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].

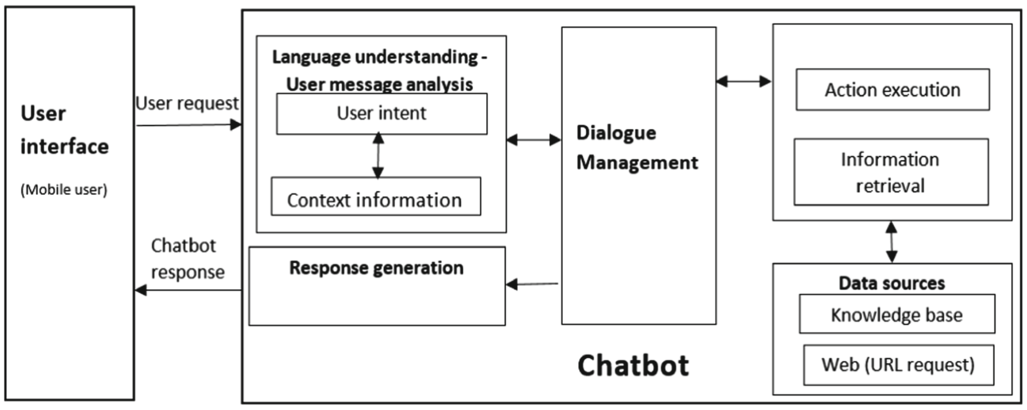
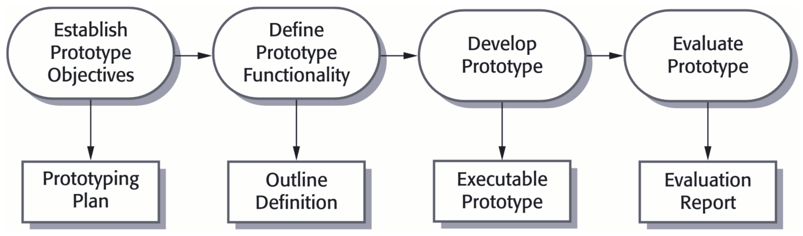
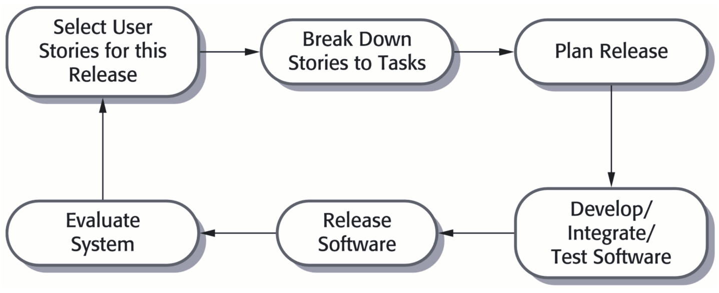


Figure 2.1: An Example Architecture of Chatbots in General
Source: Adamopoulou and Moussiades [AM20]



(a) Prototype Development Process


Select User Stories for this Release
Break Down Stories to Tasks
Plan Release
Develop/ Integrate/ Test Software
Release Software
Evaluate System

(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.

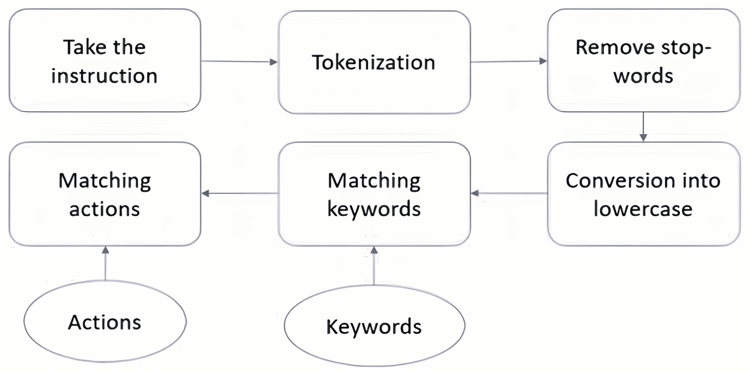


Figure 2.3: A Simple NLP Pipeline for a Chatbot
Source: Baby et al. [BKS17]

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.

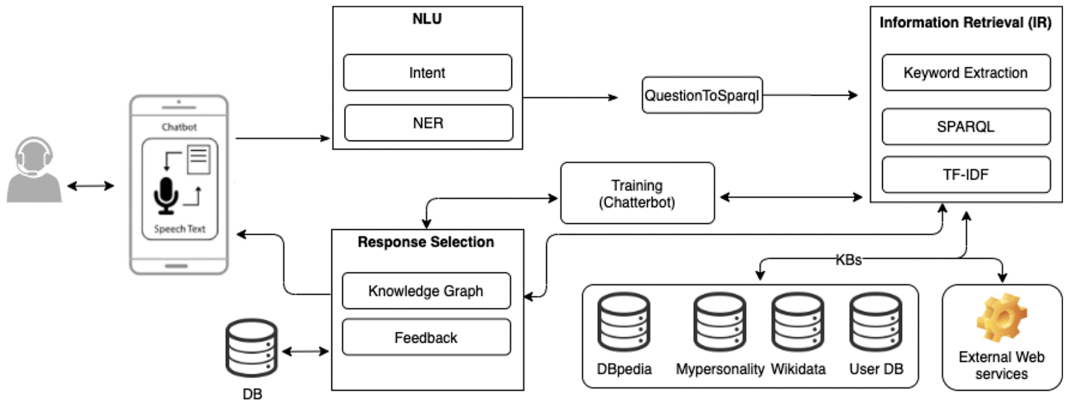


Figure 2.4: Detailed Architecture of KBot
Source: Ait-Mlouk and Jiang [AJ20]

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.

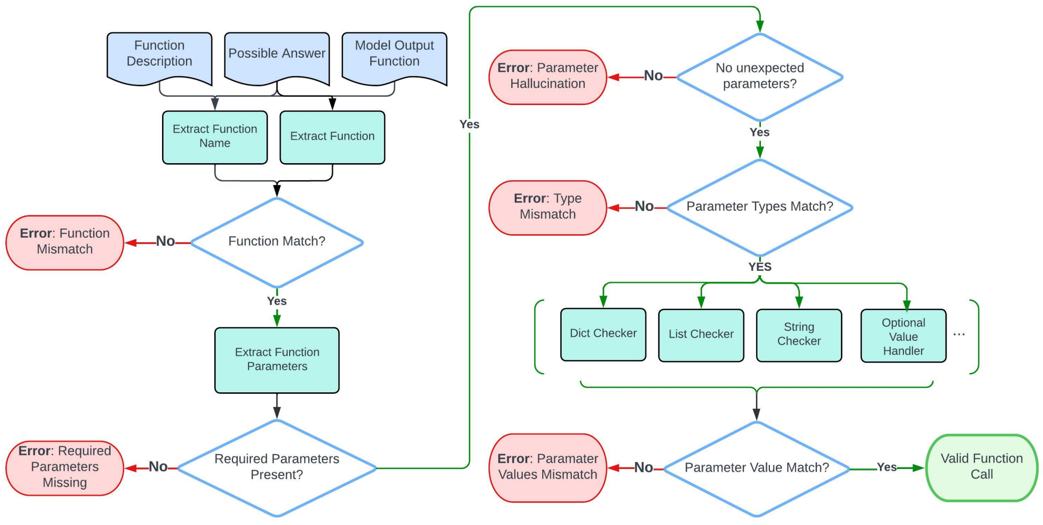


Figure 2.5: Visualization of the AST Evaluation
Source: Yan et al. [YMJ+24]

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.