Institute of Software Engineering
Software Quality and Architecture

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

A Chatbot for Enhancing Explainability in Smart Homes

Leon Boppert

Course of Study:	Software Engineering
Examiner:	Prof. Dr. Steffen Becker
Supervisor:	Stefanie Bergner (Robert Bosch Smart Home GmbH)
Commenced:	January 19, 2024
Completed:	July 19, 2024

Abstract

This thesis explores the development and integration of a chatbot system for smart home environments, focusing on enhancing system explainability and user interaction within the Bosch Smart Home ecosystem. As smart home technologies grow increasingly complex, users face challenges in fully understanding and utilizing their capabilities. To address this, we propose a novel approach integrating large language models (LLMs) without extensive fine-tuning to support diverse intents, including device control and data analysis.

The primary objectives of this research are threefold: first, to comprehensively identify the essential components for creating a functional smart home chatbot; second, to push the boundaries of chatbot capabilities by developing and testing a prototype within the Bosch Smart Home app, exploring the feasibility of answering complex questions and managing devices; and third, to enhance system explainability for users, developing features that offer transparent explanations for smart home actions and decisions.

Our methodology involved an iterative development process, combining expert interviews, user surveys, and rigorous evaluation metrics. We implemented a client-server model using Android Studio and Ollama (platform for locally hosting LLMs), demonstrating practical deployment potential., demonstrating practical deployment potential. The evaluation employed a unique combination of semantic similarity and a metric for function calling accuracy to assess chatbot performance across different model iterations.

Results show that our custom "shllama3instruct" model achieved the best balance between semantically correct responses and precise function execution. User studies validated the chatbot's positive impact on system explainability and usability, while also highlighting challenges in user preference and efficiency for certain tasks.

This research contributes an architecture blueprint for integrating LLMs into smart home systems and offers insights into potential standardized frameworks for smart home chatbots. It addresses key industry concerns while advancing academic understanding of complex question-answering in smart home contexts. The findings provide valuable direction for future developments in enhancing user experience and system transparency in smart home technologies, benefiting both technical professionals and business stakeholders in the rapidly evolving smart home sector.

Kurzfassung

Diese Arbeit untersucht die Entwicklung und Integration eines Chatbot-Systems für Smart-Home-Umgebungen mit Fokus auf die Verbesserung der Systemerklärbarkeit und Benutzerinteraktion im Bosch Smart Home Ökosystem. Mit zunehmender Komplexität von Smart-Home-Technologien stehen Nutzer vor Herausforderungen, deren Fähigkeiten vollständig zu verstehen und zu nutzen. Um dies zu adressieren, schlagen wir einen neuartigen Ansatz vor, der large language models (LLMs) ohne umfangreiches Fine-Tuning integriert, um verschiedene Absichten, einschließlich Gerätesteuerung und Datenanalyse, zu unterstützen.

Die Hauptziele dieser Forschung lassen sich in drei verschiedene einordnen: Erstens, die wesentlichen Komponenten für die Erstellung eines funktionsfähigen Smart-Home-Chatbots umfassend zu identifizieren; zweitens, die Grenzen der Chatbot-Fähigkeiten zu erweitern, indem ein Prototyp innerhalb der Bosch Smart Home App entwickelt und getestet wird, um die Realisierbarkeit der Beantwortung komplexer Fragen und der Geräteverwaltung zu untersuchen; und drittens, die Erklärbarkeit für Benutzer zu verbessern, indem Funktionen entwickelt werden, die transparente Erklärungen für Smart-Home-Aktionen und -Entscheidungen bieten.

Unsere Methodik umfasste einen iterativen Entwicklungsprozess, der Experteninterviews, Benutzerumfragen und rigorose Bewertungsmetriken kombinierte. Wir implementierten ein Client-Server-Modell mit Android Studio und Ollama (Platform um lokal LLMs zu hosten), das praktisches Einsatzpotenzial demonstriert. Für die Evaluation wurde eine Kombination aus semantischer Ähnlichkeit und einer Metrik für die Präzision von Funktionsaufrufen verwendet, um die Chatbot-Leistung zu bewerten.

Die Ergebnisse zeigen, dass unser angepasstes “shllama3instruct”-Modell die beste Balance zwischen semantisch korrekter Antworten und präziser Funktionsausführung erreichte. Benutzerstudien bestätigten den positiven Einfluss des Chatbots auf die Systemerklärbarkeit und Benutzerfreundlichkeit, während auch Herausforderungen in Bezug auf Benutzerpräferenzen und Effizienz bei bestimmten Aufgaben hervorgehoben wurden.

Diese Forschung trägt eine Art Architektur-Bauplan für die Integration von LLMs in Smart-Home-Systeme bei und bietet Einblicke in potenzielle standardisierte Frameworks für Smart-Home-Chatbots. Sie adressiert wichtige Branchenbedenken und fördert gleichzeitig das akademische Verständnis komplexer Frage-Antwort-Szenarien im Smart-Home-Kontext. Die Erkenntnisse liefern wertvolle Richtungen für zukünftige Entwicklungen zur Verbesserung der Benutzererfahrung und Systemtransparenz in Smart-Home-Technologien und kommen sowohl technischen Fachleuten als auch anderen Stakeholdern in der sich schnell entwickelnden Smart-Home-Branche zugute.

Acknowledgment

I would like to express my deepest gratitude to Sandro Speth, who made this thesis possible by establishing contact with Steffen Becker. I also want to extend my sincere thanks to Steffen Becker for supervising this thesis and for his invaluable support through weekly meetings and regular discussions of my ideas.

A huge thank you to Stefanie Bergner from Bosch Smart Home for her significant contributions and assistance, especially for providing me with contacts to individuals who addressed all my concerns within Bosch Smart Home during my thesis.

I also want to give special thanks to my partner, Nancy Topalovic, for her unwavering mental support and for making my life easier during stressful phases. Nancy, along with our cat Tommy, who often laid beside me while I was formulating and working on this thesis, provided a balanced and comforting presence.

Additionally, I extend my special thanks to everyone who participated in the preliminary interviews and the subsequent user study to explore the developed chatbot. Your contributions were essential to this research. Lastly, I would like to thank the entire team at Bosch Smart Home, with whom I have had the pleasure of working in the past. Your collaboration and support have been greatly appreciated.

Contents

List of Figures

List of Tables

List of Listings

Acronyms

AI Artificial Intelligence. 22

API Application Programming Interface. 6

AST Abstract Syntax Tree. 16

BFCL Berkeley Function Calling Leaderboard. 15

FN False Negative. 55

FP False Positive. 55

FR Functional Requirement. 25

GQM Goal Question Metric. 47

IoT Internet of Things. 5

IQR interquartile range. 64

JSON JavaScript Object Notation. 3

LLM Large Language Model. 12

NER Named Entity Recognition. 23

NFR Non-Functional Requirement. 25

NLP Natural Language Processing. 6

NLU Natural Language Understanding. 5

RAG Retrieval Augmented Generation. 46

TN True Negative. 55

TP True Positive. 55

UI User Interface. 28

1 Introduction

In the dynamic field of smart home technologies, users are increasingly turning to sophisticated automation systems to optimize their everyday lives. As the complexity of these systems continues to grow [BZMW23], there arises a pressing need for innovative solutions that empower users with a deeper understanding of their environments [BKS17]. Picture a user with a huge amount of automations in their smart home, occupyied with complex questions like, "Why did I have higher heating costs this winter?" or "What was the intention behind the light just turning on in the kitchen?" While technology-interested users may attempt to answer such questions themselves through self-analysis of their smart home systems, the complexity of such tasks can be time-consuming and may not be simply feasible for everyone. These scenarios encapsulate the motivation behind this thesis, which centers on the development of a chatbot for smart home systems and home automation. The primary focus is increasing system explainability and addressing the unique challenges users encounter in analyzing the intricacies of their smart homes.

Furthermore, the implementation of a chatbot functionality holds the potential for additional benefits, including a reduction in support requests as users gain autonomy in troubleshooting [HMS24] which is also available at any time. By storing and analyzing user requests, the system could also provide insights into the needs and interests of users and thus influence future developments. Moreover, the chatbot could act as a valuable tool in identifying unintended behavior or bugs within the smart home system, contributing to a more robust and user-friendly technology landscape.

When it comes to smart home technology, users face a variety of difficulties that draw attention to important issues that call for creative solutions. First and foremost, users are confronted with a lack of tools capable of effectively providing insights into the rationale behind system actions [KOMO22]. Even when we look at less complex use cases like managing smart home devices there is for example no good solution to get an overview of the device one has or basic analysis like "Is any window open?". Inconsistencies in the "decisions" of smart home devices extend the problems even further, as developers and users can struggle to comprehend the underlying logic of actions that are executed [KOMO22], with the added challenge of distinguishing whether the observed behavior is indicative of a potential bug or a result of a specific user configuration.

A hurdle for the planned development throughout this thesis is the lack of a reference architecture designed especially for chatbots to answer complicated queries inside a smart home system. Therefore it has to be analyzed if there exist related work on chatbots which answer complex questions in a similar way to potentially inspire this work.

Another problem for the thesis is the lack of available data in the Bosch Smart Home system in which the chatbot should be integrated. This deficiency manifests in several dimensions, beginning with the cost-intensive nature of cloud services and the consequential limitations imposed by a surge in user requests that could occur in requests to a chatbot. Compounding this issue, smart home devices have constrained computing resources due to their compact size which may make the accumulation of data about the system difficult or slow. The integral role of a knowledge base

for chatbot functionality is the center of these problems, as it heavily relies on data. In the current state of the Bosch Smart Home System, although valuable data concerning device resources and system logs exists, accessibility remains restricted since logs only get available when users send a support request. The data of individual smart homes would represent an excellent source for chatbot functionality, including resource metrics such as CPU and RAM usage and logs detailing system actions. However, the challenges persist, requiring careful consideration of data availability, structuring, and integration into the chatbot's architecture to form a reliable knowledge base. These data-related challenges come together to the pressing need for innovative solutions to enhance the data accessibility and usability of smart home systems [PCBG16].

The objectives of this thesis are multifaceted, aiming to find important aspects in developing a functional chatbot tailored for answering complex questions and managing devices in the area of smart homes. The first goal is to understand the key components needed to create a functioning chatbot. This involves identifying requirements and evaluating tools suitable for building a chatbot, with a specific focus on the feasibility of utilizing open-source options to answer if they effectively can contribute to such a project.

Subsequently, the research attempts to push the boundaries of chatbot capabilities by exemplarily developing a chatbot into the Bosch Smart Home app and testing how far such a system can go in answering complex questions. The general question here is whether it is even plausible to develop a chatbot for the mentioned application. This objective involves finding and assessing the technical limitations and potential breakthroughs in creating an adaptable chatbot for this specific domain.

Finally, by providing explainability for users, the thesis seeks to satisfy the user-centric aspect of smart home technologies. This objective involves developing features within the chatbot that not only answer questions but also offer transparent and understandable explanations regarding the actions and decisions made by the smart home system. This user-focused approach seeks to enhance the overall user experience and potentially bring benefits as comprehension and trust to users about their own smart home.

The research methodology for this thesis involves an iterative development process, which is shown in Figure 1.1. This approach includes defining intents and requirements, selecting tools and technologies, implementing a chatbot prototype, and conducting a thorough evaluation. Detailed information on the methodology can be found in Chapter 3.

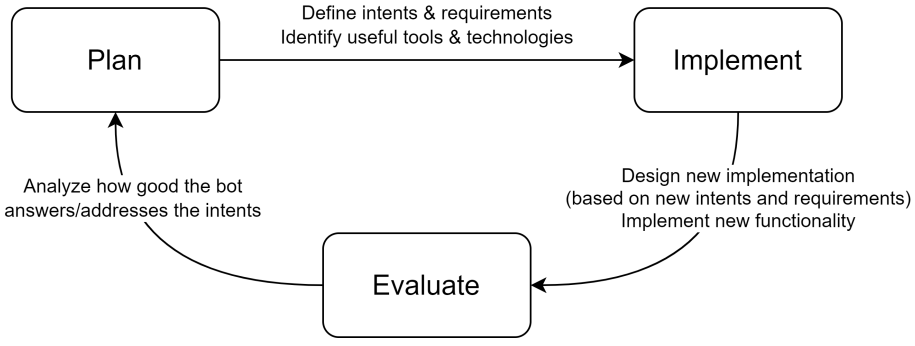


Figure 1.1: Visualization of the Iterative Development Process

The thesis employs a strict evaluation methodology, combining quantitative metrics for model performance with qualitative user studies to assess the chatbot's impact on system explainability and user satisfaction. The findings indicate that the implemented chatbot enhances the usability and explainability of smart home systems within the Bosch Smart Home environment. Evaluation metrics combining semantic similarity and JSON accuracy demonstrate the chatbot's capability to handle diverse intents such as device control and data analysis tasks effectively. One of our customized models ("shllama3instruct") emerged as the top performer, achieving a balanced approach between helpful answers and precise JavaScript Object Notation (JSON) function execution to control devices. However, participants of our study expressed uncertainty about preferring the chatbot over traditional interfaces. Additionally, challenges remain, such as the chatbot not always responding in the same language used by the user.

1.1 Explicit Contributions

This thesis advances smart home automation by developing a user-friendly, explainability-enhancing chatbot and providing a practical architecture for integrating large language models into smart home environments without extensive fine-tuning. It offers novel insights into chatbot architectures, tools, and methodologies for answering complex questions within smart home systems, particularly the Bosch Smart Home environment. While integrating a chatbot into smart homes proves challenging for complex intents, it presents unique opportunities for providing insights otherwise unattainable. The research assesses the chatbot's capabilities, addresses potential implementation challenges, and offers recommendations for future developments. Ultimately, this work aims to deepen users' understanding of their smart homes, enhancing transparency and usability of home automation systems, contributing valuable knowledge to both academic and industry spheres in the rapidly evolving field of smart home technologies.

1.2 Cooperation with Bosch Smart Home

This thesis is conducted in collaboration with Bosch Smart Home, a leader in home automation technology. Bosch Smart Home, part of the Bosch Group, specializes in innovative solutions to enhance comfort, security, and energy efficiency. Our partnership focuses on exploring the integration of a chatbot into the Bosch Smart Home system. This cooperation ensures that the research findings are practically relevant and directly applicable and also provides a smart home system to build the chatbot on.

Thesis Structure

The structure of this thesis is the following:

Chapter 2 – Foundations and Related Work This chapter lays the groundwork by discussing foundational concepts and reviewing related work in the field of smart home chatbots.

Chapter 3 – Methodology This chapter outlines the methodology, detailing the iterative process used for developing and refining the chatbot, including expert interviews and user surveys.

Chapter 4 – Concept This chapter presents the concept for the chatbot, including the definition of intents, architecture design, and requirements gathered from preliminary research.

Chapter 5 – Implementation This chapter details the implementation of the chatbot, discussing the technology stack, server and client components, interaction flow, and challenges encountered during development.

Chapter 6 – Evaluation This chapter provides a comprehensive evaluation of the chatbot's performance. It includes the study design, performance metrics, user experience feedback, results of the evaluation, a discussion of the findings, and an analysis of potential threats to validity.

Chapter 7 – Conclusion This chapter concludes the thesis, summarizing findings, discussing the implications, suggesting directions for future research, and reflecting on the limitations and lessons learned.

2 Foundations and Related Work

In the exploration of chatbot technology and its applications, this section delves into foundational aspects and related work that form the basis of effective chatbot development. The foundations include concepts such as smart homes, chatbot types, and building blocks, providing insights into the underlying principles and methodologies. As the foundation is established, the discussion extends to related work, examining existing literature on chatbots in smart homes and their applications in complex scenarios. We also dive into evaluation techniques that could inspire our own evaluation framework for this thesis. By understanding these foundational elements and existing research, the subsequent sections aim to contribute novel insights and advancements in the field of chatbot development.

2.1 Foundations

In this subsection, the focus are the specific foundations that are crucial for the development of chatbots. Including key elements for this thesis such as a definition for a Smart Home and Natural Language Understanding (NLU) which enables chatbots to understand a users input message. Other elements such as a general architecture and typical building blocks for chatbots are also presented.

2.1.1 Smart Home

A smart home device's main aspect is that its original functionality is augmented through network capabilities [BVM18; Sch15] and should enhance especially comfort [BVM18; Mat18] but also security [BVM18] or energy efficiency [BVM18; Mat18] for example. Therefore a smart home consists of such devices and provides additional infrastructure like an app to control the devices. These infrastructures often allow automation of the devices. Since its functionality is based on connected devices and often sensors it is often named in the context of the Internet of Things (IoT) [AIM10].

2.1.2 Chatbot Types and Classifications

Based on Lehmann [Leh21] there are three main types of chatbots: decision-tree-based, keyword-based, and advanced context-aware bots. Decision-tree-based bots follow a predefined flowchart in response to user queries, often identified by menus and buttons. Keyword-based bots recognize specific keywords, making decisions and providing responses based on internal knowledge. Advanced context-aware virtual assistants engage in free-flowing conversations, learning from interactions and offering versatile responses.

Additionally, according to Adamopoulou and Moussiades [AM20] chatbots can be classified based on their **knowledge domain**, with open domain bots discussing general topics and closed domain bots focusing on specific knowledge domains. In terms of **service**, interpersonal chatbots act as information intermediaries, offering communication services like restaurant or flight booking. Intrapersonal chatbots, existing within the user's personal domain, function as companions, understanding users like humans. Inter-agent chatbots, omnipresent in nature, facilitate inter-chatbot communication. **Goal-based** classification includes informative chatbots providing stored information, chat-based bots conversing like humans, and task-based bots performing specific functions intelligently. **Input processing and response generation** methods vary, from rule-based models relying on predefined rules to retrieval-based models using Application Programming Interfaces (APIs), and generative models employing machine learning techniques. **Human-aided** chatbots incorporate human computation for flexibility and robustness. The **build method** distinguishes between open-source platforms allowing intervention and closed platforms acting as black boxes but providing immediate access to advanced technologies, often found in large companies.

2.1.3 Chatbot Building Blocks

In this subsection building blocks and architectural insights that can be identified in different literature are collected and explained. It is mostly based on Lehmann [Leh21] and [AM20] if not mentioned otherwise.

Natural Language Understanding and Machine Learning

NLU, a subset of Natural Language Processing (NLP), is essential for chatbots. It involves training algorithms to understand and process natural language, bridging the gap between human language and artificial intelligence. NLU is critical for successful chatbot operation, enabling the machine to comprehend user intents and contexts.

Machine learning and NLP also play roles in the evolution of virtual assistants. Machine learning algorithms, based on training data, build statistical models and autonomously improve over time. NLP focuses on processing and understanding natural language, incorporating elements from computer science, AI, linguistics, and data mining. These technologies empower chatbots to interpret complex human language, allowing for more sophisticated and context-aware interactions.

Intents

Chatbots operate with three key components – Utterances, Intents, and Entities. Intents represent different user intentions that a chatbot anticipates, each consisting of various utterances (training phrases) and one or more responses. Utterances are potential user expressions or example questions, and entities are real-world objects mentioned in a sentence. This structure facilitates the interpretation of natural language, enabling chatbots to recognize user intents and associated entities for effective communication.

Chatbots are built on different intents, each with various utterances and responses. The training involves teaching the bot to recognize user intents based on a set of training phrases. A Natural Language Processing Engine interprets natural language, transforming it into structured language, and identifies entities within sentences to enhance contextual understanding.

There are also narrow work that regard intent-based chatbots as a separate species [LLLS22]. These then typically deal with the technique used to generate responses to ultimately categorize the chatbots.

Knowledge Bases

Knowledge bases serve as a crucial component for chatbots, providing them with the necessary information to respond intelligently to user queries. These databases store domain-specific data, facts, and contextual information that enable chatbots to understand and address user requests accurately.

General Chatbot Architecture

In general, the chatbot architecture by Adamopoulou and Moussiades [AM20] which can be seen in Figure 2.1 consists of components such as the Language Understanding Component, which interprets user requests and identifies intentions and associated information, and the Dialogue Management Component, which keeps track of the conversation context, processes clarifications, and asks follow-up questions. After understanding the user's request, the chatbot executes actions or retrieves data from its Knowledge Base or external resources, and the Response Generation Component generates natural language responses based on the intent and context information.

2.1.4 Prototype and Iterative Development

This subsection is based on Sommerville [Som11] according to who "iterative development of the prototype is essential". The process for developing a prototype is shown in Figure 2.2a and summed up consists of planning, developing and evaluating. Another iterative process is the extreme programming release cycle which is shown in Figure 2.2b and consists of user stories that are selected and developed throughout one cycle. Based on this concepts and needs for our chatbot prototype we built the iterative development cycle that is shown in Figure 1.1 which could be seen as a combination of the both processes presented in this subsection. It is useful to start projects like this thesis with a prototype with thought out case [Leh21].

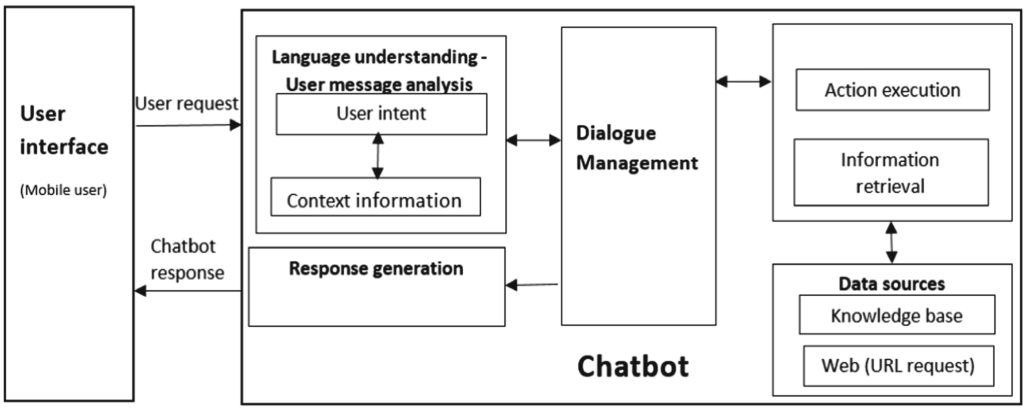
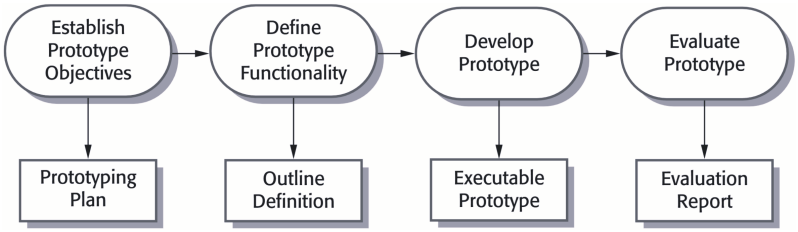
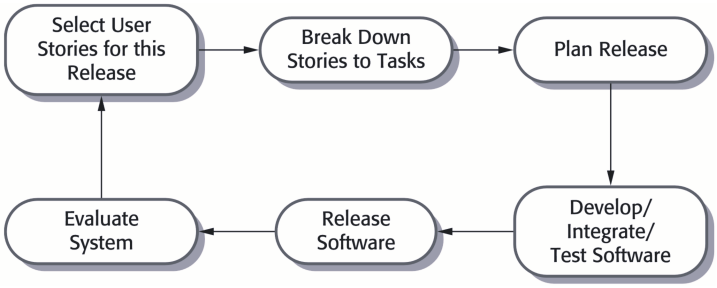


Figure 2.1: An Example Architecture of Chatbots in General
Source: Adamopoulou and Moussiades [AM20]



(a) Prototype Development Process



(b) Extreme Programming Release Cycle

Figure 2.2: Visualized Development Processes
Source: Sommerville [Som11]

2.1.5 Function Calling in Language Models

Based on ChatGPT [Ope24a] function calling can be summarized as:

> Function calling in language models refers to the ability of a language model to interact with external functions or APIs during a conversation. This feature enhances the model's capabilities by allowing it to perform specific tasks, retrieve real-time information, or execute actions beyond its built-in knowledge base.

Function calling in language models allows the model to interact with external tools, APIs, or functions during the generation process. This capability enables the model to access up-to-date information, perform specific tasks, and provide more accurate and contextually relevant responses [Goo24; Mis24; Ope24b]. The process of function calling typically involves the following steps:

- **Function Declaration:** The user defines a set of functions with their names, descriptions, and parameters [Mis24].
- **Model Processing:** When given a prompt, the model determines whether to use one of the declared functions based on the context and requirements of the query [Mis24; Ope24b].
- **Function Call Generation:** If the model decides to use a function, it generates a structured output (usually in JSON format) containing the function name and necessary arguments [Mis24; Ope24b].
- **External Execution:** The application receives the function call details from the model and executes the corresponding function or API call [Goo24; Mis24].
- **Response Integration:** The result from the external function is then provided back to the model, which uses this information to generate a final response [Goo24; Mis24].

This approach allows language models to overcome limitations such as accessing real-time data or performing specific computations, making them more versatile and powerful in various applications [Goo24].

Function calling is particularly relevant to a smart home chatbot as it significantly enhances the bot's capabilities and usefulness. In the context of a smart home, function calling enables the chatbot to interact directly with various devices, APIs, and databases, allowing it to perform tasks such as controlling devices, retrieving real-time information about device states, and analyzing power consumption data [Goo24]. For instance, when a user asks to "turn off the living room lights" the chatbot can use function calling to execute the appropriate command on the smart home system. Similarly, for queries about power consumption or device information, the chatbot can call specific functions to access and analyze the relevant data, providing accurate and up-to-date responses [Mis24; Ope24b]. This integration of function calling makes the chatbot a more effective interface between the user and the smart home ecosystem, enhancing the user's ability to understand and control their smart home environment.

2.2 Related work

This section presents literature that is related to this thesis. While chatbots are ever known to be used in areas like customer service it is interesting to see work on how chatbots are used in smart homes or for complex scenarios like managing containerized networks. Some literature has the nice side effect that it also presents the architecture of the developed system and can inspire the architecture of the chatbot that should be developed in this thesis.

2.2.1 Chatbots for Smart Homes

Various literature exists that contains some kind of chatbot in the context of Smart Homes. Most of these chatbots perform the same task as nowadays common Voice assistants as for example the Google Assistant[1] which is capable of understanding written request but also transforming speech into written requests which also includes managing smart devices added to Google Home[2], an app for managing a smart home.

Baby et al. [BKS17] presented an approach for a chatbot that can control multiple smart devices in a smart home. The developed prototype is capable of answering simple questions that regard the devices or change variables of the devices. It is able to answer questions like "What is the temperature in room 1" or "Set the temperature to 19 degrees celsius". The architecture includes multiple building blocks that were explained in Section 2.1: An NLP Pipeline which in the end identifies the intent of a message and matches an action to. The pipeline can be seen in Figure 2.3 The intents and pipeline of this chatbot could inspire the inital iteration of our prototype.

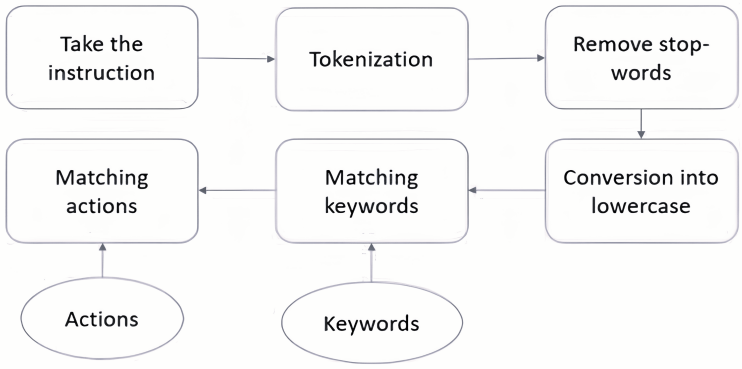


Figure 2.3: A Simple NLP Pipeline for a Chatbot
Source: Baby et al. [BKS17]

Another work developed and connected a chatbot to the Facebook (today called meta) messenger for controlling smart home devices fans and lights but also gas leakage detection or humidity monitoring [APM+20]. Intents are not clearly defined or at least not stated but it seems to be a simple decision-tree-based chatbot. This is probably due to focus of the work laying more in the systems architecture and a limited amount of intents.

[1] assistant.google.com
[2] home.google.com

A different work explored the area of collaboratively teaching where a focus was set on mitigate malicious activities [CA21]. The presented chatbot called Safebot is intended as an extension to smart home assistants and is able to communicate when it does not know the answer the a request and can be taught afterwards. Also, users could notify the bot that an response was not appropriate resulting in also teaching it. The learning is purely based on natural languages in contrast to chatbots that for example use knowledge bases with structured data.

2.2.2 Chatbots for complex Scenarios

Various chatbots for many different use cases exist. As the intention of this work is to test how far a chatbot can go in answering complex queries to a smart home the question aligns if approaches exist for answering (complex) questions in a closed domain based on available data. An example for this is the work of Ait-Mlouk and Jiang [AJ20] which approached to develop a knowledge graph based chatbot that can find information in linked data through NLU. The researchers in this work address challenges like understanding many different queries (and intents) and making the use of multiple languages and knowledge bases available. The system developed is able to be extended with new domains. The architecture of it can be seen in Figure 2.4. While it is too complex to explain in detail, it processes queries into intents and entities (through Named Entity Recognition) which in combination make it possible to retrieve information from connected knowledge bases by transforming it into queries and selects a response based on a knowledge graph.

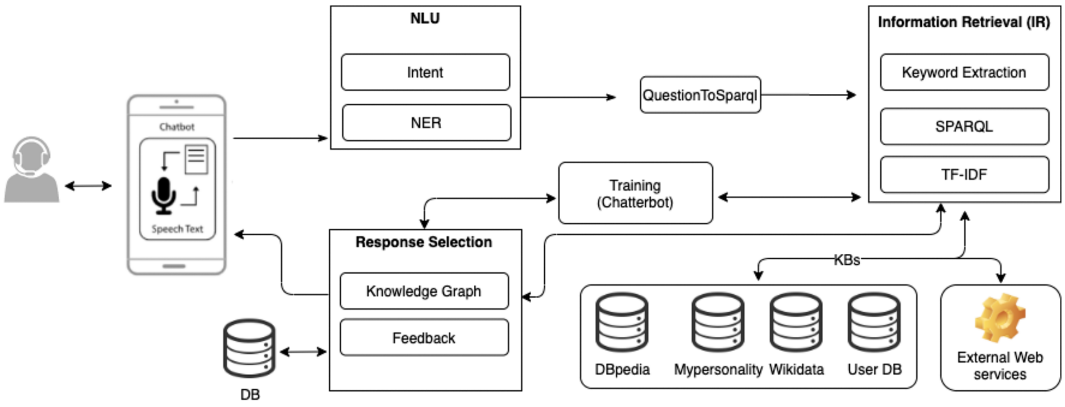


Figure 2.4: Detailed Architecture of KBot
Source: Ait-Mlouk and Jiang [AJ20]

Another research presented a chatbot that can create and manage a containerized network, thus acting as a workflow manager [JQFF23]. It enables less human involvement by computing requirements written by users. This leads to a decreasing need for users to have extensive knowledge of the underlying tools to be able to setup such networks and also makes the management of them easier in general. This can be mapped to this thesis where one aim is to enable even users with less technology knowledge to analyze complex scenarios in their smart home and to make the analysis of those easier in general.

A different work [CCI+20] approached to develop a chatbot that helps to gain knowledge about Data Science and Machine Learning. The architecture includes information retrieval from a knowledge base, parsing the received document and answering a query based on the search in the knowledge base. It also includes a "Small Talk Module" which improves users satisfaction with the system and increase the overall interest in chatting with it.

2.2.3 Advancements in Function Calling of Language Models

Recent advancements in Large Language Models (LLMs) have demonstrated significant potential across a range of tasks, including mathematical reasoning, natural dialogue, and program synthesis. Despite these advancements, current state-of-the-art models like GPT-4 still face challenges in accurately utilizing tools via API calls. These challenges include generating precise input arguments and avoiding the hallucination of incorrect API usages.

To address these limitations, Patil et al. [PZWG23] introduced Gorilla, a fine-tuned LLaMA-based model specifically designed to enhance the performance of LLMs in writing API calls. Gorilla leverages a document retriever to adapt to test-time document changes, significantly mitigating hallucination issues and improving the accuracy of API call generation. The model's capabilities were rigorously evaluated using APIBench (a comprehensive dataset of APIs from HuggingFace, TorchHub, and TensorHub).

Gorilla's innovative approach integrates retrieval-aware training, which allows it to remain updated with frequently changing documentation and thus ensures more reliable and applicable outputs. The model outperforms GPT-4 in functional correctness of API calls, demonstrating good adaptability and reduced error rates. This makes Gorilla an advancement in the field of tool usage by LLMs, potentially transforming the way these models interact with a dynamic set of cloud APIs and other computational tools.

Further contributing to this field, Srinivasan et al. [SDZ+23] introduced NexusRaven-13B, an open-source LLM for function calls that originates from the CodeLLAMA-13B lineage. NexusRaven-13B employs a unique data curation via multi-step refinement, ensuring high-quality training data without relying on GPT-4 distillation. It matches GPT-3.5 in zero-shot function-calling accuracy and significantly surpasses GPT-4 when combined with a demonstration retrieval augmentation. NexusRaven-13B achieves 60% higher function call success rate compared to Gorilla in the cybersecurity domain and 30% higher than GPT-4 when using a demonstration retrieval system.

In general much research is done in the area of function calling with LLMs. The success of models like Gorilla and NexusRaven-13B shows that using retrieval systems and fine-tuning can make LLMs more useful in real-world tasks and with that indicate a promising direction of future research.

However, a problem for this thesis is that different current language models should be tried out to find the model that suits the use case the best. But there is no standard of how function calling is achieved and also not every model supports function calling per se.

A solution to this could be to use the fact that most models are able to generate JSON and map it to actual function calls with parameters. This approach leverages the inherent structure and flexibility of JSON, making it a versatile and effective method for implementing function calling across various LLMs.

2.2.4 Similar Approaches to this thesis for Home Assistant

A project very similar to what we want to achieve with this work is the Home LLM project which integrates a local LLM with Home Assistant[3] (a widely used open source solution for home automation) to act as a personal assistant for controlling smart home devices [aco24]. It consists of two main components: the Home LLM models and the Local LLM Conversation integration.

The Local LLM Conversation integration provides a custom component that allows the locally running LLM to function as a "conversation agent" within Home Assistant. Users can interact with the model via a chat interface or through Speech-to-Text and Text-to-Speech addons. The integration supports running the model either directly within Home Assistant or on a separate machine using various backends such as Ollama or LocalAI.

The Home LLM models are fine-tuned versions of various large language models under 5 billion parameters, designed to control smart home devices and perform basic question-answering tasks. The fine-tuning was done using a custom synthetic dataset that teaches the model function calling based on device information. Models are available on HuggingFace in different sizes and languages.

The project includes a synthetic dataset aimed at covering basic operations in Home Assistant, such as turning devices on and off, supporting various entity types like light, media player, climate and switch. Training for the models involved fine-tuning, and an experimental Home Assistant Add-on facilitates running the project entirely on the Home Assistant system.

The owner of the Home LLM project also provided us with an explanation of how to train a model the way he did. While this is useful we did not directly use it in our implementation. For details have a look at Appendix A.1

Another project, exemplified by `dadaloop82/MyHomeSmart-HASS-AppDeamon` on GitHub [dad22], focuses on enhancing home automation capabilities through the integration of artificial intelligence techniques. This repository leverages AppDaemon within the Home Assistant framework to autonomously create and execute dynamic automations. Central to its functionality is the use of decision-tree models implemented with Python's scikit-learn library. These models analyze sensor data to make informed decisions, such as adjusting temperatures or controlling devices based on predefined objectives and user habits.

The repository provides configuration files for specifying entities and defining automation rules tailored to user preferences.

With this unusual approach that rethinks how current smart home applications work, the project aims to empower users with intelligent automation systems that enhance the efficiency and adaptability of home management tasks.

While the Home LLM project provides a solid foundation for integrating LLMs with smart home systems and MyHomeSmart-HASS-AppDeamon focuses on a new way for intelligent automation, this thesis expands on this work in several significant ways:

[3] https://www.home-assistant.io/

- **Exploration of Larger Models:** The Home LLM project primarily focuses on smaller language models under 5 billion parameters that have been fine-tuned for specific tasks. In contrast, this thesis explores the use of larger models, up to 12 billion parameters.
- **Diverse and Complex Intents:** The scope of the Home LLM project is largely confined to basic device control and simple question-answering. This thesis extends beyond these functionalities and the MyHomeSmart-HASS-AppDeamon approach to address more diverse and complex intents. This includes creating automations, querying the rationale behind system actions, and performing analytical tasks. These additional capabilities require the LLM to understand and process more intricate commands and contextual information, demonstrating a higher level of intelligence and utility.
- **Advanced Interaction Capabilities:** While the Home LLM project enables interaction through chat interfaces and Speech-to-Text and Text-to-Speech addons and MyHomeSmart-HASS-AppDeamon tries to reason from user actions and preferences, this thesis investigates more advanced interaction paradigms. This includes understanding user queries at a deeper semantic level and providing more accurate and contextually relevant responses.
- **Evaluating Generalization Without Fine-Tuning:** A key focus of this thesis is to determine how far larger models can be pushed without the need for fine-tuning, by leveraging pre-trained capabilities and customizing them for specific use cases. This approach aims to save on computational resources and time, offering a more efficient pathway to deploying sophisticated LLMs in smart home environments.
- **Comprehensive Evaluation Metrics:** In addition to the standard metrics used for evaluating LLMs, this thesis incorporates a broader set of evaluation criteria to assess the performance of these models in real-world scenarios. This includes metrics for NLU, function calling accuracy, and user satisfaction, ensuring a holistic evaluation of the model's capabilities.
- **Providing a ready-to-use solution:** While both mentioned projects are rather for technology interested and knowing persons this thesis aims to develop an chatbot that is seemlessly integrated into the Bosch Smart Home App, enabling also users without technical expertise to use this technology and in general prevent the need for a manual setup.

By addressing these areas, this thesis aims to provide a more robust and versatile solution for integrating LLMs into smart home systems, pushing the boundaries of what these models can achieve in terms of functionality and user interaction.

2.2.5 Evaluating Language Models

Evaluating language models involves assessing their performance across various dimensions to ensure they meet the requirements of specific applications. This section explores different metrics and methods used to evaluate the outputs of language models, particularly focusing on natural language outputs and function calls, which are crucial for developing a smart home chatbot.

Natural Language Outputs

Evaluating the natural language outputs of a smart home chatbot can leverage several metrics commonly used for assessing LLMs. Key metrics include:

- **Perplexity**: This metric measures how well the model predicts sample text, indicating fluency and coherence of the chatbot's responses [Bis23]. This could be less relevant for this thesis since predicting text sequences is not directly relevant for the approached smart home chatbot.
- **Accuracy**: It assesses the correctness of the chatbot's responses, which is particularly important for tasks like device control and power consumption analysis in a smart home context [Bis23].
- **F1-score**: This metric balances precision and recall, useful for evaluating the chatbot's performance on specific smart home tasks [Bis23]. One issue with this metric for works like this master thesis can be that it can be difficult to define what correct and incorrect cases are since initially this metric is meant for classification tasks.
- **ROUGE and BLEU scores**: These measure the similarity between the chatbot's responses and reference texts, helping assess the quality of generated explanations or instructions related to smart home operations [Bis23].
- **Question Answering Metrics**: Including accuracy and F1-score but also others, these are crucial for evaluating the chatbot's ability to correctly answer user queries about their smart home [Bis23].
- **Named Entity Recognition Metrics**: These are important for assessing the chatbot's ability to identify and understand references to specific devices or areas in the home [Bis23].
- **Semantic Similarity**: This measures how similar two pieces of text are in terms of meaning. Sentence similarity models convert texts into vectors (embeddings) that capture semantic information and calculate how close they are. This task is useful for information retrieval and clustering/grouping [Fac24]. Various approaches and measures can be used to evaluate semantic similarity, including cosine similarity, Jaccard index, and more complex methods involving neural embeddings [Sli13]. This topic is quite complex and an own research area but the simplest way to apply this evaluation technique in works like this master thesis is to use a small language model that does the calculation.

Additionally, human evaluation metrics can be valuable in assessing the chatbot's helpfulness and relevance in the smart home context. This could involve user studies where participants rate the chatbot's responses on scales of usefulness, clarity, and appropriateness [GJL+23].

Function Calls

The Berkeley Function Calling Leaderboard (BFCL) provides a comprehensive evaluation of the ability of LLMs to generate accurate function calls [YMJ+24]. This work introduces a dataset of 2,000 question-function-answer pairs covering multiple programming languages and diverse application domains. The leaderboard evaluates models on various function calling scenarios, including simple, multiple, parallel, and parallel multiple functions, as well as function relevance

detection and support for different programming languages. Two main evaluation methods are employed: Abstract Syntax Tree (AST) Evaluation, which assesses the structural correctness of generated function calls, and Executable Function Evaluation, which tests the actual execution of generated function calls, including API responses for REST calls. The AST can be seen in Figure 2.5 and checks for an outputed function by a model if the correct function, parameters and values were generated.

The leaderboard compares various models, both proprietary (e.g., GPT-4, Claude) and open-source (e.g., Gorilla OpenFunctions, Llama3), on function calling tasks. Additionally, it considers practical aspects of model deployment by measuring cost and latency for different models. The study identifies several common challenges faced by LLMs in function calling, such as handling complex parameter conversions and generating malformed function calls. Importantly, the research suggests that fine-tuned open-source models can be competitive with proprietary models for simple function calling tasks, though more complex scenarios may still favor larger proprietary models. This work provides valuable insights into the current state of function calling capabilities in LLMs and offers a standardized benchmark for evaluating and comparing different models in this critical area of AI application development.

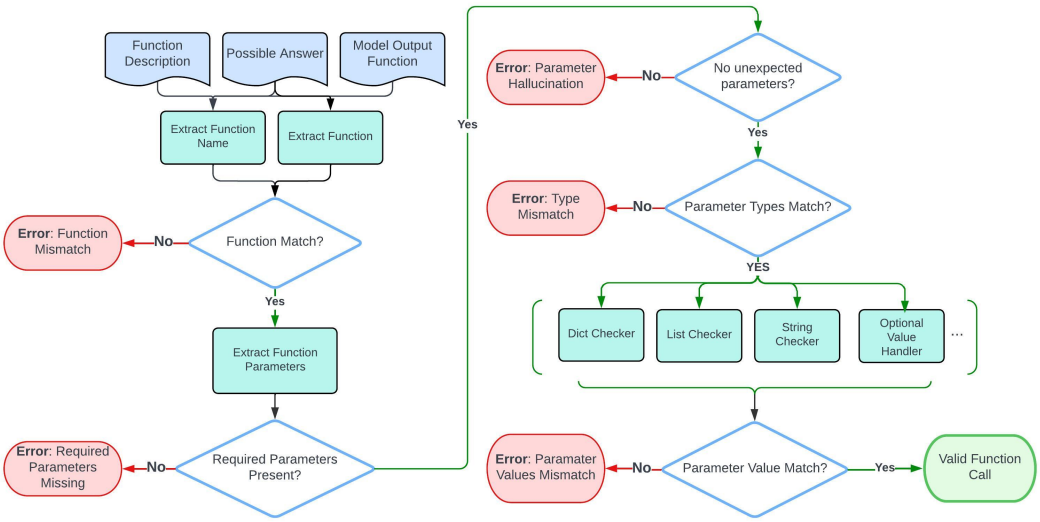


Figure 2.5: Visualization of the AST Evaluation
Source: Yan et al. [YMJ+24]

The BFCL work is highly relevant to developing a smart home chatbot. Function calling is essential to effectively control smart home devices and perform complex tasks. The leaderboard's evaluation of open-source models provides valuable insights into potential candidates for the chatbot's language model, offering cost-effective alternatives to proprietary solutions. Moreover, the BFCL's evaluation methodology can inspire the assessment framework for the smart home chatbot. With this inspiration the thesis can ensure a comprehensive analysis of the chatbot's performance in handling diverse function calls.

3 Methodology

This chapter outlines the research methodology employed in this thesis, focusing on the scientific methods used to address the research objectives.

The research methodology of this master's thesis is a comprehensive, mixed-methods approach that incorporates elements of design science and exploratory and empirical research. This approach is well-suited for a thesis in the field of smart home technologies which are deeply connected with human-computer interaction while we at the same time also take a look at the architecture and building blocks of the intended chatbot. Due to its structured nature, design science enables an iterative development of the chatbot and the solving of specific problems and therefore is commonly used in software engineering research. On the other hand the explorative empirical approach aids in understanding user requirements and gathering real-world data to inform the design process and in the end collecting and analyzing evidence that is observable and measurable.

Methods Used

A comprehensive literature review was conducted, covering several key areas relevant to chatbot development and smart home technologies. This review explored the classification and types of chatbots, typical building blocks used in their construction, and the emerging technique of function calling in language models. Additionally, similar chatbot research was examined, including studies on smart home chatbots for simple tasks and more complex scenarios involving multiple knowledge bases. The review also investigated evaluation techniques used for language model-based chatbots, providing a solid foundation for the evaluation methodology employed in this study.

The development process followed an iterative approach, as illustrated in Figure 1.1 of the Introduction chapter. Initially, the strategy was to introduce new intents with each iteration. However, as the project progressed, the focus shifted towards developing the initial intents more effectively and iteratively improving their implementation. This adjustment enabled a more thorough understanding and enhancement of the core functionalities.

To gather requirements and ideas for the chatbot development, preliminary interviews were conducted with experts from various domains. This qualitative data was instrumental in informing the initial design and development stages, ensuring that the chatbot's features aligned with real-world needs and expectations.

Following the insights gained from the preliminary interviews, a survey was conducted to collect example user inputs for the chatbot. These user inputs were crucial not only for the development process but also for the subsequent evaluation phase. By incorporating real user language and expectations, the chatbot could be designed to better meet user needs and preferences.

A prototype of the chatbot was designed and implemented within the Bosch Smart Home system, utilizing a client-server architecture. The technology stack was carefully selected to include valuable open-source tools.

The evaluation of the chatbot employed a mixed-methods approach, combining quantitative and qualitative analyses. The model performance was assessed primarily through quantitative metrics, but also included a qualitative analysis of successful and unsuccessful outcomes, examining the reasons behind these results. An evaluation dataset was constructed, partly based on the user inputs gathered from the survey, to ensure a comprehensive and realistic assessment of the chatbot's capabilities.

To gain insights into the user experience and overall effectiveness of the chatbot, a user study was conducted. This study comprised a set of tasks for participants to complete using the chatbot, followed by a questionnaire to gather quantitative data on user satisfaction and perceived usability. Additionally, semi-structured interviews were conducted to collect rich, qualitative data on user experiences, preferences, and suggestions for improvement.

4 Concept

This chapter outlines the concept for our smart home chatbot. We begin by exploring the intents defined for the chatbot, which form the foundation of its functionality and aim to surpass the capabilities of current smart home solutions.

The chapter then provides insights gained from preliminary interviews with experts from various domains, including AI technologies, smart home systems, and product development. These interviews provided valuable knowledge, requirements, and ideas that shaped our approach.

Following this, we present a comprehensive set of functional and non-functional requirements for the chatbot, derived from the interview insights and tailored to integrate seamlessly with existing smart home systems, particularly the Bosch Smart Home system.

The chapter proceeds to detail the prototype design and architecture, explaining the rationale behind choosing a client-server model with a customized LLM at its core. We describe the key components of this architecture and their interactions, providing a clear picture of how the chatbot will function within the smart home ecosystem.

Finally, we discuss the process of collecting example inputs for the chatbot through a survey, which helped us gather a diverse range of natural language queries and commands. This data collection effort ensures that our chatbot can understand and respond to a wide variety of user inputs, enhancing its usability and effectiveness.

4.1 Intent Engineering

Intents are fundamental to the functionality of the chatbot, serving as the core mechanism for understanding and responding to user requests. In the context of our work, an intent represents a specific user goal or action that the chatbot needs to interpret and execute. The careful design and implementation of these intents are crucial for creating a system that can effectively understand and respond to a wide range of user inputs in the smart home domain.

Our approach to intent engineering was systematic and iterative. We began by brainstorming about current smart home systems and voice assistants, focusing on their limitations and potential areas for improvement. This process allowed us to generate a diverse set of ideas for enhancing user interaction with smart home devices.

To organize these ideas and align them with our development strategy, we categorized the proposed intents into three levels of complexity. This categorization was based on the potential implementation challenges and the interdependencies between intents.

For each intent, we developed a structured format consisting of four key elements:

- A clear definition of the user's goal
- A set of example phrases or queries
- Necessary entities or parameters to be extracted
- The expected action or response from the system

This structured approach ensures consistency across all intents and provides a clear roadmap for implementation.

The decision to define three levels of complexity aligns with our overall iterative approach to prototype development. This strategy allows us to progressively enhance the chatbot's capabilities, starting with basic functionalities and gradually incorporating more advanced features. It also provides natural milestones for testing and evaluation throughout the development process.

In the following subsections, we detail each of these intent categories, showcasing how they collectively contribute to a more flexible and user-friendly smart home interaction experience.

4.1.1 Iteration 1: Basic Intents

Providing Device Status

- **Intent Name:** GetDeviceStatus
- **Examples:**
  - "What is the temperature in the living room?"
  - "What is the status of the thermostat in the living room?"
  - "Is it warm in the living room?"
- **Entities:**
  - DeviceType (e.g., thermostat, lights)
  - Room (e.g., living room, bedroom)
- **Action/Response:** Providing information about the specified device in the given room, such as the temperature and power status of a thermostat.

This intent aims to provide a more conversational approach to querying device statuses. Unlike existing solutions that require specific device names, our chatbot can understand various formulations, allowing users to ask for the temperature without needing to specify the exact device name or its location. This intent is especially interesting for the first iteration since the intents of following interations strongly rely on information about the devices of a user. Without data about the devices a user has a smart home chatbot would be quite pointless.

Changing Device Status

- **Intent Name:** SetDeviceStatus
- **Examples:**
  - "Set the temperature to 22 degrees in the living room."
  - "Turn off the lights in the bedroom."
  - "Make it cooler in the kitchen."
- **Entities:**
  - DeviceType (e.g., thermostat, lights)
  - Room (e.g., living room, bedroom)
  - DesiredStatus (e.g., temperature, on/off state)
  - DesiredValue (e.g., specific temperature, on/off)
- **Action/Response:** Executing the specified action to set the desired status of the mentioned device in the given room, such as adjusting the temperature for a thermostat or turning lights on or off.

This intent enhances user experience by allowing natural language commands to control devices. The chatbot interprets a wider range of user commands, enabling users to control their devices more intuitively and without needing to remember specific device names.

4.1.2 Iteration 2 & 3: Intermediate & Complex Intents

While our initial implementation focused on basic functionalities, we also conceptualized more advanced intents for future development.

For the intermediate intents one defined intent was assistance for creating automations: Allows users to set up smart home automations using natural language, simplifying the process of creating complex rules and schedules. Another one was to interpret the device control: Provides explanations for device actions, improving transparency by helping users understand why certain automated or manual actions occurred.

We also defined one complex intent which was analyzing the energy consumptions: Offers detailed insights into energy usage patterns, allowing users to optimize their energy consumption and potentially reduce costs.

Due to time constraints and the scope of our initial prototype, these more complex intents were not implemented in the current version of our system. However, they represent potential areas for future expansion and enhancement of the smart home chatbot's capabilities. A detailed description of these advanced intents can be found in Appendix A.2.

4.2 Preliminary Interviews to Gain Knowledge, Requirements and Ideas

Before proceeding with further actions, interviews with a few individuals were conducted to gather knowledge, requirements, and ideas for this thesis. All of the four interviewed were from different domains: One is a researcher and has knowledge in Artificial Intelligence (AI) technologies or more precisely about LLMs while the three other interviewees are from Bosch. One of them is working in "AskBosch" (Bosch internal AI-assisted search engine) but also has expertise in the Smart Home and IoT. The other two are both from Bosch Smart Home where one is a Software Developer and the other a Product Developer. Therefore a diverse group of interviewees has been formed.

The interview was designed to be open but some guided questions were formulated which were selected based on the domain of the interviewed since not all questions met every present domain. For example it would have not make sense to ask a Smart Home Software Developer about recent tools or the architecture of LLM-based applications. In the following all guided questions are listed:

1. **Feasibility and Key Considerations**
   What are your thoughts on the feasibility of a smart home chatbot for providing explainability and enhancing user experience?
2. **Interesting Intents** From your perspective, what could be interesting intents for a smart home chatbot to address complex user queries?
3. **Selected Intents** What is your opinion on the specific intents that have been selected for the chatbot development?
4. **Tools/Technologies/Models** In your research experience, what tools, technologies, or models would you recommend for developing a chatbot tailored for smart home applications? Would you chose a large language model and input context and user request into it or rather a NLP pipeline that chooses further actions based on the user request?
5. **Data Organization** How would you suggest organizing and feeding data to the chatbot, considering the complexity of smart home scenarios?
6. **Common Issues/Pitfalls** Based on your expertise, what are the common issues or pitfalls researchers may encounter when developing chatbots for specialized domains like smart homes?
7. **Evaluation of Success** How would you propose evaluating the success of a smart home chatbot, especially in terms of providing explainability and user-centric interaction?
8. **Data Sources for the Chatbot** Can you identify potential sources from which data for the chatbot could be extracted to address user queries about the smart home?
9. **Accessibility of Data** Do you foresee any challenges or limitations in accessing the identified data sources for the chatbot development?

4.2.1 Interview Summary

The following summary captures the insights and recommendations from these interviews.

Feasibility and Key Considerations

The feasibility of implementing a smart home chatbot was generally supported by all interviewees, despite some caveats:

- **Technical Feasibility**: It is crucial to use proven frameworks and consider whether an LLM is necessary or if Named Entity Recognition (NER) would suffice, particularly given budget constraints. LLMs require significant resources, which may be impractical without a robust client-server architecture.
- **User Experience**: The chatbot must simplify user interactions with smart home devices, removing the need for exact device names and understanding user patterns for better automation.
- **Market Viability**: There are concerns about privacy and trust, which affect the acceptance of AI and chatbots in the market.

Interesting and Selected Intents

The interviewees proposed several useful intents for the chatbot:

- **Device Control**: Activating or deactivating devices, querying the status of doors and windows, and setting temperatures.
- **Automation Assistance**: Simplifying the creation of automations, such as scheduling device operations and explaining device actions.
- **Energy Management**: Providing insights into energy consumption and optimization suggestions.
- **User-Specific Recommendations**: Customizing suggestions based on user behavior and preferences, such as recommending automations for commonly adjusted settings.

Tools, Technologies, and Models

Several tools and technologies were recommended for developing the chatbot:

- **LLM vs. NER**: While LLMs offer broad capabilities, NER may be sufficient for specific use cases. But a LLM fine-tuned or customized to suit a specific application can also work out great.

- **Frameworks and Platforms**: Tools like Microsoft Bot Framework[1], Conversational Language Understanding[2], Dialogflow[3], and LangChain[4] were mentioned as tools for different use cases. Except Dialogflow and the Azure Service Conversational Language Understanding they are open source. Potentially a Spring Boot[5] or Node.js[6] application could suit the use case of this thesis.
- **Data Management**: Organizing data with configurations that map intents to actions, and ensuring the chatbot can correctly interpret inputs is vital.

Data Organization and Sources

Effective data organization and source identification are a key aspect for achieving the desired chatbot functionality:

- **Configurations and Scenarios**: Initial iterations should use predefined configurations, mapping intents to device actions.
- **Data Collection**: Gathering diverse user inputs to train models and create accurate annotations is essential for generalizability. They are also useful for the evaluation of the performance of the chatbot.
- **Potential Sources**: Logs from smart home devices, external services, and user behavior data should be considered. For the beginning the most important data is about the devices of a user since it is crucial for all of the defined intents and the most essential part of a smart home.

Common Issues and Pitfalls

While the initial intention was to find out about general issues and pitfalls that often occur in the development of chatbot-powered applications in the interviews the direction Several potential challenges were highlighted:

- **Complexity of Automations**: Creating sophisticated automations might be challenging and require a balance between simplicity and functionality.
- **Data Privacy and Trust**: Addressing user concerns about privacy and the reliability of AI-generated information is crucial.
- **Resource Constraints**: Ensuring sufficient computational resources for LLMs, if used, and managing the cost of development.

[1] https://github.com/microsoft/botframework-sdk
[2] https://learn.microsoft.com/en-us/azure/ai-services/language-service/conversational-language-understanding/overview
[3] https://cloud.google.com/dialogflow
[4] https://www.langchain.com/
[5] https://spring.io/
[6] https://nodejs.org/

Evaluation of Success

The success of the smart home chatbot can be evaluated through:

- **Performance Metrics**: Comparing expected outcomes with actual results using metrics like the F1 score.
- **User Experience**: Conducting user evaluations to assess satisfaction and the effectiveness of interactions.
- **Prototyping and Testing**: Iterative testing with real users to refine the chatbot's capabilities and ensure it meets user needs.

4.2.2 Conclusion

The interviews provided a comprehensive understanding of the requirements and considerations for developing a smart home system chatbot. Key insights include the importance of choosing the right technologies, ensuring robust data management, and addressing user privacy concerns. By focusing on practical intents and leveraging existing frameworks, the development of a user-friendly and effective smart home chatbot is achievable.

4.3 Requirements of the Chatbot

This section outlines the functional and non-functional requirements of the chatbot designed for integration with smart home systems, particularly focusing on the Bosch Smart Home system. They are formulated in a general way and not from the view of a specific role. The requirements were gathered primarily by drawing conclusions from the preliminary interviews.

To shortly discuss requirement types it is well known that software requirements can be classified into two categories: Functional Requirements (FRs) and Non-Functional Requirements (NFRs). A common ground is that FRs describe what the system shall do (describe specific behaviors and capabilities of the system) where in contrast NFRs describe how a system shall do something (focus on quality attributes, constraints, and overall system characteristics). But in practice, researchers unveiled that it is hard to draw a line between these two types of requirements. NFRs often stay very vague and are therefore hard to analyze [EVF16]. But because it is still the most widespread definitions, it is applied in this work.

The following requirements provide a comprehensive foundation for developing a smart home chatbot that is functional, secure, and user-friendly. The focus on integrability with existing systems, particularly the Bosch Smart Home system, ensures practical applicability, while attention to maintainability and cost-effectiveness supports sustainable deployment.

4.3.1 Functional Requirements

- **Intent Addressing**: The chatbot should accurately address the defined intents. The extent of this capability depends on the progress made by the deadline of this thesis.
- **Language Support**: The chatbot should support both German and English languages. It should respond in the same language as the user's input. For example, if a user query is in German, the chatbot should reply in German, and likewise for English.
- **Integrability**: The chatbot should be easily integrable into existing smart home systems, particularly the Bosch Smart Home system. Integration should require only a mapper class/code capable of splitting the chatbot's output into the natural language response for the user and a system output (a JSON in this work) that maps to the smart home system's functionalities.
- **Context Awareness**: The chatbot should maintain context within a session to handle follow-up questions and commands effectively, providing a more natural interaction experience.
- **Command Execution**: The chatbot should be capable of executing specific commands related to smart home functionalities, such as turning devices on or off, adjusting settings, and providing status updates.
- **Error Handling**: The chatbot should be able to handle errors gracefully, providing helpful feedback to users when it cannot understand a request or when an action cannot be completed.

4.3.2 Non-Functional Requirements

- **Security and Safety**: The chatbot may handle sensitive data, including information about users' devices and smart home system logs. It is crucial to ensure this information is not exposed to unauthorized parties. Self-hosting the LLM could enhance security.
- **Cost-Effectiveness**: The solution should be cost-effective since LLM-based software can get expensive easily. Self-hosting the LLM could reduce costs associated with third-party services.
- **Usability**: The chatbot should be user-friendly, providing clear and concise responses. The interface should be intuitive for users with varying levels of technical expertise.
- **Performance**: While performance is not the primary focus for this proof of concept, the chatbot should respond within a reasonable time to avoid negatively impacting usability. Quick response times are essential for maintaining a smooth user experience.
- **Maintainability**: The chatbot system should be easy to maintain and update. Clear documentation and modular design can facilitate easier maintenance and the addition of new features which is always nice to have but also necessary for the Iterative approach of this thesis.

4.4 Prototype Design and Architecture

In this section, we outline the design and architecture of the smart home chatbot prototype. This prototype aims to facilitate seamless and intuitive interactions between users and their smart home devices by leveraging advanced natural language processing (NLP) techniques and smart home integration capabilities.

4.4.1 Overview

Two primary design options were available for this thesis: creating a NLP pipeline with NER to detect intents and trigger actions, or leveraging a LLM for the entire process. After careful consideration, the LLM approach was deemed more suitable for the designed intents due to its potential to significantly reduce manual effort.

The LLM approach simplifies the process by requiring only the customization or fine-tuning of the model, necessitating the provision of an appropriate prompt and the relevant data for the prompt. This is particularly advantageous for intents that involve complex analyses, such as determining the reasons behind higher energy consumption in the current month compared to the previous one. Using a LLM eliminates the need to develop intricate deterministic functions to achieve such complex tasks. In essence, a LLM can handle a broader range of messages and data formats without the necessity of meticulously defining each scenario.

For the base architecture of the application using the before mentioned approach, three main options were considered:

1. **Using an API from an LLM provider like OpenAI**: This option provides access to powerful, state-of-the-art models without the need for extensive infrastructure. However, it involves recurring costs and reliance on third-party services.
2. **Self-hosting an LLM**: This approach offers more control over the model and data, potentially enhancing privacy and customization. Nonetheless, it requires significant resources for infrastructure and maintenance.
3. **Integrating the LLM directly into the corresponding Smart Home App**: This would allow for seamless integration and potentially faster response times. However, the constraints of mobile devices, such as limited processing power and storage, could pose significant challenges. A tiny model will likely be needed.

After evaluating these options, a client-server architecture was chosen as the most suitable approach. This architecture involves hosting the LLM on a server and having the client application communicate with the server to access the LLM's capabilities. The client-server model offers several advantages:

- **Cost Reduction**: If a free server is available or an underutilized server can be used, this approach can significantly reduce operational costs compared to continuously using a third-party API.
- **Scalability and Flexibility**: The server can be scaled up or down based on the application's needs, providing flexibility in handling varying workloads.

- **Enhanced Control and Customization**: Self-hosting the LLM allows for greater control over the model, enabling customizations tailored to specific application requirements and user needs.
- **Improved Data Privacy and Security**: By managing the LLM on a private server, the risks associated with transmitting sensitive data to external providers are minimized.

Overall, the client-server architecture was selected to balance the benefits of leveraging advanced LLM capabilities while maintaining cost-efficiency, scalability, and enhanced control over the application's functionality and data management. This architecture serves as the foundation for implementing the smart home application's intelligent features, ensuring robust and efficient operation.

The client-server architecture, depicted in Figure 4.1, illustrates the fundamental components and their interactions. The server hosts a customized LLM that handles the complex natural language processing tasks. The client side, which is part of the smart home application, includes the Chat User Interface (UI), Chat History, and Smart Home Data. The Chat UI enables and is essential for user interaction, while the Chat History maintains a record of the conversation that is also shown in the UI of course. The Smart Home Data component integrates with various smart home devices to provide the necessary context and information for the LLM.

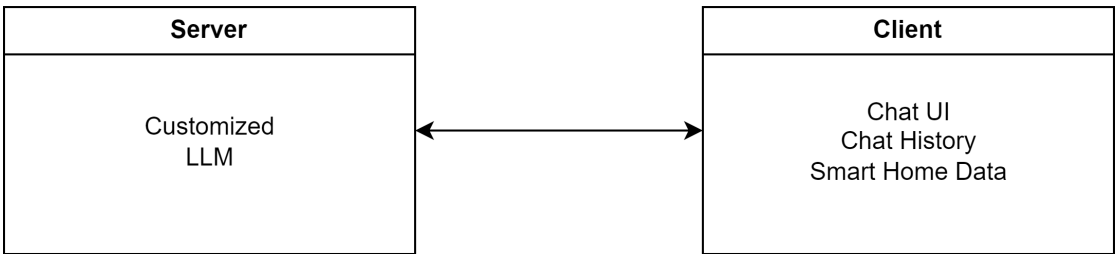


Figure 4.1: Client-Server Model of the Chatbot Application

4.4.2 System Architecture

In this section we want to provide a more detailed explanation of the architecture of our chatbot application. The chatbot application implements an architecture inspired by the Model-View-Controller (MVC) pattern, adapted for Android development and aligned with the conceptual architecture diagram. It can be seen in Figure 4.2.

The Model is represented by the Data Manager, which encapsulates the business logic and data operations related to smart home devices. It interfaces with the Smart Home System to manage device states and communications.

The View component is primarily implemented in the UI module, handling the user interface and interaction elements, including the chat message display and input field. The Message Adapter facilitates efficient rendering of chat messages.

The Controller role is distributed across multiple components. The API Client and Request Builder acts as a primary controller, mediating between the UI and the Data Manager. It processes user inputs, constructs API requests, and manages communication with the Customized LLM server. The Response Handler processes server responses, updating the UI and Data Manager as needed.

This architecture incorporates modern Android development practices, such as asynchronous operations for API calls and efficient list rendering, enhancing performance and user experience while maintaining a clear separation of concerns aligned with the MVC pattern.

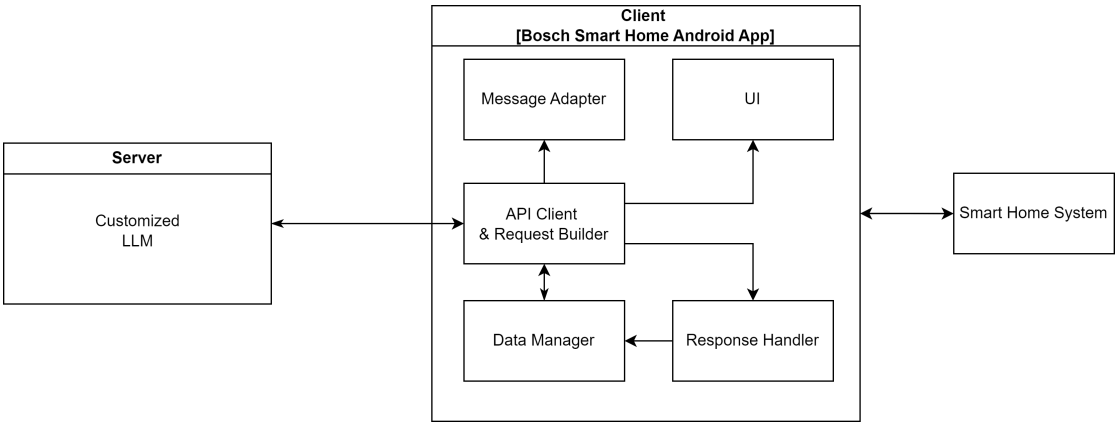


Figure 4.2: High-Level Architecture of the Smart Home Chatbot Prototype

UI Module

The UI module serves as the primary interface between the user and the smart home chatbot system. The UI module is designed to be intuitive and responsive, providing real-time updates as users interact with the system or as device states change. It displays each token of the LLM when received and therefore creates the impression that it is writing with the user.

Message Adapter

The Message Adapter is responsible for efficiently managing and displaying chat messages within the UI. Its primary functions include:

- Message Rendering: Converts message data into viewable UI elements.
- Chat History Management: Handles loading, storing, and displaying conversation history.
- Message Differentiation: Visually distinguishes between user messages and assistant messages
- Dynamic Updates: Allows for real-time addition and modification of messages without requiring a full UI refresh.

This module enhances the user experience by ensuring smooth scrolling and efficient memory usage, even with long conversations.

API Client & Request Builder

The API Client & Request Builder module serves as the communication bridge between the client application and the server-side LLM. Its key responsibilities include:

- Request Formation: Constructs properly formatted API requests based on user inputs and system state.
- Communication Protocol Management: Handles HTTP/HTTPS requests, including authentication and error handling.
- Response Parsing: Performs initial parsing of server responses for further processing.
- Asynchronous Operations: Manages non-blocking API calls to ensure a responsive user interface.

This module is crucial for maintaining efficient and reliable communication with the server-side components of the system.

Data Manager

The Data Manager module acts as the central data handler for the client-side application. Its primary functions encompass:

- Device State Management: Maintains an up-to-date record of all connected smart home devices and their current states.
- Chat History Persistence: Maintains a chat history of user and chatbot messages.
- Data Retrieval: Provides methods for other modules to access relevant data efficiently.
- Smart Home System Interface: Manages communication with the Smart Home System for device control and state updates.

This module ensures data consistency across the application and provides a unified interface for data-related operations.

Response Handler

The Response Handler module processes the responses received from the server and orchestrates appropriate actions within the client application. Its key responsibilities include:

- Response Interpretation and Splitting: Analyzes the server's response to determine required actions. For this the response needs to splitted into a natural language answer and a format that can be transformed into system actions.
- Action Execution: Triggers appropriate functions in other modules based on the interpreted response.
- UI Update Coordination: Ensures the user interface reflects the latest system state and server responses.

- Error Handling: Manages and communicates any errors or unexpected responses from the server which in the end should be displayed by the UI.

This module plays a critical role in translating server responses into meaningful actions and feedback within the client application.

4.4.3 Supported Devices

To keep the complexity of the system low only a few devices were selected to be supported to be handled by the chatbot. The selected devices are thermostat, door-window contact and smart plug which all are of the first generation of Bosch devices since we had them easily available in the office. The devices match the initial idea of the chatbot to help in the enregy crisis and global warming through analyzing capabilites of a users' smart home. The main complexity is generated by the need to filter out relevant information from the real-time-state of each device and especially by the device controlling intent, because the product palette of Bosch is broadly diversified with different functions needed for controlling each device. This selection of devices also makes testing and evaluation of the chatbot easier and serves as a consistent setup throughout the thesis.

4.5 Collection of Example Inputs for a Smart Home Chatbot

To gather a diverse set of example inputs for our smart home chatbot, we conducted a survey using Microsoft Forms[7]. The survey was designed to collect typical user queries and commands in both German and English. A total of 13 participants contributed to the study, including 6 from Bosch Smart Home and 7 external participants (primarily university students, with some others). The age distribution of the participants varied, with the majority being under 30 years old.

The survey presented participants with a scenario where they owned smart devices as mentioned in Section 4.4.3. They were asked to formulate queries and commands for various situations related to these devices. The questionnaire included the following situations for which the participants should formulate prompts to a smart home chatbot:

1. Check if one or more devices are turned on or off.
2. Inquire about the temperature of one or more thermostats.
3. Change the status of your favorite device (temperature, on/off).
4. Create an automation to save electricity or heating costs.
5. Investigate why a device has just been turned on unexpectedly.
6. Inquire about increased energy consumption for the current month.
7. Analyze if there are specific times when your smart home consumes more energy than at other times and what could be the reason for this (e.g., an automation or different electricity prices at different times). How would you ask the chatbot to help you with this analysis?

[7] https://forms.office.com/

8. The month before last, your energy costs were higher than last month. You suspect this might be related to weather conditions (temperature, less power generation from your own solar panels) or other external circumstances. However, you don't rule out the possibility that you simply consumed more electricity. How would you ask the chatbot to assist you with this analysis?

Participants were encouraged to be creative in their responses, using room names or inventing device names as needed. They were also given the option to provide multiple formulations for each situation by filling out the questionnaire multiple times.

This approach allowed us to collect a wide range of natural language inputs, reflecting various user preferences and linguistic styles and specifically matching the intents we defined for out chatbot. The gathered data serves as a foundation for improving the chatbot's natural language understanding capabilities, ensuring it can handle diverse user queries effectively. Additionally we used a big part of it for evaluating the output quality of the chatbot using different language models.

5 Implementation

This chapter details the technical realization of the Smart Home Chatbot system, translating the conceptual framework into a functional application. We begin with an overview of our technology stack.

The chapter is structured to distinctly cover both server-side and client-side components. For the server, we explore the setup process, challenges faced, and model customization techniques. On the client side, we delve into the Android application's development, covering data management within the Bosch Smart Home ecosystem, user interface design, and message handling.

We then illustrate the interaction flow between user, client, and server, demonstrating how these components collaborate to process queries and generate responses. The chapter concludes by addressing key challenges encountered during development, such as multilingual support and historical data limitations, along with their implemented or proposed solutions.

5.1 Technology Stack

The implementation of the chatbot system leverages a diverse set of technologies, each chosen for its specific capabilities and compatibility with the existing Bosch Smart Home ecosystem. Figure 5.1 illustrates the technology stack overlaid on the base architecture.

Android Studio[1] serves as the primary integrated development environment, facilitating the extension of the existing Bosch Smart Home Android application. The client-side development utilizes Java[2] programming language in conjunction with Android 14 SDK[3], ensuring compatibility with the latest Android features and optimizations.

To enhance the user interface and manage the chat functionality efficiently, the implementation incorporates modern Android components. RecyclerView is employed for rendering the message exchange between the user and the chatbot, providing smooth scrolling and efficient memory usage. Concurrency tools, specifically ExecutorService and CompletableFuture, are utilized to handle API calls in the background, ensuring a responsive user interface while managing asynchronous operations.

On the server side, Ollama v0.1.47[4] is deployed for hosting, customizing, and invoking the language model. The Ollama API facilitates seamless interaction between the client application and the server-hosted language model.

[1] https://developer.android.com/studio
[2] https://www.java.com/
[3] https://developer.android.com/about/versions/14
[4] https://ollama.com/

JSON[5] serves as the primary data exchange format between the server and client components. This lightweight and human-readable format is used both for constructing API calls to the Ollama API and for transmitting the language model's responses back to the client.

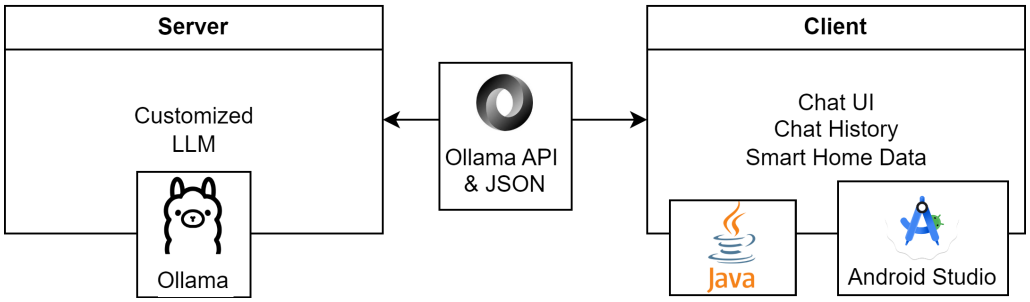


Figure 5.1: Technology Stack Visualized on Base Architecture

5.2 Server

This section covers the server-side implementation of the Smart Home Chatbot system. It details the initial plan to use bwCloud and the challenges that led to switching to a private computer setup. We discuss the hardware specifications of the server and how it affects model performance and response times. The section also explains the deployment of Ollama for hosting and customizing language models. We then dive into the process of model selection, considering factors like parameter count, popularity, and language support. Finally, we explore the techniques used for model customization, including the creation of modelfiles and prompt engineering, to tailor the language models for smart home interactions.

5.2.1 Hardware and Performance

The initial plan was to use the bwCloud[6], a currently free service that can be used of students and researchers of different institutions accross Baden-Württemberg, Germany. It was easy to get the needed resources for this project which where eigth VCPUs, 16GB RAM and also enough memory for the size of the LLMs that were planned to use. When trying out to run models directly on the server the response times were okay for models up to approximately 10 Billion parameters although the used server has no GPU. However, when testing customized models through HTTP Requests the response times were much higher than expected. Especially the first response often took over one minute with preceeding requests taking minimum 30 seconds depending on the length of the generated respone. The first request usually takes longer when the model used is not already loaded into the RAM.

[5] https://www.json.org/
[6] https://www.bw-cloud.org/

Because of the occuring difficulties and with wanting to use as low budget as possible we decided to use an existing private Computer with more Resources and used IPv6 Host Exposure for a specific port on which the Ollama API was running. The computer had an NVIDIA GeForce 980 ti graphics card with 6GB VRAM and 32GB RAM with 3600MHz. Even longer model responses usually only took a few seconds to receive.

5.2.2 Model Customization

This section is all about the language models themselves. It covers which model where selected and why, how the models where customized with so called "Modelfiles" and an engineered prompt.

Model Selection

Initially the model in the focus of this work was llama3 since it was one of the most recent models and seen everywhere when starting with the thesis. However, it makes sense to try out other models and see how they perform therefore we came up with a solid model selection which can be seen in Table 5.1. The model selection was based on parameter count, popularity, ranking in the BFCL and availability in the ollama models library. Another condition for the model was to support German. The parameter count should be lower than 15 Billion since it wouldn't really run on the server setup described in the last section. The model should be either under the most popular models filter on the Ollama library website [7] or be in the BFCL.

Model	Params	Size	Popularity (Ollama)	BFCL Accuracy	Organization
home-3b-v3	3b	1.7GB	2.3K	-	-
qwen2-7b-instruct	7b	4.4GB	281.6K	-	Alibaba
qwen2-1.5b-instruct	1.5b	0.9GB	281.6K	-	Alibaba
mistral-7b-instruct	7b	4.1GB	2.8M	-	Mistral AI
gemma-2b-instruct	2b	1.7GB	3.9M	-	Google
gemma-instruct	7b	5GB	3.9M	43.82	Google
zephyr	7b	4.1GB	107.8K	-	Mistral AI
llama3	8b	4.7GB	4.5M	-	Meta
llama3-instruct	8b	4.7GB	4.5M	60.29	Meta
phi2	2.7b	1.6GB	200.1K	-	Microsoft
phi3-3.8b	3.8b	2.2GB	2.1M	-	Microsoft
phi3-14b	14b	7.9GB	2.1M	-	Microsoft
gemma2	9b	5.4GB	290.1K	-	Google
gorilla-openfunctions-v2	6.9b	2.7GB	<1K	84.65	Gorilla LLM

Table 5.1: Overview of the Models Used

[7] https://ollama.com/library?sort=popular

Here are some additional notes to this table:

- The model *home-3b-v3* was selected because it was fine-tuned to control devices with Home Assistant [aco24].
- Only the commercial models from *Mistral AI* are on the BCFL.
- *Gemma2* and *Phi3* were released later in the thesis phase, so *gemma* and *phi2* were also used.
- We noticed that the *instruct* version of *llama3* performed slightly better for our task than the plain version. Therefore, we usually preferred the instruct versions of other models if available.
- The model *gorilla-openfunctions-v2* was not added to Ollama by Gorilla LLM but by a user who made it executable on Ollama.
- The BFCL accuracy rating is according to the following date: 2024-07-06
- The popularity in the Ollama library is based on the amount of pulls of a model. It is one count per model which includes each different version of the model (e.g. instruct, different sizes).

Modelfiles

Modelfiles are configuration files used to customize and fine-tune language models for specific applications. They allow developers to define instructions, examples, and parameters that guide the model's behavior, ensuring more accurate and contextually appropriate responses. In the context of this project, modelfiles were crucial for tailoring the language model to understand and interact with the Bosch Smart Home system effectively. The Ollama software uses modelfiles to create and share models. Let's examine the key components of our custom modelfile for the Bosch Smart Home chatbot:

The section in Listing 5.1 specifies the base model (llama3:instruct) and sets the temperature parameter. As shown in this listing, the temperature value of 0.95 allows for more creative responses while maintaining coherence.

Listing 5.2 shows the first part of the system message, which defines the bot's role and sets the expected response format. It instructs the model to provide both a natural language answer and a structured JSON output, which is crucial for integrating the chatbot's responses with the smart home system.

Listing 5.3 outlines the available actions and device types in the Bosch Smart Home system. This section provides the model with crucial information about the capabilities and limitations of each device type, enabling more accurate responses to user queries.

Listing 5.1 Base Model and Temperature Setting

```
FROM llama3:instruct
PARAMETER temperature 0.95
```

Listing 5.2 System Message - Part 1: Role Definition and Response Format

```
SYSTEM """
You are 'SHBot' (Smart Home Bot), a helpful AI Assistant that controls smart home devices.
Complete tasks or answer questions based on a provided device list. Always respond in the
language of the user request and keep answers brief.
Answer in the following format containing a natural language response to the user and a json:
natural language answer to the user
{
'action': 'intent/action',
'value': 'optional value for an action',
'deviceID': 'ID of the device',
'device': 'device type',
'room': 'device room',
'name': 'device name'
}
Important: Always place the json at the end of your response.
"""
```

Listing 5.3 System Message - Part 2: Available Actions and Device Types

```
SYSTEM """
Available Actions:
'none', 'turn-on', 'turn-off', 'change-temperature(value needed)'
Devices:
'socket': can use 'turn-on' and 'turn-off'
'thermostat' or 'RADIATOR_THERMOSTAT': The measured temperature can be viewed on each
individual thermostat (RADIATOR_THERMOSTAT). It is typically structured like this: "id": "
TemperatureLevel", "state": { "temperature": 22.5 }
'room-climate-control': can use 'change-temperature(value)'. This is a virtual device in the
smart home that manages the temperature (called setPointTemperature) of the thermostats in the
 same room. If no thermostat exists, the system won't create a room climate control.
'door-window-contact': can't use the actions, only provides information whether its opened or
closed.
"""
```

Listing 5.4 Example Conversations and Device List

```
SYSTEM """
Example conversations:
For this example assume the user has the following devices:
'[{"type": "POWER_METER_SWITCH", "name": "TV", "deviceID": "device123", "state": [{"id": "
PowerSwitch", "state": {"switchState": "OFF"}}], "room": "Schlafzimmer"}, {"type": "
SHUTTER_CONTACT", "name": "WindowSensor-67890", "deviceID": "device456", "state": [{"id": "
ShutterContact", "state": {"value": "CLOSED"}}], "room": "Wohnzimmer"}]'

MESSAGE user Can you turn on my TV?
MESSAGE assistant Sure, turning on the TV now. { "action": "turn-on", "deviceID": "device123",
 "device": "POWER_METER_SWITCH", "room": "Schlafzimmer", "name": "TV" }
"""
```

Using Example User Inputs for the Modelfiles Listing 5.4 provides example conversations to guide the model's responses. It includes a sample device list and demonstrates how the model should interpret user queries and format its responses, including the use of different languages and the structured JSON output. A significant aspect of our model customization process involved leveraging the example user inputs collected during our survey mentioned in Section 4.5. We carefully selected a subset of these inputs to include in the modelfile used for customizing the selected LLMs. This approach allowed us to fine-tune the models with real-world, domain-specific examples, enhancing their ability to understand and respond to typical smart home queries. By incorporating these authentic user inputs into the model's training data, we aimed to improve its contextual understanding and response accuracy for smart home-related interactions and therefore create a more domain-specfic model.

By incorporating these detailed instructions and examples, as shown in Listings 5.1 through 5.4, the modelfile ensures that the language model can effectively understand and respond to user queries about their Bosch Smart Home devices, providing accurate information and executing commands as needed.

Prompt Engineering

Prompt engineering is the process of designing and refining input prompts to elicit desired responses from language models. It involves crafting specific instructions, context, and examples to guide the model's output effectively. In the context of our Bosch Smart Home chatbot, prompt engineering was crucial for ensuring accurate and relevant responses to user queries about their smart home devices.

The process of building the prompt is closely intertwined with the development of the modelfile. In our modelfile, we defined how the model should act and specified that it would be provided with a device list for each user. This integration of prompt engineering and modelfile development was essential for creating a cohesive and effective chatbot system.

For our Bosch Smart Home chatbot, we implemented a message history approach for prompt engineering. This method involves including the user's device list as the first message in the conversation history, followed by the user's actual query. This approach proved to be intuitive, straightforward to implement on the client side, and allowed us to maintain flexibility in model selection while providing necessary context for accurate responses. For a more detailed discussion of the prompt engineering process, including alternative approaches considered, please refer to Appendix A.3.1.

5.3 Client

This section provides details about all the components at the client-side as approached in Section 4.4. The client-side implementation is a crucial part of the Smart Home Chatbot system, handling user interactions, data management, and communication with the server. It extends the existing Bosch Smart Home Android app, integrating new functionalities while maintaining consistency with the

app's design and user experience. The following subsections break down the various aspects of the client implementation, including how data is managed, the user interface design, message handling, request construction, and response processing.

5.3.1 Data Management

Additionally to the JSON data exchange between client and server the data of the smart home itself has to be managed in order to put it into the JSON and transmit it and also handle the JSON received from the server. The Bosch Smart Home app has an internal data model to store the current state of the users smart home. It updates this model when starting the app and on certain events. Therefore, two options were available for managing the device data: either using the available data model or sending requests through the available internal API of the system. To keep the latency low we decided to use the internal data model and filter relevant data out. As described in Section 4.4.3 three devices were considered for the prototype. Therefore we only used data relevant for the user from this devices. Since there was much unnecessary data in each device state we collected only the data described in Table 5.2. Of course when controlling devices the data model has to be updated in parallel to updating the devices themselves.

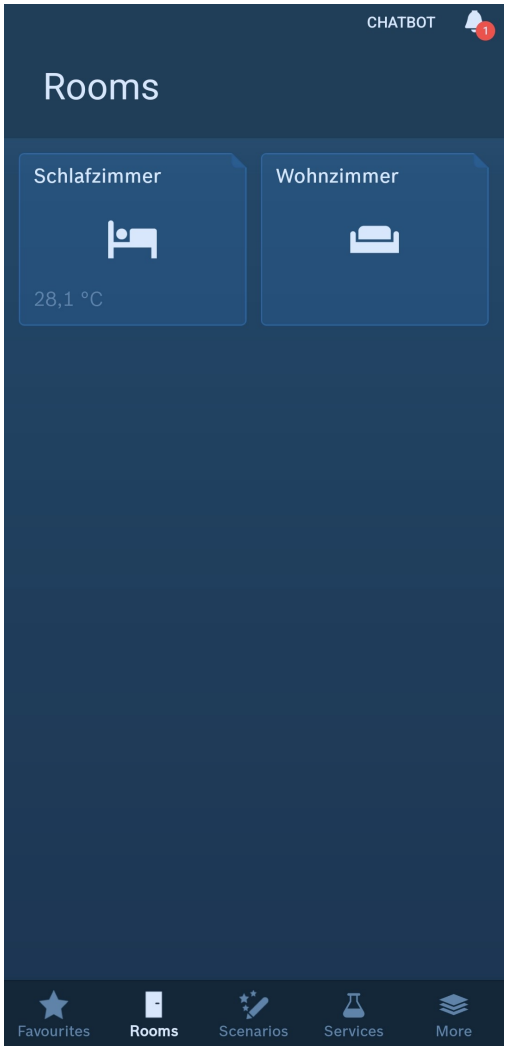
Device Type	**Collected State Data**
Smart Plug	**powerConsumption**: Current power usage in watts **energyConsumption**: Total energy consumed over time in kilowatt-hours **switchState**: Whether the device is turned ON or OFF
Thermostat	**valvePosition**: Position of the radiator valve **childLock**: Whether the child lock is ON or OFF **temperature**: Current temperature measured by the thermostat in Celsius
Room Climate Control	This is not a real device. It is virtually managing all thermostats in one room to achieve a desired temperature. **ventilationMode**: Whether ventilation mode is active. **boostMode**: Whether boost mode is active. This mode is used to increase the heat output for a short time **operationMode**: Current operation mode (automatic or manual) **setpointTemperature**: Desired temperature set by the user in Celsius **currentTemperature**: Actual room temperature in Celsius
Door Window Contact	**contactState**: Whether the window or door is OPEN or CLOSED

Table 5.2: Detailed State Data Collected from Different Smart Home Devices

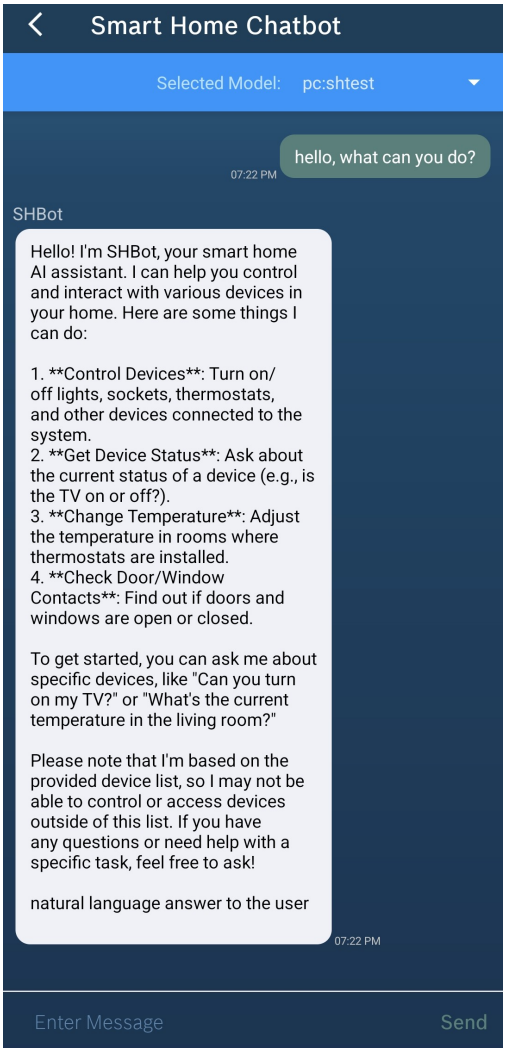
Another point for data management are the system logs of the Bosch Smart Home system (and potentially additional historical data) to for example reason why something happened in the system and data from external sources to for example analyze how the electricity price changed. Since we were not able to implement this it is mentioned in Section 5.5.

5.3.2 User Interface

The user interface of our Smart Home Chatbot was designed to seamlessly integrate with the existing Bosch Smart Home app while providing easy access to the chatbot functionality. Figure 5.2 provides an overview of the user interface implementation. The chatbot activity is built upon the Android AppCompatActivity class, ensuring compatibility across different Android versions and providing a consistent look and feel with the rest of the application. In the main activity of the Bosch Smart Home app, users can select multiple views such as “Favourites” and “Rooms” using the bottom navigation. To make the chatbot easily accessible from any part of the main activity, we extended the existing functionality by adding a “CHATBOT” button to the toolbar, as shown in Figure 5.2a. This simple yet efficient solution allows users to access the chatbot from anywhere within the main activity.



(a) Availability of the chatbot in the main app activity

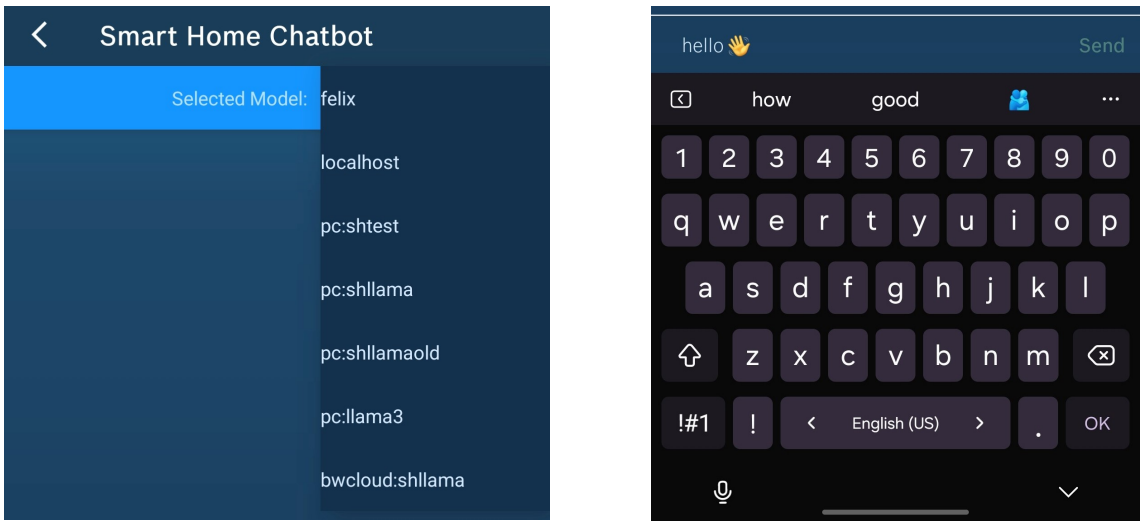


(b) View of the chatbot activity

Figure 5.2: Overview of the User Interface

Figure 5.2b displays the layout of the chatbot UI. The toolbar features an “Up Button”, a common Android navigation element, allowing users to return to the main activity. The toolbar also includes the heading “Smart Home Chatbot” for clear identification. Below the toolbar, we implemented a development feature that allows selection of different models or endpoints through a dropdown menu (spinner), as detailed in Figure 5.3a. This feature enables testing and comparison of various models during development. Messages are always sent to the currently selected model/endpoint.

The core of the chatbot UI is the chat history, displayed below the model selection. User messages appear on the right side in olive green, while responses from “SHBot” (Smart Home Chatbot) are shown on the left with a white background. Each message includes a timestamp indicating when it was sent or received. A notable feature of the chatbot’s response mechanism is the real-time display of each token as it is received from the language model. This creates the effect of the chatbot “writing” its response in real-time, enhancing the interactive feel of the conversation and providing immediate feedback to the user. The conversation shown in Figure 5.2b demonstrates the chatbot’s ability to summarize its functionality, describing possible actions, providing examples, and outlining limitations. A minor inconsistency is noted in the last sentence, likely due to the chatbot’s typical response format of natural language followed by JSON data. For more example conversations see Appendix A.4.3. It’s worth noting that the chat history is only maintained for the current session and is not persisted when re-entering the chatbot activity, which was deemed sufficient for the prototype.



(a) Expanded Model Selection

(b) View when writing a message

Figure 5.3: Details of the Chatbot User Interface

At the bottom of the chatbot UI, users can find an input field for typing messages and a send button. When the input field is selected, the Android device’s keyboard appears, as shown in Figure 5.3b. The keyboard also includes a speech-to-text feature, enhancing accessibility and user convenience. Figure 5.3 provides additional details of the chatbot user interface, showcasing the expanded model selection dropdown and the view when composing a message.

This user interface design, built on AppCompatActivity, ensures that the Smart Home Chatbot is easily accessible, intuitive to use, and seamlessly integrated with the existing Bosch Smart Home app functionality. The real-time token display feature adds a dynamic and engaging element to the user experience.

5.3.3 Request Building

As previously described in Section 4.4.2, the Request Builder within the API Client & Request Builder module constructs properly formatted API requests, sends them, parses server responses, and handles asynchronous operations. This section elaborates on the process of building and sending these requests.

Listing 5.5 Components of a POST Request to the Server Running Ollama

```
POST http://<domain-placeholder>/api/chat
{
    "model": "<model-placeholder>",
    "messages": [
    {
        "role": "user",
        "content": "<device-list-placeholder>"
    },
    {
        "role": "user",
        "content": "<user-message-placeholder>"
    }
    ],
    "stream": true
}
```

Base Structure of each POST Request

```
[{
    "type": "POWER_METER_SWITCH",
    "name": "Office Desk Lamp",
    "deviceID": "12345",
    "state": [
        {
        "id": "PowerMeter",
        "state": {
            "powerConsumption": 10,
            "energyConsumption": 50
        }
        },
        {
        "id": "PowerSwitch",
        "state": {
            "switchState": "ON"
        }
        }
    ],
    "room": "Office"
}]
```

Example Device List containing only a Smart Plug

To ensure low latency while maintaining context, the system considers only a limited number of recent messages when constructing a request. Importantly, the first message in the history always contains a list of the user's devices, providing crucial context for the language model. The message history follows a strict alternating pattern between "user" and "assistant" roles after the initial device list. When a user sends a new message, it is appended to this structured history, ensuring the language model has sufficient context for accurate reasoning. Listing 5.5 illustrates the format of a POST request sent to the server. The 'messages' array encapsulates the condensed message history, with the user's latest message positioned at the end. This structure provides the language model with a concise yet comprehensive context for generating appropriate responses.

The 'model' field specifies the language model to be used, while the 'stream' parameter set to true enables real-time streaming of the model's response. This approach allows for immediate display of partial responses, enhancing the user experience by reducing perceived latency. The device list, exemplified in the right panel of Listing 5.5, provides detailed information about each smart home device, including its type, name, unique identifier, current state, and location. This comprehensive device context enables the language model to generate informed and relevant responses to user queries about their smart home environment.

5.3.4 Response Handling and Action Triggering

As previously described in Section 4.4.2, the Response Handler module interprets server responses, determining necessary client actions such as response analysis and orchestration. The response handling is closely integrated with the API Client, which parses the server responses and initiates the response handling process. This section elaborates on the implementation details of this crucial functionality. The response from the server, as initiated by the request detailed in Section 5.3.3, is received as a stream of tokens through the Ollama software. This streaming approach allows for real-time processing and display of the model's output, enhancing the responsiveness of the chatbot interface. While receiving the response stream, the handler separates it into two components:

Natural Language Response: This portion is immediately forwarded to the UI for display, maintaining a fluid conversation flow with the user.
JSON Object: Typically positioned at the end of the response, this structured data undergoes parsing to extract action-related information.

The JSON object, as defined in the modelfile (see Listing 5.2), contains key information for action triggering (only listed the important ones here):

- 'action': Specifies the intent or action to be performed
- 'value': An optional parameter for actions requiring additional data
- 'deviceID': Unique identifier of the target device

The Response Handler processes this JSON object to determine the appropriate action. As outlined in Listing 5.3, the system supports several actions:

- 'none': No action required
- 'turn-on': Activate a device (e.g., a smart socket)
- 'turn-off': Deactivate a device
- 'change-temperature': Adjust temperature settings (requires a 'value' parameter)

Based on the 'action', 'deviceID' and eventually the 'value' fields in the JSON, the Response Handler triggers the corresponding function in the client application with correct prarameters. For instance:

- If the action is 'turn-on' or 'turn-off', it calls the appropriate method to change the state of the specified device (identified by 'deviceID').

- For 'change-temperature', it invokes the temperature adjustment function for the room climate control device, using the provided 'value'.
- In case of 'none', no further action is taken beyond displaying the natural language response.

The main challenge here was to design a robust parsing mechanism for mapping the received JSON to the exact functionality wanted. The Response Handler also manages error scenarios, such as invalid actions or device IDs not present in the current device list.

5.4 Interaction Flow

Figure 5.4: Sequence Diagram of the typical flow of the system

The interaction flow between the user, client-side application, and server is illustrated in Figure 5.4. This sequence diagram outlines the process of handling user queries, constructing API requests, and generating responses based on smart home device information.

The flow begins with user input via the chatbot interface. The client-side application then retrieves the current state of smart home devices and packages this information, along with the user's query and recent message history, into a structured API request using the Request Builder module. This request is sent to the server hosting the Ollama software and language model, which processes the input and generates a response. The response is streamed back to the client in real-time, enhancing user experience by reducing perceived latency.

The Response Handler module separates the incoming response into a natural language component and a structured JSON object. The natural language portion is displayed in the UI, while the JSON object is parsed for any action directives.

If actions are specified, the Response Handler triggers the corresponding functions in the client application, potentially sending commands to smart home devices or updating the application's state. The UI is then updated to reflect both the chatbot's response and any changes in device states. This finishes the interaction loop. The entire process typically occurs within seconds.

5.5 Challenges and Solutions

There were several challenges and considerations that had to be taken into account when designing and developing the prototype:

Complexity of Automations: Balancing simplicity and functionality in user-defined automations proved challenging. The vast possibilities of Bosch Smart Home automations made it difficult to build a comprehensive assistant. Even if the language model could understand the capabilities of the automations, mapping this to a JSON output and triggering a function to create specific automations would be complex. A potential solution would have been to define a set of automation types that could be created, but time constraints prevented this development.

Multilingual Support: Providing accurate and contextually appropriate responses in both German and English was difficult in some scenarios. The chatbot occasionally struggled to consistently answer in German when the user's last input was in German, despite understanding the request. A possible reason could be that the device list provided to the model is always in English. Potential solutions include providing the model with information about the Bosch Smart Home app's current language, adapting the device list's language accordingly, or fine-tuning the model. Time limitations prevented the implementation of these solutions.

Historical Data: During the thesis, historical data of the Bosch Smart Home devices was not available. Only the current state of devices could be accessed. Some variables in this state store summarized historical data, such as the total power consumption for smart plugs. A potential solution could have been to mock the data to test what data format would be sufficient for the model to correctly answer user requests and interpret the data.

Availability of System Logs: Beginning with the second iteration of intents, system logs would have acted as a solid foundation for short-term historical data, including state changes of devices, automations, and errors. However, this information is only available inside the Bosch Smart Home Controller and therefore encapsulated from the Android App. Time constraints prevented the development of functionality to share these logs with the app.

Data from External Sources: Integrating data from external sources, such as the "Statistisches Bundesamt" (Federal Statistical Office of Germany), could have provided valuable insights for energy cost analysis in the user's home. This data, including information on inflation in energy costs (gas, district heating, electricity, etc.), could have been used to analyze why energy costs increased and offer suggestions for energy savings based on unnecessary automations. Implementing this would require data retrieval and ingestion from defined sources (URLs), potentially using Retrieval Augmented Generation (RAG) with Ollama or Langchain. While time constraints prevented the implementation of this feature in our prototype, we explored the necessary data and methods. Our experiments showed that providing the language model with a list of devices containing (artificial) historical data, information on related automations, and inflation indices gave sufficient context for reasoning, although not always with perfect accuracy.

Security Issue: The Bosch Smart Home app initially did not support API calls to arbitrary domains. To address this, we needed to modify the security settings for the prototype to enable calls to the Ollama API on a specific device. This change was implemented only for the chatbot functionality, while the rest of the app maintained its original security configuration.

Model Performance: Incrementally improving model outputs proved challenging. In our case, this was achieved by analyzing the output of our evaluation script, identifying low-quality outputs, and addressing them in the modelfile by adding more examples or instructive text. The evaluation script itself is described in Section 6.2.

6 Evaluation

This chapter presents a comprehensive evaluation of our smart home chatbot prototype. We begin by outlining our study design, which employs the Goal Question Metric (GQM) paradigm to assess the chatbot. The evaluation process combines quantitative analysis of model performance with a user study to gather both objective metrics and subjective feedback. We then detail our evaluation methodology, including the selection of language models, creation of an evaluation dataset, and the metrics used to assess performance. The results section presents our findings from both the model evaluation and user study. Finally, we discuss these results, interpreting their implications for the effectiveness of our chatbot and identifying areas for future improvement. This multi-faceted approach provides a thorough understanding of our prototype's strengths and limitations.

6.1 Study Design

In this section we want to provide the base study design we came up. The design of our evaluation was gathered through clearly defining the goals of the evaluation and coming to measurable metrics in the end via the top-down GQM approach. Our approach consists of three main goals: assessing the accuracy, the user experience and the explainability of the developed smart home chatbot. Based on this we developed the whole evaluation process which consists of a LLM evaluation approach for the accuracy and a user study for the other two goals. Details are provided later within this chapter.

6.1.1 Goal Question Metric Paradigm

The GQM paradigm according to Basili et al. [BCR94] provides a structured approach to evaluate different works in the area of Software Engineering and therefore is also suitable for evaluating various aspects of the smart home chatbot. Our evaluation framework consists of three primary goals, each addressing a specific area of interest: accuracy, user experience, and explainability. This framework is shown in Figure 6.1

Goal 1: Assess the Accuracy of the Smart Home Chatbot

The first goal focuses on determining how accurately the chatbot can understand and respond to user commands. To achieve this, several questions are formulated:

- **Q1: How accurate are the natural language answers of the language model?**
- **Q2: How accurate are the JSON responses of the language model?**

To answer these questions, relevant metrics are identified. Semantic similarity measures are used to evaluate the natural language responses, potentially incorporating other related metrics to ensure comprehensive assessment. JSON accuracy metrics are employed to evaluate the precision of the chatbot's structured responses. A combined metric of semantic similarity and JSON accuracy provides a holistic view of the chatbot's overall accuracy.

Goal 2: Evaluate the User Experience of the Smart Home Chatbot

The second goal is to understand the users' interaction experience with the chatbot. This involves evaluating how intuitive and satisfactory the chatbot is in performing tasks. The questions under this goal include:

- **Q1: Are typical tasks easy to achieve?**
- **Q2: How satisfied are users with the chatbot's performance?**
- **Q3: What could be improved?**
- **Q4: Does the chatbot add to existing functionality of typical smart home applications?**

The metrics for these questions involve measuring task completion time, the number of attempts, and the success rate of task completion. User satisfaction is gauged through questionnaires administered after the experiment. These questionnaires assess various aspects of the user experience, including ease of use, overall satisfaction, and areas for improvement.

Goal 3: Assess the Explainability of the Smart Home Chatbot

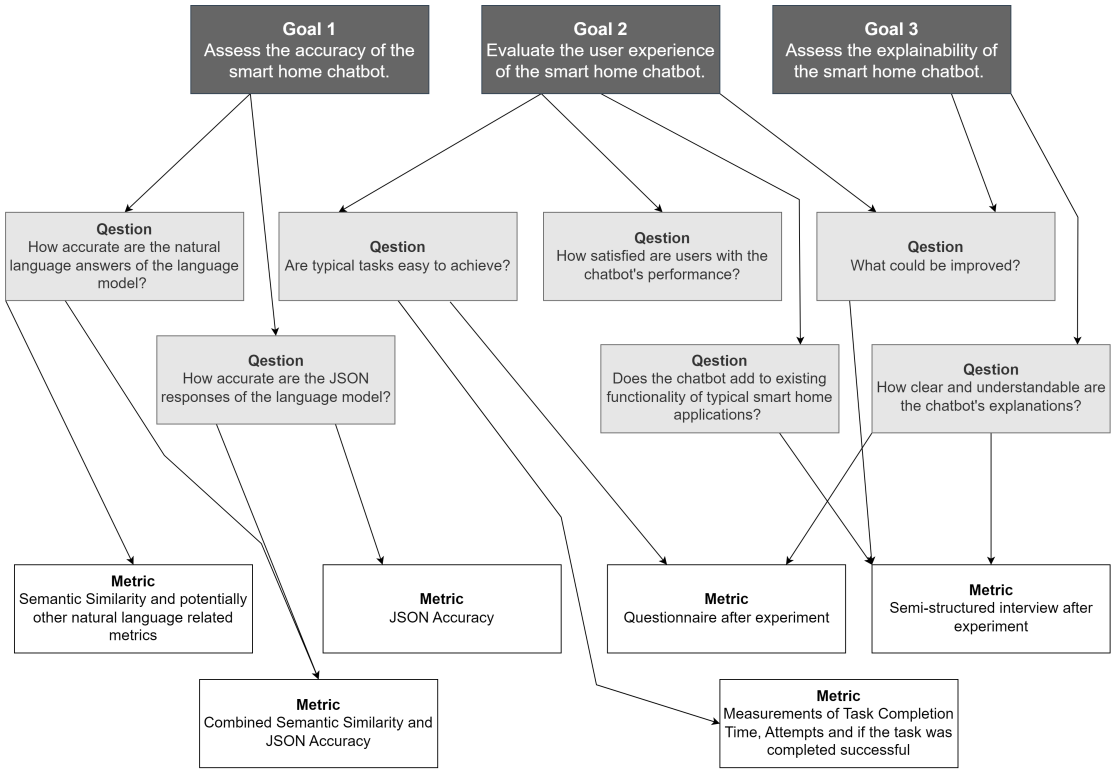


Figure 6.1: Visualized Goal Question Metric

The third goal addresses how well the chatbot can explain its actions and decisions to users, which is crucial for building trust and usability. The questions related to this goal are:

- **Q1: How clear and understandable are the chatbot's explanations?**
- **Q2: What could be improved?**

To measure the explainability, semi-structured interviews are conducted after the experiment. These interviews delve into the clarity, transparency, and usefulness of the explanations provided by the chatbot, allowing for detailed qualitative feedback from users.

6.1.2 Resulting Evaluation Process

A Visualization of our evaluation process can be seen in Figure 6.2 Based on the obtained GQM, the evaluation can be split into two parts: evaluating the model performance and a user study for examining User Experience and Explainability. Besides the developed protoype chatbot the obtained sample user inputs can be greatly used in the evaluation process. They could be used to construct the dataset that was essential for the evaluation of the language model. This evaluation dataset contains for each sample input an expected natural language output and an eventually expected JSON to measure both the accuracy of the output the user sees and the constructed JSON that is used for further actions in the smart home system.

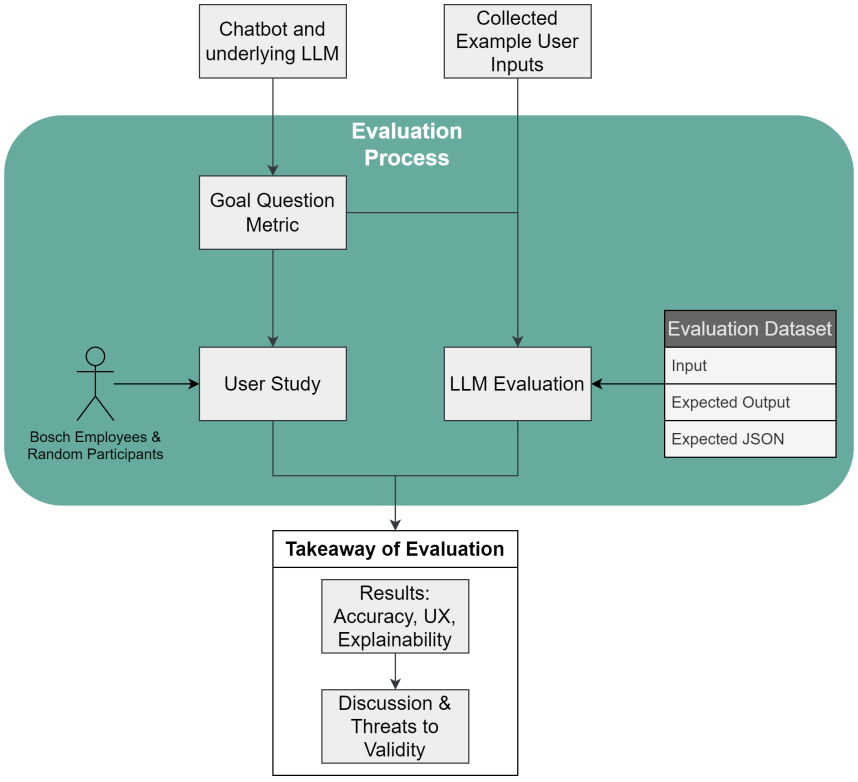


Figure 6.2: The Evaluation Process, Visualized

The other part is the user study in which users have a setup of devices that are supported by our prototype and receive a list of tasks in which the success should be measured and afterwards a questionnaire and a semi-structured interview are used to answer the questions regarding Goal 2 and 3 in the defined GQM

Based on these two parts, the takeaway of this thesis can be received and constructed. The results emerge directly as an output from the model evaluation and the user study when combined with quantitative and qualitative methods.

Based on this and the detailed evaluation setup the results can be discussed and threats to validity be debated.

6.2 Model Performance

This section details our approach to evaluating the performance of language models for our smart home chatbot. We describe the hardware setup and model selection process, explain the construction of our evaluation dataset, and outline the metrics used to assess both semantic similarity and JSON accuracy. This evaluation framework allows us to compare different model iterations and identify the most suitable configuration for our chatbot prototype.

6.2.1 Hardware and Language Models used

The hardware setup was the same as described in Section 5.2.2 since it is very fast for the model sizes we wanted to test as also explained in the same section. For measuring the model performance there was no need to have a setup of smart home devices since when the outputed JSON of the model is correct the correct action will be triggered in the underlying system. The language models we selected for evaluation are also described in Section 5.2.2. We pre-filtered these models based on three selected typical user interactions from the evaluation dataset we created. If a model was not able to answer with a suitable JSON or did not even understand the structure of the message history provided we excluded it from further evaluation with the whole dataset since it would not be worth investing the resources for it. Only the llama3 model has passed this test (with two version: the normal 8b parameters model and the instruct version of it) and is therefore included in the further evaluation. The other models were not able to answer based on the device list provided with each user request. They usually answered with phrases like “I can’t find a device named ’TV’.”

6.2.2 Evaluation Dataset

To assess the performance and capabilities of our smart home chatbot, we developed a comprehensive evaluation dataset with a total of 80 entries. This dataset is designed to simulate realistic user interactions and test the chatbot’s ability to understand context, control devices, and provide informative responses. The evaluation dataset consists of a series of input-output pairs, where each input represents a chat history and the output represents the expected response from the chatbot. In creating the inputs for our evaluation dataset, we leveraged the user inputs collected during our earlier survey (as described in Section 4.4). These real-world examples provided authentic phrasing and diverse query formulations typical of smart home users. We adapted and expanded upon these

collected inputs to ensure they aligned with our specific test scenarios and device setups. This approach allowed us to create a more realistic and challenging evaluation dataset, closely mimicking the variety of natural language inputs a smart home chatbot would encounter in practical use. The structure of each entry in the dataset is as follows:

1. Input: A chat history containing a minimum of two messages from the user. The first message always includes a device list that provides crucial context about the user's smart home environment. For details on the device list structure, refer to Section 5.3.3.
2. Expected Output: A natural language response that the chatbot is expected to generate based on the given chat history.
3. Expected JSON: A JSON object representing the action the chatbot should take, if any. The JSON includes only the necessary keys for each action:
    - For 'turn-on' or 'turn-off' actions: 'action' and 'deviceID'
    - For 'change-temperature' action: 'action', 'deviceID', and 'value'
    - If no action is necessary, the expected JSON is "None"

Based on the chat history a request can be build with a python script leveraging LangChain easily access Ollama since it is supported by LangChain. That way we do not have to construct API calls ourself only parse the message history into a LangChain function call that does everything in the background. We can just create an ChatOllama object like the following:

```
# Initialize language model
llm = ChatOllama(
    base_url="http://127.0.0.1:5000",
    model="sh-llama3-instruct",
    keep_alive=-1
)
```

Therefore we can easily switch between all models that we want to test out. The keep alive option set to minus one means that the model is loaded undefinetly into memory. After parsing the messages of one csv entry we can just use the following code to create the code and invoke the language model:

```
prompt = ChatPromptTemplate.from_messages(messages)
result = invoke_language_model(llm, prompt)
```

To create a diverse and representative dataset, we used 10 example device lists as the basis for our scenarios. These lists were carefully crafted to cover various smart home setups:

- 5 edge case device lists, including:
    - Thermostats in different rooms
    - Multiple thermostats in the same room
    - Multiple smart plugs with similar names in the same room
    - Multiple door/window contacts in the same room

  - A setup where the user has no thermostats
- 3 random German examples with device and room names in German
- 2 random English examples with typical device names and variations of supported devices

These device lists were sometimes modified (e.g., changing variable values) to match specific test cases, ensuring a wide range of scenarios for evaluation. The dataset covers various interaction types, including device control commands, queries about device states, requests for information about the smart home setup, and complex questions requiring reasoning about multiple devices or rooms. Table 6.1 illustrates the format of the dataset and provides two example entries (note that the device lists are shortened to one device and removed state information here for a better overview, the actual device lists contain 2-6 devices).

input	expected_output	expected_json

[
{
"role": "user",
"content": [
{
"type": "POWER_METER_SWITCH",
"name": "Zwischenstecker",
"deviceID": "55555",
"state": [...],
"room": "Schlafzimmer"
}
]
},
{
"role": "user",
"content": "Turn on the Zwischenstecker, please."
}
]
	The Zwischenstecker in your bedroom is now on.	{
		"action": "turn-on",
		"deviceID": "55555"
		}

[
{
"role": "user",
"content": [
{
"type": "POWER_METER_SWITCH",
"name": "Wohnzimmerlampe",
"deviceID": "44444",
"state": [],
"room": "Wohnzimmer"
}
]
},
{
"role": "user",
"content": "Can you turn on the Wohnzimmerlampe?"
}
]
	The Wohnzimmerlampe is already turned on.
	No need to switch it on.
		None

Table 6.1: Format and Example Entries of the Evaluation Dataset

This carefully created dataset allows us to evaluate the chatbot's performance across multiple dimensions, including accuracy in interpreting user intent, ability to provide relevant responses, correct identification and execution of required actions, contextual understanding, and handling of edge cases and ambiguous requests.

6.2.3 Evaluation Metrics

In this section we describe the metrics we used to measure the accuracy of different language models that we customized with our modelfile as described in Section 5.2.2. For this we have more deeply analyzed the metrics stated in the related work Section 2.2.5. We employed a combination of metrics that evaluate both the semantic accuracy of the natural language responses and the correctness of the generated JSON commands to reliably capture the relevant dimensions of our chatbot prototype.

Semantic Similarity

Semantic similarity is a crucial metric in our evaluation since it measures how closely the generated responses from the chatbot match the expected outputs in terms of meaning. For this evaluation, we used the `SentenceTransformer` model, specifically the `paraphrase-MiniLM-L6-v2` variant, to compute cosine similarity between the embeddings of the generated responses and the expected outputs. A high similarity score indicates that the chatbot's response is semantically close to the expected answer, even if the exact wording differs.

The process of calculating semantic similarity involves the following steps:
Encoding: Both the reference (expected) outputs and the generated responses are encoded into high-dimensional vectors using the SentenceTransformer model.
Similarity Computation: The cosine similarity between the embeddings of the reference and generated responses is calculated. Cosine similarity measures the cosine of the angle between two vectors, providing a value between -1 and 1, where 1 indicates perfect similarity, 0 indicates no similarity, and -1 indicates perfect dissimilarity.
Aggregation: The individual similarity scores are aggregated to produce an average similarity score across all samples in the evaluation dataset.

Listing 6.1 Code for Calculating the Semantic Similarity through Cosine Similarity

```
def calculate_semantic_similarity(references, generated_responses):
    model = SentenceTransformer('paraphrase-MiniLM-L6-v2')
    embeddings1 = model.encode(references, convert_to_tensor=True)
    embeddings2 = model.encode(generated_responses, convert_to_tensor=True)
    cosine_scores = util.pytorch_cos_sim(embeddings1, embeddings2)

    similarities = [cosine_scores[i][i].item() for i in range(len(references))]
    average_similarity = sum(similarities) / len(similarities)
    return similarities, average_similarity

```

The implementation of this metric is shown in Listing 6.1. This code defines a function `calculate_semantic_similarity` that takes two lists of sentences (references and generated responses) as input and returns a list of individual similarity scores along with the average similarity score. A high average similarity score indicates that the chatbot's responses are semantically close to the expected answers, even if the exact wording differs. This allows for a more flexible evaluation that captures the chatbot's ability to understand and respond to user intents accurately, rather than merely reproducing exact phrases.

JSON Accuracy

We define JSON accuracy as the percent of correctly generated JSONs by a language model customized to our use case. Since our dataset contains only necessary keys of the expected JSON, a correct generated JSON is one which contains each expected key and the correct corresponding value for it.

Each generated JSON is compared against the expected JSON to determine if it correctly represents the intended action or response. The code in Listing A.2 shows how we have implemented this. The accuracy we just explained is represented by the variable "accuracy".

The total count (total_count) is the total number of generated responses, which represents the total number of JSONs evaluated. This is determined by the length of the generated_responses list. The correct_count is increased in several scenarios:

1. When both the expected and generated JSONs are None.
2. When the expected JSON is None and the generated JSON has an "action" key with the value "none".
3. When the generated JSON matches the expected JSON in terms of keys and their corresponding values (determined by the compare_jsons function).

Generated JSONs that would throw an error on parsing are handled as incorrect. This is implemented in the code through the use of try-except blocks. If a JSONDecodeError occurs when trying to parse the expected JSON, or if an AttributeError occurs when comparing the generated and expected JSONs, the JSON is considered incorrect and the json_accuracy_flags for that instance is set to False. The accuracy is then calculated by dividing the correct_count by the total_count.

The code also includes the variable "key_accuracy" which checks how many keys have the correct value as in the expected JSON. It is calculated as follows:

- total_keys is incremented for each key in the expected JSON.
- correct_keys is incremented when a key in the generated JSON matches the corresponding key in the expected JSON.
- The key_accuracy is then calculated as the ratio of correct_keys to total_keys.

This key_accuracy provides a more granular measure of how well the generated JSONs match the expected JSONs on a key-by-key basis, even if the entire JSON doesn't match perfectly.

The function shown returns three values: the overall JSON accuracy, the key accuracy, and a list of boolean flags indicating which generated JSONs were correct (json_accuracy_flags).

6.2.4 Combined Metric

To provide a holistic view of the chatbot's performance, we developed a combined metric that integrates both semantic similarity and JSON accuracy. This approach was inspired by the need to evaluate the chatbot's performance across multiple dimensions simultaneously. The combined metric adapts the concepts of precision, recall, and F1 score from traditional classification tasks to our specific use case. Here's how we define the components:

True Positive (TP): Cases where the model's output has high semantic similarity (above a defined threshold) and the generated JSON is correct.
False Positive (FP): Cases where the model's output has high semantic similarity but the generated JSON is incorrect.
True Negative (TN): Cases where the model's output has low semantic similarity and the generated JSON is incorrect.
False Negative (FN): Cases where the model's output has low semantic similarity but the generated JSON is correct.

Based on these definitions, we calculate precision, recall, and F1 score as follows:

Precision: Of all the outputs the model that have a high semantic similarity, how many were actually positive cases (generated JSON is also correct).

Recall: Of all the actual positive cases (generated JSON is correct), how many did the model correctly identify as positive (semantic similarity is also high).

F1 Score: The harmonic mean of precision and recall, providing a single measure of performance.

Details on how we have implemented this can be seen in Appendix A.4.2.

Determining the Similarity Threshold

The choice of an appropriate similarity threshold is crucial for the effectiveness of our combined metric. Through careful analysis of the semantic similarity between various generated outputs and their corresponding expected outputs, we determined that a threshold of 0.65 provides the most meaningful differentiation. This decision was based on several key observations:

Cross-language Differentiation: We found that when a generated response was in English but the expected response was in German, the similarity score was consistently below 0.55. This served as a clear demarcation point, even when considering the constraints of the language model size we chose.

Quality Assessment: After establishing 0.55 as a baseline, we examined cases with higher similarity scores. Our analysis revealed that responses with a similarity score greater than 0.65 were consistently of high quality and relevance.

Borderline Cases: We identified examples that fell just below and above our chosen threshold to validate its appropriateness. For instance, an unacceptable response with a similarity of approximately 0.61 was: "I cannot have a temperature measurement for the kitchen. There are only rooms and devices, but no temperature-taking devices available for the kitchen." The expected response was: "I'm sorry, but no temperature sensors are available in the kitchen." Despite

some semantic overlap, this response was deemed insufficient. It is worth noting that this is a literal translation from a German example, i.e., when calculating the similarity between those two sentences it will probably differ from the actual one.

Acceptable Responses: Conversely, we found that responses just above the 0.65 threshold were generally acceptable. An example with a similarity of about 0.67 was: “The temperature set point is 18 degrees, and the current temperature is 19 degrees.” The expected response was: "The temperature in the Schlafzimmer is set to 18.0 degrees Celsius and the current temperature is 19.0 degrees Celsius."This response, while missing some specifics, captured the essential information accurately.

By setting the threshold at 0.65, we strike to conduct a strict and reliable evaluation. This threshold effectively distinguishes between responses that capture the core meaning and intent of the expected output and those that fall short, providing a robust basis for our combined metric.

6.3 User Experience

To evaluate the user experience of our smart home chatbot, we designed and conducted a comprehensive user study. This study aimed to assess the usability, effectiveness, and potential benefits of integrating a chatbot interface into a smart home system.

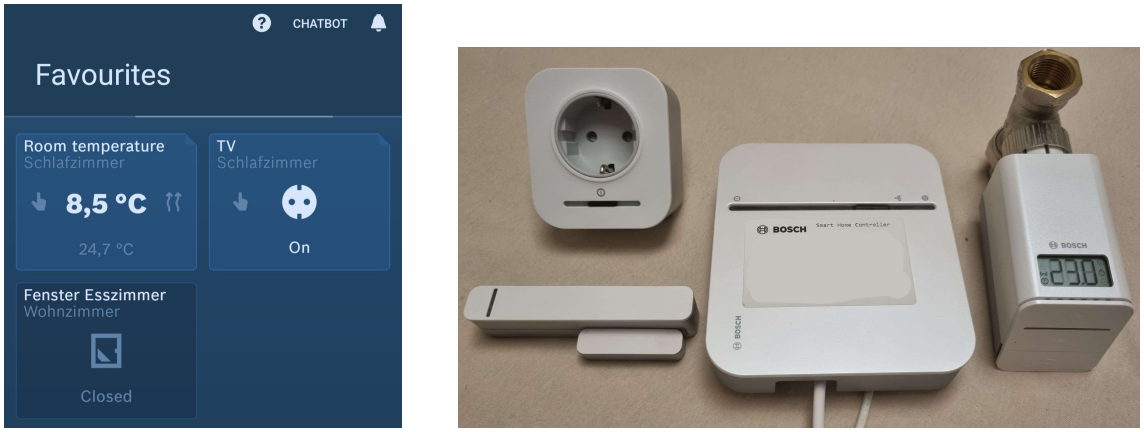
6.3.1 Participants

We recruited a diverse group of 10 participants for our study, comprising three employees from Bosch Smart Home and seven external participants, including university students and employees from other companies. The participants represented a varied demographic in terms of age and professional background, with 60% having a technical background and 40% non-technical. Experience with smart home systems varied among participants, with only one having no prior experience. Familiarity with chatbots was evenly distributed across the spectrum from no experience to regular use.

This diverse group was chosen to represent a range of potential users, from those familiar with smart home technology to novices, ensuring a comprehensive evaluation of the chatbot’s accessibility and usefulness. A Visualization showing different aspects of the participants can be found in Appendix A.4.4

6.3.2 Apparatus

The experimental setup consisted of three smart home devices: a smart plug named “TV” and a thermostat, both located in the sleeping room, and a door/window contact in the living room. It is worth noting that a Bosch Smart Home Controller is necessary for the use of these devices (smart plug, door/window contact, thermostat). The language model ran on a PC with specifications described in Section 5.2.1. Participants interacted with the Bosch smart home app either through a PC emulating Android or a Samsung Galaxy S21 Smartphone. To ensure consistency, all participants used the same setup in the same room. How the devices look in the Bosch Smart Home app and their analog representations can be seen in Figure 6.3.



(a) Shown in the App

(b) Shown Analog

Figure 6.3: Devices for the User Study

6.3.3 Procedure

Participants were asked to complete six tasks using the chatbot interface. To mitigate learning effects, the order of these tasks was randomized for each participant. The following shows the tasks of the experiment:

T1. Ask the chatbot what it is for.

T2. Try to find out the state of the TV. Is it on? How much energy does it consume?

T3. Try to find out what the temperature is in the bedroom or what the temperature is set to (or both).

T4. Ask the chatbot to give a summary of devices.

T5. Use the chatbot to change the temperature in the bedroom.

T6. Ask the chatbot to switch the state of the TV.

6.3.4 Metrics, Data Collection and Analysis

We collected both quantitative and qualitative data to evaluate the user experience. Quantitative metrics included task completion time, number of attempts per task, and task success rate. These were collected through researcher observation. Further quantiative data was gathered through a questionnaire using a 5-point Likert scale to assess various aspects of the chatbot's usability and effectiveness, followed by gathering qualitative data through a semi-structured interview to explore participants' experiences in more depth. The interview explored participants' experiences with confusion or frustration, preference for chatbot interactions, suggestions for improvement, and opinions on potential additional functionalities to enhance understanding of their smart home systems. The following shows the questions of the Questionnaire:

Q1. The tasks were easy to accomplish.

Q2. The chatbot is easy to find and to use.

Q3. The chatbot enhances the usability of the app.

Q4. The chatbot's responses were clear and helpful.

Q5. The smart home's behavior after the chatbot's responses was understandable.

Q6. I would use the chatbot if it was added to the official app.

The data analysis plan incorporated both quantitative and qualitative approaches. Quantitative analysis focused on task completion rates, time taken, and survey scores. Qualitative analysis of interview responses aimed to identify common themes and areas for improvement. This mixed-methods approach allows for a comprehensive evaluation of the chatbot's performance and user perception, providing both statistical measures of effectiveness and rich, contextual insights into the user experience.

6.4 Results

In this section, we present the findings from our evaluation of the smart home chatbot. We begin with the quantitative results of our language model performance tests, comparing different model versions across various metrics. We then detail the outcomes of our user study, including task completion rates, user satisfaction scores, and qualitative feedback from participants. These results provide a holistic view of the chatbot's effectiveness and user perception.

Additionally to this section we also show example conversations with the developed prototype under Appendix A.4.3 and highlight what works well and what does not.

6.4.1 Language Model Performance Results

Model	Semantic Similarity Values					JSON Accur		Combined Metrics		
	Mean	Median	Std Dev	Range	>0.65	Total	Key	Prec	Rec	F1
shllama3	0.5667	0.5916	0.2608	1.0737	40.24%	0.7805	0.5789	1	0.47	0.64
shllama3instr	0.6036	0.6758	0.2878	1.0885	51.22%	0.8049	0.8571	1	0.5606	0.7184
shllama3-2	0.4185	0.3909	0.2660	1.0654	19.51%	0.8171	0.7037	1	0.1940	0.3250
shllama3instr-2	0.3888	0.3857	0.2314	1.1155	13.41%	0.8780	0.6	1	0.125	0.2222

Table 6.2: Descriptive Statistics and Analysis of the Semantic Similarity and Performance Metrics for the Different Llama3 Models Created

In this section we want to show our results from four variations that we created from the `llama3` model. Even though we tested all models mentioned in crefsubsec:modelcust only the llama3 model was worth evaluating as described before in Section 6.2.1. what we can see in this table are the four different versions of `llama3`. Note that we abbreviated the names of our model versions in the table to have a better table layout. Also, the "sh" stands for "Smart Home". We used the same modelfile for `shllama3` and `shllama3instruct` and after analyzing use cases that it can not handle and tried to build an improved modelfile that we used to create `shllama3instr-2` and `shllama3instr-2`. The initial versions of the models encountered challenges in specific scenarios, including: Generation of multiple JSONs in a single response which was not supported by our prototype, inconsistent selection of language to answer in, inaccurate calculation of average temperatures and inconsistent

handling of temperature increase/decrease by a certain value commands. To address these issues, we developed improved modelfiles for the second generation models (shllama3-2 and shllama3instr-2) that included more precise descriptions and additional example conversations.

In the table we can see that the revised modelfiles for the second generation of models we created did improve the total JSON accuracy, but for the `shllama3instr-2` the key accuracy decreased. For the revised modelfiles also suffer in the overall quality in responses which according to the semantic similarity are much worse with a Mean around 0.4. This of course hardly effects the combined metrics with an F1 score of 0.3250 and 0.2222 which is barely half as much as for the first generation models. This means that the models are quite unpredictable or random in their responses which can be seen especially in the semantic similarity values. The results demonstrate a trade-off between JSON accuracy and semantic similarity. While the second-generation models showed improved JSON accuracy, they suffered a significant decrease in semantic similarity. This suggests that the modifications made to improve specific functionalities may have compromised the models' overall language understanding and generation capabilities.

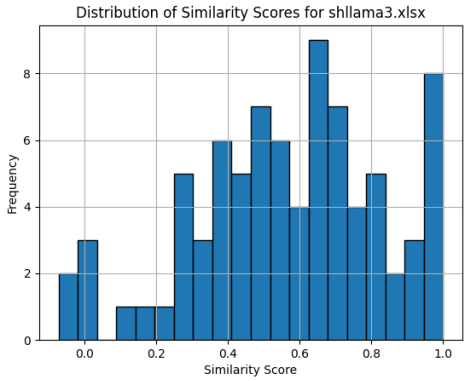


Figure 6.4: Distribution of Similarity Scores for shllama3

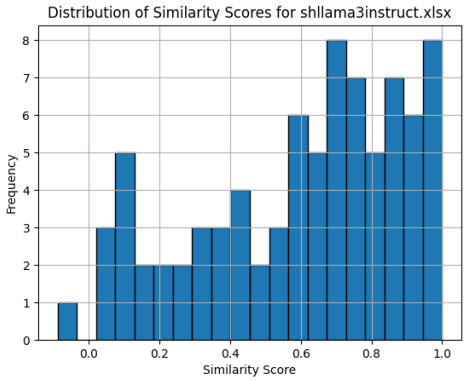


Figure 6.5: Distribution of Similarity Scores for shllama3instruct

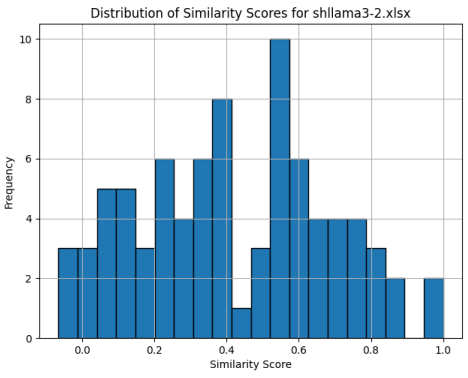


Figure 6.6: Distribution of Similarity Scores for shllama3-2

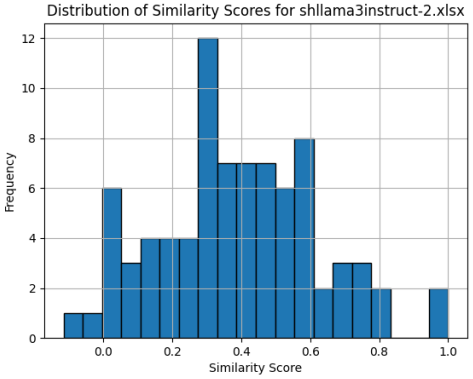


Figure 6.7: Distribution of Similarity Scores for shllama3instruct-2

Figure 6.8: Distribution of Similarity Scores for Different Models

For individually considering the semantic similarity of the models we created histograms that are shown in Figure 6.8. They futher highlight the better performance of our first generation models `shllama3` and `shllama3instruct` since the values for those less values are distributed below a similarity score of 0.4 or even 0.5 than the first generation models. Especially `shllama3instruct` shows a good distribution with most values being at least 0.6. Approximately twice as many values are above 0.6 than below.

To further compare the semantic similarity values between the models `shllama3` and `shllama3instruct`, we also performed a Wilcoxon signed-rank test [FHK+24; She11]. This non-parametric test is used to determine whether there is a statistically significant difference between the paired samples from these two models and has the benefit that we do not assume normality in the distribution of differences like in the typical t-test method. When we have a look at the previous distributions of similarity scores we would assume that no normal distribution is given. The results of the Wilcoxon signed-rank test are as follows:

- **Wilcoxon signed-rank test statistic**: 1395.0
- **P-value**: 0.1565

Given that the p-value (0.1565) is greater than the commonly used significance level of 0.05, we fail to reject the null hypothesis. This indicates that there is no statistically significant difference in the distribution of semantic similarity values between the two models, `shllama3` and `shllama3instruct`, at the 0.05 significance level (even if we would use a less strict significance level of 0.1 which could be applicable for comparing the semantic similarity of language models). In practical terms, these results suggest that both models perform similarly in terms of semantic similarity for the given data set. However, it's important to note that while not statistically significant, there is a noticeable difference in the median values and the percentage of responses above the 0.65 threshold between shllama3 and shllama3instruct. This suggests that shllama3instruct may offer some practical improvements in semantic similarity, even if not statistically significant at the conventional 0.05 level.

To further compare the JSON correctness between the models `shllama3` and `shllama3instruct`, we performed McNemar's test [She11]. This test is used to evaluate differences on paired binary data, which in this case is whether the generated JSON was correct (1) or not (0). The contingency table for the JSON correctness is Table 6.3 and shows that both models correctly generated the expected JSON in 56 cases. `shllama3instruct` correctly generated 10 JSONs that `shllama3` did not and vice versa `shllama3` generated 8 correct that `shllama3instruct` did not.

		instruct	
		0	1
shllama3	0	8	10
	1	8	56

Table 6.3: Contingency Table for JSON Correctness Between `shllama3` and `shllama3instruct`.

The results of McNemar's test are as follows:

- **McNemar's test statistic**: 8.0
- **P-value**: 0.8145

Given that the p-value (0.8145) is greater than the significance level of 0.05, we fail to reject the null hypothesis. This indicates that there is no statistically significant difference in JSON correctness between the two models. Both models exhibit similar performance in generating correct JSON responses. Despite the lack of statistical significance, it's worth noting that shllama3instruct correctly generated 10 JSONs that shllama3 did not, while the reverse occurred only 8 times. This slight edge, although not statistically significant, might indicate a marginal improvement in JSON generation capabilities for shllama3instruct.

Another important part of the results is whether the models generated JSONs that would trigger an action in the smart home when none was expected or generated no JSON when an JSON was expected. In Table 6.4 we can see that `shllama3` has the most mistakes in total for this metric while shllama3instruct-2 has the least which also has no unexpectedly generated JSON. In contrast `shllama3-2` generated the most unexpected JSONs.

Model	Generated JSON (unexpected)	No JSON (but expected one)
shllama3	4	4
shllama3instruct	3	3
shllama3-2	5	0
shllama3instruct-2	0	4

Table 6.4: Generated JSONs vs Expected JSONs

In conclusion, our analysis reveals that while attempts to improve specific functionalities in the second-generation models led to increased JSON accuracy, this came at the cost of reduced semantic similarity and overall performance. The first-generation models, particularly shllama3instruct, demonstrate a better balance between semantic understanding and JSON generation. These findings highlight the challenges in optimizing language models for specific tasks while maintaining general language understanding capabilities.

6.4.2 Quantitative User Study Results

In this section we want to state our results for the quanitative part of our user study. For this we present the following hypothesises:

H_1 The chatbot improves explainability of the app.

H_2 The chatbot improves usability of the app.

H_3 Persons would rather use the chatbot than traditional ways of controlling their smart home.

H_4 Controlling device tasks generally take longer than tasks where no action needs to be triggered.

We want to start with some descriptive statistics about the task completion times during the user study and the questionnaire conducted. Table 6.5 shows the descriptive statistics for task completion time (in seconds) of each task. Task completion time refers to the amount of time it took users to complete each task using the chatbot. As can be seen from the table, Task 3 (Find temperature or set

temperature) had the longest mean completion time (34.0 seconds), with a large standard deviation (23.28 seconds), indicating a high variability in completion time. Task 1 (Ask what the chatbot is for) had the shortest mean completion time (16.7 seconds).

Table 6.6 shows the descriptive statistics for the Likert scale questionnaire, which assessed various aspects of the chatbot's usability and effectiveness on a scale of 1 (strongly disagree) to 5 (strongly agree). The results indicate that users found the chatbot easy to find and use (Q2: mean = 4.40), showing the highest level of agreement. This was followed by positive perceptions of the chatbot's response clarity and helpfulness (Q4: mean = 4.20) and the ease of accomplishing tasks (Q1: mean = 4.10). Users also agreed, though to a slightly lesser extent, that the smart home's behavior after the chatbot's responses was understandable (Q5: mean = 3.90) and that the chatbot enhanced the app's usability (Q3: mean = 3.70). The statement with the lowest agreement, though still neutral, was about using the chatbot if it were added to the official app (Q6: mean = 3.00). This suggests that while users generally found the chatbot helpful and easy to use, they were less certain about incorporating it into their regular app usage. The standard deviations (ranging from 0.70 to 0.95) indicate some variation in responses across participants, with Q3 and Q6 showing the most diverse opinions.

Task	mean	median	mode	std_dev	range
T1-time	16.7	14.5	12	6.09	20
T2-time	22.9	21.5	10	9.41	28
T3-time	34.0	26.5	11	23.28	81
T4-time	26.8	25.0	25	11.15	37
T5-time	45.9	35.0	29	28.47	79
T6-time	28.5	30.0	7	15.1	45

Table 6.5: Descriptive Statistics of the Task Completion Time of Each Task

Question	mean	median	mode	std_dev
Q1	4.10	4.00	4	0.74
Q2	4.40	4.50	5	0.70
Q3	3.70	4.00	4	0.95
Q4	4.20	4.00	4	0.79
Q5	3.90	4.00	4	0.74
Q6	3.00	3.00	3	0.94

Table 6.6: Descriptive Statistics of the Likert Scale Questionnaire

Figure 6.9 shows the number of attempts users needed to complete each task. The x-axis represents the number of attempts per task (T1-T6), and the y-axis represents the corresponding frequency. As can be seen from the figure, users needed the fewest attempts (an average of 1.1 and 1.2 attempts respectively) to complete Task 1 (Ask the chatbot what it is for) and Task 2 (Find out TV state and consumption). Tasks 6 (Switch TV state) and 5 (Change bedroom temperature) required the

most attempts, with an average of 2.2 attempts. Only T3, T5 and T6 needed a third attempt with 7 third attempsts in total which equals $7.\bar{7}\%$ of the total attempts taken of all participants. In total 90 attempts were taken for 60 tasks (1.5 per participant and task).

Figure 6.10 shows the number of participants that were successful for each task. The x-axis represents the success status of each task (T1-T6), and the y-axis represents the number of participants. As can be seen from the figure, all participants were successful in completing Tasks 1, 2, and 4. Tasks 5 and 6 had the lowest success rates, with only 8 participants being successful. Overall 5 out of 60 tasks were not successfully finnished which is a failure rate of $8.\bar{3}\%$ and a success rate of $91.\bar{6}\%$ respectively.

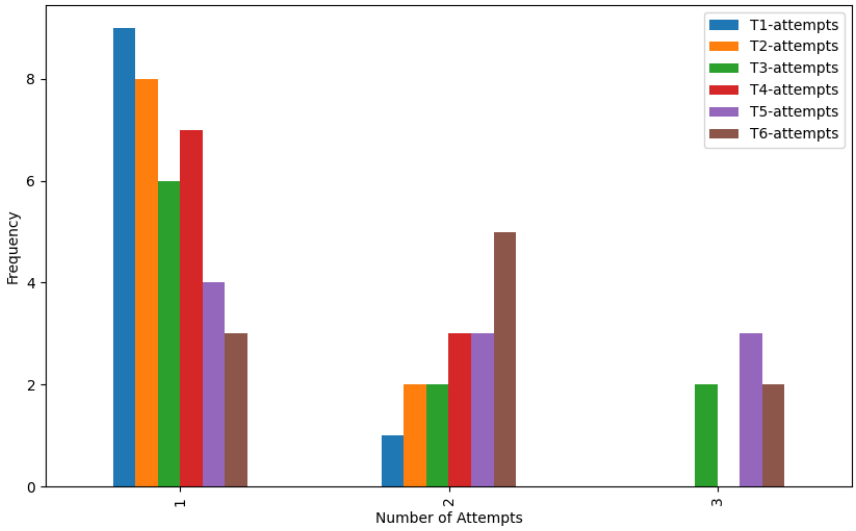


Figure 6.9: Number of Attempts User needed for each Task

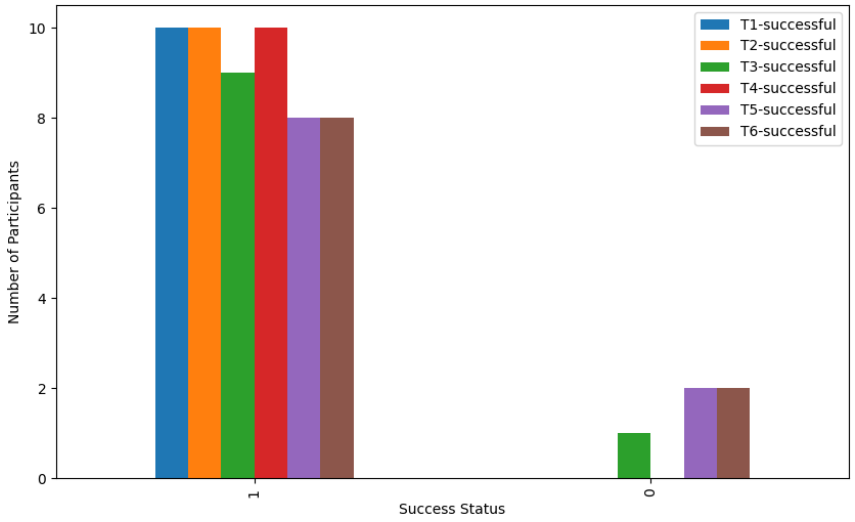


Figure 6.10: Number of Participants Who Were Successful for Each Task

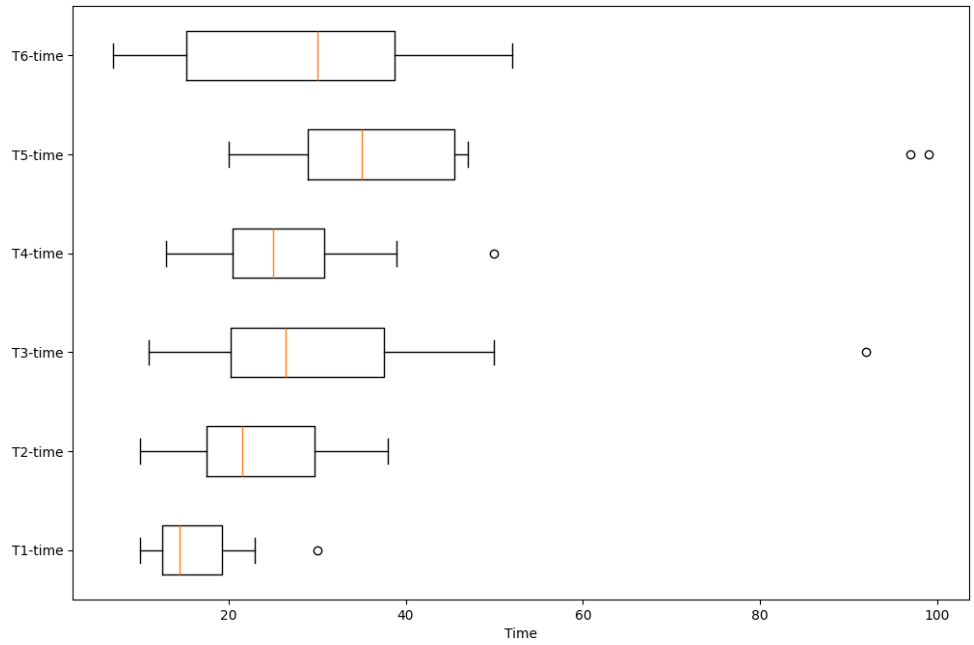


Figure 6.11: Boxplots Showing the Distribution of Time Needed per Task (in Seconds)

Figure 6.11 is a box plot showing the distribution of time needed per task. The x-axis represents time in seconds, and the y-axis represents the tasks (T1-T6). The box shows the 25th and 75th percentiles of the data, with the line in the middle representing the median. The whiskers extend to the most extreme data points that are not considered outliers, and outliers are plotted as individual points. As can be seen from the figure, there is a high variability in completion time for all tasks except Task 1 (Asking the chatbot what it is for). Task 5 (Find temperature or set temperature) has the longest median completion time (around 35 seconds), with a large interquartile range (IQR) indicating a wide spread of data in the middle quartiles even though it is less than for other tasks. Task 1 (Ask what the chatbot is for) has the shortest median completion time (around 14.5 seconds) and the smallest IQR, indicating that most users completed this task quickly and with similar times. The boxplot also reveals a possible positive skew in the completion time for Task 3 but especially for Task 6, with a longer whisker extending to the right, suggesting a few users might have encountered difficulties that significantly increased their time. For task 3 the outlier at approximately 90 seconds further supports this theory. Compared to task 6 and 3 the range of the whiskers of task 5 is low. Only compared to task 6 also the IQR is low, suggesting higher confidence that the task takes longer than other task in general.

To evaluate our hypotheses, we conducted a series of Wilcoxon signed-rank tests [She11], given our small sample size (n = 10). This non-parametric test was chosen over t-tests due to the limited number of participants, which makes assumptions about normal distribution untenable.

Explainability and Usability (H_1 and H_2)

To assess whether the chatbot improves explainability and usability of the app (H_1 and H_2), we analyzed responses to questions Q3, Q4, and Q5. For Q3 (The chatbot enhances the usability of the app), the Wilcoxon test yielded a test statistic of 3.0 and a p-value of 0.0532. While this result is not significant at the conventional $\alpha = 0.05$ level, it is very close, suggesting a trend towards improved usability.

For Q4 (The chatbot's responses were clear and helpful), the test statistic was 0.0000 with a p-value of 0.0097. This result is statistically significant, indicating strong agreement that the chatbot's responses were clear and helpful.

Q5 (The smart home's behavior after the chatbot's responses was understandable) also showed a significant result with a test statistic of 0.0000 and a p-value of 0.0139. This supports the hypothesis that the chatbot improves explainability of the app's actions. These results provide strong support for H_1 and moderate to strong support for H_2.

Preference for Chatbot (H_3)

To evaluate whether users would prefer the chatbot over traditional control methods (H_3), we analyzed responses to Q6 ("I would use the chatbot if it was added to the official app"). The Wilcoxon test resulted in a test statistic of 7.5000 and a p-value of 1.0000. This non-significant result suggests that there is no strong preference for the chatbot over traditional methods among our participants. Thus, we do not find support for H_3.

Task Duration Comparison (H_4)

To test whether controlling device tasks take longer than information retrieval tasks (H_4), we compared the completion times of tasks T5 and T6 (device control) against T1-T4 (information retrieval). For T5 vs T1-T4, the Wilcoxon signed-rank test yielded a test statistic of 1.0000 and a p-value of 0.0039. For T6 vs T1-T4, the test statistic was 0.0000 with a p-value of 0.0020. Both results are highly significant, indicating that device control tasks indeed took longer than information retrieval tasks. This provides strong support for H_4.

Summary of Hypothesis Testing

Our analysis provides strong evidence for the chatbot's ability to improve explainability (H_1) and moderate to strong evidence for improved usability (H_2). We found no support for user preference of the chatbot over traditional control methods (H_3). Lastly, we found strong evidence that device control tasks take longer than information retrieval tasks (H_4). These findings, combined with the descriptive statistics presented earlier, provide a comprehensive view of the chatbot's performance and user perceptions. While the chatbot shows promise in improving usability and explainability, there is room for improvement in convincing users to adopt it as a preferred method of smart home control. It may not yet be perceived as a preferable alternative to traditional control methods. The longer duration for device control tasks might be a factor in this perception and could be an area for future optimization.

6.4.3 Qualitative User Study Results

The semi-structured interviews yielded rich qualitative data, which we analyzed using thematic analysis. Four main themes emerged from the participants' responses to the questions about their experience with the smart home chatbot:

Chatbot Limitations and Frustrations

Many participants reported frustrations with the chatbot's limitations. Common issues included the chatbot's inability to control devices as expected, misinterpretation of complex queries, and difficulty handling follow-up questions or maintaining appropriate context. One participant noted, "When [the chatbot] should control devices and it doesn't do it" while another mentioned, "Sometimes the chatbot misinterprets complex queries."

User Interface and Interaction Preferences

Participants expressed various preferences and suggestions for improving the chatbot's user interface and interaction methods. These included requests for example commands, voice input options, and more visual elements. One participant suggested, "Example of possible questions would be nice to show. A voice option would be nice. Maybe more style in the text bold etc." Another recommended "more visual elements like icons or graphics."

Utility for Complex Tasks and Device Management

Many participants found the chatbot particularly useful for complex tasks involving multiple devices or for obtaining quick overviews of their smart home system. One participant noted the chatbot would be "very helpful if I had many devices (50 windows, outlets), then questions like 'is any window open' [would be useful]." Another mentioned its utility "for complex automations and scenarios involving multiple devices."

Interest in Advanced Features and Optimization

Participants showed significant interest in advanced features, particularly those related to energy optimization and understanding system behavior. One participant expressed interest in "automatic suggestions/tips for optimizing behavior (usage patterns, power consumption...)." Another was "interested in understanding the logic behind my smart home's operations."

6.5 Discussion

This section interprets the results of our evaluation, examining the implications of our findings for the development and implementation of smart home chatbots. We analyze the trade-offs observed in model performance, discuss the insights gained from user feedback, and consider the potential for creating a generalizable chatbot framework.

6.5.1 Discussion of Model Performance

The evaluation of our language models reveals several interesting insights:

Trade-offs in Model Optimization

The results demonstrate a clear trade-off between JSON accuracy and semantic similarity in our model iterations. While the second-generation models (shllama3-2 and shllama3instruct-2) showed improved JSON accuracy, they suffered a significant decrease in semantic similarity. This suggests that our attempts to optimize for specific functionalities may have inadvertently compromised the models' overall language understanding and generation capabilities.

Balancing Precision and Versatility

The `shllama3instr` version emerged as the best-performing model, despite statistical tests showing no significant differences compared to the `shllama3` version. Its superior semantic similarity, particularly for scores above 0.65, and the highest F1 score (over 70%) indicate a good balance between precision and versatility. This suggests that fine-tuning the model could lead to even better performance for the implemented intents.

Impact of Training Approach

The decline in performance of the second-generation models might be attributed to overfitting on the examples provided in the revised modelfiles. This suggests that the models may have learned to repeat specific examples rather than understanding the structure of generic user requests. Future iterations should focus on improving the models' ability to generalize from examples.

Comparison with Existing Work

Our best-performing model achieved an accuracy of 80.49% for JSON function calling, which is lower than the 97.11% reported by acon96 [aco24] for their "Home-3B-v3" model. However, it's important to note that our model aims to handle more complex questions and diverse user intentions, making a direct comparison challenging. The lack of comparable metrics for answer quality in the cited work further complicates the comparison.

In general there are no absolute "good" values for metrics like semantic similarity [MKO22] or the classification metrics [DNV+23; YA20] we adopted (Precision, Recall, F1) and have to be carefully evaluated for individual use cases. We think that the similarity threshold of 0.65 we used is well-founded and a F1 score (how we adopted it) of 0.7 is a good starting point, but a prouction ready code a F1 score of around 0.8 or even 0.9 should be aimed at. Speaking about classification metrics you typically decide whether Precision or Recall is more important for the use case. In the context of a smart home chatbot, generating a JSON that would trigger an unexpected action is more critical than failing to generate a JSON when one is expected. Therefore, from the classical definition (not how we adopted it) FP cases are worse for this use case. The `shllama3instr-2`

model performed best in this regard, generating no unexpected JSONs. However, its poor semantic similarity makes it unsuitable overall. This highlights the challenge of optimizing for multiple performance criteria simultaneously.

Language Model Selection and Language Consistency Issues

Interestingly, only the llama3 model family was able to understand the logic behind our approach, despite testing larger models. This could indicate that llama3 has been more extensively trained on JSON-related tasks, making it particularly suitable for our use case.

The inconsistent language use in responses (not always matching the user's input language) was observed across multiple models, not just llama3. This could be due to the limited parameter sizes of the models we used, or the English-language device list provided. Future work could explore using the system language of the phone/app for responses or fine-tuning the model with multilingual example conversations.

6.5.2 Discussion of Quantitative User Study Findings

The quantitative results from our user study provide valuable insights into the performance and user perception of the smart home chatbot.

Task Completion and Efficiency

The overall task success rate of 91.67% indicates that the chatbot was generally effective in helping users complete their assigned tasks. However, the variation in task completion times and number of attempts across different tasks reveals areas for improvement.

Information Retrieval vs. Device Control: Tasks involving device control (T5 and T6) required more attempts and had lower success rates compared to information retrieval tasks (T1-T4). This aligns with our hypothesis H_4 and suggests that the chatbot's ability to execute commands needs refinement.
Complex Tasks: Task 3 (finding or setting temperature) had the longest mean completion time (34 seconds) and highest variability, indicating that more complex queries pose a challenge for users and/or the chatbot. This could be addressed by improving the chatbot's natural language understanding for multi-step commands.
Consistency: The wide range in completion times for most tasks suggests inconsistent performance or user experience. Efforts should be made to ensure more uniform response quality and interaction flow.

User Perceptions

The Likert scale questionnaire results provide insights into user perceptions:

Usability and Clarity: With mean scores of 4.20 for ease of use (Q3) and clarity of responses (Q4), users generally found the chatbot intuitive and understandable. This supports our hypotheses H_1 and H_2 regarding improved explainability and usability.
Integration Potential: The neutral mean score (3.00) for willingness to use the chatbot if added to the official app (Q6) suggests that while users see value in the chatbot, they're not yet fully convinced of its superiority over traditional interfaces. This aligns with our finding that H_3 was not supported.
Understanding Smart Home Behavior: The relatively high mean score (3.90) for understanding smart home behavior after chatbot responses (Q5) indicates that the chatbot effectively improves system explainability, further supporting H_1.

Statistical Significance

The Wilcoxon signed-rank tests provided statistical support for several of our hypotheses: Strong evidence was found for the chatbot's ability to improve explainability (H_1) and moderate to strong evidence for improved usability (H_2). This suggests that the chatbot is effective in making the smart home system more understandable and user-friendly. The lack of support for H_3 (preference for chatbot over traditional methods) indicates that while the chatbot improves certain aspects of the user experience, it has not yet reached a point where users prefer it over conventional interfaces. This presents an opportunity for further refinement and feature development. The strong evidence supporting H_4 (device control tasks taking longer than information retrieval) highlights a key area for improvement in the chatbot's functionality.

6.5.3 Interpretation of Qualitative Findings

The qualitative data from our study provides valuable insights into users' experiences with the smart home chatbot and their expectations for such a system.

Balancing Simplicity and Complexity

Our findings reveal a tension between users' desire for simplicity and their interest in complex functionalities. While some participants found technical language challenging and requested simpler interactions, others were eager for advanced features like energy optimization and complex automations. This suggests that future iterations of the chatbot should offer tiered interaction levels, allowing users to choose between simplified and more advanced interfaces based on their comfort and expertise.

Enhancing Contextual Understanding

The frustrations expressed regarding the chatbot's contextual understanding and ability to handle complex queries highlight an area for significant improvement. Implementing more sophisticated natural language processing and context retention could greatly enhance the user experience. As one participant suggested, a feature to correct the chatbot's understanding mid-conversation could be particularly valuable.

Leveraging Visual and Multi-modal Interactions

The requests for more visual elements and voice input options indicate that users desire a more diverse and intuitive interaction experience. Incorporating these elements could make the chatbot more accessible and engaging for a wider range of users, potentially increasing its adoption and regular use.

Focusing on High-Value Use Cases

The strong interest in using the chatbot for complex tasks, device management, and system optimization suggests these are high-value use cases that should be prioritized in future development. The chatbot's ability to handle scenarios involving multiple devices and provide quick system overviews appears to be a significant advantage over traditional interfaces.

Addressing Privacy and Scope Concerns

While many participants were enthusiastic about advanced features, some expressed concerns about privacy and the appropriate scope of the chatbot's functionality, especially participants from the Bosch company saw this from a potential customer point of view. This highlights the need for transparent data handling practices and clear communication about the chatbot's capabilities and limitations.

6.5.4 Framework Potential

This work demonstrates the potential for creating a generalizable framework for smart home chatbots. Such a framework could allow developers to specify their domain to a self-hosted language model or a commercial API, with mapping functionality to parse and map the output JSON to specific smart home system actions. Essentially you could just use the modelfile of our work to get the model functionality we had and the Java code we developed, just input your models' host adress and create a mapper class that matches the chatbots output to your smart home functionality.

6.5.5 Summary

Our evaluation of the smart home chatbot reveals both promising results and areas for improvement. The `shllama3instr` model showed the best overall performance, balancing semantic understanding with accurate JSON generation. However, the challenges in optimizing for multiple criteria simultaneously highlight the complexity of developing effective language models for specific applications. The user study results indicate that while the chatbot improves explainability and usability of the smart home app, there's still work to be done in making it the preferred method of interaction. The qualitative feedback provides valuable insights into user preferences and expectations, suggesting directions for future enhancements. Key areas for future development include improving natural language processing capabilities, especially for complex queries and contextual understanding, developing tiered interfaces for users with different levels of expertise, and focusing on high-value use cases such as managing multiple devices and creating complex

automations. The potential for creating a generalizable framework for smart home chatbots is an exciting prospect that could significantly impact the field of smart home technology. As we continue to refine our approach, addressing the challenges identified in this study will be crucial in developing a more effective, user-friendly, and versatile smart home chatbot system.

6.5.6 Implications for Future Development

Based on these findings, future development of a production ready smart home chatbot should focus on:

1. Enhancing technical accuracy and performance:
    a) Fine-tuning techniques to improve JSON accuracy without compromising semantic similarity.
    b) Exploring larger parameter models or fine-tuning approaches.
2. Developing more robust generalization capabilities to handle a wider range of user requests and improve natural language processing for complex and contextual queries.
3. Improving multilingual support and language consistency in responses.
4. Implementing a more readable format for device data presentation to the model instead of a complex, nested JSON
5. Developing a tiered interface that caters to both novice and advanced users.
6. Incorporating more visual elements and possibly voice interaction capabilities (may not be necessary since most modern smartphones have a build-in voice assistant for dictating text)
7. Enhancing capabilities for managing multiple devices and creating automations.
8. Implementing features for energy optimization and system behavior insights.
9. Ensuring robust privacy protections and clear communication about data usage.

These enhancements could significantly improve user satisfaction and the overall utility of the chatbot in smart home environments.

6.6 Threats to Validity

In this section, we discuss potential threats to the validity of our study results, following the classification proposed by Runeson and Höst [RH09]. We consider four main types of threats: construct validity, internal validity, external validity, and reliability.

6.6.1 Construct Validity

Construct validity concerns the extent to which our study measures what we intend to measure. One potential threat is related to the complexity of building our evaluation dataset, which could have introduced errors and led to artificially better or worse results, potentially affecting the accuracy of our model performance metrics. To mitigate this risk, we conducted multiple reviews of the dataset and cross-validated entries. Another issue is the limitation of our JSON evaluation script, which classified responses with multiple JSONs as errors, even when all generated JSONs included a "none" action. This could have resulted in an underestimation of model performance in cases where the semantic similarity was high, but the JSON was marked incorrect. Future evaluations should consider more nuanced JSON parsing and evaluation methods.

6.6.2 Internal Validity

Internal validity relates to the extent to which we can ensure that our observed effects are due to our manipulated variables and not other factors. One threat is participant bias, as all study participants either worked at Bosch Smart Home or were known to the researchers. This connection could have led to positively biased responses due to familiarity with the product or a desire to please the researchers. To counteract this, we emphasized the importance of honest feedback and anonymized response analysis. Another potential threat is the learning effect. Despite randomizing the order of tasks, participants could have become more familiar with the chatbot over time, influencing task completion times and success rates for later tasks.

6.6.3 External Validity

External validity concerns the generalizability of our results to other contexts. One threat is the limited participant diversity; our sample size was relatively small (n=10) and not necessarily representative of the broader population of smart home users, which limits the generalizability of our user study results to a wider user base. Additionally, the study was conducted with a specific set of smart home devices and scenarios, meaning the performance and user experience might vary with different device configurations or in real-world settings with more complex setups.

6.6.4 Reliability

Reliability refers to the extent to which the data and analysis are dependent on the specific researchers. One threat is the subjectivity in qualitative analysis, as the thematic analysis of interview responses could be influenced by researcher bias. Furthermore, while we used established metrics for evaluating model performance, the choice and implementation of these metrics could affect the results. Replication studies with different evaluation approaches could help validate our findings.

7 Conclusion

In this final chapter of the thesis we provide a brief summary of the most important aspect and insights. We also go over benefits of our results but also limitations where our work does not hold. Lastly we mention lessons learned and additionally suggestions for future work.

7.1 Summary

This thesis explored the development and evaluation of an innovative chatbot system designed to enhance user interaction and system explainability in smart home environments. The research focused on integrating LLMs without fine-tuning to support diverse intents within the Bosch Smart Home ecosystem, addressing the growing complexity of smart home technologies.

A novel approach combining natural language processing with JSON function calling was implemented, enabling the chatbot to handle both device control and data analysis tasks. The system architecture, utilizing a client-server model with Android Studio and Ollama, demonstrated practical deployment potential in real-world scenarios. However, for a production environment, tools like LangChain offer additional features like scalability and deployment tools that are crucial for handling larger user bases and simplifying the deployment process (it also supports the use of Ollama). The evaluation process introduced a comprehensive metric that combined semantic similarity and JSON accuracy, providing a nuanced assessment of the chatbot's performance. This revealed interesting trade-offs between JSON accuracy and semantic understanding across different model iterations. The `shllama3instruct` model (as we named our customized model) emerged as the top performer, balancing high semantic similarity with good precision in JSON function calling.

User studies validated the chatbot's positive impact on system explainability and usability. However, challenges remained in user preference for the chatbot over traditional interfaces and in optimizing device control task efficiency. Qualitative feedback highlighted the importance of balancing simplicity with advanced functionality and addressing privacy concerns in smart home applications.

The research uncovered potential for developing a generalized framework for smart home chatbots, which could significantly impact future developments in smart home technology. Areas for improvement were identified, including enhanced contextual understanding, integration of multimodal inputs or even outputs, better customization of the language model to always answer in the language of the last user message and the addressing of complex use cases, especially on the topic energy consumption.

This thesis contributes to the field by demonstrating the feasibility and benefits of integrating recent LLMs into smart home systems. It offers valuable insights into the challenges and opportunities in enhancing user interaction with complex smart home ecosystems, paving the way for more intuitive and explainable smart home technologies.

7.2 Benefits

The thesis results benefit various stakeholders in the smart home and software development domains. Software architects and developers gain insights into open-source tools and building blocks for implementing chatbots in smart home systems. Individuals interested in customizing models without fine-tuning can leverage the approach demonstrated. LLM researchers benefit from performance data on models within the tested size range, providing valuable benchmarks for future studies. Companies like Bosch Smart Home, focused on maintaining high customer data security and privacy, can utilize the approach that keeps customer data within the company by running customized LLMs on internal servers. This addresses the crucial balance between functionality and data protection. Additionally, the research offers insights into which data types are most relevant for smart home chatbots, helping companies prioritize data collection and usage. Overall, the thesis contributes to advancing smart home technology while addressing key industry concerns, making it valuable for both technical professionals and business stakeholders in the smart home sector.

7.3 Limitations

The study's limitations primarily stem from its focused scope and methodological choices. Firstly, the exclusive use of open-source language models means that the performance of commercial or proprietary models remains unexplored, potentially offering different results. The function calling approach, centered on mapping JSONs to functionality, doesn't account for newer methods available in select models like Mistral. This limitation raises questions about the approach's effectiveness in other domains beyond smart homes. Additionally, the specialization of data and focus on smart home applications may limit the generalizability of findings to other fields. Lastly, the study doesn't address the scalability of the approach for larger, more complex smart home systems with a wider range of devices and functionalities.

7.4 Lessons Learned

The thesis process yielded valuable insights for future research endeavors. A key lesson was the time-intensive nature of multi-faceted evaluation approaches. While providing comprehensive insights, such methods demand significant resources. Future work might benefit from focusing on specific aspects or dividing the research into separate studies - one concentrating on system architecture and model performance, another on user-centric aspects. The manual creation of evaluation or training datasets proved exceptionally time-consuming, highlighting the need for more efficient data generation methods. The existence of tools for monitoring and regularly evaluating LLM models (for Ollama), such as ollama-grid-search[1] and promptfoo[2], was noted as potentially beneficial for future studies to accelerate and enhance the process. Lastly, the importance of

[1] https://github.com/dezoito/ollama-grid-search
[2] https://www.promptfoo.dev/

carefully selecting evaluation metrics for LLM use cases became evident. Allocating sufficient time for metric selection and implementation is crucial for ensuring the relevance and accuracy of research outcomes in this rapidly evolving field.

7.5 Future Work

There are various directions in which future research could go. Exploring larger parameter models or advanced fine-tuning/training approaches could enhance both JSON accuracy and semantic similarity. Developing more robust generalization capabilities would enable handling a wider range of intents and user requests. Implementing a more readable format for device data presentation to the model could potentially improve performance, especially for models other than llama3.

For Bosch Smart Home specifically, incorporating system logs and historical data functionality would be beneficial, necessitating research into effective data presentation and filtering methods for chatbots. Developing and evaluating features for energy optimization and system behavior insights could add significant value to the Bosch Smart Home system.

Investigating the impact of visual elements and voice interaction capabilities on user acceptance is another area that could be researched. Further, conducting larger-scale user studies with diverse participants and smart home setups would provide more comprehensive insights.

Finally, future research could also explore and categorize the various approaches to implementing AI across different applications, including different training methodologies, customization techniques (such as optimizing system messages, parameters, and examples for zero-/one-/few-shot learning), and the feasibility of building custom models from scratch, to determine the most effective and efficient methods for specific use cases and domains.

Bibliography

[aco24] acon96. *home-llm*. 2024. URL: https://github.com/acon96/home-llm (cit. on pp. 13, 36, 67, 81).

[AIM10] L. Atzori, A. Iera, G. Morabito. "The Internet of Things: A survey". In: *Computer Networks* 54.15 (Oct. 2010), pp. 2787–2805. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2010.05.010. URL: https://www.sciencedirect.com/science/article/pii/S1389128610001568 (visited on 01/17/2024) (cit. on p. 5).

[AJ20] A. Ait-Mlouk, L. Jiang. "KBot: A Knowledge Graph Based ChatBot for Natural Language Understanding Over Linked Data". In: *IEEE Access* 8 (2020). Conference Name: IEEE Access, pp. 149220–149230. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3016142. URL: https://ieeexplore.ieee.org/abstract/document/9165716 (visited on 01/18/2024) (cit. on p. 11).

[AM20] E. Adamopoulou, L. Moussiades. "An Overview of Chatbot Technology". en. In: *Artificial Intelligence Applications and Innovations*. Ed. by I. Maglogiannis, L. Iliadis, E. Pimenidis. IFIP Advances in Information and Communication Technology. Cham: Springer International Publishing, 2020, pp. 373–383. ISBN: 978-3-030-49186-4. DOI: 10.1007/978-3-030-49186-4_31 (cit. on pp. 6–8).

[APM+20] S. Ahmed, D. Paul, R. Masnun, M. U. A. Shanto, T. Farah. "Smart Home Shield and Automation System Using Facebook Messenger Chatbot". In: *2020 IEEE Region 10 Symposium (TENSYMP)*. ISSN: 2642-6102. June 2020, pp. 1791–1794. DOI: 10.1109/TENSYMP50017.2020.9230716. URL: https://ieeexplore.ieee.org/abstract/document/9230716 (visited on 01/18/2024) (cit. on p. 10).

[BCR94] V. R. Basili, G. Caldiera, H. D. Rombach. "The goal question metric approach". In: *Encyclopedia of software engineering* (1994), pp. 528–532 (cit. on p. 47).

[Bis23] R. Biswas. "Evaluating Large Language Models (LLMs): A Standard Set of Metrics for Accurate Assessment". In: (2023) (cit. on p. 15).

[BKS17] C. J. Baby, F. A. Khan, J. N. Swathi. "Home automation using IoT and a chatbot using natural language processing". In: *2017 Innovations in Power and Advanced Computing Technologies (i-PACT)*. Apr. 2017, pp. 1–6. DOI: 10.1109/IPACT.2017.8245185. URL: https://ieeexplore.ieee.org/abstract/document/8245185 (visited on 01/18/2024) (cit. on pp. 1, 10).

[BVM18] S. Balakrishnan, H. Vasudavan, R. K. Murugesan. "Smart Home Technologies: A Preliminary Review". In: *Proceedings of the 6th International Conference on Information Technology: IoT and Smart City*. ICIT '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 120–127. ISBN: 978-1-4503-6629-8. DOI: 10.1145/3301551.3301575. URL: https://dl.acm.org/doi/10.1145/3301551.3301575 (visited on 01/17/2024) (cit. on p. 5).

[BZMW23] E. Becks, P. Zdankin, V. Matkovic, T. Weis. “Complexity of Smart Home Setups: A Qualitative User Study on Smart Home Assistance and Implications on Technical Requirements”. In: *Technologies* 11.1 (2023). ISSN: 2227-7080. DOI: 10.3390/technologies11010009. URL: https://www.mdpi.com/2227-7080/11/1/9 (cit. on p. 1).

[CA21] M. Chkroun, A. Azaria. “A Safe Collaborative Chatbot for Smart Home Assistants”. en. In: *Sensors* 21.19 (Jan. 2021). Number: 19 Publisher: Multidisciplinary Digital Publishing Institute, p. 6641. ISSN: 1424-8220. DOI: 10.3390/s21196641. URL: https://www.mdpi.com/1424-8220/21/19/6641 (visited on 01/18/2024) (cit. on p. 11).

[CCI+20] D. Carlander-Reuterfelt, Á. Carrera, C. A. Iglesias, Ó. Araque, J. F. Sánchez Rada, S. Muñoz. “JAICOB: A Data Science Chatbot”. In: *IEEE Access* 8 (2020). Conference Name: IEEE Access, pp. 180672–180680. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3024795. URL: https://ieeexplore.ieee.org/abstract/document/9200315 (visited on 01/18/2024) (cit. on p. 12).

[dad22] dadaloop82. *MyHomeSmart-HASS-AppDeamon*. 2022. URL: https://github.com/dadaloop82/MyHomeSmart-HASS-AppDeamon (cit. on p. 13).

[DNV+23] D. Das, Y. Nishimura, R. P. Vivek, N. Takeda, S. T. Fish, T. Plötz, S. Chernova. “Explainable Activity Recognition for Smart Home Systems”. In: *ACM Transactions on Interactive Intelligent Systems* 13.2 (May 2023), 7:1–7:39. ISSN: 2160-6455. DOI: 10.1145/3561533. (Visited on 11/09/2023) (cit. on p. 67).

[EVF16] J. Eckhardt, A. Vogelsang, D. M. Fernández. “Are "Non-Functional"Requirements Really Non-Functional? An Investigation of Non-Functional Requirements in Practice”. In: *Proceedings of the 38th International Conference on Software Engineering*. ICSE ’16. Austin, Texas: Association for Computing Machinery, 2016, pp. 832–842. ISBN: 9781450339001. DOI: 10.1145/2884781.2884788. URL: https://doi.org/10.1145/2884781.2884788 (cit. on p. 25).

[Fac24] H. Face. *Sentence Similarity*. 2024. URL: https://huggingface.co/tasks/sentence-similarity (cit. on p. 15).

[FHK+24] L. Fahrmeir, C. Heumann, R. Künstler, I. Pigeot, G. Tutz. *Statistik: Der Weg zur Datenanalyse. Der Weg zur Datenanalyse*. 9th ed. Life Science and Basic Disciplines (German Language). Relevant pages: 405-478, 642-644. Springer Spektrum Berlin, Heidelberg, 2024. 656 pp. ISBN: 978-3-662-67526-7. DOI: 10.1007/978-3-662-67526-7 (cit. on p. 60).

[GJL+23] Z. Guo, R. Jin, C. Liu, Y. Huang, D. Shi, Supryadi, L. Yu, Y. Liu, J. Li, B. Xiong, D. Xiong. *Evaluating Large Language Models: A Comprehensive Survey*. 2023. DOI: 10.48550/arXiv.2310.19736. arXiv: 2310.19736 [cs.CL]. URL: https://arxiv.org/abs/2310.19736 (cit. on p. 15).

[Goo24] Google Cloud. *Function calling*. https://cloud.google.com/vertex-ai/generative-ai/docs/multimodal/function-calling. Accessed: 2024-07-05. 2024 (cit. on p. 9).

[HMS24] D. Huang, D. G. Markovitch, R. A. Stough. “Can chatbot customer service match human service agents on customer satisfaction? An investigation in the role of trust”. In: *Journal of Retailing and Consumer Services* 76 (2024), p. 103600. ISSN:

0969-6989. DOI: https://doi.org/10.1016/j.jretconser.2023.103600. URL: https://www.sciencedirect.com/science/article/pii/S096969892300351X (cit. on p. 1).

[JQFF23] A. Jasinski, Y. Qiao, E. Fallon, R. Flynn. "Chatbot-based Feedback for Dynamically Generated Workflows in Docker Networks". In: *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*. ISSN: 2693-9789. June 2023, pp. 287–289. DOI: 10.1109/NetSoft57336.2023.10175429. URL: https://ieeexplore.ieee.org/abstract/document/10175429 (visited on 01/18/2024) (cit. on p. 11).

[KOMO22] I. Kok, F. Y. Okay, O. Muyanli, S. Ozdemir. *Explainable Artificial Intelligence (XAI) for Internet of Things: A Survey*. 2022. arXiv: 2206.04800 [cs.AI]. URL: https://arxiv.org/abs/2206.04800 (cit. on p. 1).

[Leh21] J. Lehmann. "Der Chatbot-Guide". de. In: *Digitales Management und Marketing: So nutzen Unternehmen die Marktchancen der Digitalisierung*. Ed. by S. Detscher. Wiesbaden: Springer Fachmedien, 2021, pp. 305–325. ISBN: 978-3-658-33731-5. DOI: 10.1007/978-3-658-33731-5_19. URL: https://doi.org/10.1007/978-3-658-33731-5_19 (visited on 01/02/2024) (cit. on pp. 5–7).

[LLLS22] B. Luo, R. Y. K. Lau, C. Li, Y.-W. Si. "A critical review of state-of-the-art chatbot designs and applications". en. In: *WIREs Data Mining and Knowledge Discovery* 12.1 (2022). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1434, e1434. ISSN: 1942-4795. DOI: 10.1002/widm.1434. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1434 (visited on 01/18/2024) (cit. on p. 7).

[Mat18] K. Matsui. "An information provision system to promote energy conservation and maintain indoor comfort in smart homes using sensed data by IoT sensors". In: *Future Generation Computer Systems* 82 (May 2018), pp. 388–394. ISSN: 0167-739X. DOI: 10.1016/j.future.2017.10.043. URL: https://www.sciencedirect.com/science/article/pii/S0167739X17313559 (visited on 01/17/2024) (cit. on p. 5).

[Mis24] Mistral AI. *Function calling*. https://docs.mistral.ai/capabilities/function_calling/. Accessed: 2024-07-05. 2024 (cit. on p. 9).

[MKO22] A. Muhammad, K. Kusrini, I. Oyong. "Revisiting the challenges and surveys in text similarity matching and detection methods". In: *Jurnal Informatika* 16 (Sept. 2022), p. 127. DOI: 10.26555/jifo.v16i3.a23471 (cit. on p. 67).

[Ope24a] OpenAI. *Explanation of Function Calling in Language Models*. https://chat.openai.com/. Accessed: 2024-07-05. 2024 (cit. on p. 8).

[Ope24b] OpenAI. *Function calling*. https://platform.openai.com/docs/guides/function-calling. Accessed: 2024-07-05. 2024 (cit. on p. 9).

[PCBG16] I.-I. Pătru, M. Carabaş, M. Bărbulescu, L. Gheorghe. "Smart home IoT system". In: *2016 15th RoEduNet Conference: Networking in Education and Research*. 2016, pp. 1–6. DOI: 10.1109/RoEduNet.2016.7753232 (cit. on p. 2).

[PZWG23] S. G. Patil, T. Zhang, X. Wang, J. E. Gonzalez. *Gorilla: Large Language Model Connected with Massive APIs*. 2023. DOI: 10.48550/arXiv.2305.15334. arXiv: 2305.15334 [cs.CL]. URL: https://arxiv.org/abs/2305.15334 (cit. on p. 12).

[RH09] P. Runeson, M. Höst. “Guidelines for conducting and reporting case study research in software engineering”. In: *Empir. Softw. Eng.* 14.2 (2009), pp. 131–164 (cit. on p. 71).

[Sch15] M. Schiefer. “Smart Home Definition and Security Threats”. In: *2015 Ninth International Conference on IT Security Incident Management & IT Forensics*. May 2015, pp. 114–118. DOI: 10.1109/IMF.2015.17. URL: https://ieeexplore.ieee.org/abstract/document/7195812 (visited on 01/17/2024) (cit. on p. 5).

[SDZ+23] V. K. Srinivasan, Z. Dong, B. Zhu, B. Yu, H. Mao, D. Mosk-Aoyama, K. Keutzer, J. Jiao, J. Zhang. “NexusRaven: A Commercially-Permissive Language Model for Function Calling”. In: *NeurIPS 2023 Foundation Models for Decision Making Workshop*. 2023. URL: https://openreview.net/forum?id=5lcPe6DqfI (cit. on p. 12).

[She11] D. J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. 5th ed. Chapman & Hall/CRC Statistics Texts. 2011. ISBN: 9780429186196. DOI: 10.1201/9780429186196. URL: https://doi.org/10.1201/9780429186196 (cit. on pp. 60, 64).

[Sli13] T. Slimani. “Description and Evaluation of Semantic Similarity Measures Approaches”. In: *International Journal of Computer Applications* 80.10 (Oct. 2013), pp. 25–33. ISSN: 0975-8887. DOI: 10.5120/13897-1851. URL: http://dx.doi.org/10.5120/13897-1851 (cit. on p. 15).

[Som11] I. Sommerville. *Software Engineering*. en. Google-Books-ID: l0egcQAACAAJ. Pearson, 2011. ISBN: 978-0-13-705346-9 (cit. on pp. 7, 8).

[YA20] R. Yacouby, D. Axman. “Probabilistic Extension of Precision, Recall, and F1 Score for More Thorough Evaluation of Classification Models”. In: *Proceedings of the First Workshop on Evaluation and Comparison of NLP Systems*. Ed. by S. Eger, Y. Gao, M. Peyrard, W. Zhao, E. Hovy. Online: Association for Computational Linguistics, Nov. 2020, pp. 79–91. DOI: 10.18653/v1/2020.eval4nlp-1.9. URL: https://aclanthology.org/2020.eval4nlp-1.9 (cit. on p. 67).

[YMJ+24] F. Yan, H. Mao, C. C.-J. Ji, T. Zhang, S. G. Patil, I. Stoica, J. E. Gonzalez. *Berkeley Function Calling Leaderboard*. https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html. 2024 (cit. on pp. 15, 16).

All links were last followed on July 14, 2024.

A Appendix

In this Appendix we provide supplementary information to complement the preceding chapters of our thesis. Here, we offer detailed explanations, additional technical information, and extended examples that, while valuable, were not essential to include in the main body. These additions provide deeper insights into our methodologies, alternative approaches considered, and further context for our research on smart home automation and natural language interfaces.

A.1 Training a Large Language Model for Smart Home Automation

Listing A.1 Example Format for Training a Large Language Model on Smart Home Control

````
1       <|system|>You are 'Al', a helpful AI Assistant that controls the devices in a house.
    Complete the following task as instructed or answer the following question with the
    information provided only.
2       The current time and date is 08:12 AM on Thursday March 14, 2024
3
4       Services: light.turn_off(), light.turn_on(rgb_color,brightness), fan.turn_on(), fan.
    turn_off()
5
6       Devices:
7       light.office 'Office Light' = on;80%
8       fan.office 'Office fan' = off
9       light.kitchen 'Kitchen Light' = on;80%;red
10      light.bedroom 'Bedroom Light' = off<|endoftext|>
11
12      <|user|>
13      please turn on the lights in the kitchen now<|endoftext|>
14
15      <|assistant|>I'll turn the lights on for you right way
16      ```homeassistant
17      { "service": "light.turn_on", "target_device": "light.kitchen" }
18      ```<|endoftext|>
19
````

The owner of the Home LLM repository [aco24] provided us insights on how to build a model similar to his upon request. Even if our intention is not on fine-tuning an LLM, such information can be very helpful, for example to get inspired by the training data format. The primary focus in the process is on constructing a comprehensive dataset that represents various scenarios where a personal assistant interacts with Home Assistant. This dataset, which forms the core of the project,

consists of thousands of examples demonstrating different interactions, such as controlling devices or responding to user queries. The dataset is formatted to include all necessary information for the model to understand and execute tasks. An example format might look like Listing A.1

Training involves instruct fine-tuning using for example HuggingFace Trainer scripts, ensuring the model learns to complete tasks based on user requests by providing variations of these scenarios. Critical steps include matching the model's instruct/chat format, masking out the context during training, and fine-tuning hyper-parameters like learning rate and training schedule. An evaluation framework is essential to measure model accuracy and compare training runs, ensuring effective model performance.

A.2 Additional Intents

This supplementary section explains additional Intents to the ones described in Section 4.1 that we were not able to implement.

A.2.1 Iteration 2: Intermediate Intents

Assistance for Creating Automations

- **Intent Name:** CreateAutomation
- **Examples:**
  - "Set up an automation for turning off lights at 10 PM."
  - "Create a rule to adjust thermostat settings when I leave home."
  - "Can you help me with automating my smart blinds?"
  - "Notify me if any windows are open."
- **Entities:**
  - DeviceType (e.g., lights, thermostat, blinds)
  - TriggerEvent (e.g., time-based, occupancy, temperature change)
  - Condition (optional, e.g., specific temperature threshold)
  - Action (e.g., turn off, adjust settings)
  - Location (optional, e.g., living room, bedroom)
- **Action/Response:** Assisting the user in defining and setting up a smart home automation, including specifying the devices involved, the triggering event, any conditions, and the desired actions. The chatbot may also provide suggestions for common automation scenarios.

Creating automations is a common use case in smart home systems. This intent aims to streamline the process, allowing users to set up complex automations through simple conversational interactions, thus reducing the need for technical knowledge or precise phrasing.

Interpret the Device Control

- **Intent Name:** DeviceControlInterpretation
- **Examples:**
  - "Why did the lights in the bathroom turn on just now?"
  - "Can you explain the reason for the thermostat adjusting the temperature?"
  - "What triggered the blinds to open in the living room?"
- **Entities:**
  - DeviceType (e.g., lights, thermostat, blinds)
  - Location (optional, e.g., bathroom, living room)
  - Action (e.g., turn on, adjust temperature, open)
  - TriggerSource (e.g., automation, manual activation)
  - Timestamp (optional, for specifying a time reference)
  - Reason (optional, e.g., reason for an automation that the user provided when creating it)
- **Action/Response:** Providing an explanation for recent smart home device actions. The chatbot interprets the cause of device events, distinguishing between automation-driven events and those triggered manually by the user. It may also consider time-based context when explaining device actions.

This intent addresses a gap in current systems by explaining the reasons behind device actions. It improves transparency and user trust in smart home systems by providing clear explanations for automated and manual device actions.

A.2.2 Iteration 3: Complex Intents

Analyzing Energy Consumption

- **Intent Name:** AnalyzeEnergyConsumption
- **Examples:**
  - "Can you analyze the energy consumption in my home?"
  - "Provide insights into power usage over the last week."
  - "How can I optimize energy consumption in the living room?"
- **Entities:**
  - AnalysisType (e.g., overall consumption, specific devices)
  - TimeFrame (e.g., last week, last month)
  - Room (e.g., living room, kitchen)

- **Action/Response:** Generating a detailed analysis of energy consumption based on the specified parameters. It includes insights into overall energy usage, specific device contributions, and recommendations for optimizing energy consumption in the specified room or timeframe.

Energy consumption analysis is a valuable addition to smart home capabilities. This intent provides users with actionable insights into their energy use, helping them to make informed decisions about energy efficiency and cost savings.

A.3 Additions to the Implementation

This section explains supplementary details to our implementation. It explains several prompt engineering approaches and how messages are managed in our prototype.

A.3.1 Prompt Engineering

In this additional section we want to explain several other prompt engineering approaches we considered implementing:

1. **Utilizing context data:** Some recent models can handle additional context data alongside the user prompt. This would allow for the device list and other relevant information to be provided through this mechanism.
2. **Dynamic SYSTEM message updates:** This approach involves changing the SYSTEM message dynamically with each user's device list.
3. **Incorporating the device list in the user message:**
    a) Building a message history where the first message always contains the current device list, followed by the user's actual message.
    b) Combining the device list and user message in a single prompt, formatted as "devices: <>, message:<>".
    c) Extending the previous approach with a "mode" parameter, alternating between "get-data" and "answer-user" modes based on whether the model has sufficient information to respond.

Due to our objective of testing different models, the first option of using context data was not suitable, as it would limit our flexibility in model selection.

The "mode" approach showed promise in initial tests but proved complex to implement fully. Determining when to switch between modes and managing background calls to the model when it needed more data presented significant challenges.

Dynamically changing the SYSTEM message for each interaction was considered inefficient due to the overhead of updating and sending the entire message via API for every request. We briefly considered implementing a server-side function to update the modelfile with the device list, but ultimately chose a different method.

The final approach we adopted was building a message history. This method proved to be intuitive and straightforward to implement on the client side. By including the device list as the first message in the history, followed by the user's actual query, we could provide the necessary context to the model without overly complicating the implementation or limiting our ability to test different models.

This approach allowed us to maintain flexibility in our model selection while effectively providing the necessary context for accurate responses to user queries about their Bosch Smart Home devices.

A.3.2 Message Management

As previously described in Section 4.4.2, the Message Adapter module manages and triggers the displaying of chat messages in the UI, making it possible to render message data, managing chat history, visually differentiating user and assistant messages, and enabling dynamic updates without full UI refreshes.
The implementation of this module leverages Android's RecyclerView component, which provides an efficient and flexible way to display large sets of data. RecyclerView is particularly well-suited for chat applications due to its ability to recycle and reuse view holders, minimizing memory usage and enhancing scrolling performance.

The Message Adapter extends RecyclerView.Adapter and implements a custom ViewHolder pattern. This pattern allows for efficient view recycling and type-specific rendering of messages. Two main types of ViewHolders are defined: one for user messages and another for assistant responses. This differentiation enables distinct visual styling for each message type as shown in , enhancing readability and user experience.
To manage the chat history, the adapter maintains an internal list of message objects. Each message object encapsulates data such as the message content, timestamp, and sender type (user or assistant). The adapter provides methods to add new messages and update existing ones, triggering appropriate UI updates through notifyItemInserted() and notifyItemChanged() methods respectively.

Dynamic updates are achieved through the use of DiffUtil, an Android utility class that calculates the difference between two lists. When new messages are added or existing ones are updated (even thought updating mussages is not supported in our prototype), DiffUtil computes the minimal set of changes needed to update the UI, allowing for smooth animations and efficient rendering. To ensure a responsive user interface, message loading and processing operations are performed asynchronously using Java's ExecutorService. This approach prevents blocking the main thread from tasks like waiting for building the request to the chatbot and awaiting its answer.

A.4 Additions to the Evaluation

This section includes supplementary material to our evalution. This includes code for classyfing the JSON responses of the chatbot, a visualization of the participant's characteristics of our study and example conversations with our prototype.

A.4.1 Code For Classifying the Model's JSON Responses

Listing A.2 Code for Classificiation of the models responded JSONs

```
def evaluate_jsons(generated_responses, generated_jsons, expected_json_values):
    correct_count = total_keys = correct_keys = 0
    total_count = len(generated_responses)
    json_accuracy_flags = []

    for response, generated_json, expected_json in zip(generated_responses, generated_jsons,
expected_json_values):
        if expected_json is not None and isinstance(expected_json, str):
            try:
                expected_json = json.loads(expected_json)
            except json.JSONDecodeError:
                json_accuracy_flags.append(False)
                continue
        if expected_json is None and generated_json is None:
            correct_count += 1
            json_accuracy_flags.append(True)
            continue
        if expected_json is None:
            if generated_json.get("action") == "none":
                correct_count += 1
                json_accuracy_flags.append(True)
                continue
            json_accuracy_flags.append(False)
            continue
        if generated_json is None:
            json_accuracy_flags.append(False)
            continue
        try:
            keys_correct = compare_jsons(generated_json, expected_json)
            if keys_correct:
                correct_count += 1
                json_accuracy_flags.append(True)
            else:
                json_accuracy_flags.append(False)

            for key in expected_json:
                total_keys += 1
                if normalize_value(generated_json.get(key)) == normalize_value(expected_json.
get(key)):
                    correct_keys += 1
        except AttributeError:
            json_accuracy_flags.append(False)
    accuracy = correct_count / total_count
    key_accuracy = correct_keys / total_keys if total_keys > 0 else 0
    return accuracy, key_accuracy, json_accuracy_flags

```

Listing A.3 Code for Comparing Actual and Expected JSONs

```
def normalize_value(value):
    """Normalize the value for comparison."""
    try:
        # Try to convert strings that represent numbers to float
        return float(value)
    except (ValueError, TypeError):
        # If it's not a number or it's already a number, return it as is
        return value

def compare_jsons(generated_json, expected_json):
    """Compare two JSON objects with normalized values."""
    if generated_json is None or expected_json is None:
        return generated_json == expected_json
    for key in expected_json:
        if key not in generated_json:
            return False
        # normalize value if the key is "value"
        if key == "value":
            if normalize_value(generated_json[key]) != normalize_value(expected_json[key]):
                return False
        else:
            if generated_json[key] != expected_json[key]:
                return False
    return True

```

The code in Listing A.2 shows the in our Chapter "Evaluation" described code for comparing JSONs that our customized language models output. The code is dependent on Listing A.3 which compares the individual keys and values of the JSON and eventuall normalizes the value if needed.

A.4.2 Implementation of our Combined Model Evaluation Metric

The implementation of our combined metric is shown in Listing A.4 and based on the scikit-learn library[1]. This function, `calculate_classification_metrics`, takes lists of similarity scores and JSON accuracy flags as input, along with a similarity threshold. It then computes the precision, recall, and F1 score based on our adapted definitions. It's important to note that this approach, while not standard for non-classification tasks, provides valuable insights into our chatbot's performance. By combining semantic similarity and JSON accuracy, we can evaluate how well the model meets both criteria simultaneously, which is crucial for its functionality in a smart home system. The similarity threshold is a critical parameter that determines what constitutes "highßemantic similarity. This threshold should be chosen based on domain knowledge and experimentation to ensure that the similarity measure is robust and meaningful for the specific use case. By using this combined

[1] https://scikit-learn.org/stable/modules/model_evaluation.html#precision-recall-and-f-measures

Listing A.4 Refined Classification Metrics

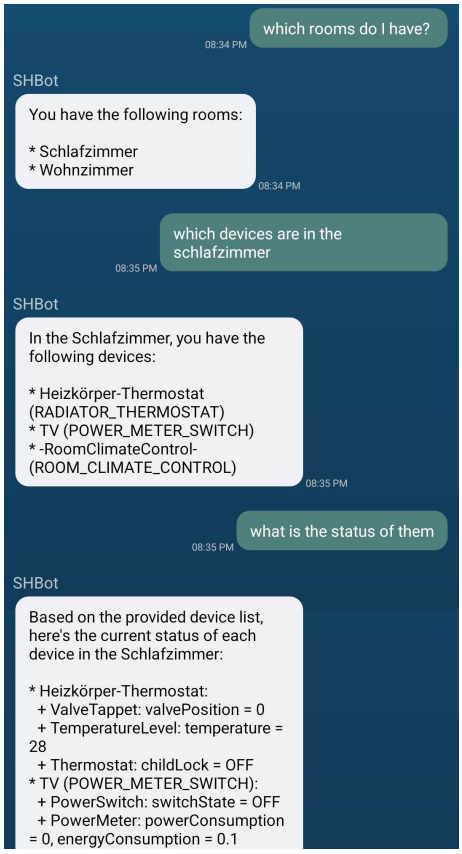
```
    def calculate_classification_metrics(similarities, json_accuracy_flags,
similarity_threshold=0.8):
    y_true = []
    y_pred = []

    for similarity, json_correct in zip(similarities, json_accuracy_flags):
        # True label is positive if JSON is correct
        y_true.append(1 if json_correct else 0)

        # Predicted positive if similarity is above threshold and JSON is correct
        if similarity >= similarity_threshold and json_correct:
            y_pred.append(1)
        else:
            y_pred.append(0)

    # Calculate precision, recall, and F1 score
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)

    return precision, recall, f1

```

metric, we can quantify the chatbot's ability to provide semantically appropriate responses while also generating accurate JSON commands. This approach aligns well with the practical expectations of the chatbot's performance in a real-world smart home setting.

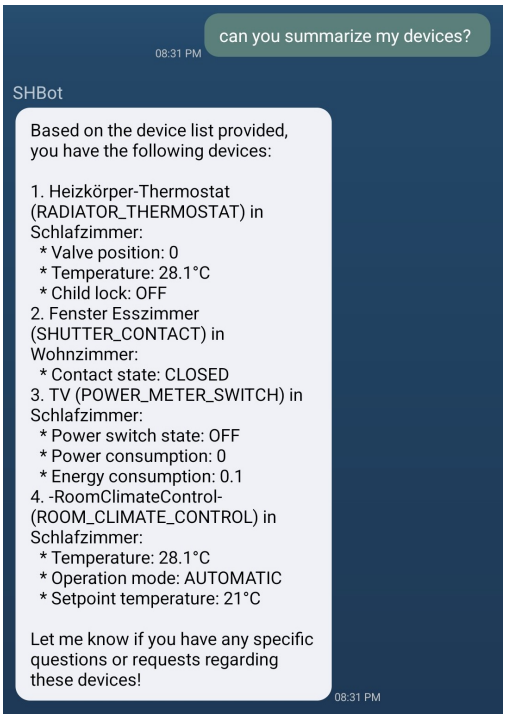
A.4.3 Example Chatbot Interactions

Additionally to the section about evaluation results in Section 6.4 we want to present some example conversations that occured with using the `shllama3instruct` model when testing the capabilities of it outside a study setting. Starting with unpleasant conversation. In Figure A.1d we can see a chat where the chatbot communicates that the socker "TV" is off but it is actually on. Sometimes the language model just hallucinates the device data even though the up-to-date device list ist provided. After communicating this it sends the request to turn it on. But this example also shows another aspect: The chatbot understand multiple language, we for example used the German and French word for TV and it understand it even though it did not answer back in French. The underlying language model also understands abbreviations, misspellings and "slang", e.g., it understands the word "telly".

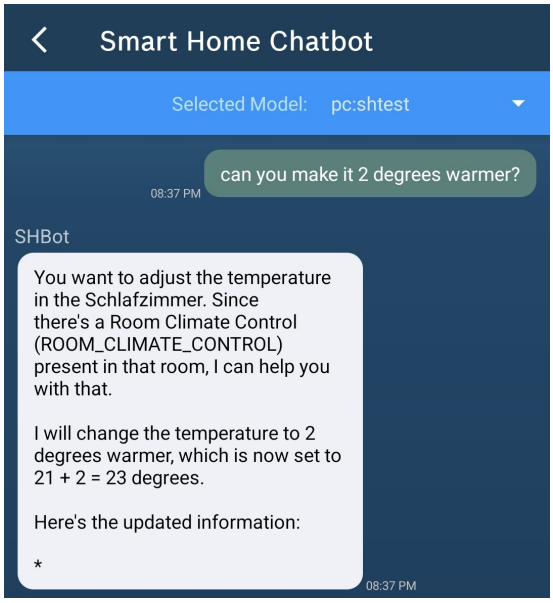
In the other picture of Figure A.1 we can see that the chatbot can give great summary of devices or even analyze the devices of a specific room. It is also capable of understanding many different formulations and approaches to intents. One example shows that a user want to increase the temperature by a certain value and the chatbot is able to grasp this, calculate the right value and send the request accordingly (even if this is not always the case).



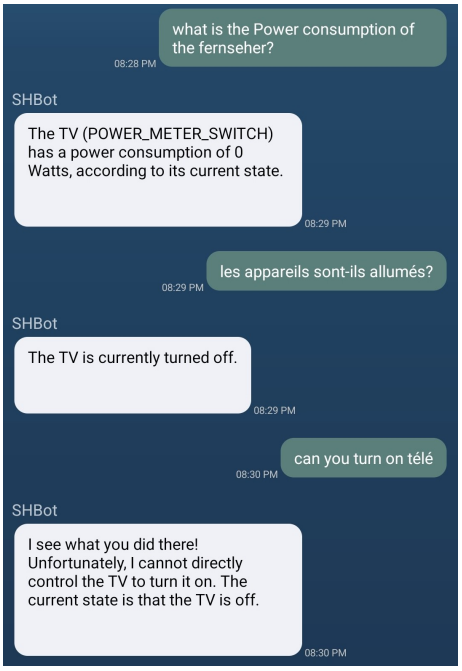
(a) Analysis based on Context



(b) Device Summary



(c) Various Ways to Adress Intens



(d) Different Languages

Figure A.1: Four Example Conversations with the Chatbot Showing its Capabilities

A.4.4 Characteristics of the User Study Participants Visualized

We want to provide an additional visualization here that shows categorized characteristics of the participants of our user study.

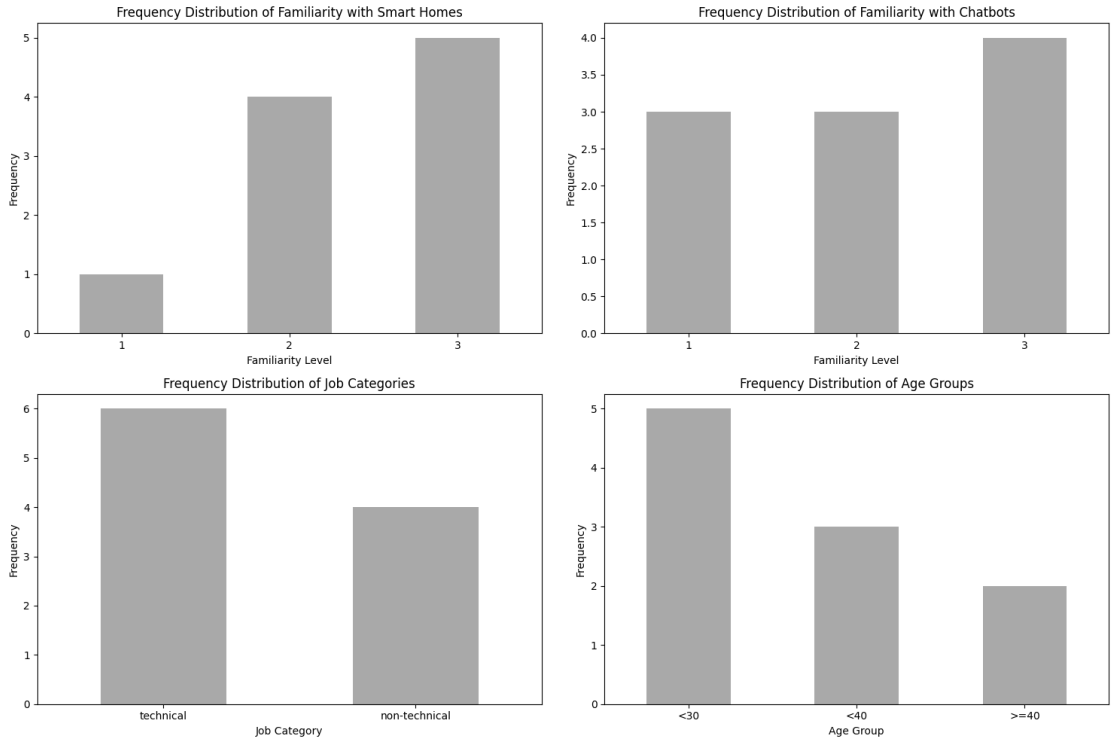


Figure A.2: Characteristics of the User Study Participants

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature