LLM. These instructions are necessary for both highlight the necessary web elements in the web application through the user guidance and textually displaying the instructions to the user. In this sections, first, an appropriate textual format for the UI instructions will be described. Afterward, the way this format is described to the LLM, and a method to extract these instructions from the textual output are elaborated.

4.5.1. Textual Format for UI Instructions

In user guidance, it is crucial to highlight the necessary elements in the web application and display the textual instructions to the user in the chatbot view. Therefore, a specific format shall be specified to encapsulate both objectives.

The robotic process automation is explained in Chapter 2 referencing the Robot Framework. In the Robot Framework language, for a browser-based web application, each step comprises an action for the step and a CSS selector to identify the web component. As Robot Framework

language is used extensively in robotic process automation and graphical UI tests for browser-based web applications, it is helpful to follow a similar textual format to describe the necessary UI instructions for the user and specify the corresponding elements.

In the proposed textual format, a single step is composed of three main parts respectively: the action to be performed by the user, the name of the web UI element, such as Settings Button, Color Dropdown, etc., and the CSS selector of the web UI element. This textual format shall be prompted to the LLM so that the necessary steps of instructions can be returned according to this specified format. Specifying these fields in plain text without special characters or markers confuses the LLM, as the prompt describes this format with in context learning. It is convenient to wrap this format in special characters and markers to enforce this format to LLM. Therefore, the LLM can understand the necessary attributes, and generate each step according to them in the proposed grammar described in Section 4.4.

The textual format with special characters is:

$$$`ComponentUserAction ComponentUserName`=`ComponentSelector`$$$

These special characters are chosen because the $, =, and ` characters will not be included in the values of the ComponentUserAction, the ComponentUserName, and the ComponentSelector.

An example step according to this proposed textual format is:

$$$`Press Test Environment Tab Button`=`bh-ra-scroll-cell:nth-child(1)`$$$

4.5.2. Prompt LLM with the Proposed Textual Format

The textual format is proposed in Subsection 4.5.1, but to receive each necessary step with this textual format, the LLM shall be explicitly prompted, so that it can return responses and instructions obeying this textual format.

The prompt that is used for this purpose is:

Return format shall be like
$$$'ComponentUserAction ComponentUserName'='ComponentSelector'$$$.
Please only return the steps like this format and do not forget to add triple $ sign.

The triple $ sign is expressed twice in the prompt because this character combination is essential to mark each step's beginning and end. As LLMs have an indeterministic nature, it is beneficial to highlight the necessity of these signs in the prompt. In several prompts that do not mention this requirement, the $ sign is not added to the response, so it is reemphasized in the last sentence of the prompt.

4.5.3. Extracting UI Instructions from LLM's Textual Output

The proposed textual format to describe each step is prompted to the LLM with the prompt explained in Subsection 4.5.2. The major problem encountered is that LLMs have a probabilis-

tic nature. Although the textual format is explicitly defined above, the LLM may change the ` character to ', ", or an empty string character and return the steps according to these changed characters. Therefore, in the chatbot application, directly expecting these characters for each step may lead to an error, although the returned steps are correct, and the necessary fields are provided.

Therefore, a regular expression is constructed to capture these slight changes in the textual format and adapt to them. So, there will not be any interruptions in the flow of the chatbot application, which will be described briefly in the following chapters, caused by these slight changes. The constructed regular expression is:

```
/\$\$\$([`'"]*)([^'=]+)(\1)\s*=\s*([`'"])([^'`]+)(\4)\$\$\$/g
```

Figure 4.10.: Regular Expression for UI Instructions

In this regular expression, starting and ending a single step with a triple **$** sign is necessary. The 'ComponentUserAction' and the 'ComponentUserName' must be separated by a single space character, and the opening character of their composition must be the same as their closing character, these marker characters are expressed in the regular expression. The 'ComponentSelector' must have the same opening and closing characters. An equation mark separates these two parts with varying numbers of space characters. As the LLM returns the list of steps of instructions, the global option for this regular expression is set, so that the regular expression matches each step and not only the first occurrence. Group number 2, the second parenthesis, captures the necessary user instructions to be displayed, and Group number 5, the fifth parenthesis, captures the relevant CSS selector. Therefore, both the user instructions and the identifiers of the web components are extracted.

4.6. Interaction with UI Components in a Web-based Application

This section describes the necessary procedure for the chatbot application to follow to determine whether a component is visible on the current web page and whether it is component disabled or not. In addition, highlighting the necessary components is essential for the scope of the application. Therefore, this section also tackles how the border colors of the component are dynamically modified and restored afterward by the chatbot application.

Just as discussed in Chapter 2, in a browser-based web application, the current page's elements are placed in the current page's HTML. DOM allows interaction with the web document and finds whether the element is inside the current HTML page. DOM can be controlled or modified using the jQuery library. Chatbot application provides a CSS selector string to the relevant jQuery function to find the corresponding web element. According to the function's output, the chatbot application determines whether the web element is currently visible on the web page. Also, if the element is found, the chatbot application can decide if the element is disabled or not. For example, the chatbot application can decide whether a button element is clickable.

As mentioned in Chapter 2, DOM can be manipulated with the jQuery library and the relevant functions. To highlight the components for user guidance, the border color and thickness shall be modified. The chatbot application stores the border style of the component to be highlighted to restore its style afterward. If the component does not have a border property, border is created to highlight the component, and is cleared afterward. Then, the chatbot application changes the component's border with the specified color and thickness using the relevant jQuery function so the component becomes highlighted. To remove the highlight, the border style of the component is restored.

4.7. Architecture Overview

This section describes the design of the chatbot application, defines all the subcomponents, and explains the interfaces between the components and the data they exchange. Figure 4.11 is the component diagram for the overall architecture and visualizes the components with their interfaces.

The chatbot application has two main components: the '**Chatbot Popup View**' and the '**User Assistance Service**'.

The '***Chatbot Popup View***' is a popup view that has an UI similar to a messaging application. The user can type their prompt into the popup, and all the required user actions are listed, and in the guidance mode each relevant step is displayed. The prototype proposal is included in Section 4.3 and the captures of the real implementation will be included in the following sections.

The '***User Assistance Service***' is the component that performs the necessary operation to communicate with the LLM and achieve the user guidance. This service contains three sub-components: the '**Prompt Agent**', the '**Instruction Adapter**', and the '**Instruction Agent**'.

The '***Prompt Agent***' is the component that interacts with the LLM. It receives the textual prompt that the user typed from the 'Chatbot Popup View', wraps the user's prompt with the description of the web application by using the grammar defined in the previous section, the current page information, and the textual format to receive the responses from the LLM. Then, it sends the structured prompt to the LLM, and they are passed to the 'Instruction Adapter'. An example for this structured prompt will be given in the following sections

The '***Instruction Adapter***' is the component that converts the received response from the LLM to a set of user instructions and query selectors for HTML elements. Also, it verifies whether the returned result is in a valid format. These converted instructions are passed to the 'Instruction Agent'.

The '***Instruction Agent***' can be considered as the most innovative component that performs user guidance. After receiving the instructions, it first tries to find the necessary step of instruction to guide the user, by checking the visible elements in the current page view. According to the user's interaction with the web application, it iterates the instructions and finds the corresponding next step. Even when the user performs a faulty interaction with the web application, it can find the correct step. In case the component in the current step is disabled or not configured, it communicates to the '**Prompt Agent**' how to enable or configure

this component, and the '**Prompt Agent**' communicates with the LLM. Further details of the user guidance algorithm will be given in depth in the following section. According to the current step, the '**Instruction Agent**' interacts with the '**Chatbot Popup View**' to display the user instruction and interacts with the web application to highlight the necessary element on the web page. Both finding the element and changing the border color are achieved via the jQuery functions with the CSS selectors, as described in the previous section.

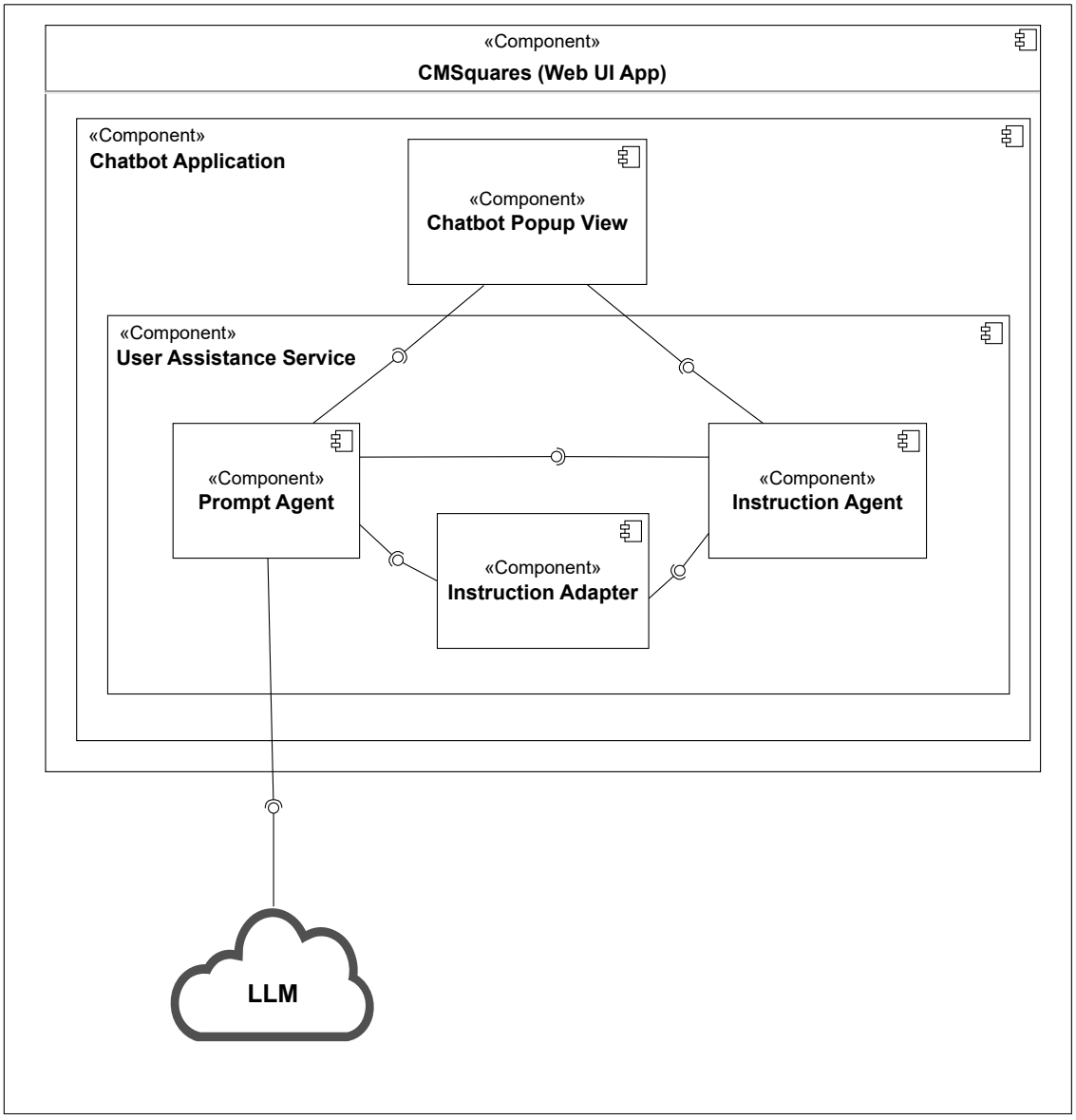

Figure 4.11.: Component Diagram for the Chatbot Application

4.8. Algorithm for User Guidance

This section introduces the actions performed and taken by the 'Instruction Agent' and different states that the agent transitions according to different conditions. Also, this section tries to briefly explain the details of how the user is guided according to the provided steps of instructions. Figure 4.12 is the state machine diagram for the algorithm of the user guidance and visualizes the steps described below.

In the beginning, the 'Instruction Agent' stays in idle mode and waits for steps of instructions. After receiving the list of instructions, the 'Instruction Agent' sets up the event listeners for the web document to capture the document events from the user, such as clicking on any part of the page or pressing the keyboard keys. After the setup, the agent iterates from the end of the list and tries to find the achievable instruction step. The purpose of iterating backward is to find the closest achievable step toward the target step; therefore, the user won't need to perform redundant instructions. The agent can understand whether the step is achievable by using the jQuery function and the CSS selectors to check whether the related element is visible on the current web page. If the current step is not found, although the list has been traversed, the 'Instruction Agent' communicates with the 'Prompt Agent' for a recalculation. Also, if the current step is found but the element is disabled, the 'Instruction Agent' communicates with the 'Prompt Agent' to ask how to enable the element in the current step. As another edge case, if the last instruction, finish instruction, is the current step, the user guidance is finished, and an exit message is displayed to the user.

If these edge cases are not the case, the desired execution of the user guidance is started. The element related to the current step is highlighted, and the user instruction is displayed in the popup view. When the user interacts with the UI, the event is captured, and the agent tries to find the new current step. It first checks whether the element in the next step is visible. If the element is not visible, it checks whether the element in the current step is still visible, and if it is visible, it continues to highlight the same element. If it is also not visible, then the agent understands that the user accidentally changed the view of the application. Therefore, just like in the initial calculation step, it traverses the steps of instructions backward and tries to find the current step. Also, it communicates with the 'Prompt Agent' if necessary, as mentioned in the previous phase. If the element in the next step is visible, the current step is set to the next step. If it is not disabled, this element is highlighted, and the new user instruction is displayed in the popup. Otherwise, the 'Instruction Agent' communicates with the 'Prompt Agent' about how to enable it.

If the current step is the last instruction, finish instruction, the user guidance is finished by slowly removing the element's highlight and displaying an exit message to the user. As the user guidance is finished, the event listeners for the web document are cleared and removed.

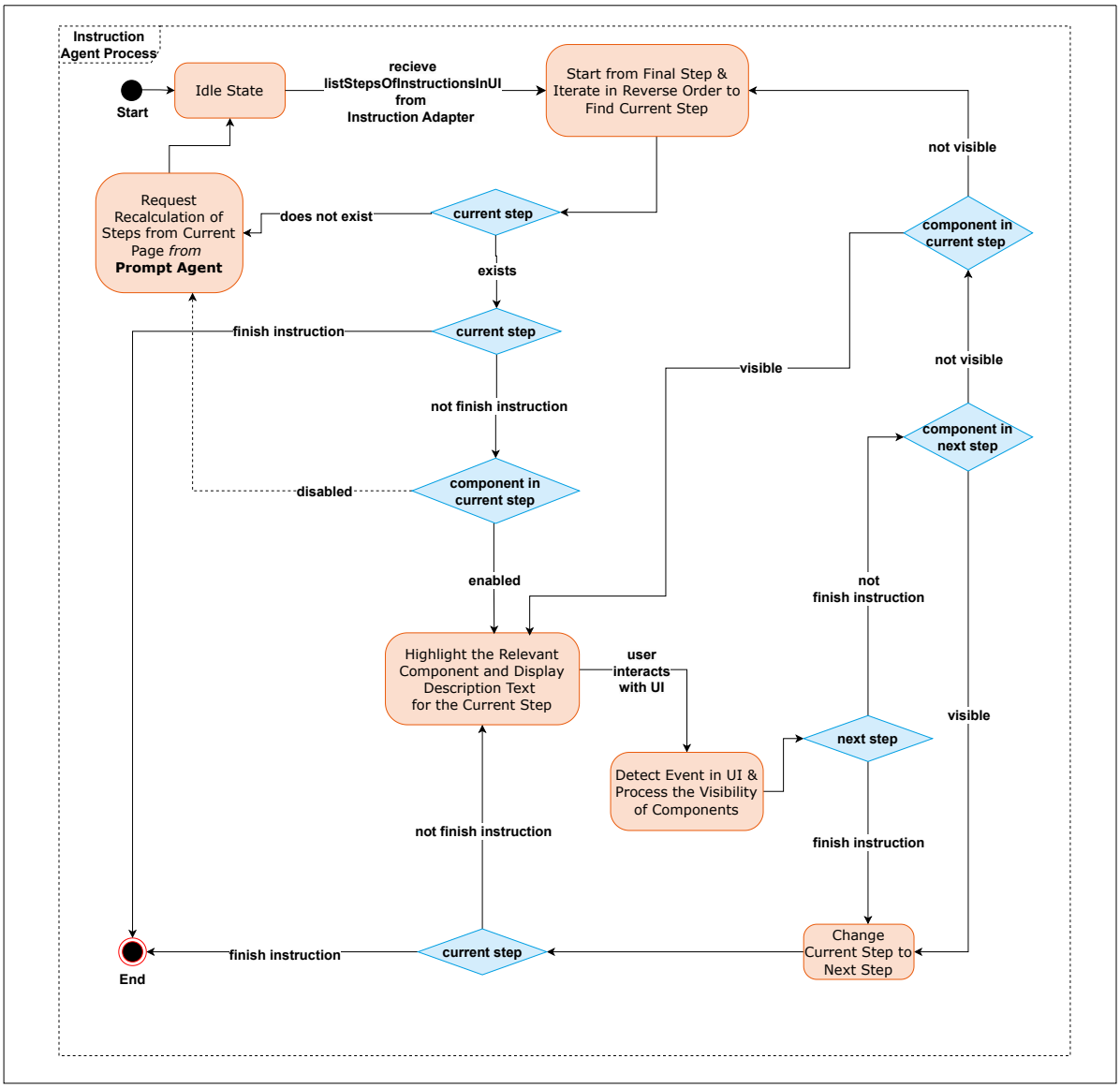

Figure 4.12.: Instruction Agent Finite State Machine Diagram

4.9. User Interaction and Application Flow

This section introduces the user's step-by-step interaction with the chatbot application. All details regarding the user's interaction with the application will be provided, and the internal processes in the application and the communication between the components that happen afterward will be explained. Figure 4.13 is the sequence diagram that visualizes the user's interactions and corresponding internal processes.

The user clicks the chatbot application icon from the web application's top bar. The popup view of the chatbot application is opened with a greeting message and is ready to take

prompts from the user. The user types the prompt, and then the popup view of the chatbot application sends the text to the 'Prompt Agent'.

The 'Prompt Agent' structures and wraps the user's prompt by adding extra information, such as the web application description with the proposed grammar and path of the current page. Then, the 'Prompt Agent' sends the structured prompt to the LLM, and the LLM calculates the necessary instructions to be performed to guide the user in the web application. After receiving the response from the LLM, the 'Prompt Agent' checks whether the returned response from the LLM matches the required format by using the 'Instruction Adapter'. It communicates with the LLM using the same structured prompt if it does not match and continues sending the structured prompts until a certain error threshold. If the error threshold is exceeded, the 'Prompt Agent' stops sending the prompts to the LLM, and an error message is displayed in the popup view. If the returned response from the LLM matches the required format, the response is passed to the 'Instruction Adapter'.

The 'Instruction Adapter' converts the response to a set of user instructions and CSS selectors. The converted steps of instructions are then passed to the 'Instruction Agent'.

The 'Instruction Agent' performs the necessary user guidance steps after receiving the set of user instructions and CSS selectors. The steps to be performed by the 'Instruction Agent' are explained in detail in the previous section, the 'Algorithm for User Guidance'. After the algorithm is completed, an exit message is displayed in the popup view, highlighting that all necessary steps have been completed.

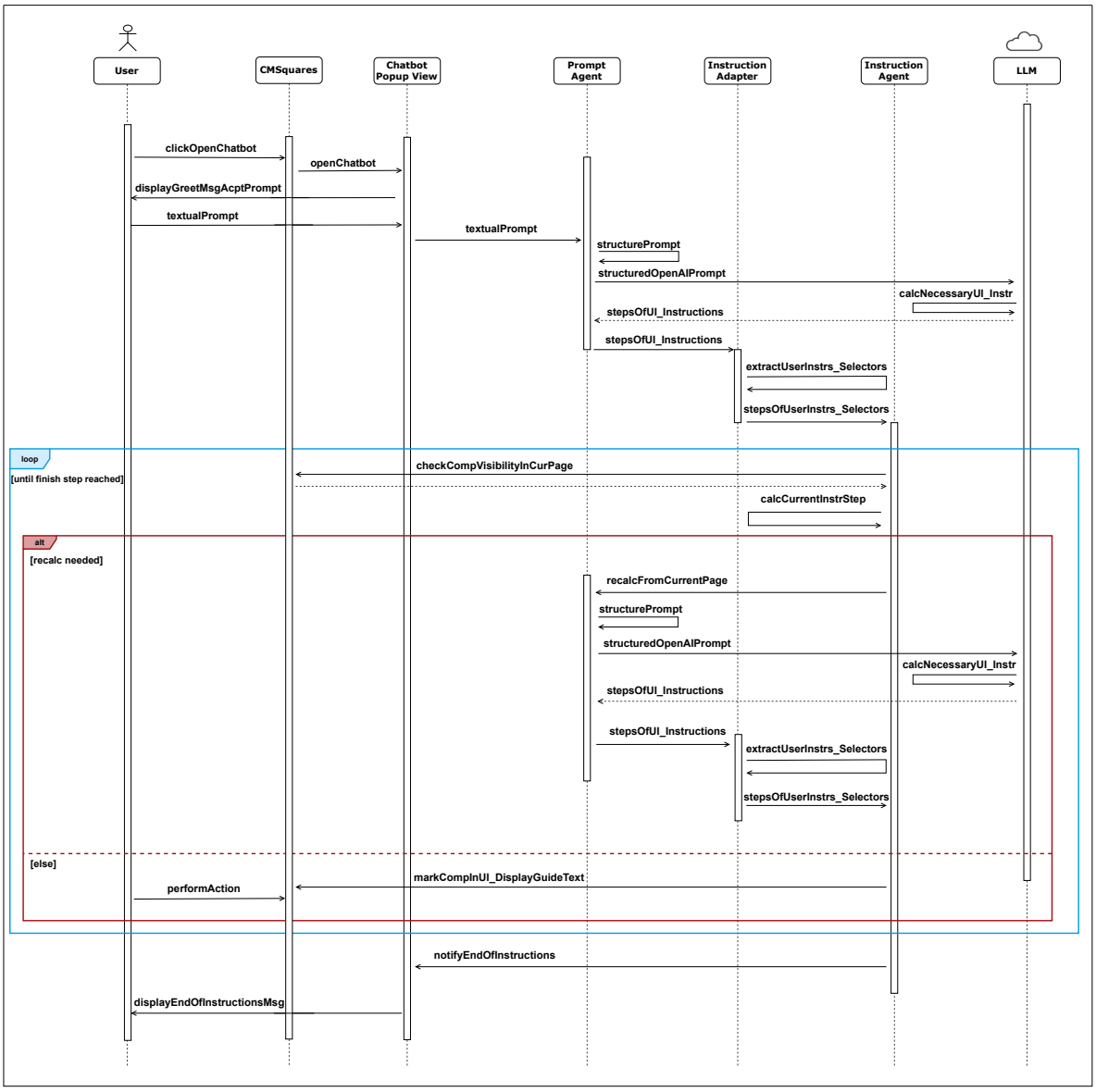

Figure 4.13.: Sequence Diagram for User Interaction and Internal Processes

4.10. Implementation of the Prototype

This section introduces the implementation details of the chatbot application. First, it introduces the pseudo-codes of the main functionalities and routines of the components. Afterward, this section gives insights into specific considerations that are made to improve the performance and efficiency of the application.

4.10.1. Design and Implementation of Algorithms

The pseudo-codes regarding the functionalities and routines of the architectural components are defined in this section. Several details regarding the implementation are omitted in order to keep the focus on the functional logic of the components and the interactions between the components. Functions regarding the visibility of the components, whether they are disabled or not, and highlighting are achieved through jQuery functions.

In Algorithm 1, the user prompt, which has a string type, is provided as an input. There is an internal prompt stack that the 'Prompt Agent' uses to store the prompt of the user, and if any web component in one of the instruction steps is disabled and need configuration, a new prompt to configure it is inserted to the top during the execution. If the function is called with a non-empty input, it means that either input is from the user or from the 'Instruction Agent' to enable a component, and it is placed at the top of the stack. A combined prompt is generated by using the web application definition with the proposed grammar, current page information, the UI instruction format explanation, and the prompt at the top of the prompt stack. Both the web application definition and the UI instruction format are statically coded in the source code. The generated prompt is sent to the LLM, and the returned textual format is checked by using the method of the 'Instruction Adapter'. In case it does not adhere to the expected format, it means that the LLM did not generate the required steps because the prompt is unachievable in this web application, or an error is encountered because of its indeterministic nature. To ensure that this prompt is not achievable, the same prompt is sent until the error threshold is exceeded. In case the returned response adhere to the expected format, it is passed to the 'Instruction Adapter' to convert to UI instructions.

```
Function receiveUIInstrs(textUserPrompt):
    if len(textUserPrompt) > 0 then
        this.promptStack.append(textUserPrompt);
    end
    if len(this.promptStack) == 0 then
        return;
    end
    curPrompt ← this.promptStack.top();
    currentPage ← getCurrentWebPagePath();
    combinedPrompt ← generatePrompt(APP_DEF, UI_INSTR_FORMAT, curPrompt,
     currentPage);
    llmResp ← await sendPromptToLLM();
    if instructionAdapter.isRespFormatCorrect(llmResp) == False then
        if this.errCounter ≤ ERR_THRESHOLD then
            this.errCounter ← this.errCounter + 1;
            receiveUIInstrs();
        else
            notify error;
        end
    else
        instructionAdapter.convertRespToUIInstr(llmResp);
    end
```

Algorithm 1: Prompt Agent's Routine

In Algorithm 2, the function's input is the LLM response obtained from the 'Prompt Agent.' The statically coded regular expression is used to extract both the list of user instructions and web component selectors from the provided input. After collecting all of them into separate lists, they are passed to the 'Instruction Agent' to execute the user guidance algorithm.

```
Function convertRespToUIInstr(llmResp):
    if len(llmResp) == 0 then
        return;
    end
    userInstrsList ← [];
    webCompSelectorsList ← [];
    while ((uiInstrMatch = this.uiInstrRegex.exec(llmResp))! == null) do
        userInstrsList.append(uiInstrMatch.userInstr);
        webCompSelectorsList.append(uiInstrMatch.webComponentSelector);
    end
    instructionAgent.execUserGuidanceAlgo(userInstrsList, webCompSelectorsList);
```

Algorithm 2: Instruction Adapter's Routine

In Algorithm 3, the event listeners are set to capture document events, such as clicking and entering an input. Afterward the user guidance algorithm has started its execution as a main routine, and it continues its execution until the guidance for the user's prompt is completed. This part of the algorithm is the implementation of the logic that is explained in details in Subsection 4.8. In *highlightWebComponent*(...) function, if the provided input element is currently not highlighted, its original border color CSS properties are stored inside an internal map, and its border is changed to the highlight color. In *removeHighlightWebComponent*(...) function, if the element is currently highlighted, its original CSS border properties are restored by using the internal map. The $fadeHighlightWebComponent(...)$ is similar function, but the element's color slowly fades away, and *removeHighlightWebComponent*(...) function is called at the end.

```
Function execUserGuidanceAlgo(userInstrsList, webCompSelectorsList):
    setDocumentEventListeners();
    this.currentStepInd ← −1;
    while len(promptAgent.promptStack) > 0 do
        while document not clicked && not first iteration do
            do nothing;
        end
        if this.currentStepInd >= 0 then
            removeHighlightWebComponent(webCompSelectorsList[this.currentStepInd]);

        end
        this.currentStepInd ← findCurrentStepIndex(webCompSelectorsList);
        if this.currentStepInd == −1 then
            promptAgent.receiveUIInstrs();
        else
            if compDisabled(webCompSelectorsList[this.currentStepInd]) then
                promptAgent.receiveUIInstrs("Enable " +
                  webCompSelectorsList[this.currentStepInd]);
            end
            displayUserInstrInChatbotPopupView(userInstrsList[this.currentStepInd]);
            highlightWebComponent(webCompSelectorsList[this.currentStepInd]);
            if this.currentStepInd == len(webCompSelectorsList) − 1 then
                fadeHighlightWebComponent(webCompSelectorsList[this.currentStepInd]);

                promptAgent.promptStack.pop();
                if len(promptAgent.promptStack) == 0 then
                    displayAllInstructionsCompletedInChatbotPopupView();
                    return;
                else
                    promptAgent.receiveUIInstrs();
                end
            end
        end
    end
```

Algorithm 3: Instruction Agent's User Guidance Routine

In Algorithm 4, the objective is to determine whether the component is visible in the currently viewed page. Therefore, if this function is not called for the first time, and there is a previous step that has been found, visibility of the web component in the next step is checked. If it is not visible, it is interpreted that the user did not performed the instruction provided, so the visibility of the previously highlighted component is checked. If it is also not visible, it can be inferred that the user changed the view completely, so the list of instructions are

traversed backwards, and the corresponding visible component is found. The same procedure is applied if this function is called for the first time. The main reason the list is traversed backwards is to find the step closest to the final step.

```
Function findCurrentStepIndex(webCompSelectorsList):
    if this.currentStepInd >= 0 then
        if compVisible(this.currentStepInd + 1) then
            return this.currentStepInd + 1
        end
        if compVisible(this.currentStepInd) then
            return this.currentStepInd
        end
    end
    stepInd ← −1;
    for i = len(webCompSelectorsList) − 1 to 0 do
        if compVisible(i) then
            stepInd ← i;
            break;
        end
    end
    return stepInd;
```

Algorithm 4: Instruction Agent's Find Current Step Routine

4.10.2. Special Details and Considerations for Implementation

Prompt Sent by the Prompt Agent

The OpenAI API supports two types of prompts: system and user prompts. In the implementation of the 'Prompt Agent', the system prompt contains the initial message from the application to inform the LLM about the use case, the web UI description with the proposed grammar, and the format of the UI instructions. On the other hand, the user prompt contains the input that is typed by the user and the current web page that the user views, which is fetched by the chatbot application. The system and the user prompt are sent together by the 'Prompt Agent'.

Figure 4.14 displays the prompt that is sent to give introductory instructions to the LLM and adapt it to this certain use case.

```
I will describe a Web App. I will describe it using a yaml.
Afterwards I will ask you the required steps to perform tasks according to the description.
```

Figure 4.14.: Initial Prompt

Figure 4.15 displays the prompt that contains the description of the web application with

the definition of the UI instruction format. The web application is not included in the image because of its length, but several parts of the descriptions are presented in Figure 4.8 and Figure 4.9.

```
The app description is:\\n\\n
'
WEB_APP_DESCRIPTION
'\\n\\n\\n
Please explain the steps without long descriptions
 but just answer them with list of ComponentUserName and ComponentSelector.
Return format shall be like $$$\`ComponentUserAction ComponentUserName\`=\`ComponentSelector\`$$$.
Please only return the steps like this format and do not forget to add triple $ sign.
Please return a response if the provided prompt is similar to a given use case described in the app description.
Otherwise don't return list of steps.
```

Figure 4.15.: Web App Description Prompt

Figure 4.16 visualizes the prompt that contains the current page information and requests the shortest path of instructions from the current page.

```
Current Path is '${currentUrl}'.
Consider the Path for the shortest list of steps.
Please consider path of each component to find shortest steps of instructions from the current path.
Please double check for shortest steps.
```

Figure 4.16.: Current Page Prompt

Figure 4.17 presents the format of the system prompt.

```
${INIT_PROMPT}\n\n${WEB_UI_DESCRIPTION_PROMPT}
```

Figure 4.17.: System Prompt

Figure 4.18 demonstrates the format of the user prompt.

```
${this.promptStack[this.promptStack.length - 1]}. ${currentPagePrompt}
```

Figure 4.18.: User Prompt

Request Recomputation of Steps

The 'Request Recomputation of Steps' is an essential part of the algorithm to be considered, as this prompt is not taken from the user, and the necessary prompt is generated from the chatbot application. There are two reasons for this to be requested. If all steps of instructions are traversed and none of the elements in these instructions are visible on the current web

page, the recomputation of steps is requested. Another reason for this request is if an element in a step is visible but disabled. A common approach is designed to cover both cases. As it is crucial to not forget the original prompt that the user typed, the chatbot application contains a prompt stack that contains both the original prompt from the user and the prompt generated by the chatbot application. The prompt generated by the user is always the bottom prompt in the prompt stack, and the chatbot application continues its execution until the prompt stack is not empty.

The logic in implementing a solution for the first reason for the 'Request Recomputation of Steps' is more straightforward. When no element is found from the instruction steps on the current page, it means that LLM did not generate the correct instruction steps. The most likely reason for this problem is the indeterministic nature of the LLMs. Therefore, it is logical to send the previous prompt to the LLM and accept the correct response. So, the prompt at the top of the prompt stack is sent to the LLM, and new calculations are made according to the returned response.

Implementing a solution for the second reason for the 'Request Recomputation of Steps' is trickier. As the element in the current step is disabled, enabling this element is required, but it is impossible to know which elements are disabled before finding the element on the web page. So when a necessary element for an instruction step turns out to be disabled, a prompt to enable it is sent to the LLM to receive instruction steps. To perform this, the 'Instruction Agent' communicates with the 'Prompt Agent' about the disabled element, and then the 'Prompt Agent' constructs a new prompt to enable this element with the current page information and places it at the top of the prompt stack. This newly constructed prompt is sent to the LLM to receive the new instruction steps. After the user guidance is finished enabling the element, this prompt pops from the top of the stack. The prompt that is at the top of the prompt stack is the previous prompt, and it is sent to the LLM with the current page information, and the new instructions steps are calculated again to perform the user's prompt. Also, this prompt stack maintains a structure to perform user guidance for multiple disabled elements recursively, as it stores the previously constructed prompts.

Minimizing Communication with the LLM

This chatbot application is functional by using the LLM, and it can perform user guidance according to the responses returned. Each prompt sent causes an inevitable network delay and computational delay from the LLM. Each delay causes temporary freezes in the chatbot application, and to make the user experience as smooth as possible, it is essential to minimize communication with the LLM and communicate only when necessary.

To minimize the communication, the 'Instruction Agent' uses the user guidance algorithm to find the necessary steps according to the returned steps of instructions. The algorithm is implemented mainly to find the current step according to the list of returned steps and not ask each step to the LLM. Communication with the LLM after the initial prompt only occurs if the necessary step cannot be found or the element in the current step is disabled. This significantly reduces the delay of execution and increases the user experience as there are only a few temporary freezes.

Specifying the Current Page

A web application may have multiple paths to navigate to a specific web page. Therefore, it is essential to specify from which page the user tries to perform the prompted action to reach the target. So, the chatbot application must pass on information regarding the current web page. The path variable in the URL is used to define the web page. Also, in the proposed grammar, it is easy to define which web elements are located in the given path of the URL. The paths of the web elements are also declared in the grammar for the web application definition. The current page information is sent to the LLM by the 'Prompt Agent' to receive the list of instructions from the given current page. This enables receiving the correct steps of instructions from the LLM with the provided prompt, as the LLM can calculate the necessary path from the current page. This chatbot application is designed to work with a browser-based web application, so the 'Prompt Agent' uses the 'window' object, which is built into TS, to fetch the path name of the current URL viewed on the browser.

Prompt Error Handling

In the chatbot application, both the user and the LLM may have unpredictable behaviors. The user may ask for a prompt irrelevant to the current web application definition, the LLM may not be able to calculate the necessary steps of instructions in the first iteration, or it may return a response that does not match the expected response format. As there is more than one unpredictable behavior, a general solution that covers all these cases is implemented. Because of the unpredictable behaviors, displaying an error message directly to the user in the first error is inappropriate, and several tries shall be made to ensure an error is encountered.

In all three unexpected behaviors, the returned response from the LLM is guaranteed to not match the response format, and the 'Prompt Agent' interacts with the 'Instruction Adapter' to verify if the returned response format is valid, as explained in the previous section. The 'Prompt Agent' contains an internal counter to count the consecutive requests made to the LLM. If the returned response does not match the response format, the 'Prompt Agent' understands that an error is encountered, so the internal counter is incremented. If the error threshold is not exceeded, the 'Prompt Agent' communicates with the LLM and sends the same prompt to the LLM again; otherwise, the error message is displayed to the user. The main reason to use such a threshold value is to minimize the likelihood of error caused by the LLM but increase the probability of returning an error message if the requested prompt by the user is irrelevant to the current web application definition.

The threshold value that is used in the implementation is 3. Although several studies and resources have been researched to find the optimal number, the threshold value may change according to the complexity of the prompts, and there is no certain answer. According to the trials that are performed with the implementation, the responses returned from the LLM become false positives after 3 trials, and steps of instructions are returned even if the prompt is not achievable. Moreover, each try requires time to complete, and the user experience degrades after 3 trials as the loading indicator is displayed for a long time. Therefore, the threshold value is set to 3 in the implementation.

4.11. Real-Life Prototype and User's Journey

The chatbot application's technical aspects and theoretical foundations are described in the previous sections. This section changes focus to highlight a user's journey while using the chatbot application. The screen captures of the chatbot application are included in this section, along with the user interactions performed. This section aims to give insight into the usage of the chatbot application by including both desired and undesired interactions by the user with the web application. The chatbot will have different states and behaviors according to the user guidance algorithm defined in the previous sections.

In the example given below, the user wants to change the background color of the web application. The chatbot will guide the user step-by-step.

In Figure 4.19, the user views the top bar of the web application. To open the chatbot application the user clicks the 'Assist' button in the top bar.

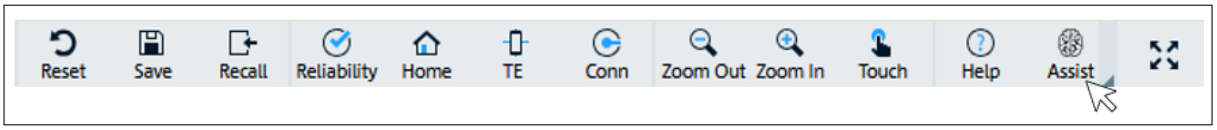

Figure 4.19.: Real-Life Prototype: Assist Button to Open Chatbot App

In Figure 4.20, the chatbot is opened with a greeting message. It is ready to accept prompts from the user.

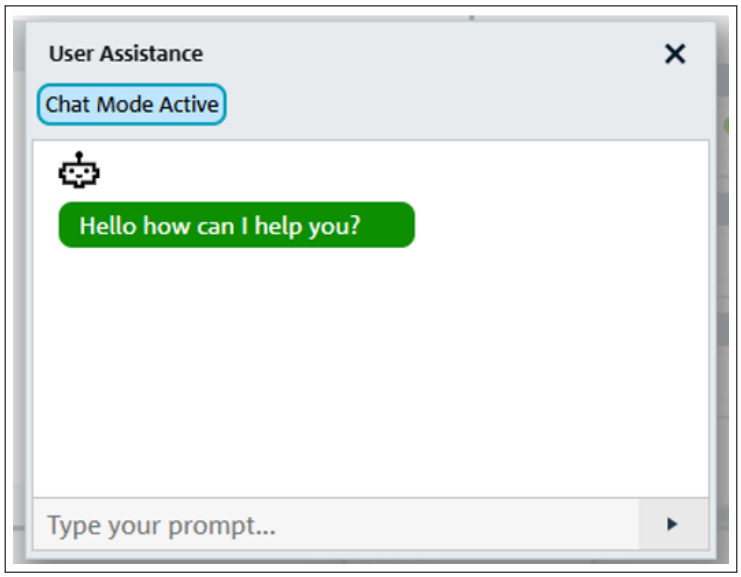

Figure 4.20.: Real-Life Prototype: Display Greeting Message

In Figure 4.21, the user types the desired prompt. For this use-case the user wants to change the background color of the application and desires virtual guidance. After typing the prompt, clicks to the send button to send the prompt.

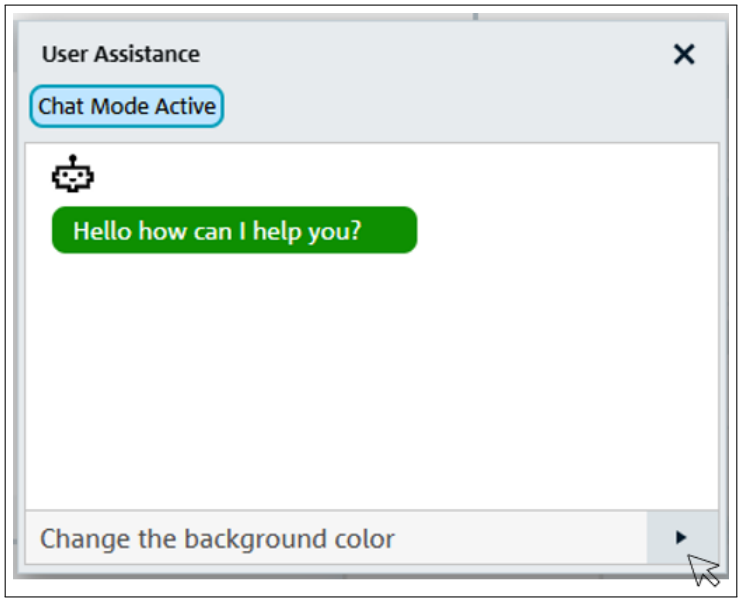

Figure 4.21.: Real-Life Prototype: User Types Prompt

In Figure 4.22, the chatbot is communicating with the LLM to fetch the required steps of instructions. The loading indicator is displayed to the user.

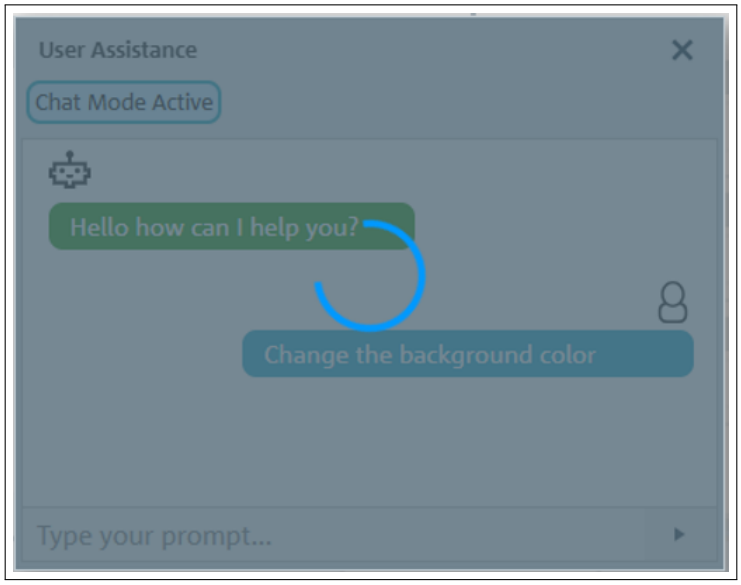

Figure 4.22.: Real-Life Prototype: Display Load Indicator

In Figure 4.23, the chatbot displays all the required steps of instructions. If the instructions are not clear enough, the user shall activate the user guidance mode so that the user will be able to view the highlighted components step by step. The user shall click the 'Activate User Guidance' button.

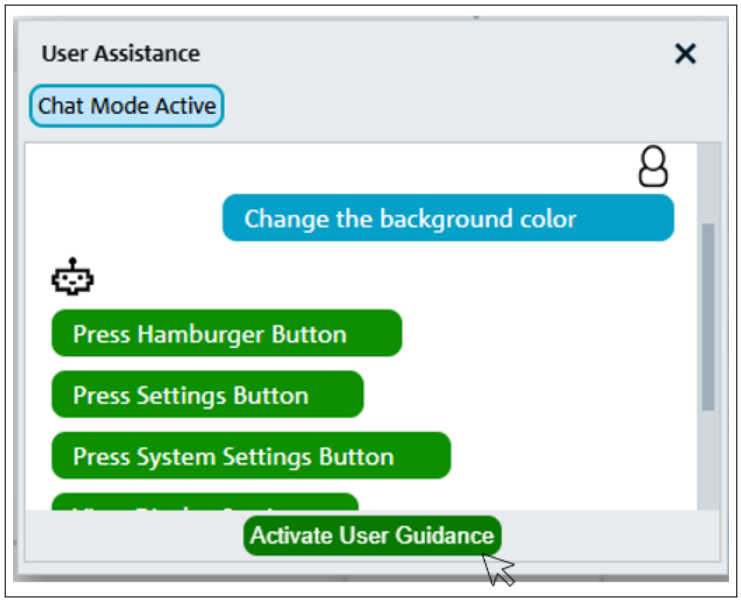

Figure 4.23.: Real-Life Prototype: Display List of Instructions

In Figure 4.23, the user clicks the 'Activate User Guidance' button, and the chatbot has been changed to 'User Guidance Mode'. In Figure 4.24, the chatbot displays the required step and highlights the relevant element. In this step, the chatbot highlights the hamburger button on the top left and instructs the user to press the button.

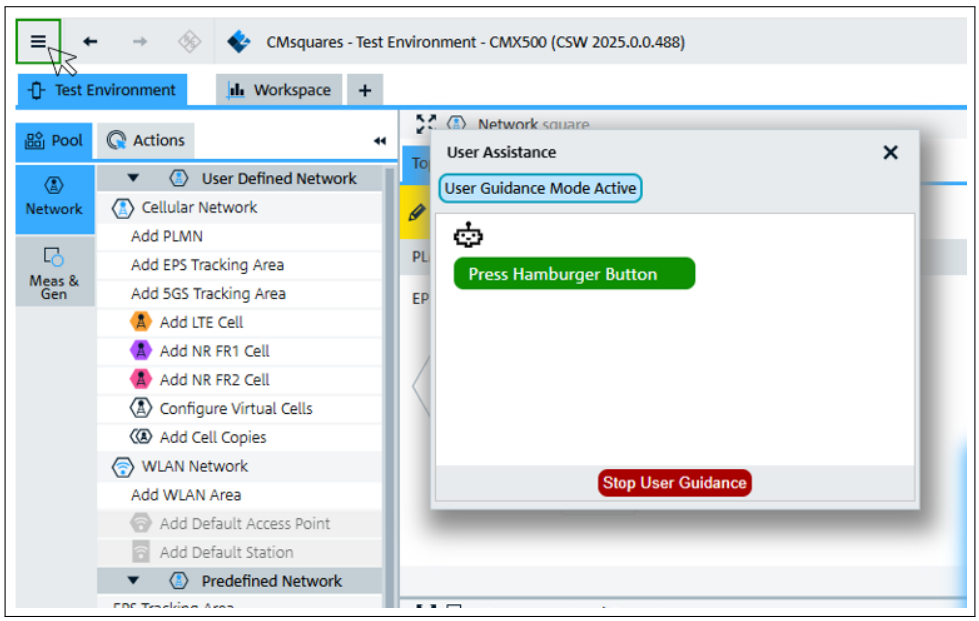

Figure 4.24.: Real-Life Prototype: Highlight Hamburger Button

In Figure 4.24, the user clicks on the hamburger button, and the menu is expanded. In

Figure 4.25, the chatbot displays the next required step and highlights the element. The 'Settings' button is highlighted, and the chatbot instructs the user to press the button. Also, the old instruction, 'Press Hamburger Button', is greyed out, and the recent instruction is displayed in green. Also, in the next steps, only the most recent instructions are displayed in green, and the old instructions are greyed out.

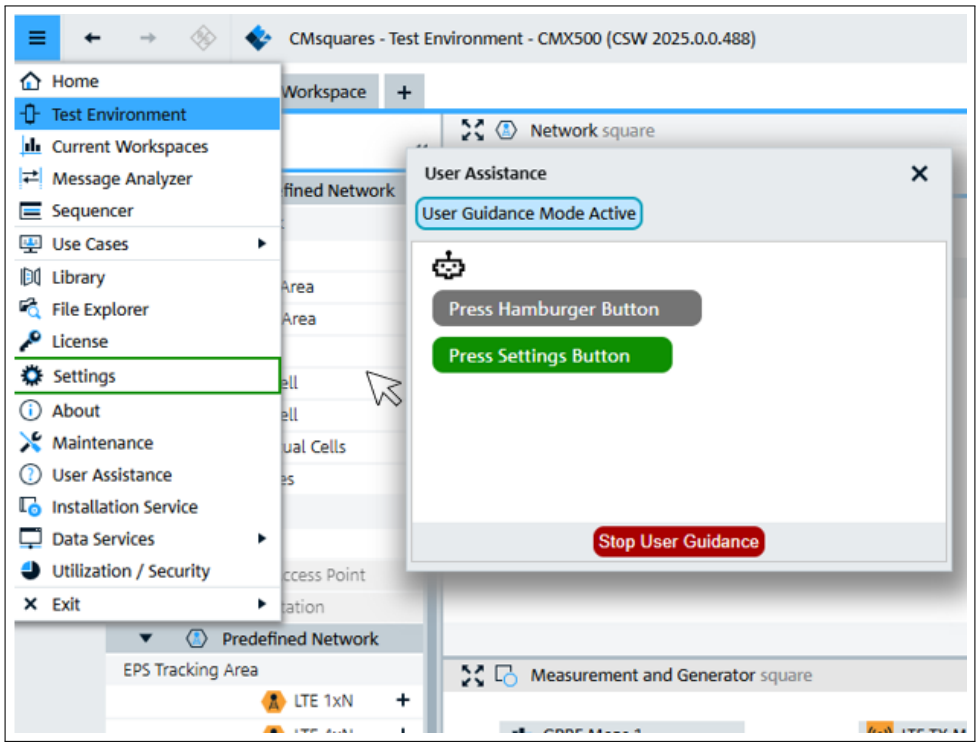

Figure 4.25.: Real-Life Prototype: Highlight Settings Button

In Figure 4.25, the user accidentally clicks on another part of the web page, and the menu disappears. The chatbot recognizes that the 'Settings' button is no longer visible, and recalculates the corresponding next step. After the calculation, in Figure 4.26, the chatbot displays the next step and highlights the element. In this step, the chatbot highlights the hamburger button on the top left and instructs the user to press the button.

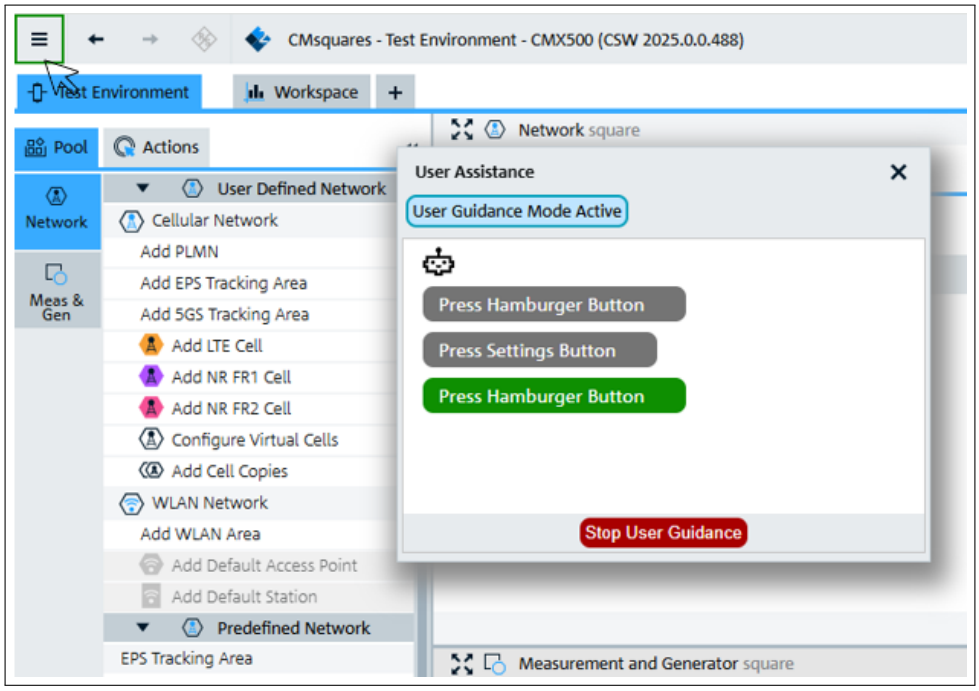

Figure 4.26.: Real-Life Prototype: Highlight Hamburger Button

In Figure 4.26, the user clicks on the hamburger button, and the menu is expanded. In Figure 4.27, the chatbot displays the next required step and highlights the element. The 'Settings' button is highlighted, and the chatbot instructs the user to press the button.

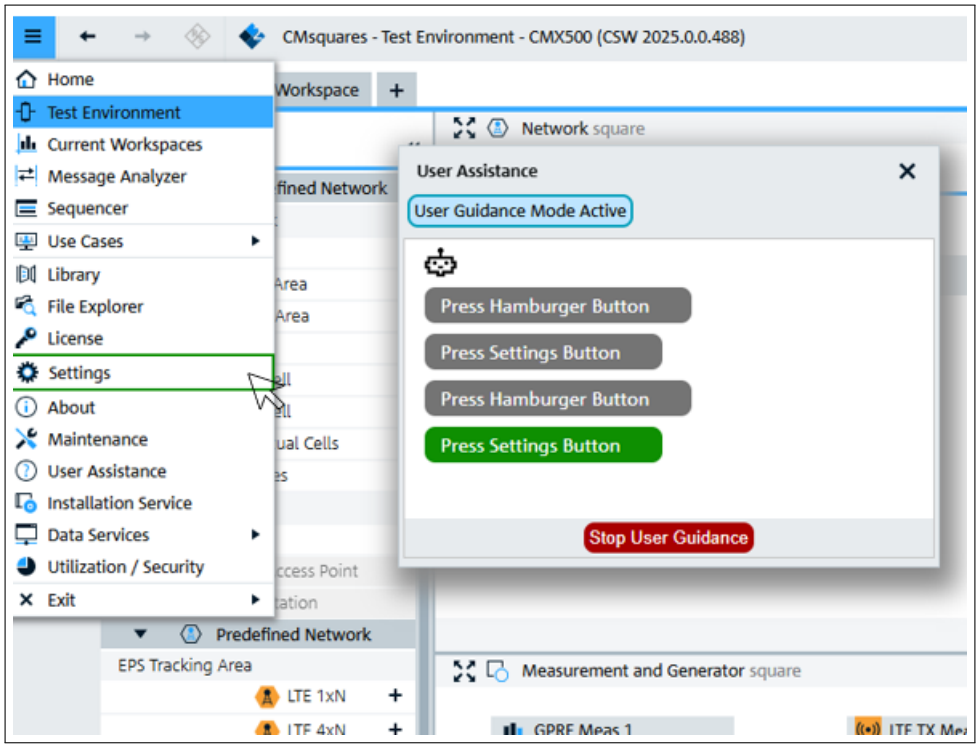

Figure 4.27.: Real-Life Prototype: Highlight Settings Button

In Figure 4.27, the user clicks on the 'Settings Button', and the 'Settings' page is opened. In Figure 4.28, the chatbot displays the next required step and highlights the element. The 'Color Set Dropdown' is highlighted, and the user is instructed to press the button.

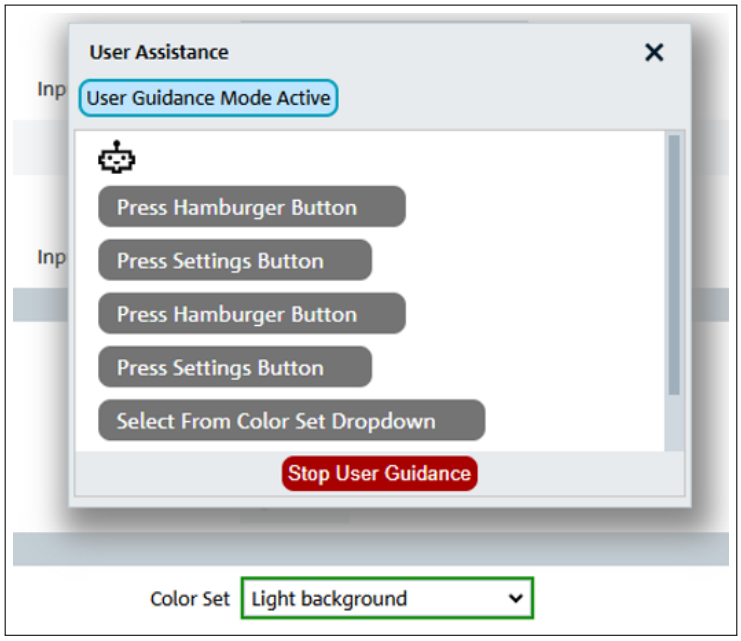

Figure 4.28.: Real-Life Prototype 10

In Figure 4.29, the chatbot displays that all steps of instructions are completed, an continues to highlight the element.

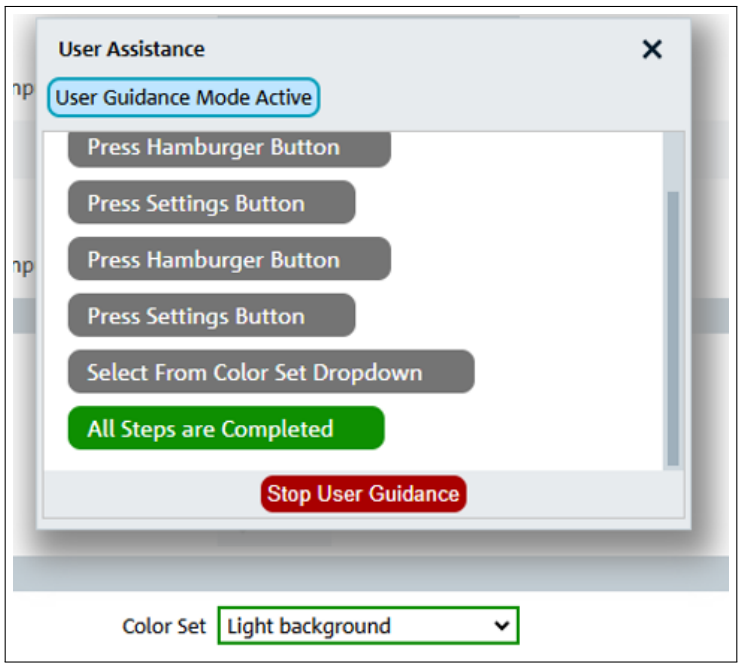

Figure 4.29.: Real-Life Prototype: Highlight Color Set Dropdown

In Figure 4.30, the chatbot continues to display that all steps of instructions are completed. After 2 seconds, the highlight color fades and changes to a darker color.

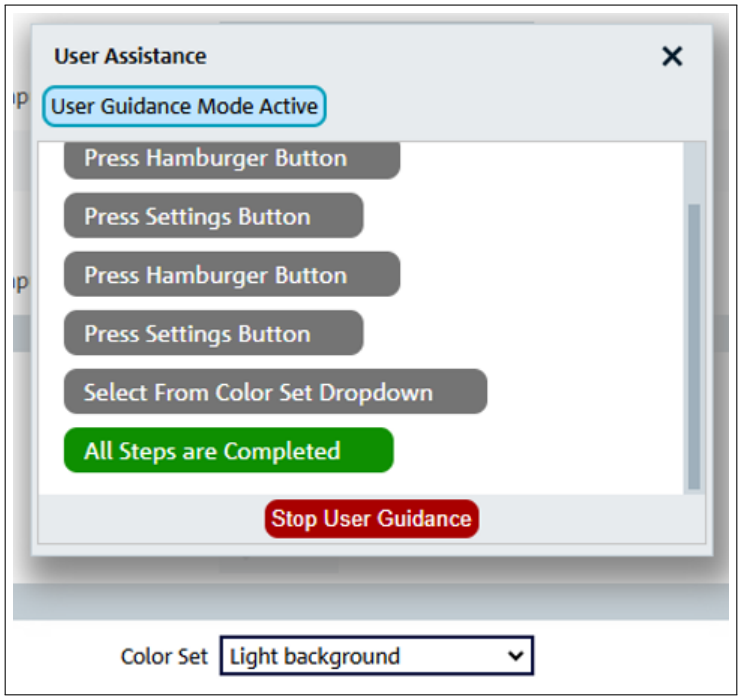

Figure 4.30.: Real-Life Prototype: Highlight Color Fades

In Figure 4.31, the chatbot continues to display that all steps of instructions are completed. After 2 seconds, the highlight color fades, and the highlight is removed.

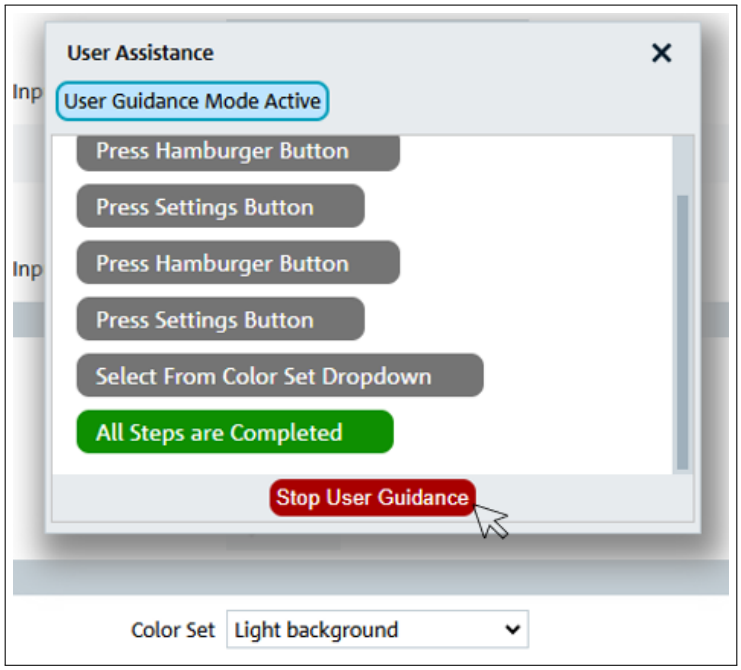

Figure 4.31.: Real-Life Prototype: Highlight Color Disappears

In Figure 4.31, the user clicks on the 'Stop User Guidance' button. In Figure 4.32, the user guidance is stopped, the chatbot changes to the 'Chat Mode', and the view is reset.

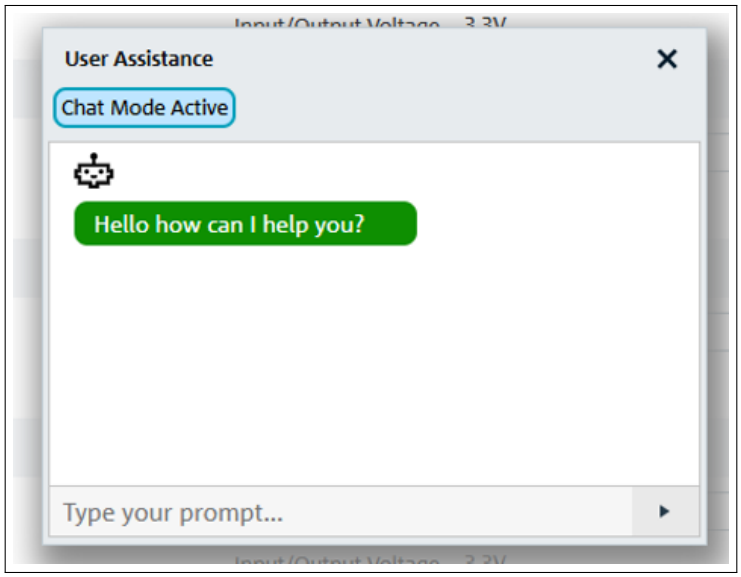

Figure 4.32.: Real-Life Prototype: Reset Chatbot View

4.12. Evaluation Methods

This section explains the methodologies used to obtain results for the research questions. Both the quantitative and qualitative methods used and constructed for this research are described and explained in detail.

4.12.1. Measuring the Understandability of the Grammar

Modeling the UI components and constructing grammar are key aspects of this thesis. The proposed grammar and how components are described by using this grammar are described in detail in the previous sections. To check that the proposed grammar successfully defines the web application and is understandable by the LLM, several parts of the web application are described with the proposed grammar. These described parts with different questions are prompted to the LLM, and the answers of the LLM are manually checked.

Different questions are prepared to check whether the LLM understands the given web application definition and can answer the relevant questions. The questions are grouped under two categories: *direct questions* and *navigational questions*. The first type of questions can be answered and interpreted without requiring navigation between the web components. The answers to these questions can be directly obtained from the definition of the component or its parent component. The second type of question requires navigation between the web components. To answer these questions, the LLM must identify which components are visible on which page and perform necessary navigation between web components and pages.

The definition of the several parts of the web application is manually prepared with the proposed grammar and prompted to the LLM, and each question is asked about the corresponding web pages and components. The questions are directly prompted to the LLM's browser-based web application. After a set of questions is prompted, the history of the LLM is cleared, and the same set of questions are asked again. This process is repeated three times to ensure that the prior prompts do not affect each other and performed measurements are accurate.

The asked questions are repeated for many web components and web pages that are found in the web application definition, so to reduce repetition, in the following questions, the web components' names are replaced with the letter X, the web pages' names are replaced with the letter Y, and specific use cases are replaced with letter Z.

The questions related to the web component's direct attributes are omitted as they are trivial to answer for LLM. The asked direct questions are:

1. What is the possible user action for component X?
2. What elements are visible on web page Y?
3. Which component is web component X's parent component?
4. Is use case Z achievable with component X?

The asked navigational questions are:

1. If component X1 is visible from the current page, is component X2 visible?
2. From page Y, what are the necessary steps to reach component X?
3. If component X1 is visible from the current page, what are the necessary steps to reach component X2?
4. If component X can be disabled or enabled, which component can enable or disable it?

4.12.2. Measuring the Generation of UI Instructions from the LLM

Generating UI instructions from the LLM is another key aspect of this thesis. This measurement is implemented by extending the previous measurement. The web application definition from the previous measurement are reused.

A snippet of code is implemented to send the prompts to the LLM. The returned textual LLM output is extracted to UI instructions using the method defined in the previous sections. These instructions and compare the returned instructions with the expected ones. Unlike the previous measurement, this focuses on test cases that generate UI instructions. Hence, the prompts in the test cases simulate user prompts and use cases of the web application. Therefore, although this measurement also measures the understandability of the grammar, it mainly measures the generation of UI instructions.

Different test cases are defined to test the accuracy of these results with different difficulties. These test cases contain the user prompt and the current page information. These test cases are wrapped with the web application definition and sent to the LLM. The returned instructions are automatically compared with the expected instructions, and the accuracy of the LLM responses is calculated. Therefore, the success of the generation of UI instructions can be derived from the result of the test cases.

Unlike the previous measurement, this used the OpenAI API to interact with the LLM instead of the web application. The chatbot application code is used to interact with the LLM and generate UI instructions from these outputs, so the OpenAI API is used to achieve it. The code snippet for this measurement uses the module used in the chatbot application for both operations. Using this module, the snippet sends the test-case prompts to the LLM and then compares the obtained UI instructions with the expected results.

For this measurement, use-case scenarios are defined with different difficulty levels.

- **Use Case Easy:** Change Background Color
- **Use Case Medium:** Change IPv4 Method
- **Use Case Hard:** Create Measurement and View Test Results
- **Use Case Error:** Order a Hamburger, Create an IP Router/ a New Color/ a Movie

The descriptions and requirements of the use-cases are defined in Table 4.3.

The web application definition is prompted to the LLM, but as there is a limitation in the token size, the web application definitions for the 'Use Case Easy' and the 'Use Case Medium'

Table 4.3.: Use Cases and Definitions Table

Use Case	Description
Easy	Requires navigation between web components and pages. Needs 1-10 steps of instructions to complete the task.
Medium	Requires the configuration of a web component, navigation between web components and pages. Needs 1-10 steps of instructions to complete the task.
Hard	Requires the configuration of a web component, navigation between web components and pages. Needs 1-25 steps of instructions to complete the task.
Error	Cannot be achieved by the web application. An empty list of instructions shall be obtained. The LLM is expected to return a textual output, but this output shall not be UI instructions according to the defined format.

are combined. The combination of these definitions and the web application definition for the 'Use Case Hard' are prompted separately for the test cases. The web application definitions mentioned contain the relevant web components for the use cases, and several irrelevant components that are found on the pages containing the relevant components. These irrelevant components are distractors and are added to mislead and challenge the application.

Test cases contain the prompts that are sent to the LLM. Each prompt comprises the corresponding use case, web application definition, and the name of the tested current page. Four web pages are tested as the current page, and each test case is repeated three times to obtain accurate results. Therefore, each use case is tested 4*3 times. For the 'Use Case Error', there are 4 questions to be asked, and the current page is not changed, but each question is asked three times. Therefore use case is tested 4*3 times.

4.12.3. Conducting Interviews about the Chatbot Application

This thesis tries to develop an approach to provide user guidance by considering software engineering principles. Therefore, just like considering the technical aspects, it is crucial to consider the user experience. The thesis tries to develop user-friendly user guidance and improve it by gathering end-user feedback. A list of interview questions is prepared to collect user feedback and improve the chatbot application further. The interviewees are asked the interview questions to gather their verbal feedback.

The chosen interviewees are employees working at Rohde & Schwarz. The interviewees are selected from different teams and are familiar with the web application and domain used. Different groups of employees are selected to evaluate the user guidance unbiasedly. Three different target groups are formed according to their familiarity with the web application:

- **Participants Unfamiliar with the Web Application:** The interviewees in this target group are less familiar with the web application and have less domain knowledge than the other groups. Eight people have been selected for this target group, and they work in different teams.
- **Participants Familiar with the Web Application:** The interviewees in this target group are highly familiar with the web application. Five people have been selected for this target group, and they work in the team responsible for developing the web application.
- **Participants Familiar with the Web Application and Chatbot Application:** The interviewees in this target group are highly familiar with the web application and have more familiarity with the chatbot application. Two people have been selected for this target group, and they work in the team responsible for developing the web application. They have also supervised the development of the chatbot application. They have shared their feedback throughout each development phase.

The interviewees are asked to try three different use cases with different difficulties. The test cases are:

- **Use Case Easy:** Change Background Color
- **Use Case Medium:** Change IPv4 Method
- **Use Case Hard:** Create Measurement and View Test Results

The interview questions are grouped under 'General Information about the User,' 'Design,' 'Navigation and Flow,' and 'Feedback and Improvement.' The interview questions are:

- **General Information about the user**
  - What is your role in the company?
  - How familiar are you with the web app and the domain?
  - What is your age?
- **Design**
  - Are the design (color, highlights, etc.) elements helpful to complete the task?
  - Do you find any design element feel confusing?
  - What is good about the user guidance and design elements?
  - Would you have preferred it a different way, and if so, how?
- **Navigation and Flow**
  - Is the user guidance helpful to complete the task?
  - Are the instructions clear and easy to understand/follow?

- **Feedback and Improvement**
  - Would you prefer using the tool over traditional help/manual/documentation, etc.?
  - What kind of tasks/use cases would you use it for?
  - What could be improved? What features are missing?
  - If you could redesign one part of this user guidance, what would you change and why?
  - Would voice assistance (input/output) be helpful?

Unlike other target groups, the 'Participants Familiar with the Web Application and Chatbot Application' are asked questions about the source code and package structure. In contrast to other interviews, these topics are discussed as a feedback session without any pre-defined set of questions. Their answers for 'Design," Navigation and Flow,' and 'Feedback and Improvement' are reflected in the 'Participants Familiar with the Web Application' in Chapter 5.

The average interview time for low-experience interviewees is around 25 minutes, while for high-experience interviewees it is around 20 minutes. The other demographics information about interviewees are represented in Table 4.4.

Table 4.4.: Interviewee Demographics and Interview Details

Demographics	**Experience Level**		**Age Ranges**			**Background**			**Total**
	Low	High	20-26	27-39	40+	EE	CS	PM	
No of Participants	8	7	5	7	3	6	5	4	15

[a] EE stands for Electrical Engineering
[b] CS stands for Computer Science
[c] PM stands for Product Management

5. Results

In this chapter, the obtained results of the proposed evaluation methods are explained briefly and discussed. The results obtained from the tests are aggregated and presented as a whole. This chapter will explain the results of the 'Understandability of the Grammar' and the 'Generation of UI Instructions from the Textual LLM Output.' Afterward, the user's feedback about usability and the results of the conducted interviews will be discussed.

5.1. Understandability of the Grammar

In this section, the main focus of the evaluation is measuring the understandability of the proposed grammar by the LLM. As mentioned in the Subsection 4.12.1, two categories of questions are prepared: direct and navigational.

In order to accurately assess and retrieve the outcomes of direct questions, the questions are prompted multiple times, and number of trials are presented in Table 5.1. Figure 5.1 visualizes the corresponding results.

Table 5.1.: Direct Questions and Number of Repetitions Table

Direct Question	**No of Repetitions**
1) What is the possible user action for component X?	8 * 3 = 24 times
2) What elements are visible on web page Y?	4 * 3 = 12 times
3) Which component is web component X's parent component?	6 * 3 = 18 times
4) Is use case Z achievable with component X?	8 * 3 = 24 times

[a] The web components' names are replaced with the letter X.
[b] The web pages' names are replaced with the letter Y.
[c] Specific use cases are replaced with letter Z.

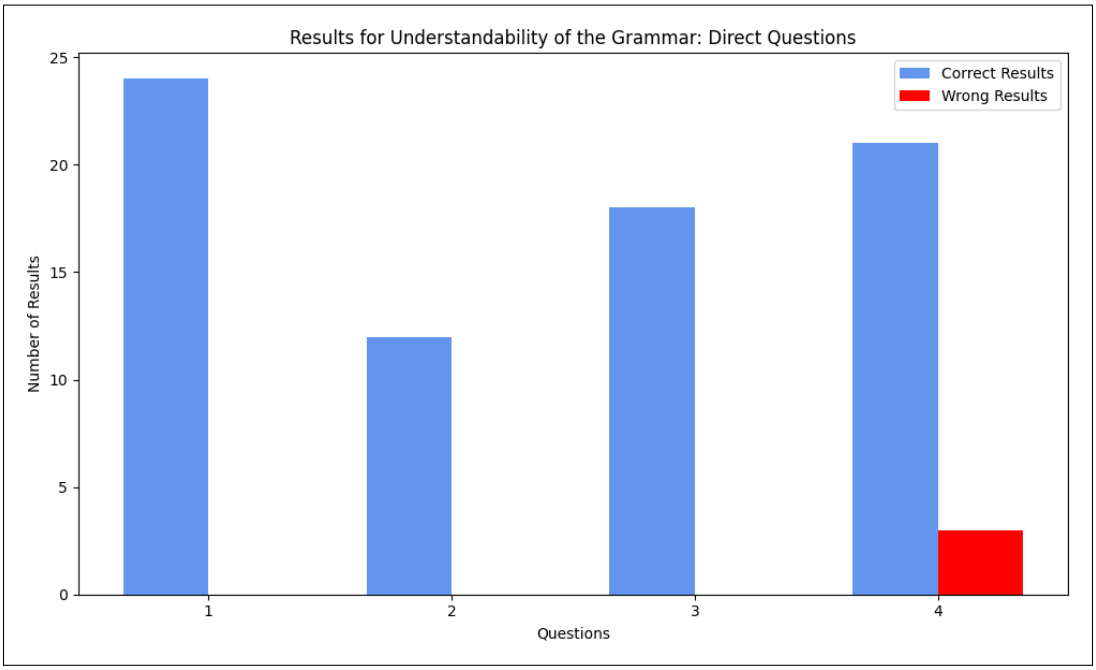

Figure 5.1.: Direct Question Results

Although there are no wrong results from questions 1, 2, and 3, several results are wrong for question 4. The most probable reason for this error is polysemy confusion. The LLM generates results that are false positives. The LLM misinterprets the meanings of the web components and the use cases. Semantic drift is also a possible cause but less likely than polysemy confusion as new questions are not likely to drift to change the interpretation of the LLM based on the evolving conversation. An example for the mentioned error is given in Figure 5.2.

Input:
Is a hamburger can be ordered with the Hamburger Button?

Output:
Yes, if we interpret your question in a playful or metaphorical way, you could say that a 'hamburger' (the food item) can be ordered with the 'Hamburger Button' if your application also has a feature that allows food ordering through that button.

Figure 5.2.: Polysemy Confusion Example

To obtain the results of the navigational questions and accurately evaluate them questions are prompted repeatedly, and the number of repetitions are represented in Table 5.2. The results are illustrated in Figure 5.3.

Table 5.2.: Navigational Questions and Number of Repetitions Table

Navigational Question	**No of Repetitions**
1) If component X1 is visible from the current page, is component X2 visible?	8 * 3 = 24 times
2) From page Y, what are the necessary steps to reach component X?	8 * 3 = 24 times
3) If component X1 is visible from the current page, what are the necessary steps to reach component X2?	8 * 3 = 24 times
4) If component X can be disabled or enabled, which component can enable or disable it?	5 * 3 = 15 times

[a] The web components' names are replaced with the letter X, X1, and X2.
[b] The web pages' names are replaced with the letter Y.

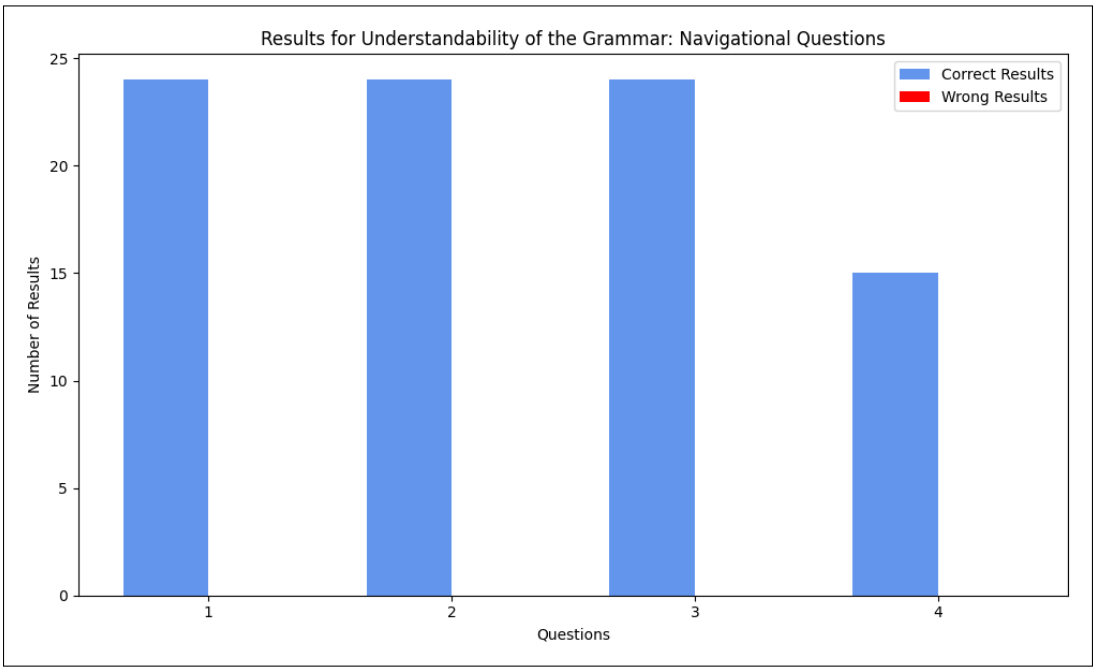

Figure 5.3.: Navigational Question Results

These questions did not return any wrong results. The LLM can interpret the connections between the web components and pages. Therefore, the LLM clearly understands the navigations from the proposed grammar.

Therefore, from the results obtained, the LLM can infer that the proposed grammar and the definition of the web application are understandable by the LLM. The obtained wrong results are not caused by the proposed grammar but by the probabilistic nature of the LLMs.

5.2. Generation of UI Instructions from the LLM

In this section, the main goal of the evaluation is the generation of UI instructions from the textual LLM output. As explained in the Subsection 4.12.2, test cases consist of the user prompts. Each prompt encompasses the corresponding use case, web application definition, and the name of the current web page. As highlighted in the previous section, four web pages are tested as the current page, and each test case is repeated three times. For 'Use Case Error,' the current page information is not changed, but each question is asked three times. Each use case and number of each use case's repetition is represented in Table 5.3.

Table 5.3.: Use Cases and Number of Repetitions Table

Use Case	**No of Repetitions**
Easy: Change Background Color	4 * 3 = 12 times
Medium: Change IPv4 Method	4 * 3 = 12 times
Hard: Create Measurement and View Test Results	4 * 3 = 12 times
Error: Order a Hamburger, Create an IP Router/ a New Color/ a Movie	4 * 3 = 12 times

The snippet checks whether the received UI instructions are exactly the same as the expected results. If the results do not match exactly, the received UI instructions are manually checked to decide the type of wrong answer. An answer may be evaluated as false if:

1. A shorter list of instructions is expected.
2. UI instructions are expected, but no instruction is returned.
3. There are missing steps of instructions.
4. An empty list of UI instructions is expected, but a list of instructions is returned.

For case 1, the list of instructions obtained is correct but is not the shortest list from the current page. The chatbot application can find the step from the current page that leads to the shortest path with the user guidance algorithm with the given steps of instructions. Therefore, this type of error does not cause an error in the execution of the application.

For case 2, although the use case is achievable, no instruction is obtained. The chatbot application interacts with the LLM until the error threshold is reached. Therefore, this type of error does not always cause an error in the execution of the application.

For case 3, although the use case is achievable, there are missing steps in the execution path. When a missing step is encountered, the chatbot application interacts with the LLM to receive new instruction steps. Therefore, this type of error does not always cause an error in the execution of the application.

For case 4, although the use cases are not achievable, a list of instructions is obtained. This case causes false positive, and the chatbot application starts guiding the user for a different

use cases, although the use case are not achievable. Therefore, this type of error causes an error when encountered in the execution of the application.

To summarize, case 1 does not cause any flaw in the execution of the chatbot application as the user guidance algorithm recovers this type of error internally. Cases 2 and 3 cause an extra delay as additional prompts are sent to the LLM. Therefore, it has a slight effect on UX. The case 4 is the only unrecoverable type of error. The indeterministic nature of the LLMs causes these unrecoverable errors. As in the Section 5.1, polysemy confusion is the most probable cause for these errors. Both the number of correct results and different cases of wrong results for each of the use case are visualized in Figure 5.4.

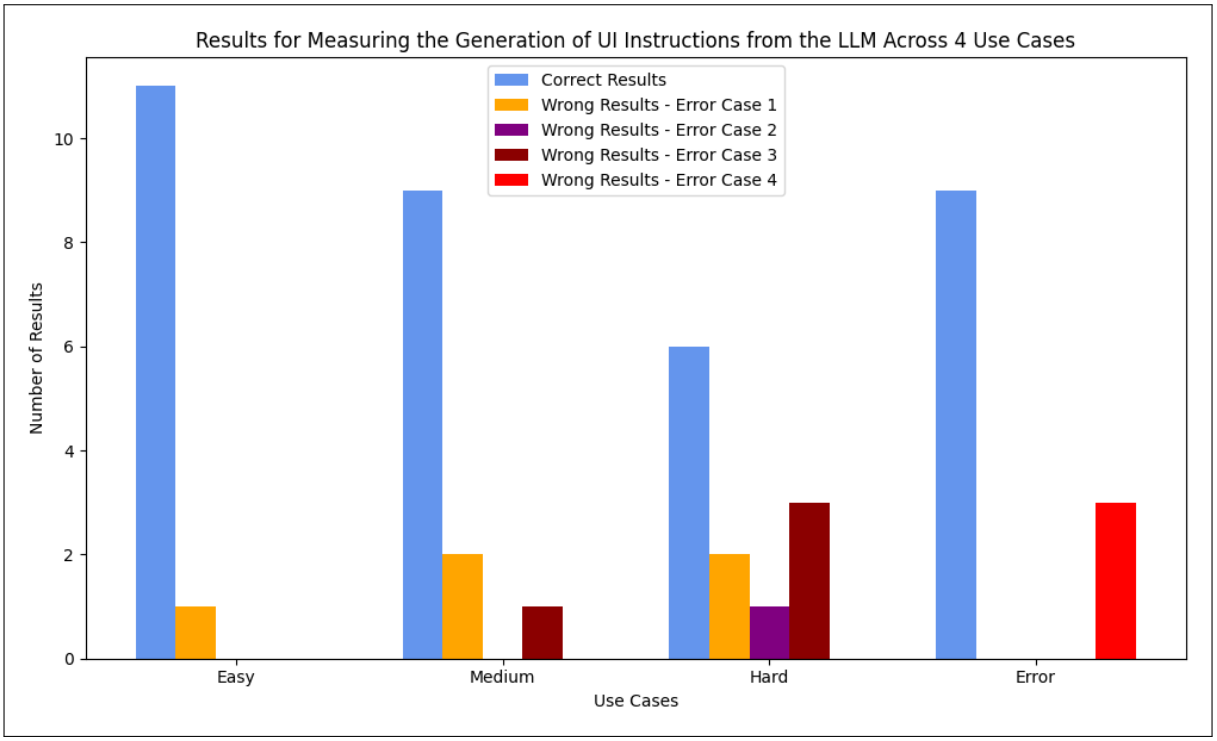

Figure 5.4.: Generation of UI Instructions from the Textual LLM Output Results

According to the results represented in Figure 5.4, the number of correct results is significantly larger than the number of wrong ones. The majority of the wrong results are in case 1. Therefore, the chatbot application's execution flow is not always affected. Only a few test cases affect the user experience or the overall execution of the chatbot application.

5.3. Results of the Interviews

In this section, the main objective is to gather feedback about the chatbot application's user experience and usability. The questions are asked to the specified interviewees in the target groups, and their feedback is collected according to the proposed questions. The feedback is

collected, and results are aggregated according to the target groups and the questions' topics: 'Design,' 'Navigation and Flow,' and 'Feedback and Improvement.'

5.3.1. Design

Feedback from Participants Unfamiliar with the Web Application

Are the design (color, highlights, etc.) elements helpful to complete the task? Do you find any design element feel confusing?

The design elements are helpful in guiding the user through the experience. They make the necessary information stand out. Especially, the color choice for highlighting the following steps is effective in terms of drawing the user's attention and it matches the rest of the design on the device. Overall, the design elements are not confusing for the users.

What is good about the user guidance and design elements?

Highlighting the corresponding web component and step-by-step displayed user instructions make the guidance easy to follow. The textual user instructions are well-organized. It is helpful to repeat the steps when the user cannot follow them properly or makes an error, and highlighting the elements on the web application's UI, significantly increases the quality of the user guidance.

Would you have preferred it a different way, and if so, how?

Several feedback are proposed by the participants to enhance the user experience. One participant suggests that a more interactive approach which splits the process into smaller steps may be designed to improve the usability and make the guidance easier to follow. Several participants recommend that when the user makes a mistake, the previous step can be highlighted in a different color, indicating the error, and inform the user about the faulty behavior. Moreover, the outdated steps' user instructions are greyed out in the chatbot popup view. A participant proposes that these steps can be highlighted with a different color in the web application, so that the user can view the previously interacted elements.

Feedback from Participants Familiar with the Web Application

Are the design (color, highlights, etc.) elements helpful to complete the task? Do you find any design element feel confusing?

Overall, the 'User Guidance Mode' contains design elements which are very helpful for the user guidance and enriches the experience. Displaying the user instruction of the current step in green and the old one in grey improves the user's journey. Also, highlighting the next web element with green color is favorable as it captures attentions and makes it distinguishable from the other web elements. The design elements make the experience easy to follow and they are not confusing.

What is good about the user guidance and design elements?

Several participants point out that in such a big web application environment, it is not straightforward to search all the elements individually. Therefore, highlighting the web element of the current step is innovative. Moreover, they believe that it is really nice to have that chatbot application can interpret the viewable web components that reside in the current page and perform highlighting with that knowledge. For example it is an impressive feature for the participants, that the chatbot application recognizes when an expander collapses and adjusts the highlighting. Another anticipated property of the chatbot is that it goes step-by-step to explain the flow expectation to the user even if the user addresses the wrong user behavior, and it does not perform the recalculations without requiring a new prompt from the user.

Would you have preferred it a different way, and if so, how?
Several interviewees mention that, currently, highlighting the web component draws enough attention, but an indicator like an animation can be used to catch the user's attention even further. Also, there has been a feature suggestion about auto-scrolling to the highlighted element in the web page may improve the speed of the user guidance and the user experience.

5.3.2. Navigation and Flow

Feedback from Participants Unfamiliar with the Web Application

Is the user guidance helpful to complete the task?
The guidance is helpful and straightforward for completing the prompted task. For a user who is quite new to the project and without a background related to the domain, it is often tough to understand, locate and navigate to the relevant components. Therefore, according to the collected feedback, the chatbot application is beneficial to guide both more uncomplicated use cases like changing the color settings and more complex tasks involving instructions regarding the specific domain.

Are the instructions clear and easy to understand/follow?
Most of the instructions are clear, easy to understand, and straightforward. However, several improvements for the clarity of the instructions can be made by simplifying some text and breaking them into bullet points. Also, some of them can be more concise to avoid information overload.

Feedback from Participants Familiar with the Web Application

Is the user guidance helpful to complete the task?
The guidance is resourceful as it provides enough details to complete the task. Participants believe that the dual mode of the chatbot application with 'Chat Mode' and the 'User Guidance Mode' is beneficial and it is a nice addition to give the users enough flexibility to chose their method of guidance.

Are the instructions clear and easy to understand/follow?

Instructions are unambiguous and intuitive. Most of the participants state that displaying each instruction step-by-step is a lovely property which enhances the user's journey. One participant comes up with a potential improvement about changing user instructions in a way that they display several "check" conditions. Therefore, the user instructions will be like "after that, you should see xy", and it may decrease the chance of misinterpretation by the user.

5.3.3. Feedback and Improvement

Feedback from Participants Unfamiliar with the Web Application

Would you prefer using the tool over traditional help/manual/documentation, etc.?

Most of the participants mention that the user guidance through the chatbot application is more interactive and visually engaging than traditional documentation. It provides a convenient and faster way to complete the task, and finding something you're looking for looks easier than doing it in documentation. However, some of the participants point out that a quick-access FAQ would be beneficial and improve the usability.

What kind of tasks/use cases would you use it for?

Interviewees anticipate that this tool is helpful for both onboarding the use cases for the web application and troubleshooting. It would be helpful to discover new use cases of the web application or when particular settings must be configured on the device. Moreover, the chatbot application can be used to guide some actions that are not straightforward in the UI.

What could be improved? What features are missing? If you could redesign one part of this user guidance, what would you change and why?

According to the collected feedback, this chatbot application may be improved with more customization options and a quick-search function for finding specific information faster. It can also be helpful to provide some in depth explanations on specific settings or features. Several redesign ideas that would improve the design of the chatbot application are about preventing the information overload by having have collapsible sections or a more dynamic layout. These ideas may improve user guidance and make navigation easier for the user.

Would voice assistance (input/output) be helpful?

Several of the participants consider that voice assistance feature can be helpful, especially for accessibility and hands-free use cases. However, this feature should be optional and customizable, because most of the participants do not believe that it will make the guidance easier than written instructions. These participants believe typing the instructions is much more straightforward.

Feedback from Participants Familiar with the Web Application

Would you prefer using the tool over traditional help/manual/documentation, etc.?

Some participants believe using the chatbot application rather than documentation is more convenient. On the other hand, some believe documentation is more handy as it is nice to get a deeper understanding. Both of the parties agree that user guidance is helpful in finding paths.

What kind of tasks/use cases would you use it for?
This tool may be an asset for finding components that the user doesn't know where the component is, and it would be convenient to discover and experiment with new features added to the application. Moreover, it may be extended to every feature and application use case. It would be beneficial to guide users for all the web application features.

What could be improved? What features are missing? If you could redesign one part of this user guidance, what would you change and why?
Gathered feedback highlight that this chatbot application may be further developed to perform automatic configurations and setups. Therefore, the user will have two options to view each necessary step individually and automatically perform the desired task instantly. It may improve the application's usability as it extends the functionality of the chatbot application. Most of the participants are primarily happy with the application's design as it fits the overall design of the web application. Some suggestions are that receiving all instructions at the beginning may be optional, according to the user's request. Therefore, the user won't need to see the list of instructions in the beginning if the user is willing to be guided.

Would voice assistance (input/output) be helpful?
Most of the participants believe that voice assistance is unnecessary. They believe that textually interacting with the chatbot application is much easier. Also, most customers who are using the web application prefer not to have voice assistance in the application.

5.3.4. Software Packages and Integration

Package structure of the chatbot application fits the package structure of the main application. The 'Popup View Component' and the 'User Guidance Service' packages are separated. Therefore, packages and source codes for the controller and view are separated. This allows a more effortless extension for potential features.

Dividing the 'User Guidance Service' to sub-components: the 'Prompt Agent', the 'Instruction Adapter', and the 'Instruction Agent', makes the application design to follow the 'Single Responsibility Principle'. These sub-components only perform their tasks and provide necessary programming interfaces for other components. The design patterns, such as observer and singleton patterns, are used, so the chatbot application adheres the best practices for software development. Moreover, several parts of the source code may be reused in further development. The constants and dependencies are packed together, and no magic values are used inside the code. This improves the maintainability of the source code.

The source code of the chatbot application is integrated into the web application's source code without making extensive changes to the primary source code. Therefore, no bugs in

the main application are introduced, and this property enables to perform easier integration and end-to-end testing.

6. Discussion

In this section, the results obtained in the previous chapter are discussed, and the interpretations of these results are briefly explained.

6.1. Understandability of the Grammar

The grammar is descriptive and successfully defines the attributes and properties of the web components. This can be inferred from the results obtained in the previous chapter. After prompting the web application definition to the LLM with the proposed grammar, it responds correctly in defining the possible use cases of the web component and the relevant user actions related to the web component.

Web pages in the web application are not explicitly defined; instead, the information regarding which web component is included in which page is given in the web component's definition. When the LLM is prompted to ask which components are included in the given page, the LLM can interpret the relations between the pages and the components and return the correct results.

In grammar, the child web components included inside the parent web component are defined, but each web component's parent component is not directly defined. The parent web component of a child web component can be inferred from the grammar, and LLM can easily make this interpretation.

The questions regarding whether a use case is achievable by a web component or not produce correct results most of the time, but several wrong answers are produced, which are false positives. The LLM can correctly identify if a use case is achievable by the web component so it does not produce false negatives. The reason several results are false positives is because of the probabilistic nature of the LLMs. Some possible LLM-based errors that may cause this issue are polysemy confusion and semantic drift. Considering the false positive results, the most probable cause for this problem is polysemy confusion. The LLM misinterprets the meanings of the web components and the use cases. Semantic drift is another possible LLM-based error that may cause this problem. Nevertheless, it is less likely than polysemy confusion as new questions are not likely to drift to change the interpretation of the LLM based on the evolving conversation. Therefore, this problem is caused by the probabilistic nature of the LLM instead of the grammar.

The results regarding the navigational questions display that all obtained results are correct. These results show that the grammar successfully reflects the relations between different web components and navigation between web components and pages. This property is especially essential in navigating different web pages and configuring web components in the chatbot

application. Therefore, the proposed grammar is appropriate in terms of supporting these functionalities.

These results and interpreted analysis demonstrate that the LLM understands the proposed grammar, captures the significant properties of the described web application, and successfully defines and describes the web application.

6.2. Generation of UI Instructions from the LLM

Most of the time, the results of the 'Use Case Easy' are the correct results. The UI instructions generated from the corresponding test cases match the expected instructions. Some of the test cases generate longer instructions than the expected instructions, and it is caused because of the probabilistic nature of the LLMs. The instructions returned from the LLM start from the web application's main page, and the LLM does not use the current page information in the prompt. Although, the returned instructions do not match the expected ones, with the user guidance algorithm, the correct step can be found in the given list of instructions. Therefore, the chatbot application's execution flow is not interrupted by an error, and the user experience is unaffected.

Generally, the results of the 'Use Case Medium' are the correct results. Like in the previous use case, several test cases generate longer instructions than the expected instructions, and the user guidance algorithm enables finding the correct step from the given list of instructions. Some test cases return wrong results because several steps of instructions are missing. The user guidance algorithm tries to find an appropriate step from the instructions in such a case, and if it cannot find it, the chatbot application interacts with the LLM to receive new steps of instructions until the threshold is reached. Therefore, only an error in the execution of the chatbot application occurs when the thresholds are reached and no correct steps of instructions are returned. Otherwise, most of the time, the chatbot application's execution flow is not interrupted. Only slight delays and interruptions in the user experience may occur.

Predominantly, the 'Use Case Hard' results are correct. Similar to the previous use case, several test cases generate longer instructions than the expected instructions, and several steps of instructions are missing from the expected list of instructions. Unlike the previous use case, some test cases return wrong results because of returning an empty list of instructions, although the use case is achievable. The most probable reason for this is that, as the use case is more complex than the previous ones, the LLM may be unable to interpret the necessary steps in each prompt because of its probabilistic nature. When the empty list of instructions is returned, the chatbot application interacts with the LLM to receive new steps of instructions until the threshold is reached. Therefore, just like in the previous use case, the only error in the execution of the chatbot application occurs when the thresholds are reached, and no correct steps of instructions are returned. Most of the time, the chatbot application's execution flow is not interrupted, and only slight delays and slight interruptions in the user experience may occur as the user waits until the correct list of steps of instructions is returned.

For the most part, the 'Use Case Error' results are correct. Unlike the previous cases, it returns a list of instructions, although an empty list of UI instructions is expected. This

error is the only error that causes an error in the execution of the chatbot application. The chatbot application guides the user through the list of steps obtained, although the empty list of steps shall be returned from the LLM. The cause of this error is not the user guidance algorithm, but just like in the previous section, the cause of this error is the probabilistic nature of the LLMs, resulting in false positive values. Although the web application has no feature corresponding to the prompt, the LLM generates instructions for a different use case. The most probable explanation for this error is polysemy confusion. The LLM misinterprets the achievable use cases.

Considering the overall results, the crafted prompts are successful in generating UI instructions, and the LLM can return them according to the proposed instruction definition. Moreover, these instructions can be extracted effectively from the textual response and grouped to the relevant user instructions and CSS selectors.

6.3. Usability of the Chatbot Application

The participants' feedback reflects that design elements and choices regarding the chatbot application help to guide the user through the journey. The color chosen for the highlight helps to attract the user's attention and matches the rest of the design on the device. Moreover, when the user does not follow the instructions as expected, automatic adjustment of the chatbot application and highlighting the relevant component improves the user's journey. To further improve the design, highlighting can be changed with an animation to draw the user's attention, and a feature like auto-scrolling for the highlighted component may be added.

The navigation and flow of the chatbot application meet the non-functional requirements. The dual mode of the chatbot application is beneficial for both users who are familiar and unfamiliar with the web application. Moreover, the instructions are clear, concise, and easy to follow, and the chatbot application can successfully guide users in uncomplicated use cases and more complex tasks with the help of step-by-step instructions. A potential improvement that may enhance the quality of the user guidance is simplifying some instructions and breaking them into bullet points to improve clarity and avoid information overload.

Further feedback reflects that the chatbot application has a high potential in onboarding users to the web application and introducing new features. Also, most participants believe the chatbot application is more convenient than traditional methods like reading documentation, and with further improvements, the documents could be replaced fully. Some potential improvements are adding more customization options to the chatbot application, implementing a quick search functionality to perform the prompted actions instantly instead of providing user guidance and adding in-depth explanations to several use cases and configurations in the user guidance.

6.4. Software Packages and Integration

The software package structure matches the package structure of the overall application. Also, best practice is followed regarding packaging as view and controller components are

separated. Moreover, the source code is easier to maintain as the required constants and dependencies are packed together, and no magic values are used inside the code. The usage of design patterns and design principles enables the reusability of code and allows further feature developments. Therefore, the developed chatbot application is properly integrated, considering the software development principles, into the main source code of the web application.

6.5. Limitations and Future Work

The limitations of this study include:

- **Token and Prompt Text Size:** The web application definition is constantly given with the prompt and is long regarding the token size. Therefore, it leaves little room for the user's input due to its size.
- **Applicability to the All Project:** The web application definition is prompted with the prompt. Therefore, only a certain portion of the web application can be defined to fit the prompt's limited token size. Also, as the proposed grammar is written manually, it takes significant time and effort to fully describe the web application.
- **Infrastructure:** The chatbot application is connected to an LLM deployed on a cloud server. For this chatbot application to function correctly, it shall be connected to an LLM, which is reachable by the chatbot application. Therefore, this design and implementation requires special consideration for systems that are air-gapped or contain extra security measures.
- **Sample Size:** This study's findings regarding usability are based on conducted interviews with several employees from Rohde & Schwarz. Therefore, a different target group with a different sample size may lead to different insights into usability.

Several suggestions for future research include:

- **Fine Tuning the LLM:** Most of the limitations are caused by the web application description being prompted. If the LLM is fine-tuned with the web application definition and the proposed grammar, the chatbot application's usage can be extended to all web application features, and the user can type longer inputs.
- **Generating a Tool to Convert Web UI to the Proposed Grammar:** The web application currently contains many features with many web components. As the web application is defined manually with the proposed grammar, it requires massive effort to describe the whole web application. Therefore, a further study may be conducted to generate a tool that automatically converts the web components and the features to the proposed grammar. Moreover, the newly developed features of the web application would be easily integrated with the usage of this tool.

- **Interview with Larger Target Group:** More detailed interviews with larger target groups can be conducted to collect design preferences and requirements regarding the chatbot application. Also, more general feedback can be gathered directly from the customers to meet their requirements.
- **Adding Software Tests:** Currently, the included test covers the understandability of the grammar and generation of UI instructions from the LLM. Further unit tests can be implemented to ensure code quality and compatibility. These tests can also be added to the deployment pipelines for further development.

7. Conclusion

7.1. Summary

The introduction of the LLM and the development of chatbot applications based on them has enabled the development of lots of new applications on different domains that are interactive and user-friendly. This research develops a methodology and a chatbot application that can guide the user in the web application step-by-step according to the user prompts. First, user requirements are collected with the supervisors from Rohde & Schwarz, and a mockup representing the prototype is designed. After clarifying the requirements, a grammar that defines the web application and is understandable by the LLM is proposed. Afterward, the prompt structure is defined that encapsulates both the web application definition, the current page information, the desired UI instructions' format, and the user' prompt, and an approach is developed that can extract the returned textual output from the LLM and converts them to UI instructions. Then, a user guidance algorithm is designed for the chatbot application to guide the user through the web application using the returned list of steps of instructions. After designing and iterating over these parts of development, the necessary components of the chatbot application are designed, partially implemented, and integrated into the main web application. Moreover, quantitative results are collected by performing experiments on both the understandability of the grammar and the generation of UI instructions from the textual LLM output, as well as qualitative results are obtained by conducting interviews and collecting feedback from the end users of the chatbot application. The results presented in the previous chapters will now address the stated research questions.

7.2. Findings

RQ1) How to model UI components to feed LLM to generate the required steps to perform UI actions?

An approach to model the web UI components is investigated by searching the related work and extending this work by proposing a grammar to define the components. The proposed grammar encapsulates the relevant information regarding the web components of the web application and is understandable and interpretable by the LLM. These attributes of grammar, whether they encapsulate all the crucial details of the web application and its understandability by the LLM, are checked by prompting the LLM with the definitions of the web application and asking questions regarding the web application to the LLM. The results show that the grammar is descriptive enough for the LLM to interpret the general layout of

the web application, the attributes of the components, the relations among the components, and the web pages.

RQ2) How to generate UI instructions from the textual output generated from the LLM?

A format to represent UI instructions is designed, a method to extract these instructions from the textual output is implemented, and the proposed format is added to the prompt to explain the format specification to the LLM. Afterward, a user guidance algorithm is implemented so that the extracted UI instructions can be followed step-by-step by the developed chatbot application. These instructions are quantitatively evaluated, and it has been concluded that the necessary UI instructions can be generated for the web application and extracted in most of the test cases. The evaluations and discussions highlight that the chatbot application can recover from most error cases with the usage of the user guidance algorithm. As a follow-up, end users are interviewed, and their feedback is gathered to qualitatively analyze the chatbot application's navigation and flow, and it has been acknowledged that the generated UI instructions are also accurate regarding enhancing the user experience.

RQ3) How to integrate this chatbot into the product by following and considering software engineering principles?

Considerations regarding the software engineering principles are separated into two groups. The first group contains the concepts regarding software development principles, package structures, and software integration, and the second one contains aspects of user experience and usability. To deliberate the first group of considerations, the chatbot application is designed and developed by considering desing principles, patterns, source code package structures. This implemented code base is integrated into the main web application code base by following its package structure. To take into account the second group of considerations, the design of the chatbot application is prototyped by considering the design of the overall application and matching design elements are chosen. Also, enhancing the usability and the user experience is strongly considered in the design and implementation. Both of these considerations are qualitatively evaluated by conducting interviews with both the supervisors of the project and the target groups, and their overall feedback reflect that the developed chatbot application is successfully integrated to the main product by considering the software engineering principles.

A. Appendix

A.1. Measurement for Generation of UI Instructions from the LLM

```python
def main():
    service = OpenAIService()
    successCtr = 0
    for t in TEST_CASES:
        try:
            # Constructing the current page prompt
            current_page_prompt = \
                f"Current Path is '{t['baseURL']}'."\
                f"Consider the Path for the shortest list of steps."\
                f"Please consider path of each component to find shortest steps of instructions from the current path."\
                f"Please double check for shortest steps."

            # Constructing the generated system prompt
            generated_system_prompt = f"{INIT_PROMPT}\n\n{WEB_UI_DESCRIPTION_PROMPT}"
            # Constructing the generated user prompt
            generated_user_prompt = f"{t['prompt']}. {current_page_prompt}\n\n"
            result = service.send_prompt(system_prompt=generated_system_prompt, user_prompt=generated_user_prompt)
            userSteps, tsSteps = respParser(result)
            expUserSteps, expTsSteps = t['result']['userSteps'], t['result']['tsSteps']
            if stepsAreEquiv(userSteps, expUserSteps) and stepsAreEquiv(tsSteps, expTsSteps):
                successCtr += 1
        except Exception:
            continue
    print('Success: ' + str(successCtr) + ' / Total: ' + str(len(TEST_CASES)))
```

Figure A.1.: Python Measurement Script for Generation of UI Instructions from the LLM

```
{
    "prompt": "Change background color",
    "baseURL": "/deviceui/testenvironment",
    "result": {
        "userSteps":  ['Press Hamburger Button',
                       'Press Settings Button',
                       'Press System Settings Button',
                       'View Display Section',
                       'View Color Set Div',
                       'Select From Color Set Dropdown'],
        "tsSteps": ['bh-burger-menu bh-btn',
                    'bh-burger-menu div.list>a:nth-child(8)',
                    'bh-settings bh-header-slot div.item:nth-child(1)',
                    'bh-display-settings bh-expander',
                    'bh-display-settings div.contentWrp',
                    'bh-display-settings div.contentWrp bh-settings-line .settingsLineContent bh-drop-down']
    }
}
```

Figure A.2.: Example Test Case

A.2. Question Examples for Understandability of Grammar

- What is the possible user action for Color Set Dropdown?
- What elements are visible on Settings page?
- Can a new color created with Color Set Dropdown?
- If Color Set Dropdown is visible from the current page, is Hamburger Button visible?
- If Settings Button is visible from the current page, what are the necessary steps to Color Set Dropdown?

List of Figures

4.1. Prototype Proposal: Chat Mode Send Prompt . . . 14
4.2. Prototype Proposal: Display List of Instructions . . . 14
4.3. Prototype Proposal: Highlight Hamburger Button . . . 15
4.4. Prototype Proposal: Highlight Settings Button . . . 16
4.5. Prototype Proposal: Display that All Steps are Completed . . . 16
4.6. Proposed Grammar Definition . . . 20
4.7. Base Components' Grammar Definition . . . 20
4.8. Definition of the Hamburger Button and the Menu Section with the Grammar 21
4.9. Definition of the Color Set Components with the Grammar . . . 22
4.10. Regular Expression for UI Instructions . . . 24
4.11. Component Diagram for the Chatbot Application . . . 26
4.12. Instruction Agent Finite State Machine Diagram . . . 28
4.13. Sequence Diagram for User Interaction and Internal Processes . . . 30
4.14. Initial Prompt . . . 35
4.15. Web App Description Prompt . . . 36
4.16. Current Page Prompt . . . 36
4.17. System Prompt . . . 36
4.18. User Prompt . . . 36
4.19. Real-Life Prototype: Assist Button to Open Chatbot App . . . 39
4.20. Real-Life Prototype: Display Greeting Message . . . 39
4.21. Real-Life Prototype: User Types Prompt . . . 40
4.22. Real-Life Prototype: Display Load Indicator . . . 40
4.23. Real-Life Prototype: Display List of Instructions . . . 41
4.24. Real-Life Prototype: Highlight Hamburger Button . . . 41
4.25. Real-Life Prototype: Highlight Settings Button . . . 42
4.26. Real-Life Prototype: Highlight Hamburger Button . . . 43
4.27. Real-Life Prototype: Highlight Settings Button . . . 44
4.28. Real-Life Prototype 10 . . . 45
4.29. Real-Life Prototype: Highlight Color Set Dropdown . . . 45
4.30. Real-Life Prototype: Highlight Color Fades . . . 46
4.31. Real-Life Prototype: Highlight Color Disappears . . . 47
4.32. Real-Life Prototype: Reset Chatbot View . . . 47

5.1. Direct Question Results . . . 54
5.2. Polysemy Confusion Example . . . 54
5.3. Navigational Question Results . . . 55

5.4. Generation of UI Instructions from the Textual LLM Output Results 57

A.1. Python Measurement Script for Generation of UI Instructions from the LLM . 70
A.2. Example Test Case 71

List of Tables

4.1. Proposed Grammar's Mandatory Fields and Definitions Table 18
4.2. Proposed Grammar's Optional Fields and Definitions Table 19
4.3. Use Cases and Definitions Table 50
4.4. Interviewee Demographics and Interview Details 52

5.1. Direct Questions and Number of Repetitions Table 53
5.2. Navigational Questions and Number of Repetitions Table 55
5.3. Use Cases and Number of Repetitions Table 56

Acronyms

API application programming interface. 5, 6, 9, 10, 12, 35, 49

CSS Cascading Style Sheets. 7, 9, 22–24, 26, 27, 29, 33, 65

DOM Document Object Model. 6, 7, 9, 24, 25

HTML HyperText Markup Language. 7, 18, 19, 24, 25

LLM Large Language Model. v, vii, viii, 1–6, 8–12, 17, 18, 20, 22–26, 29, 31, 32, 35, 37, 38, 40, 48–50, 53–57, 63–69, 73

RPA Robotic Process Automation. 7

UI user interface. v, vii, viii, 1–4, 6–9, 11–13, 17–19, 22–25, 27, 31, 35, 36, 48–50, 53, 56–58, 60, 64–69, 72, 73

Bibliography

[1] J. Daniel and H. Blake. "Explainability of Responses in GPT-4 and LLaMA". In: (Jan. 2025).

[2] H. Xu, Y. J. Kim, A. Sharaf, and H. H. Awadalla. "A paradigm shift in machine translation: Boosting translation performance of large language models". In: (2024).

[3] R. Shan and T. Shan. "Enterprise LLMOps: Advancing Large Language Models Operations Practice". In: *2024 IEEE Cloud Summit*. 2024, pp. 143–148. DOI: 10.1109/CloudSummit61220.2024.00030.

[4] M. R. J, K. VM, H. Warrier, and Y. Gupta. *Fine Tuning LLM for Enterprise: Practical Guidelines and Recommendations*. 2024. arXiv: 2404.10779 [cs.SE]. URL: https://arxiv.org/abs/2404.10779.

[5] Z. Lu, S. Mysore, T. Safavi, J. Neville, L. Yang, and M. Wan. *Corporate Communication Companion (CCC): An LLM-empowered Writing Assistant for Workplace Social Media*. 2024. arXiv: 2405.04656 [cs.HC]. URL: https://arxiv.org/abs/2405.04656.

[6] J. Eschbach-Dymanus, F. Essenberger, B. Buschbeck, and M. Exel. "Exploring the Effectiveness of LLM Domain Adaptation for Business IT Machine Translation". In: *Proceedings of the 25th Annual Conference of the European Association for Machine Translation (Volume 1)*. Ed. by C. Scarton, C. Prescott, C. Bayliss, C. Oakley, J. Wright, S. Wrigley, X. Song, E. Gow-Smith, R. Bawden, V. M. Sánchez-Cartagena, P. Cadwell, E. Lapshinova-Koltunski, V. Cabarrão, K. Chatzitheodorou, M. Nurminen, D. Kanojia, and H. Moniz. Sheffield, UK: European Association for Machine Translation (EAMT), June 2024, pp. 610–622. URL: https://aclanthology.org/2024.eamt-1.51/.

[7] M. Brachman, A. El-Ashry, C. Dugan, and W. Geyer. "How Knowledge Workers Use and Want to Use LLMs in an Enterprise Context". In: *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. CHI EA '24. Honolulu, HI, USA: Association for Computing Machinery, 2024. ISBN: 9798400703317. DOI: 10.1145/3613905.3650841. URL: https://doi.org/10.1145/3613905.3650841.

[8] R. Figliè, T. Turchi, G. Baldi, and D. Mazzei. *Towards an LLM-based Intelligent Assistant for Industry 5.0*. Affiliations: 1. Computer Science Department, University of Pisa, Pisa, 56127, Italy; 2. Zerynth, Pisa, 56124, Italy. 2024.

[9] A. T. Neumann, Y. Yin, S. Sowe, S. Decker, and M. Jarke. "An LLM-Driven Chatbot in Higher Education for Databases and Information Systems". In: *IEEE Transactions on Education* 68.1 (2025), pp. 103–116. DOI: 10.1109/TE.2024.3467912.

[10] S. E. Løvås. "AI Chatbots in Health: Implementing an LLM-Based Solution to Promote Physical Activity". Permanent link: https://hdl.handle.net/10037/34247; Date: 2024-06-16. Master thesis. Unknown, June 2024.

[11] J. Sánchez Cuadrado, S. Pérez-Soler, E. Guerra, and J. De Lara. "Automating the Development of Task-oriented LLM-based Chatbots". In: *Proceedings of the 6th ACM Conference on Conversational User Interfaces*. CUI '24. Luxembourg, Luxembourg: Association for Computing Machinery, 2024. ISBN: 9798400705113. DOI: 10.1145/3640794.3665538. URL: https://doi.org/10.1145/3640794.3665538.

[12] B. Alsafari, E. Atwell, A. Walker, and M. Callaghan. "Towards effective teaching assistants: From intent-based chatbots to LLM-powered teaching assistants". In: *Natural Language Processing Journal* 8 (2024), p. 100101. ISSN: 2949-7191. DOI: https://doi.org/10.1016/j.nlp.2024.100101. URL: https://www.sciencedirect.com/science/article/pii/S2949719124000499.

[13] K. Pandya. "Automating Customer Service using LangChain: Building custom open-source GPT Chatbot for organizations". In: *Proceedings of the 3rd International Conference on Women in Science & Technology: Creating Sustainable Career*. Submitted, 28–30 December 2023. Affiliation: Birla Vishvakarma Mahavidyalaya, Gujarat, India. Dec. 2023.

[14] A. Jonnala. *How Large Language Models (LLM) help enterprises enhance customer experiences*. Received: 15 Aug 2024. 2024.

[15] M. K. d. Melo, A. V. Almeida Faria, F. A. R. de Oliveira, F. Rezende Celestino, and V. Rafael Rezende Celestino. *Improving Customer Journeys: Data-Driven LLM Chatbot Customization*. Published on December 13, 2024. Available at SSRN: https://ssrn.com/abstract=5055005 or http://dx.doi.org/10.2139/ssrn.5055005. 2024.

[16] S. Naidu. "Role of Artificial Intelligence in Service Innovation: Challenges and Opportunities in Indian Context". In: (Feb. 2025). Article published in February 2025.

[17] K. Kumar, N. Kuhar, and M. Sharma. *Artificial Intelligence in the Indian Banking System: A Systematic Literature Review*. Available at SSRN: https://ssrn.com/abstract=5088937 or http://dx.doi.org/10.2139/ssrn.5088937; Date: November 15, 2024. Nov. 2024.

[18] T. Erdmann, S. Zecevic, N. Park, B. Ransom, H. Bui, K. Lionti, J. Hedrick, and K. Schmidt. "ChemChat—Recent Advances in Democratizing and Facilitating Access to Domain-Specific AI/ML Through LLM-Powered Conversational Assistants". In: *Proceedings of the MRS Fall Meeting 2024*. Presentation Date: 01 Dec 2024. Dec. 2024.

[19] Rohde & Schwarz. *CMX500 5G One Box Signalisierungstester*. https://www.rohde-schwarz.com/de/produkte/messtechnik/mobilfunktester-netzwerkemulator/cmx500-5g-one-box-signalisierungstester_63493-601282.html. Accessed: February 23, 2025.

[20] Rohde & Schwarz. *Demystifying 5G Unified User Experience with the R&S CMSquares*. https://www.rohde-schwarz.com/ae/knowledge-center/videos/demystifying-5g-unified-user-experience-with-the-r-s-cmsquares-video-detailpage_251220-706944.html. Accessed: February 23, 2025.

[21] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, et al. "A Survey on Evaluation of Large Language Models". In: *ACM Transactions on Intelligent Systems and Technology* 15 (2024), pp. 1–45.

[22] J. Deng and Y. Lin. "The Benefits and Challenges of ChatGPT: An Overview". In: *Front. Comput. Intell. Syst.* 2 (2023), pp. 81–83.

[23] J. Yu, S. Sun, X. Hu, J. Yan, K. Yu, and X. Li. *Improve LLM-as-a-Judge Ability as a General Ability*. 2025. arXiv: 2502.11689 [cs.CL]. URL: https://arxiv.org/abs/2502.11689.

[24] V. Nguyen Van, H. Luong Thi Minh, V. Nguyen The, T. Mong Quoc, T. Bui Anh, T. Le Anh, C. Phan Thi, and K. Nguyen Huu. "Revolutionizing education: An extensive analysis of large language models integration". In: *International Research Journal of Science, Technology, Education, and Management* 4.4 (2024), pp. 10–21. DOI: 10.5281/zenodo.14744029. URL: https://doi.org/10.5281/zenodo.14744029.

[25] M. Ma, Y. Yang, Y. Han, X. Deng, Q. Fan, and Y. Feng. "Integrating Advanced Language Models and Economic Incentives for Domain Adaptive Medical Data Stewardship". In: *2024 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. 2024, pp. 4553–4560. DOI: 10.1109/BIBM62325.2024.10821718.

[26] J. M. Lavista Ferres, E. K. Fishman, L. C. Chu, F. Lopez-Ramirez, C. K. Crawford, and S. P. Rowe. "AI in the Era of GPT: Transforming the Future of Work and Discovery". In: *Journal of the American College of Radiology* (2025). DOI: 10.1016/j.jacr.2025.02.007. URL: https://doi.org/10.1016/j.jacr.2025.02.007.

[27] Y. Wang and F. Shi. *Logical forms complement probability in understanding language model (and human) performance*. https://arxiv.org/pdf/2502.09589. Preprint. 17 Feb 2025. Affiliations: Yixuan Wang, University of Chicago; Freda Shi, University of Waterloo, Vector Institute, Canada CIFAR AI Chair. Feb. 2025. arXiv: 2502.09589.

[28] H. Li, W. Feng, X. Zhou, and Z. Shen. *GiFT: Gibbs Fine-Tuning for Code Generation*. 2025. arXiv: 2502.11466 [cs.LG]. URL: https://arxiv.org/abs/2502.11466.

[29] S. Hummel, G. Wadhwa, S. H. Abbas, and M.-T. Donner. "Practical Applications of Fed LLM for Privacy-Preserving AI in Education". In: (Feb. 2025). DOI: 10.13140/RG.2.2.31080.17925.

[30] W. Hayder. "Highlighting DeepSeek-R1: Architecture, Features and Future Implications". In: (Feb. 2025), pp. 1–13. DOI: 10.47760/ijcsmc.2025.v14i02.001.

[31] M. Cate. "The Role of Zero-Shot and Few-Shot Learning in Enhancing LLMs for Real-World Applications". In: (July 2023).

[32] A. A. Jo, E. D. Raj, and J. Sahoo. "Efficiency and Performance Optimization in Large Language Models through IB Fine-Tuning". In: *ACM Trans. Intell. Syst. Technol.* (Feb. 2025). Just Accepted. ISSN: 2157-6904. DOI: 10.1145/3718096. URL: https://doi.org/10.1145/3718096.

[33] M. Cherukuri. "Cost, Complexity, and Efficacy of Prompt Engineering Techniques for Large Language Models". In: (Jan. 2025). DOI: 10.13140/RG.2.2.10698.48325.

[34] D. Fang, J. Qiang, Y. Zhu, Y. Yuan, W. Li, and Y. Liu. *Progressive Document-level Text Simplification via Large Language Models*. 2025. arXiv: 2501.03857 [cs.CL]. URL: https://arxiv.org/abs/2501.03857.

[35] N. Paul. *Transformer Processes: Tuning Model and Prompt Engineering*. https://medium.com/@nishi.paul.in/transformer-processes-tuning-model-and-prompt-engineering-45e4ee99c469. Accessed: February 23, 2025. 2024.

[36] J. Wang. *Prompt Types of Open AI Chat GPT*. https://medium.com/@jimwang3589/prompt-types-of-open-ai-chat-gpt-a37d4743c24e. Accessed: February 23, 2025. 2024.

[37] M. Aouini and J. Loubani. *Towards more Contextual Agents: An extractor-Generator Optimization Framework*. 2025. arXiv: 2502.12926 [cs.AI]. URL: https://arxiv.org/abs/2502.12926.

[38] Microsoft. *Overview of Azure OpenAI Service*. https://learn.microsoft.com/de-de/azure/ai-services/openai/overview. Accessed: February 23, 2025. 2025.

[39] OpenAI. *OpenAI API Pricing*. https://platform.openai.com/docs/pricing. Accessed: February 23, 2025. 2025.

[40] Microsoft. *Azure Cognitive Services: OpenAI Service Pricing*. https://azure.microsoft.com/en-us/pricing/details/cognitive-services/openai-service/. Accessed: February 23, 2025. 2025.

[41] M. Michael. *Enhancing Retrieval Augmented Generation: Tackling Polysemy, Homonyms, and Entity Ambiguity*. https://medium.com/@mollelmike/enhancing-retrieval-augmented-generation-tackling-polysemy-homonyms-and-entity-ambiguity-with-0fa4d395c863. Accessed: February 23, 2025. 2024.

[42] C. Li, B. Bi, M. Yan, W. Wang, and S. Huang. "Addressing Semantic Drift in Generative Question Answering with Auxiliary Extraction". In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Short Papers)*. ©2021 Association for Computational Linguistics. Association for Computational Linguistics. Aug. 2021, pp. 942–947.

[43] I. Gibbs, S. Dascalu, and F. C. Harris, Jr. "A separation-based UI architecture with a DSL for role specialization". In: *Journal of Systems and Software* 101 (2015), pp. 69–85. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2014.11.039. URL: https://www.sciencedirect.com/science/article/pii/S0164121214002702.

[44] J. Seixas, A. Ribeiro, and A. Rodrigues da Silva. "A Model-Driven Approach for Developing Responsive Web Apps". In: *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2019)*. Copyright c 2019 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved. SCITEPRESS – Science and Technology Publications, Lda., 2019, pp. 257–264. ISBN: 978-989-758-375-9. DOI: 10.5220/0007678302570264.

[45] A. Milicevic, D. Jackson, M. Gligoric, and D. Marinov. "Model-based, event-driven programming paradigm for interactive web applications". In: *Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software*. Onward! 2013. Indianapolis, Indiana, USA: Association for Computing Machinery, 2013, pp. 17–36. ISBN: 9781450324724. DOI: 10.1145/2509578.2509588. URL: https://doi.org/10.1145/2509578.2509588.

[46] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon, and M. Florins. *USIXML: a User Interface Description Language Supporting Multiple Levels of Independence*. http://www.isys.ucl.ac.be/bchi. Université catholique de Louvain, School of Management (IAG), ISYS-BCHI, Place des Doyens, 1 – B-1348 Louvain-la-Neuve, Belgium. 2004.

[47] P. Naik, R.T.Thorat, and G. Naik. *Honing jQuery Skills for Elevating Dynamic and Interactive Web Experience (Includes Comparison Between JavaScript and jQuery)*. Nov. 2024, pp. 1–3. ISBN: 978-93-6087-876-4.

[48] J. Siderska. "Robotic Process Automation – a driver of digital transformation?" In: *Engineering Management in Production and Services* 12.2 (2020), pp. 21–31. DOI: 10.2478/emj-2020-0009.

[49] Robot Framework. *Robot Framework*. https://robotframework.org. Accessed: February 23, 2025.

[50] Robot Framework. *Robot Framework Insurance Example*. Accessed: February 23, 2025. 2022. URL: https://docs.robotframework.org/en/latest/_examples/insurance.html.

[51] M. Brambilla and P. Fraternali. *Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2014. ISBN: 0128001089.

[52] M. Karu. "A Textual Domain Specific Language for User Interface Modelling". In: Springer, 2013. DOI: 10.1007/978-1-4614-3558-7_84.

[53] T. Silva. "Towards a Domain-Specific Language to Specify Interaction Scenarios for Web-Based Graphical User Interfaces". In: 2022, pp. 48–53. DOI: 10.1145/3531706.3536463.

[54] E. Livshitz. *YAML vs JSON: Which is More Efficient for Language Models*. https://betterprogramming.pub/yaml-vs-json-which-is-more-efficient-for-language-models-5bc11dd0f6df. Accessed: February 23, 2025. 2023.

[55] J. Franklin. "DOM Manipulation with jQuery". In: *Beginning jQuery*. Berkeley, CA: Apress, 2013, pp. 43–57. ISBN: 978-1-4302-4933-7. DOI: 10.1007/978-1-4302-4933-7_4. URL: https://doi.org/10.1007/978-1-4302-4933-7_4.

[56] K. M. Kale and N. Pise. "Generative AI Application Development Using OpenAI". In: *Generative AI: Current Trends and Applications*. Ed. by K. Raza, N. Ahmad, and D. Singh. Singapore: Springer Nature Singapore, 2024, pp. 37–65. ISBN: 978-981-97-8460-8. DOI: 10.1007/978-981-97-8460-8_3. URL: https://doi.org/10.1007/978-981-97-8460-8_3.

[57] C. Hansmann and S. Braun. *AI Chatbots in Design Thinking*. Affiliations: 1. ruff consult GmbH (christian@d-hansmann.de); 2. IMLA – Institute of Machine Learning and Analytics, Offenburg University of Applied Sciences (simone.braun@hs-offenburg.de). 2024.