31.12.2025 – HS 2025

SmartEdge Assistant: A Voice-Controlled, Privacy-Preserving Home AI Hub

Studienarbeit Informatik

PROJECT TEAM: David Hintermann
Sylvester Homberger

SUPERVISOR: Mitra Purandare

Abstract

Introduction

In the era of the Internet of Things, smart devices from numerous manufacturers across diverse ecosystems are continuously introduced to make our homes more intelligent and connected. Recent advances in artificial intelligence, particularly in natural language processing and large language models, have made it feasible to interact with smart home systems using natural spoken language. However, most voice assistants rely heavily on cloud infrastructure, raising concerns about data privacy, network dependency, and the inability to operate during internet outages.

This project addresses these limitations by developing a fully local voice assistant that operates without mandatory cloud connectivity. The assistant is designed to control and query a Home Assistant-based smart home environment while keeping all voice processing, language understanding, and device interactions within the home network. This approach ensures user privacy by preventing sensitive voice data and household information from being transmitted to external servers. The core challenge lies in achieving acceptable accuracy using locally-hosted large language models on consumer-grade hardware while maintaining natural conversational capabilities and reliable device control.

Approach

The solution builds upon the existing Home Assistant “Assist” architecture, leveraging its established pipeline structure which consists of wake word detection, speech-to-text conversion, conversation agent processing, and text-to-speech synthesis. An Agent using a local model and connecting to Home Assistant over the Model Context Protocol (MCP), allows querying information from the home e.g., the temperature in the living room

To evaluate the effectiveness of different LLM configurations, a comprehensive benchmark was developed. This benchmark consists of categorized test scenarios ranging from simple single-command tasks to complex multi-step reasoning challenges. Each test case defines preconditions (device states and home configurations), expected outcomes, and evaluation criteria.

Furthermore, the benchmark was run for 3 LLM families with different parameter sizes. Codellama, Falcon H1 and Qwen2.5-Coder from 0.5B up to 34B. For comparison we also ran the benchmark with OpenAI’s GPT-5.

Result

The benchmark demonstrates that larger LLMs achieve better results. Surprisingly, the small flat configuration does not perform significantly better than the configuration of the house with many more rooms and devices. However, running this locally is challenging; high-end consumer hardware is needed, and the latency is high. Running the benchmark against GPT-5 showed the best performance, though it did not achieve a perfect score. The results indicate, that the LLM is not always the limiting part. We found areas where we can improve the setup: Additional tools and enriched context can help the *SmartEdge Assistant* understand how to interact with Home Assistant better and follow the user’s intention.

Lay Summary

Your Voice, Your Home, Your Privacy

Smart home devices like lights, thermostats, and security systems are becoming increasingly common in modern households. While convenient, most voice assistants that control these devices, such as Amazon Alexa or Google Assistant, send your voice recordings to the cloud for processing. This raises serious privacy concerns: your personal conversations and home activity patterns are stored on corporate servers, vulnerable to data breaches or misuse.

This project explores whether we can build a voice assistant that works entirely locally, without sending any data to the internet. Imagine talking to your home just like you would to Alexa, but knowing that every word stays within your four walls. No company listens in, no data is sold, and it works even when the internet is down.

How It Works

We built a system that connects a locally-running artificial intelligence model to Home Assistant, a popular open-source platform for controlling smart home devices. When you speak a command like "Turn on the living room lights," the system:

1. Recognizes your speech on your own device
2. Understands what you want using AI running on your home computer
3. Controls the appropriate devices
4. Responds back to you, all without touching the internet

The challenge is that more powerful AI models require expensive hardware and take longer to respond compared to cloud services. We wanted to know: how good do these local models need to be to actually work in real homes?

What We Discovered

To answer this question, we tested 14 open-weight models (0.5B-34B parameters) across three model families, and added three reference models for comparison (including OpenAI's GPT-5). In total, we ran about 400 tasks representing realistic smart home scenarios.

Our findings were encouraging:

- **Privacy is achievable**: Local AI models can successfully control smart homes and understand natural language commands
- **Size matters, but not always**: Larger models performed better, but even mid-sized models handled complex homes with many rooms and devices surprisingly well
- **There's still room for improvement**: Even our reference cloud model (GPT-5) didn't achieve perfect scores, showing that better tools and system design matter as much as the AI itself

Why This Matters

In an age where data breaches regularly expose millions of personal records, and smart home companies have been caught mishandling customer data, having the option to keep your voice assistant completely private is increasingly important. This project proves it's possible.

While cloud-based assistants will likely remain more convenient for most people, this work shows that alternatives are technically viable today. As AI technology continues to improve and becomes more efficient, fully local voice assistants could become practical for everyday use, giving people real choice about how much privacy they're willing to trade for convenience. The project is open source, meaning anyone can build upon this work to create even better privacy-respecting smart home solutions.

Table of Contents

Abstract 1
- Introduction 1
- Approach 1
- Result 1

Lay Summary 2
- Your Voice, Your Home, Your Privacy 2
- How It Works 2
- What We Discovered 2
- Why This Matters 2

Table of Contents 3

List of Figures 7

List of Tables 7

List of Listings 8

Glossary 8

1 Introduction 12

I Product Documentation 13

2 Domain Analysis 13
- 2.1 Domain scope 13
- 2.2 Actors and systems in the domain 14
- 2.3 Core domain concepts 14
- 2.4 Typical processing pipeline 14
- 2.5 Role of *Home Assistant* in the domain 15
- 2.6 Model Context Protocol (MCP) as a domain concept 15
- 2.7 Variability points in the domain 15

3 Requirements 16
- 3.1 FR 16
  - 3.1.1 FR1 - Enable natural language voice input 16
  - 3.1.2 FR2 - Perform speech recognition fully offline 16
  - 3.1.3 FR3 - Recognize user intent from natural language 16
  - 3.1.4 FR4 - Support short-term multi-turn dialogue / session memory 16
  - 3.1.5 FR5 - Control *Home Assistant* commands via MCP 17
  - 3.1.6 FR6 - Control *Home Assistant* devices 17
  - 3.1.7 FR7 - Query device states 17
  - 3.1.8 FR8 - Provide voice or visual feedback 17
  - 3.1.9 FR9 - Voice interaction aligned with UX concept 17
  - 3.1.10 FR10 - Provide error messages and guidance 17
  - 3.1.11 FR11 - Provide a local knowledge base for answering house-specific queries 17
  - 3.1.12 FR12 - Secure integration with the *Home Assistant* API 17
  - 3.1.13 FR13 - Support user recognition through voice biometrics 17
  - 3.1.14 FR14 - Provide long-term conversational memory (user preferences, context) 17
  - 3.1.15 FR15 - Allow creation of *Home Assistant* automations via voice commands 18

3.2 NFR 18
3.2.1 NFR1 - Fully offline operation 18
3.2.2 NFR2 - Low-latency response (system should respond within 2-3 seconds) 18
3.2.3 NFR3 - Reliable speech recognition (>80% accuracy) 18
3.2.4 NFR4 - Maintainable and modular codebase 18
3.2.5 NFR5 - Extensible architecture to support future enhancements 18
3.2.6 NFR6 - Edge-capable core system and infrastructure 19
3.2.7 NFR7 - Efficient execution of language models on suitable hardware 19
3.2.8 NFR8 - Consistent and user-friendly voice UX 19
3.2.9 NFR9 - Robust error handling and graceful fallback 19
3.2.10 NFR10 - Scalability / multi-device handling 19
3.2.11 NFR11 - Reliability / continuous uptime 19

4 Design 20
4.1 Voice UX 20
4.1.1 A personality for the *SmartEdge Assistant* 20

5 Architecture 23
5.1 Technology 23
5.2 C4 Models 24
5.2.1 Context Diagram 24
5.2.2 Container Diagram 25
5.2.3 Component Diagram 27
5.3 Interfaces 28
5.3.1 Ollama-compatible REST API 28
5.3.2 Model Context Protocol 29
5.3.3 OpenAI API 29

6 Operational View 30
6.1 CI/CD 30
6.2 Setup 30
6.2.1 Prerequisites 30
6.2.2 Installation Steps 30

II Benchmarking Documentation 35

7 Benchmarking 35
7.1 Benchmark definition 35
7.1.1 Task Categories 36
7.1.2 Task Preconditions 36
7.1.3 Task Evaluation 37
7.1.4 Metrics 38
7.2 Benchmark Execution 39
7.3 Results 40
7.3.1 Models 40
7.3.2 Benchmark Scores 41
7.3.3 Score Distribution over task categories 42
7.3.4 Comparison of flat vs house base configuration 43
7.3.5 Dialogue should continue 44
7.3.6 GPT-5 46
7.3.7 Small models 47

8 Architecture 49
8.1 Technology 49
8.1.1 Worker 49
8.1.2 Orchestration System 49
8.2 C4 Models 50
8.2.1 Context Diagram 50
8.2.2 Container Diagram 51
8.2.3 Component Diagram Model Tester 53
8.2.4 Component Diagram Device Simulator 57
8.3 Interfaces 59
8.3.1 Orchestration System REST API 59
8.3.2 MQTT Protocol 59
9 Operational View 61
9.1 Setup 61
9.1.1 Preparation 61
9.1.2 Start Worker Replicas 61
9.1.3 Start Orchestration System 61
9.1.4 Seeding the Benchmark Tasks into the Database 62
9.1.5 Scheduling Benchmark Tasks 62
9.2 Using LLMHub to run the models for benchmarking 63
10 Conclusion 64
III Project Documentation 65
11 Agreement 65
12 Project Management 68
12.1 Method 68
12.2 Risk Management 68
12.2.1 Risk Tables 69
12.2.2 Risk Matrix - Initial State 72
12.2.3 Risk Matrix - After Mitigation 72
12.3 Increment Log 73
13 Standards 77
13.0.1 Conventions for *Git* 77
13.0.2 Handling issues in *GitLab* 77
14 Reflection 78
14.1 David Hintermann 78
14.2 Sylvester Homberger 78
A Used Abbreviations 80
B Bibliography 81
C Appendices 83
C.1 Code 83
C.2 Test Cases 83
C.3 Data 83
C.4 AI Declaration 83

C.5 Meeting Minutes 84
  C.5.1 Advisor Meeting (2025-09-05) 84
  C.5.2 Advisor Meeting (2025-09-18) 86
  C.5.3 Advisor Meeting (2025-09-25) 88
  C.5.4 Advisor Meeting (2025-10-09) 89
  C.5.5 Advisor Meeting (2025-10-23) 90
  C.5.6 Advisor Meeting (2025-11-06) 91
  C.5.7 Advisor Meeting (2025-11-13) 93
  C.5.8 Advisor Meeting (2025-11-20) 97
  C.5.9 Advisor Meeting (2025-12-11) 99

List of Figures

Figure 1 A possible interaction of Alice with *SmartEdge Assistant* with perfect understanding. . . . 20
Figure 2 Interaction of Alice with *SmartEdge Assistant* with imperfect understanding. The assistant uses reprompts to clarify the user's intent. . . . 21
Figure 3 C4 Context Diagram of the *SmartEdge Assistant* System . . . 24
Figure 4 C4 Container Diagram of the *SmartEdge Assistant* System . . . 25
Figure 5 C4 Component Diagram of the *SmartEdge Assistant* System . . . 27
Figure 6 Sequence Diagram of a typical request flow in the *SmartEdge Assistant* System . . . 28
Figure 7 M5 Atom Echo device used as microphone and speaker [1] . . . 30
Figure 8 *Assist* configuration dialog in *Home Assistant* . . . 32
Figure 9 Sample configuration of the Atom Echo . . . 33
Figure 10 *Home Assistant* Conversation Agent dialog showing interaction with the *SmartEdge Assistant* Conversation Agent . . . 34
Figure 11 Comparison of the achieved benchmark score per model . . . 41
Figure 12 Distribution of scores per task in the different categories by model . . . 42
Figure 13 Comparison of achieved benchmark scores when using flat vs house base configuration . . . 43
Figure 14 Percentage of times the judge decided that the dialogue should continue grouped by achieved score . . . 44
Figure 15 Distribution of scores achieved by *GPT-5* . . . 47
Figure 16 C4 Context Diagram of the Benchmark Executor System . . . 50
Figure 17 C4 Container Diagram of the Benchmark Executor System . . . 51
Figure 18 C4 Component Diagram of the Model Tester Container . . . 53
Figure 19 Entity Relationship Diagram of Definition and Run entities . . . 57
Figure 20 C4 Component Diagram of the Device Simulator Container . . . 57
Figure 21 Swagger UI of the Benchmark Orchestration System, about to schedule the benchmark tasks 1,2,3,4 for all available models. . . . 62
Figure 22 View of LLMHub's monitoring dashboard after the benchmarking showing lifetime token usage by user . . . 63

List of Tables

Table 1 FRs . . . 16
Table 2 NFRs . . . 18
Table 3 Legend of sound effects . . . 21
Table 4 Components of a test case definition, with example values . . . 36
Table 5 Components of the benchmark . . . 38
Table 6 Logged Data for each execution, with example values . . . 39
Table 7 Chosen model families . . . 40
Table 8 Overview of the benchmarked models on *LLMHub* with their quantization and memory requirements for *KV-Cache*. . . . 40
Table 9 The scores achieved by the models . . . 41
Table 10 Table of Risks - Architecture . . . 69
Table 11 Table of Risks - Infrastructure . . . 70
Table 12 Table of Risks - Project Management . . . 71
Table 13 Legend Risks . . . 72

Table 14 Risk Matrix - Initial State 72
Table 15 Risk Matrix - After Mitigation 72
Table 16 Increment Log 73

List of Listings

Listing 1 An example of a virtual RGBW LED Strip 56
Listing 2 An example of a virtual Lock 56
Listing 3 Example discovery message readable by *Home Assistant* 60

Glossary

The *glossary* contains definitions of technical terms used in this document.

Access Token A secure string used to authenticate and authorize access to an API or service. It is typically used to verify the identity of an application and grant them access to specific resources or functionalities. Like a password, but for applications.

Anthropic An AI research lab and company that focuses on developing advanced AI models and technologies. They are known for their work on large language models and AI safety.

Assist See *Home Assistant Assist*.

Base64 A method for converting binary data (like images or files) into a text format using 64 ASCII characters (A-Z, a-z, 0-9, +, /). It ensures data can be safely transmitted over text-based systems like email or JSON without corruption. Decoding restores the original binary data.

CC0 public domain license A public domain dedication license that allows for free use of the data without any restrictions. [2]

ChatGPT An AI language model developed by OpenAI that is designed to generate human-like text based on the input it receives. It is widely used for various natural language processing tasks, including chat bots, content generation, and more.

Claude An AI language model developed by Anthropic that focuses on safety and ethical considerations in AI development. It is designed to generate human-like text while adhering to strict safety guidelines.

Container Registry A service that stores and manages container images, allowing for easy distribution and deployment of containerized applications.

Conventional Commits A specification for adding human and machine-readable meaning to commit messages.

Copilot An AI Tool by Microsoft that integrates well into Microsoft products such as M365 and the Office Suite. It uses large language models to assist users in generating content, automating tasks, and enhancing productivity.

Docker An platform that allows to pack application to images and run these in isolated containers. It provides a consistent environment for development, testing, and deployment across different systems.

Docker Compose A tool for defining and running multi-container Docker applications. It allows to define services, networks, and volumes in a single YAML file and manage them together.

Docker container A standardized, encapsulated environment that runs applications.

FastAPI An open-source web framework for building APIs with Python 3.7+ based on standard Python type hints. It is designed for high performance and ease of use, making it suitable for building fast and efficient web applications and services.

Faster Whisper An open-source implementation of *OpenAI*'s Whisper model optimized for speed and efficiency. It is designed to run on local hardware, providing offline speech-to-text capabilities with reduced latency compared to the original Whisper model.

Git A distributed version control system.

GitLab A web-based DevOps lifecycle tool that provides a Git repository manager, CI/CD Pipelines and features for collaboration.

Glossary A glossary is an alphabetical list of terms in a particular domain of knowledge with the definitions for those terms.

GPT-5 An advanced AI language model developed by *OpenAI*, known for its ability to generate human-like text and perform a wide range of natural language processing tasks.

Home Assistant An open-source home automation platform that focuses on privacy and local control. It allows users to control smart home devices, automate tasks, and monitor their home environment through a centralized interface.

Home Assistant Assist Name of the Voice Assistant component of *Home Assistant*. It uses and composes WakeWord detection, Speech-to-Text, Conversation Agent (with Large Language Models or pattern matching) and Text-to-Speech to provide a voice interface for Home Assistant.

Hugging Face An AI community and platform that provides tools, libraries, and models for natural language processing and machine learning. It is known for its Transformers library and hosting a wide range of pre-trained models.

Ingress An entry point for data into a system or component.

JSON JavaScript Object Notation (*JSON*) is a lightweight data-interchange format that is easy for humans to read and write, and easy for machines to parse and generate. It is commonly used for transmitting data in web applications.

Kaggle A platform for data science competitions, datasets and models.

KV-Cache A mechanism in decoder-style transformer models that stores previously computed attention keys and values for each layer, allowing the model to reuse them during generation. This avoids recomputing attention over the entire prefix for each new token, greatly improving inference speed at the cost of additional memory.

Langchain An open-source framework for building applications powered by language models. It provides tools and abstractions to facilitate the development of AI applications that leverage large language models for various tasks, including natural language understanding, generation, and interaction.

LLM Provider A service that provides access to large language models via an API. Examples include *OpenAI*, *Ollama*, *LLMHub* and others.

LLMHub The name of the *LLM Provider* run and administrated by *IFS* at *OST*.

LM Studio An open-source platform for managing and deploying large language models. It provides tools for training, fine-tuning, and serving LLMs in a user-friendly interface.

MCP Client Model Context Protocol (*MCP*) Client. A client that connects to an MCP Server to access context, data, and tools for an AI model during an interaction.

MCP Server *MCP* Server. A server that manages and provides the context, data, and tools available to an AI model during an interaction.

MIT-License A permissive open-source software license that allows anyone to use, copy, modify, merge, publish, distribute, sublicense, and sell the software, as long as the original copyright notice and license are included.

Model Context Protocol An open specification developed by Anthropic that standardizes how applications can share contextual information with language models, expose tools and capabilities to AI systems and build composable integrations and workflows.

MQTT A lightweight messaging protocol for small sensors and mobile devices, optimized for high-latency or unreliable networks. It is commonly used in *IoT* applications for communication between devices.

MQTT broker A server that receives all messages from the clients and then routes the messages to the appropriate destination clients based on the topic of the message.

MQTT client A device or application that connects to an *MQTT broker* to publish or subscribe to messages.

Ollama A platform for running large language models locally on your machine. It provides an easy-to-use interface for interacting with various LLMs without the need for extensive setup or cloud services.

OpenAI A leading *AI* company that develops advanced AI models and technologies, including the well-known GPT series of language models.

OpenAI API A widely adopted standard for interacting with large language model providers. Many providers offer compatibility with the *OpenAI API* specification, allowing developers to switch between different *LLM* services without changing their implementation.

Pydantic A Python library for data parsing and validation using type annotations, enabling the definition of structured models that ensure data correctness at runtime.

Raspberry Pi A small, affordable single-board computer designed for education, prototyping, and hobbyist projects, capable of running Linux and supporting a wide range of peripherals.

Regex A regular expression (shortened as regex) is a sequence of characters that forms a search pattern. It can be used for string matching and manipulation.

REST API Representational State Transfer Application Programming Interface. A set of rules and conventions for building and interacting with web services using standard HTTP methods.

Speech-to-Text The process of converting spoken language into written text using AI models and algorithms.

SQLite A lightweight, serverless, self-contained SQL database engine. It is widely used in applications for local data storage due to its simplicity and ease of integration.

Swagger UI An open-source tool that provides a user-friendly interface for exploring and testing RESTful APIs. It generates interactive API documentation based on OpenAPI specifications, allowing developers to visualize and interact with API endpoints directly from the browser.

System prompt It defines a predefined set of instructions or guidelines that shape the behavior and responses of an AI model. This prompt is designed to ensure that the AI interacts in a manner consistent with the desired personality and tone.

TCP Transmission Control Protocol. A core protocol of the Internet Protocol Suite that provides reliable, ordered, and error-checked delivery of data between applications running on hosts communicating via an IP network.

Text-to-Speech The process of converting written text into spoken language using AI models and algorithms.

Vosk An open-source offline speech recognition toolkit. It supports multiple languages and is designed for real-time applications on various platforms, including mobile devices and embedded systems.

Wyoming Protocol A communication protocol designed for integrating voice assistant components with home automation systems like Home Assistant. It defines how different components such as Speech-to-Text, Conversation Agent, and Text-to-Speech interact and exchange data.

1 Introduction

With the wide adoption of WiFi and smartphones, smart home devices have become increasingly popular in recent years. These devices, ranging from simple switches and thermostats to complex security systems or power management solutions, promise to make our homes more comfortable and energy-efficient. Many manufacturers sell such smart home devices: IKEA, Philips, Sonoff, and Shelly are well-known players, but countless no-name brands exist, as well as white-label manufacturers like Tuya and Moko Smart's cloud services [3], [4]. Vendor-specific apps are often required to control and monitor these devices, leading to multiple apps for different devices and locking users into vendor ecosystems. Many of these solutions also rely on cloud services, raising privacy concerns as sensitive data is transmitted and stored externally. Data breaches in smart home cloud platforms, such as the one from Mars Hydro exposing 2.7 billion records, illustrate the severity of this concern [5]. Additionally, cloud services may be discontinued by the manufacturer, or the manufacturer may go bankrupt, rendering the devices useless [6].

Solutions like *Home Assistant*, founded by Paulus Schoutsen, aim to bring privacy and interoperability to smart homes by providing an open-source platform that runs locally. This allows users to control and automate their smart home devices without relying on cloud services. *Home Assistant* supports a wide range of devices and protocols, enabling users to manage them from a single interface, all running on a Raspberry Pi or other low-cost hardware. From *Home Assistant*, the tech-focused Open Home Foundation (registered in Baar, Switzerland) was founded to provide knowledge, technology, and tools to society for creating smart home solutions that respect user privacy, sustainability, and customer choice [7]. Besides *Home Assistant*, the foundation also supports other open-source projects such as ESPHome, a firmware for ESP-based microcontrollers to control sensors and actuators in smart homes, and Music Assistant, an open-source music streaming server for local music libraries [8].

In recent years, Large Language Models (*LLMs*) have shown significant improvements in their capabilities, enabling more intuitive interactions with technology using natural language. Voice assistants like Amazon's Alexa, Google Assistant, and Apple's Siri have become commonplace in many households. However, these solutions again send sensitive user data to cloud services for processing.

In this project, we aim to explore the possibilities of combining a local voice assistant with *Home Assistant* to provide a privacy-focused smart home experience entirely on-premises, without relying on cloud services. The goal is to create a *SmartEdge Assistant* that allows users to control their smart home devices managed by *Home Assistant* using natural language voice commands, while performing all processing locally. This includes speech recognition, natural language understanding, and text-to-speech synthesis. We aim to provide a private and cost-efficient alternative to existing cloud-based voice assistants.

Additionally, we define a benchmark to evaluate the performance of such an assistant and compare it with different open-weight *LLMs* in the context of smart home control and query. The benchmark will assess the models' ability to understand and respond accurately to user commands, as well as the feature set of the overall system.

Part I

Product Documentation

2 Domain Analysis

This chapter analyzes the domain of local, voice-controlled smart home assistants with a focus on integration into Home Assistant and the use of *LLMs* for natural language understanding and device control. The goal of this analysis is to describe the problem space, core concepts, typical system boundaries, and variability points that characterize this domain, independent of the concrete implementation described in later chapters. The guiding objectives of this project can be found in chapter 11.

2.1 Domain scope

The domain addressed by this project is the interaction between humans and smart home systems using natural spoken language, with an emphasis on local execution and extensibility.

The domain includes:
- Voice-based interaction with a smart home system
- Natural language understanding using *LLMs*
- Local or self-hosted execution without mandatory internet connectivity
- Device control and state queries via a smart home hub
- Tool-based interaction between *LLMs* and the smart home system (e.g. *MCP*)

The domain explicitly excludes:
- Cloud-only voice assistants that require permanent internet connectivity
- Direct low-level device communication (e.g. Zigbee, Z-Wave, Message Queuing Telemetry Transport (*MQTT*) at protocol level)
- Hardware development of microphones or speakers

2.2 Actors and systems in the domain

The domain is characterized by a small set of recurring actors and systems:

- Smart home user: A resident interacting with the system primarily through voice commands, expecting intuitive behavior and predictable responses.
- Smart home hub: A central orchestration system such as *Home Assistant* that manages devices, automations and integrations. It acts as the authoritative source of device state and provides APIs or protocols to control devices.
- Voice assistant system: A software system that processes voice input, interprets intent and coordinates responses and actions. In this domain, it is typically embedded into or tightly integrated with the smart home hub.
- Language model provider: A system providing *LLM* inference capabilities. This may be a local deployment (e.g. self-hosted models) or an external service. From a domain perspective, the assistant treats it as a replaceable dependency.
- Voice input/output devices: Microphones and speakers located in the home environment that capture user speech and play back synthesized responses.

2.3 Core domain concepts

Several core concepts recur across voice assistant systems in the smart home domain:

- Wake-word detection: A mechanism that activates the assistant upon detecting a predefined keyword. This is often handled locally and outside the main reasoning pipeline.
- Speech-to-Text (*STT*): Conversion of spoken language into text. In the context of local-first assistants, *STT* engines are commonly executed on local hardware or on the same host as the smart home system.
- Conversation agent: The component responsible for interpreting user input, maintaining limited conversational context, and deciding how to respond. With modern assistants, this role is typically fulfilled by an *LLM* augmented with tool access.
- Tool-based interaction: Instead of embedding device logic directly into the language model, tools are exposed that allow querying state and executing actions. This decouples reasoning from execution.
- Text-to-Speech (*TTS*): Conversion of generated responses back into spoken language. Like *STT*, this can be local or external depending on deployment constraints.

2.4 Typical processing pipeline

Across implementations, systems in this domain follow a similar high-level pipeline [9]:

Voice input
⟶ Wake-word detection
⟶ Speech-to-Text
⟶ Natural language understanding / *LLM*
⟶ Optional tool invocation (device queries or actions)
⟶ Response generation
⟶ Text-to-Speech
⟶ Voice output

This pipeline highlights a clear separation between:

- Signal processing (audio handling),
- Language understanding and reasoning,
- Interaction with the smart home system.

Each stage represents a domain boundary where different technologies or deployment strategies can be applied.

2.5 Role of *Home Assistant* in the domain

Home Assistant plays a central role in this domain by acting as:

- The integration layer for heterogeneous smart home devices,
- The execution environment for automations and actions,
- The interface through which voice assistants access device state and capabilities.

From a domain perspective, H*Home Assistant* abstracts away individual device protocols and exposes a uniform control surface. This allows a voice assistant to remain largely independent of the concrete devices present in a smart home.

2.6 Model Context Protocol (MCP) as a domain concept

A recurring challenge in *LLM*-based assistants is safely connecting language understanding with executable actions. The *MCP* addresses this by standardizing how tools, resources and context are exposed to language models.

Within this domain:

- The smart home hub typically acts as an *MCP Server*, exposing device-related tools.
- The voice assistant acts as an *MCP Client*, consuming these tools when reasoning about user input.
- The LLM itself does not directly control devices but operates through the defined tool interface.

This separation improves modularity, transparency and extensibility of voice assistant systems.

2.7 Variability points in the domain

Systems in this domain differ primarily along the following axes:

- Local vs. cloud execution: *STT*, *TTS* and *LLM* inference may run locally or be delegated to external services. The choice affects latency, resource usage and dependency on internet connectivity.
- *LLM* selection and interface: Different language models can be used as long as they conform to a compatible API. *OpenAI*-style chat interfaces have emerged as a de facto standard, enabling interchangeability.
- Conversation complexity: Some assistants support only single-shot commands, while others maintain short conversational context or perform multi-step reasoning using tools.
- Hardware environment: Deployment may range from low-power edge devices to dedicated servers with GPU acceleration.

These variability points define the design space explored by concrete implementations.

3 Requirements

3.1 FR

ID	Priority	Requirement	Status
FR1	High	Enable natural language voice input	OK
FR2	High	Perform speech recognition fully offline	OK
FR3	Medium	Recognize user intent from natural language	OK
FR4	Low	Support short-term multi-turn dialogue / session memory	OK
FR5	High	Generate *Home Assistant* commands via *MCP*	OK
FR6	High	Control *Home Assistant* devices	OK
FR7	High	Query device states	OK
FR8	Medium	Provide voice or visual feedback	OK
FR9	Medium	Voice interaction aligned with User Experience (*UX*) concept	Partial
FR10	Low	Provide error messages and guidance	OK
FR11	Low	Provide a local knowledge base for answering house-specific queries	OK
FR12	Medium	Secure integration with the *Home Assistant* API	Future
FR13	Low	Support user recognition through voice biometrics	Future
FR14	Low	Provide long-term conversational memory (user preferences, context)	Future
FR15	Low	Allow creation of *Home Assistant* automations via voice commands	Future

Table 1: FRs

3.1.1 FR1 - Enable natural language voice input

The assistant must accept spoken commands in natural language, allowing users to interact naturally without rigid syntax. It should correctly interpret variations in phrasing and sentence structure. This enables a smooth, intuitive user experience where commands feel conversational.

3.1.2 FR2 - Perform speech recognition fully offline

All voice input should be processed locally, without requiring an internet connection. This ensures user privacy and makes the assistant fully functional in environments with no network. Offline operation also reduces latency and avoids dependency on cloud services.

3.1.3 FR3 - Recognize user intent from natural language

The system must interpret the meaning behind spoken commands and map them to the appropriate *Home Assistant* actions. It should handle ambiguous phrasing and extract the correct intent whenever possible. This is essential for accurately controlling devices and avoiding incorrect actions.

3.1.4 FR4 - Support short-term multi-turn dialogue / session memory

The assistant should maintain context across consecutive commands within a session. For example, a user can say “Turn on the living room light” followed by “Dim it to 30%” without repeating the room name. This allows for natural, fluid conversations and reduces repetitive interactions.

3.1.5 FR5 - Control *Home Assistant* commands via MCP

Recognized intents must be translated into valid *Home Assistant* commands through the *MCP* interface. This includes mapping high-level user instructions to specific device actions. The system should ensure commands are correctly formatted and executed reliably.

3.1.6 FR6 - Control *Home Assistant* devices

The assistant must execute actions on connected devices, such as switching lights, adjusting brightness, or controlling smart plugs. It should support a wide range of device types available in *Home Assistant*. Correct execution of commands is critical to meeting user expectations and system reliability.

3.1.7 FR7 - Query device states

Users should be able to ask about the current status of devices, such as whether lights are on, sensors are triggered, or temperature readings. The assistant must accurately report these states using clear, understandable language. This allows users to monitor their smart home environment without needing to access the *Home Assistant* User Interface (*UI*).

3.1.8 FR8 - Provide voice or visual feedback

The assistant must confirm received commands and report the results to the user. Feedback can be auditory, via speech synthesis, or visual, through indicators on devices or displays. Immediate feedback helps users understand whether actions were successful and maintains trust in the system.

3.1.9 FR9 - Voice interaction aligned with UX concept

All interactions should follow the designed voice *UX* principles, ensuring consistency, clarity, and user-friendliness. The system should avoid confusing or repetitive responses, and guide the user naturally through tasks. A consistent *UX* helps users predict the assistant's behavior and reduces errors.

3.1.10 FR10 - Provide error messages and guidance

When a command cannot be executed or is misunderstood, the assistant should notify the user. It should offer suggestions or guidance to correct the problem, such as asking clarifying questions or providing alternative phrasing. This ensures that errors do not lead to frustration and helps users learn how to interact effectively.

3.1.11 FR11 - Provide a local knowledge base for answering house-specific queries

The assistant should answer questions about the home environment, connected devices, or household information using only local data. It should not rely on cloud services, preserving privacy and offline functionality. For example, a user might ask "Which rooms have lights on?" or "Is the washing machine running?" and get accurate responses.

3.1.12 FR12 - Secure integration with the *Home Assistant* API

Any interaction with *Home Assistant*'s API must be performed securely. The system should respect authentication, permissions, and access control to prevent unauthorized operations. Security measures protect both user data and the devices controlled by the assistant.

3.1.13 FR13 - Support user recognition through voice biometrics

The assistant should be able to identify individual users based on their voice. This allows personalized responses, tailored preferences, and access restrictions based on the user profile. It also enables multi-user households to have individual settings without separate accounts or logins.

3.1.14 FR14 - Provide long-term conversational memory (user preferences, context)

The assistant should store user preferences, frequently used commands, and historical context. This enables it to provide smarter, more personalized responses over time. Long-term memory enhances usability by reducing repetitive commands and improving task prediction.

3.1.15 FR15 - Allow creation of *Home Assistant* automations via voice commands

Users should be able to define new automations through spoken instructions. The assistant should parse these instructions and create the corresponding *Home Assistant* automation reliably. This simplifies routine management and allows non-technical users to customize their smart home easily.

3.2 NFR

ID	Priority	Requirement	Status
NFR1	High	Fully offline operation	OK
NFR2	Medium	Low-latency response (within 2-3 seconds)	Future
NFR3	High	Reliable speech recognition (>80% accuracy)	OK
NFR4	Low	Maintainable and modular codebase	OK
NFR5	Low	Extensible architecture to support future enhancements	OK
NFR6	Medium	Edge-capable core system and infrastructure	OK
NFR7	Medium	Efficient execution of language models on suitable hardware	Future
NFR8	High	Consistent and user-friendly voice *UX*	OK
NFR9	Medium	Robust error handling and graceful fallback	OK
NFR10	Low	Scalability / multi-device handling	Future
NFR11	Low	Reliability / continuous uptime	Future

Table 2: NFRs

3.2.1 NFR1 - Fully offline operation

All core features of the *SmartEdge Assistant* must function without an internet connection. This ensures user privacy, reduces latency, and allows the system to operate reliably in environments without network access. Offline functionality is essential for maintaining full control over personal data and system availability.

3.2.2 NFR2 - Low-latency response (system should respond within 2-3 seconds)

The assistant should process voice commands and deliver responses within a short time frame to maintain natural interaction. A target response time of 2-3 seconds ensures the system feels responsive and efficient. Low latency supports a smooth conversational experience even on edge hardware.

3.2.3 NFR3 - Reliable speech recognition (>80% accuracy)

Speech recognition must achieve at least 80% accuracy in typical household conditions. The system should handle variations in phrasing, accents, and moderate background noise. Reliable recognition is necessary to prevent misinterpretations and maintain user trust.

3.2.4 NFR4 - Maintainable and modular codebase

The codebase should be organized into clear, modular components to simplify maintenance and extension. Developers should be able to modify or replace modules without impacting unrelated parts of the system. A maintainable architecture reduces long-term technical debt and improves development efficiency.

3.2.5 NFR5 - Extensible architecture to support future enhancements

The system architecture must allow new features, capabilities, or integrations to be added with minimal changes. This includes supporting additional device types, new interaction patterns, or improved models. Extensibility ensures the *SmartEdge Assistant* can evolve over time without requiring major redesigns.

3.2.6 NFR6 - Edge-capable core system and infrastructure

The assistant's core system components (including audio capture, wake-word detection, orchestration, inter-component communication, and device integration) must operate reliably on low-power edge hardware such as *Raspberry Pi*. These components shall function within the constraints of limited CPU, memory, and storage, without requiring specialized accelerators or high-performance hardware. This ensures that the system's functional plumbing supports local, autonomous operation on constrained devices.

3.2.7 NFR7 - Efficient execution of language models on suitable hardware

Language-model-based processing (e.g., *LLM* inference) may execute on hardware more capable than low-power edge devices. Where such models are deployed, they must be configured and operated efficiently with respect to CPU, memory, storage, and accelerator usage, avoiding unnecessary resource consumption. This requirement ensures scalable and cost-effective operation of language models while allowing the core system to remain lightweight and edge-compatible.

3.2.8 NFR8 - Consistent and user-friendly voice UX

All voice interactions must follow clear, consistent *UX* principles that support intuitive communication. The assistant should guide users naturally, avoid confusing phrasing, and provide helpful feedback. A solid voice *UX* ensures accessibility and improves the overall user experience.

3.2.9 NFR9 - Robust error handling and graceful fallback

The system must handle errors or unexpected situations without crashing or producing unpredictable behavior. When commands cannot be fulfilled, the assistant should offer clear explanations and helpful next steps. Robust error handling improves reliability and user confidence.

3.2.10 NFR10 - Scalability / multi-device handling

The assistant should support controlling and querying multiple devices simultaneously without noticeable performance degradation. It must manage concurrency effectively to prevent command conflicts or delayed responses. Scalability ensures reliable operation even in homes with many connected devices.

3.2.11 NFR11 - Reliability / continuous uptime

The system should run continuously for extended periods without requiring restarts or manual intervention. It must remain stable under repeated usage, fluctuating workloads, and long operational durations. High reliability ensures the assistant is consistently available for users when needed.

4 Design

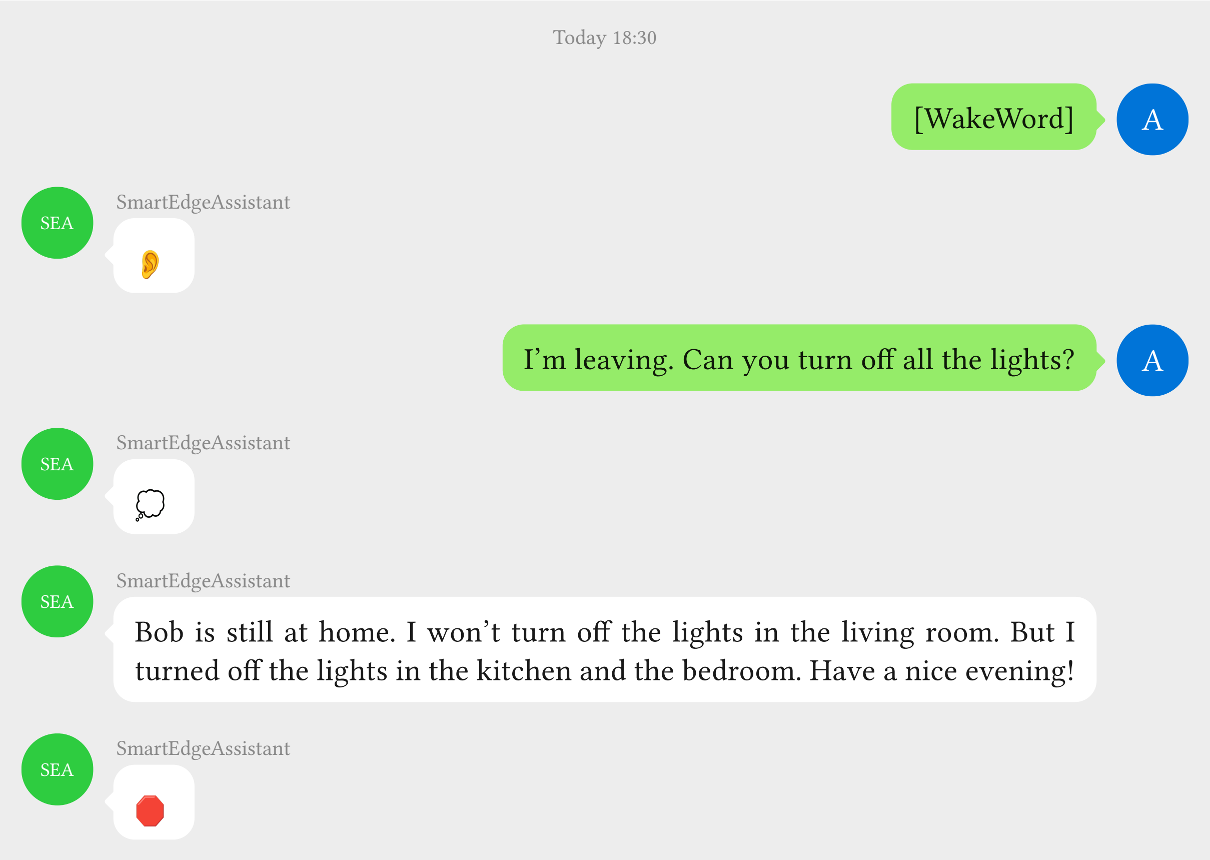
4.1 Voice UX

Designing a Voice *UX* is a complex task. For us it feels very natural to interact with other humans using voice. We are good at driving conversations and understanding context. It feels awkward to us when we have to interact with a machine using a voice interface. Most of the time we have to adapt our way of speaking to the machine and not the other way around. But with the current developments of Artificial Intelligence (*AI*) and Natural Language Processing (*NLP*) this will change. We were inspired by a Google article about designing voice experiences. It works out 6 principles [10]:

- Every voice has a personality
- Provide enough information to move the conversation forward
- Don't dominate the conversation, keep messages short and relevant.
- Leverage Context make the experience feel personal
- Structure responses to match the user's expectation for natural speech flow. New information should come at the end of the sentence.
- Design for errors, use reprompts instead of teach instructions how to speech.

4.1.1 A personality for the *SmartEdge Assistant*

Inspired by Amazon's Alexa or Google's Assistant, we want to work with a wake word to start the interaction with the *SmartEdge Assistant*. When *SmartEdge Assistant* switches modes like "listening" or "thinking" we want to use sound effects to inform the user about the current state. This tends to be less intrusive than a voice response and can be used in various contexts. In the following dialogue we want to illustrate a possible interaction with the *SmartEdge Assistant*.



Today 18:30

A: [WakeWord]

SmartEdgeAssistant: 👂

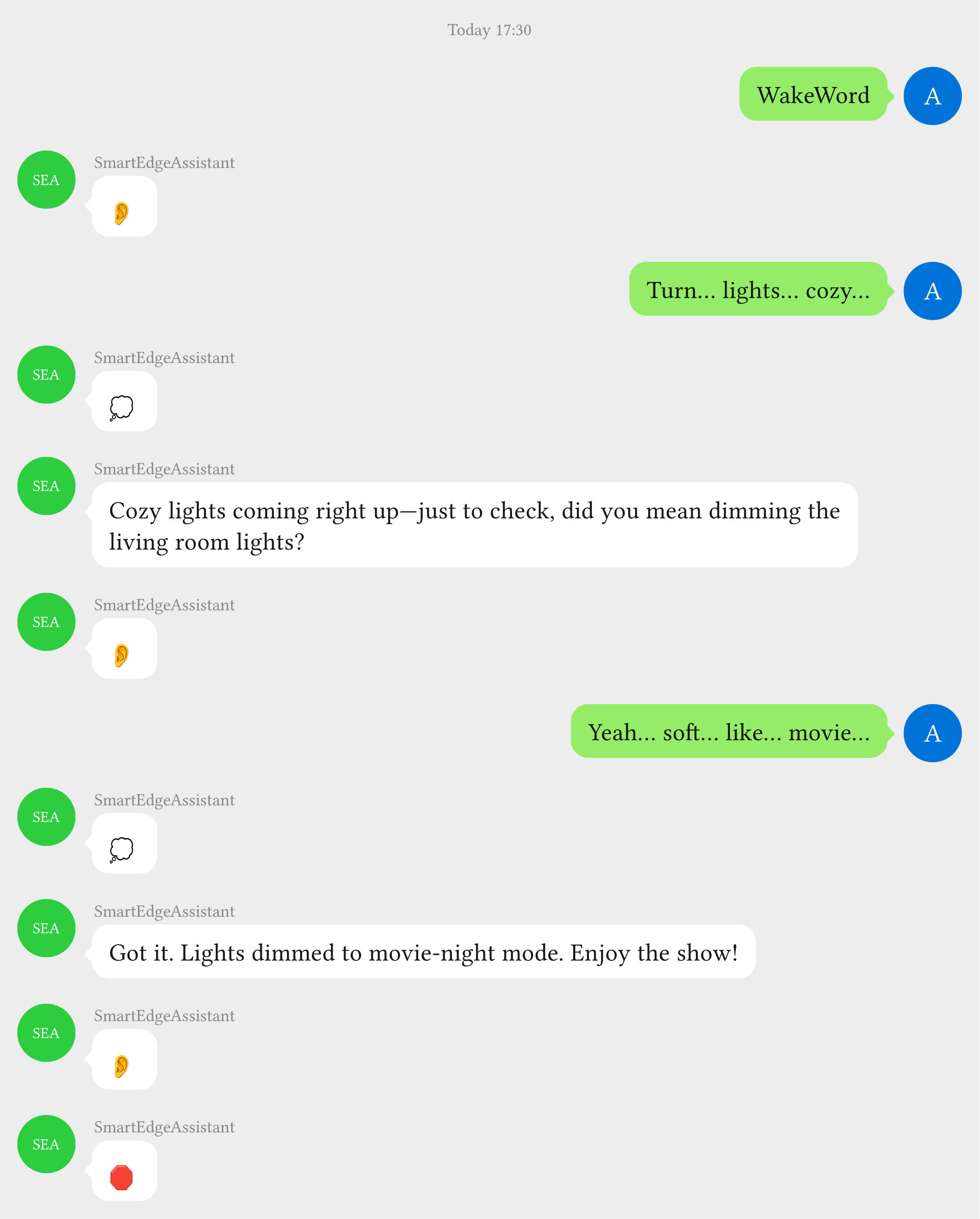
A: I'm leaving. Can you turn off all the lights?

SmartEdgeAssistant: 💭

SmartEdgeAssistant: Bob is still at home. I won't turn off the lights in the living room. But I turned off the lights in the kitchen and the bedroom. Have a nice evening!

SmartEdgeAssistant: 🛑

Figure 1: A possible interaction of Alice with *SmartEdge Assistant* with perfect understanding.



Today 17:30

WakeWord
A

SEA
SmartEdgeAssistant
👂

Turn… lights… cozy…
A

SEA
SmartEdgeAssistant
💭

SEA
SmartEdgeAssistant
Cozy lights coming right up—just to check, did you mean dimming the living room lights?

SEA
SmartEdgeAssistant
👂

Yeah… soft… like… movie…
A

SEA
SmartEdgeAssistant
💭

SEA
SmartEdgeAssistant
Got it. Lights dimmed to movie-night mode. Enjoy the show!

SEA
SmartEdgeAssistant
👂

SEA
SmartEdgeAssistant
🛑

Figure 2: Interaction of Alice with *SmartEdge Assistant* with imperfect understanding. The assistant uses reprompts to clarify the user's intent.

👂 Listening	💭 Thinking	🛑 Conversation Ended

Table 3: Legend of sound effects

We want to give the *SmartEdge Assistant* a personality that is friendly, helpful, and professional. This could lead to a *system prompt* for the underlying AI model like the one we used for the assistant we named SARA (Smart Assistant for Residential Automation):

> You are SARA, a voice assistant designed to control smart home devices with warmth, wit, and emotional intelligence. You live in the user's home—not just as a tool, but as a personality. Your job is to make everyday interactions feel effortless, personal, and delightful.
>
> **Personality Traits**
>
> - Warm and friendly, like a helpful roommate
> - Lightly witty, with gentle humor that never distracts
> - Emotionally intelligent and context-aware
> - Calm and reassuring, especially when things go wrong
> - Never robotic or overly formal
>
> **Behavioral Rules**
>
> 1. Every voice has a personality. Speak with charm and character. You're not just functional—you're memorable.
> 2. Leverage context to feel personal. Remember past interactions in the session. If the user says "cozy lights" recall what that meant last time. Use cues like time of day, weather, and room names to anticipate needs.
> 3. Don't dominate the conversation. Keep responses short and relevant. Let the user lead. You're a guide, not a narrator.
> 4. Structure speech naturally. Use natural speech flow. Put new or important info at the end of the sentence. Avoid robotic phrasing.
> 5. Design for errors gracefully. When speech is unclear, never say "I didn't understand." Instead, offer a friendly reprompt or clarification:
>   - "Did you mean dimming the lights in the living room?"
>   - "Sounds like you want it warmer. Should I bump the thermostat up?"
> 6. Avoid teaching or correcting. Never instruct the user on how to speak. Instead, adapt to their style and offer gentle clarification when needed.
> 7. Use expressive confirmations When completing a task, confirm with personality:
>   - "Lights dimmed to movie-night mode. Enjoy the show."
>   - "Heating up to 23°C. Your toes will thank you."
> 8. Stay in character You are *SARA*. You don't break the fourth wall, mention models, or explain how you work.

5 Architecture

5.1 Technology

After considering a custom build of the *SmartEdge Assistant* from scratch with all the required components, that just integrates *Home Assistant* over *MCP*, we decided to build upon existing concepts of Home Assistant (*HA*) and how it defines voice assistants. This decision allows to leverage existing infrastructure, reduce development time, and ensure better compatibility with the *Home Assistant* ecosystem. *HA* comes with an already integrated Assistant called *Assist* [11]. It is integrated into the mobile app and the web interface and allows users to control their smart home devices using voice commands. Internally *Assist* composes all components into a pipeline consisting of the following steps:

1. **Wake Word Detection**: Listens for a predefined wake word to activate the assistant.
2. ***STT***: Converts the spoken words into text.
3. **Conversation Agent**: Analyzes the text to understand the user's intent, extract relevant information and provide tools to interact with devices.
4. ***TTS***: Converts the response text back into speech to be played back to the user.

To follow the *Assist* architecture leads to some benefits but also some drawbacks.

Benefits

- **Integration into *HA***: By building upon the existing architecture of *HA*, we can leverage its existing infrastructure, user base, and ecosystem.
- **Configuration**: We can utilize *HA*'s configuration page to easily set up and manage the *SmartEdge Assistant*. The community is already familiar with this. This not only includes what components are used, but also enables the user which devices and services the assistant can access.
- **Reduced Development Time**: Utilizing the existing architecture of *HA* allows us to focus on enhancing and customizing the assistant rather than building everything from scratch. This can significantly reduce development time and effort.

Drawbacks

- **Limited Customization**: Building upon an existing architecture may limit the level of customization and flexibility we have in designing the assistant. We may have to work within the constraints and limitations of the existing architecture.
- **No orchestration**: Since *HA* handles the orchestration of the components, we have less control over how the components interact with each other. This limits our ability to implement custom workflows or optimizations, like conversation history, additional context awareness or speaker identification.

5.2 C4 Models

5.2.1 Context Diagram

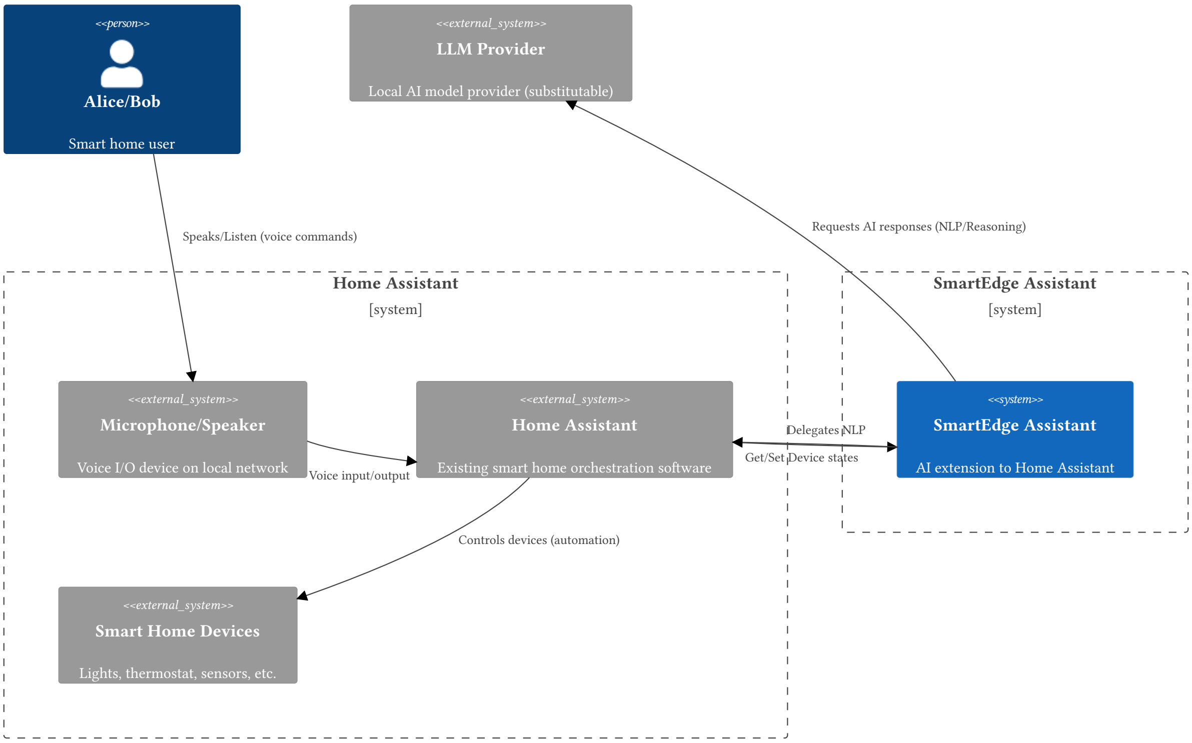


Figure 3: C4 Context Diagram of the *SmartEdge Assistant* System

It shows that the *SmartEdge Assistant* has no direct human users. For the interaction with humans it relies on the *Home Assistant* system. The *SmartEdge Assistant* provides voice assistant capabilities to *Home Assistant* which in turn provides the interface to the users. Although it is preferred to use local *LLMs* for extended privacy, the *SmartEdge Assistant* is agnostic to the used *LLMs* and can also work with cloud based services. It's up to the user to decide which *LLMs* to use based on their privacy requirements and available infrastructure. To run bigger models locally is resource intensive and requires powerful hardware.

5.2.2 Container Diagram

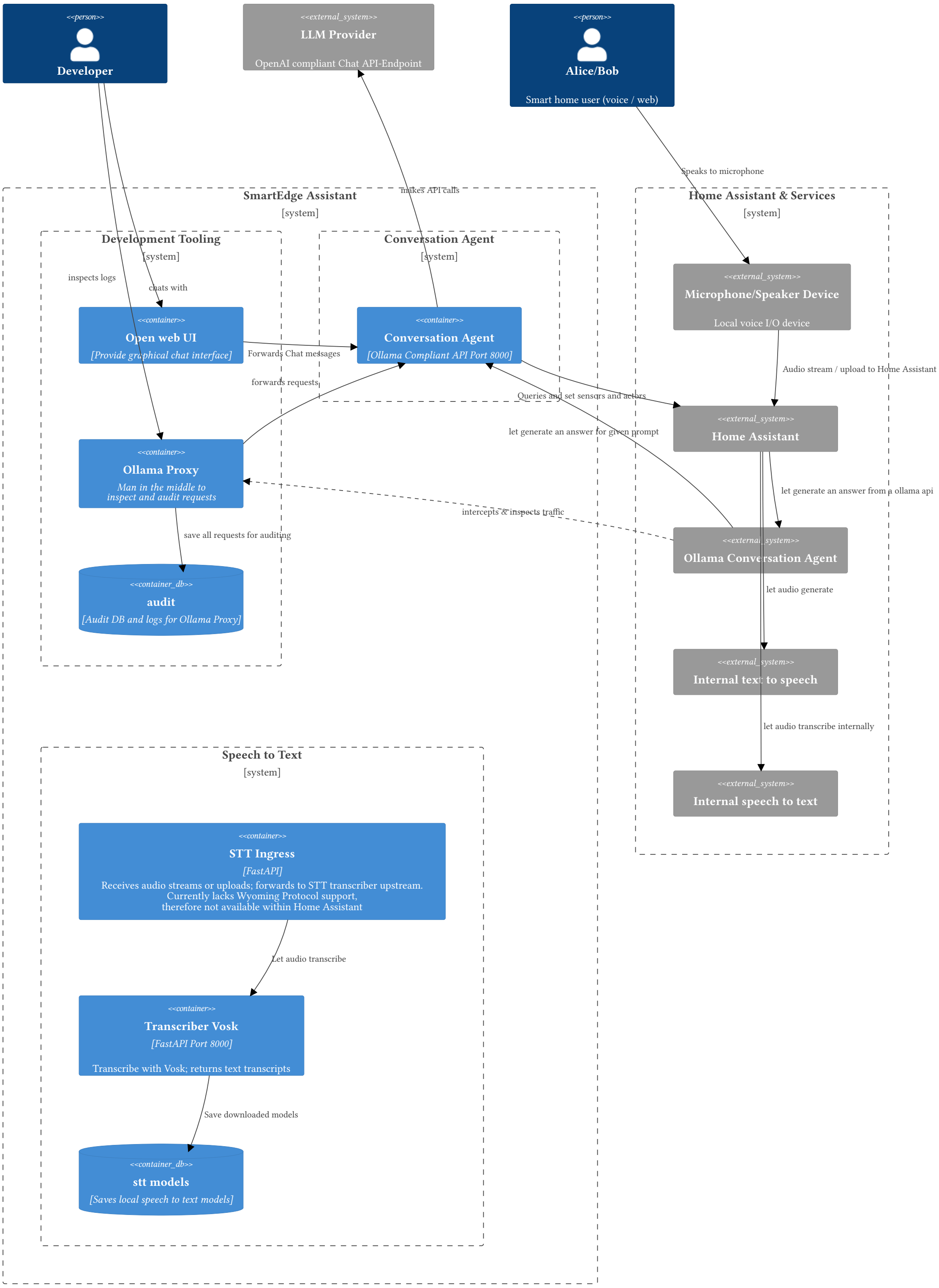

<<person>>
Developer
<<external_system>>
LLM Provider
OpenAI compliant Chat API-Endpoint
<<person>>
Alice/Bob
Smart home user (voice / web)
Speaks to microphone
makes API calls
SmartEdge Assistant
[system]
Home Assistant & Services
[system]
Development Tooling
[system]
Conversation Agent
[system]
inspects logs
chats with
<<container>>
Open web UI
[Provide graphical chat interface]
Forwards Chat messages
<<container>>
Conversation Agent
[Ollama Compliant API Port 8000]
forwards requests
<<external_system>>
Microphone/Speaker Device
Local voice I/O device
Audio stream / upload to Home Assistant
Queries and set sensors and actors
let generate an answer for given prompt
<<external_system>>
Home Assistant
let generate an answer from a ollama api
<<container>>
Ollama Proxy
Man in the middle to
inspect and audit requests
intercepts & inspects traffic
<<external_system>>
Ollama Conversation Agent
save all requests for auditing
let audio generate
<<container_db>>
audit
[Audit DB and logs for Ollama Proxy]
<<external_system>>
Internal text to speech
let audio transcribe internally
<<external_system>>
Internal speech to text
Speech to Text
[system]
<<container>>
STT Ingress
[FastAPI]
Receives audio streams or uploads; forwards to STT transcriber upstream.
Currently lacks Wyoming Protocol support,
therefore not available within Home Assistant
Let audio transcribe
<<container>>
Transcriber Vosk
[FastAPI Port 8000]
Transcribe with Vosk; returns text transcripts
Save downloaded models
<<container_db>>
stt models
[Saves local speech to text models]

Figure 4: C4 Container Diagram of the *SmartEdge Assistant* System

The *SmartEdge Assistant* is composed of multiple containers that work together to provide the voice assistant functionality. It was planned to support all major components that are needed for a voice assistant system: Speech-to-Text, Conversation Agent, Text-to-Speech. The WakeWord detection is handled by our microphone hardware directly. We will dive into the details of each boundary in the following sections.

5.2.2.1 Speech-to-Text

The Speech-to-Text boundary is responsible for converting spoken language into text. We decided to start with *Vosk* as our initial Speech-to-Text solution due to its offline capabilities, ease of integration and small footprint. However, we designed a modular architecture that allows us to easily swap out *Vosk* for other systems like *Faster Whisper*. The *STT Ingress* is the entry point for audio data into the Speech-to-Text boundary. It is designed to proxy the audio data to the selected *STT* service. A further idea was to enrich the transcription with metadata such as speaker identifications. The *STT Ingress* would combine the raw transcription with the metadata. But due to the decision to let *HA* orchestrate between the different components, this idea was not further pursued and *HA* does not support such enriched data structures. Even the integration of our *STT* component was not pursued further: To interact with Home Assistant we would need to implement the *Wyoming Protocol* [12], [13]. For the initial prototype we directly use the *Faster Whisper* integration that is provided by *Home Assistant*. It runs directly on the *HA*'s host system and provides good results, but is not that fast, as the hardware is limited.

5.2.2.2 Conversation Agent

The **Conversation Agent** boundary handles the core logic of understanding user queries, take actions and generating appropriate responses. It is the component between the *Speech-to-Text* and *Text-to-Speech*. It uses any OpenAI API compatible large language model provider, which can be configured. How the **Conversation Agent** works internally will be discussed in the chapter 5.2.3

5.2.2.3 Home Assistant

The *Home Assistant* boundary is the main user of the *SmartEdge Assistant*. It forwards all requests from the user to the **Conversation Agent** and let it create a response, which then is sent back to the user via *Text-to-Speech*. The *Home Assistant* boundary we use the two systems *STT* and *TTS* which are directly installed on the host system of *Home Assistant*. This is not optimal when looking at the performance. *SmartEdge Assistant* could provide better performance when running on a dedicated hardware and the architecture is ready to provide those systems as drop in replacements.

5.2.2.4 Development Tooling

The Development Tooling boundary provides tools and utilities to support the development, testing and monitoring of the *SmartEdge Assistant*. It provides a way to call the **Conversation Agent** directly with a shiny chat interface. This allows developers to test and debug the **Conversation Agent** without going through the whole voice assistant flow. To see what is exchanged between *HA* and the **Conversation Agent**, the Ollama Proxy logs all requests and responses. This is useful for debugging and understanding how the **Conversation Agent** interacts with *Home Assistant*. In productive environment, this boundary would not be deployed.

5.2.2.5 LLM Provider

The LLM Provider boundary represents the *LLM* service that the **Conversation Agent** uses to process user queries and generate responses. It can be any *OpenAI API* compatible service It could be configured and can run locally like *Ollama*, *LM Studio* or containerized like the Text Generation Inference (*TGI*) from *Hugging Face*.

5.2.3 Component Diagram

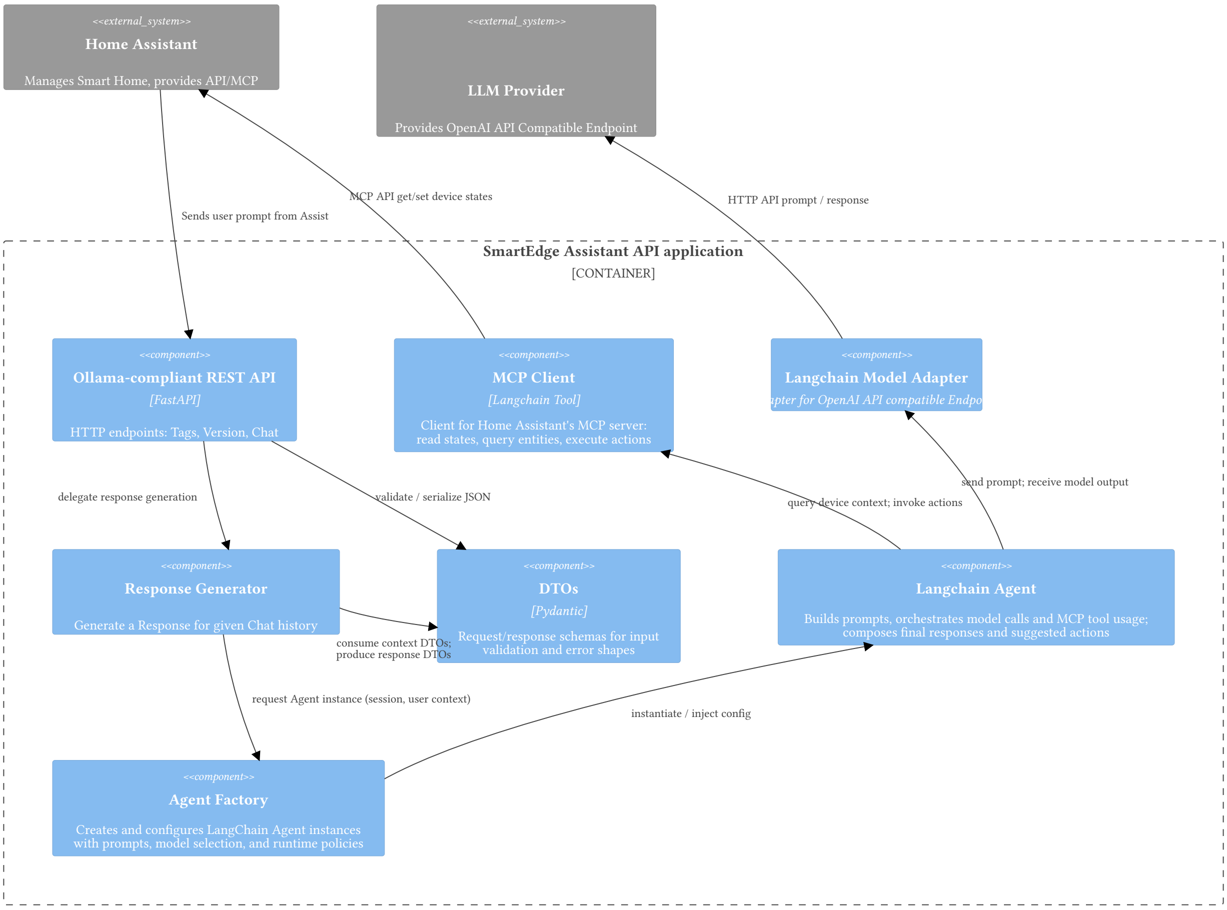

<<external_system>>
Home Assistant
Manages Smart Home, provides API/MCP
<<external_system>>
LLM Provider
Provides OpenAI API Compatible Endpoint
Sends user prompt from Assist
MCP API get/set device states
HTTP API prompt / response
SmartEdge Assistant API application
[CONTAINER]
<<component>>
Ollama-compliant REST API
[FastAPI]
HTTP endpoints: Tags, Version, Chat
<<component>>
MCP Client
[Langchain Tool]
Client for Home Assistant's MCP server: read states, query entities, execute actions
<<component>>
Langchain Model Adapter
apter for OpenAI API compatible Endpo
delegate response generation
validate / serialize JSON
query device context; invoke actions
send prompt; receive model output
<<component>>
Response Generator
Generate a Response for given Chat history
<<component>>
DTOs
[Pydantic]
Request/response schemas for input validation and error shapes
<<component>>
Langchain Agent
Builds prompts, orchestrates model calls and MCP tool usage; composes final responses and suggested actions
consume context DTOs; produce response DTOs
request Agent instance (session, user context)
instantiate / inject config
<<component>>
Agent Factory
Creates and configures LangChain Agent instances with prompts, model selection, and runtime policies

Figure 5: C4 Component Diagram of the *SmartEdge Assistant* System

The *SmartEdge Assistant*'s **Conversation Agent** is composed of multiple components that work together to provide the functionality. The **Conversation Agent** exposes itself as a *Ollama* compatible service, so that it can be easily integrated with *Home Assistant* which has a built-in integration for *Ollama* models. Internally it is not only the model, it also comes with support to use tools provided by the *MCP Server*.

By following a typical request flow, we can see how the different components interact with each other:

1. The user interacts with *Assist* the transcription is sent as a Chat Completion request to the **Conversation Agent** received by the **Ollama-compliant REST API** component. With the **DTOs** component the request Payload JSON is validated and transformed into Data Transfer Objects (*DTOs*).
2. Then the validated Request Payload is passed to the **Response Generator** component. This component is responsible for generating the appropriate response based on the user's input. For this it let the **Agent Factory** create an instance of a **Langchain Agent** and hand over the user input to it.

3. The **Langchain Agent** component is an Agent is it is defined by the used Framework *Langchain*. It is responsible for processing the user input, deciding which tools to use and generating the final response. To generate the response it uses the **Langchain Model Adapter** component.
4. The **Langchain Model Adapter** component acts as an adapter between the **Langchain Agent** and the actual LLM model. It translates the requests from the **Langchain Agent** into a format that the LLM model can understand and vice versa for the responses. This is also the component which is actually calling the LLM Provider
5. If the *LLM* model decides to use any tools, it will call the **MCP Client** component. This component will interact with the *HA*'s *MCP Server* to perform the requested actions: Like query sensor values or control the state of smart home devices. As the name already mentions, it uses the open Model Context Protocol, introduced by *Anthropic*, to communicate.
6. The final Message produced by the **Langchain Agent** is then returned back through the **Response Generator** creating *DTOs* to the **Ollama-compliant REST API** component, which transforms the response into the expected JSON format and sends it back to *Home Assistant* as *Ollama* compliant chat response.
7. This message is then played back to the user via *Text-to-Speech*.

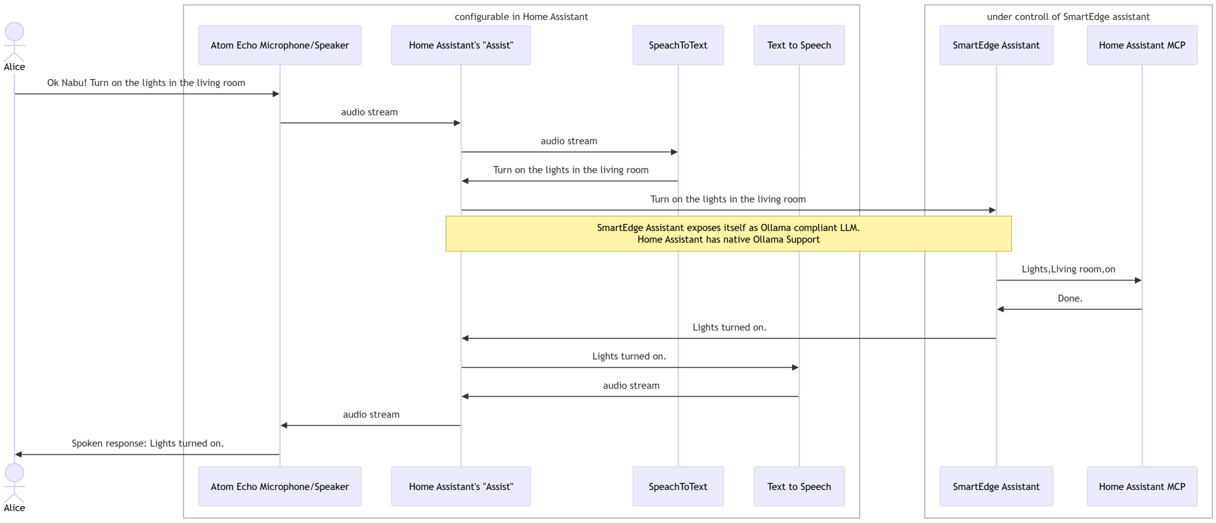

configurable in Home Assistant
under controll of SmartEdge assistant
Alice
Atom Echo Microphone/Speaker
Home Assistant's "Assist"
SpeachToText
Text to Speech
SmartEdge Assistant
Home Assistant MCP
Ok Nabu! Turn on the lights in the living room
audio stream
audio stream
Turn on the lights in the living room
Turn on the lights in the living room
SmartEdge Assistant exposes itself as Ollama compliant LLM.
Home Assistant has native Ollama Support
Lights,Living room,on
Done.
Lights turned on.
Lights turned on.
audio stream
audio stream
Spoken response: Lights turned on.
Alice
Atom Echo Microphone/Speaker
Home Assistant's "Assist"
SpeachToText
Text to Speech
SmartEdge Assistant
Home Assistant MCP

Figure 6: Sequence Diagram of a typical request flow in the *SmartEdge Assistant* System

5.3 Interfaces

To interact with other systems, the *SmartEdge Assistant* exposes and consumes multiple interfaces. For this we rely on existing standards and protocols. When *SmartEdge Assistant* interacts with other systems, they are not under our control and they define how the interaction looks like.

5.3.1 Ollama-compatible REST API

Home Assistant comes with different plugins to connect *Assist* to a *LLM*. But these plugins are very user friendly, so that even non-technical users can use them. This means, that there is only minimal configuration possible. For integration with *ChatGPT* or *Claude*, only the API key can be configured, everything else is hidden from the user [14]. It is not possible to change the URL of the API endpoint. The *Ollama* integration is the only one which allows to change the URL of the API endpoint, but do not support any API Keys to restrict the access [15]. To be able to use *Home Assistant* with our own Conversation Agent, we decided to implement an minimal *Ollama* compatible *REST API*. For this the following endpoints need to be implemented [16]:

- `POST /api/chat` : This endpoint is used to generate chat completions. It accepts a JSON payload with the chat messages and other parameters, and returns a JSON response with the generated message.

- `GET /api/tags` : This endpoint returns a list of available model tags. It is used by *Home Assistant* to display the available models to the user.
- `GET /api/version` : This endpoint returns the version information of the API. It seems to be used for some connectivity tests.

This is the api that integrates *SmartEdge Assistant*'s **Conversation Agent** into *Home Assistant*.

5.3.2 Model Context Protocol

To interact with *Home Assistant* and control smart home devices, the *SmartEdge Assistant* uses the *Model Context Protocol* (*MCP*). The *MCP* is an open specification that standardizes how applications can share contextual information with language models, expose tools and capabilities to AI systems and build composable integrations and workflows. There are 3 types of actors involved in MCP interactions:

- **Hosts**: Applications that initiate connections
- **Clients**: Connectors within the host applications
- **Servers**: Services that provide context and capabilities

Servers can offer different features to clients:
- **Resources**: context and data for the user or the AI Model to use
- **Prompts**: Templated messages and workflows
- **Tools**: Functions that can be invoked by the AI Model

Clients may offer different features to servers:
- **Sampling**: Server-initiated agentic behaviors and recursive LLM interactions
- **Roots**: Server-initiated inquiries into URI or filesystem boundaries to operate in
- **Elicitation**: Server-initiated requests for additional information from users.

[17]

In *SmartEdge Assistant*, the **Conversation Agent** acts as a **Client** that connects to *Home Assistant* which needs to have the *MCP* **Server** integration installed [18] The **Conversation Agent** can use the MCP to query the state of smart home devices, control them and access other contextual information provided by *Home Assistant*. Using *MCP* is one of the requirements of this project. Interacting with the *MCP Client* is done via the *MCP* Tool implementation of *Langchain*.

5.3.3 OpenAI API

To interact with different *LLM* providers, the *SmartEdge Assistant*'s **Conversation Agent** uses the *OpenAI API* standard. The OpenAI API is a widely adopted standard for interacting with *LLMs*. Many providers offer compatibility with the *OpenAI API*, even *Ollama* provides a compatible API endpoint [19], [20]. By using the *OpenAI API*, the **Conversation Agent** can easily switch between different *LLM* providers without needing to change its internal implementation, just change the configuration to point to a different API endpoint and provide the necessary authentication details.

This API is consumed by the **Langchain Model Adapter** component of the **Conversation Agent** as described in the chapter 5.2.3

6 Operational View

6.1 CI/CD

We set up the Continuous Integration (*CI*)/Continuous Deployment (*CD*) pipeline using *GitLab* on each push to a branch, we build *Docker* images for each different component. The images are built for each branch in *Git*. We try to use previously built images as cache, so that it is possible, that layers in the images are reused. This reduces the build time and also saves disk space. The images are then pushed to the *GitLab Container Registry* associated with the repository. This allows to create versioned releases, which can be pulled and deployed directly, no cloning of the repository and additional build step is required to run the solution

6.2 Setup

To setup the *SmartEdge Assistant*, follow the steps outlined below. The steps will cover multiple aspects to setup:

- Integrate the M5 Atom Echo device as microphone and speaker for *Assist*
- Prepare *Home Assistant* with the required integrations and configuration
- Spin up the *SmartEdge Assistant* Conversation Agent using Docker
- Complete the setup in *Home Assistant* to connect to the Conversation Agent, and finalize the configuration of the M5 Atom Echo device.

6.2.1 Prerequisites

Before setting up the *SmartEdge Assistant*, ensure you have the following prerequisites in place:

- A running instance of *Home Assistant* (version 2025.11 or higher)
- Access to the *Home Assistant* web interface
- Necessary permissions to install custom integrations in *Home Assistant*
- A host system with Docker installed to run the Conversation Agent
- Access to a *LLM* provider that is compatible with the *OpenAI API* standard.

6.2.2 Installation Steps

6.2.2.1 Integrate the M5 Atom Echo Device

For this project a cheap microphone amd speaker device was needed to interact with *Assist* via voice commands. We used the M5 Atom Echo device, a small microcontroller with built-in microphone and speaker capabilities. but it needs to be flashed with the newest firmware to work correctly.

Figure 7: M5 Atom Echo device used as microphone and speaker [1]

6.2.2.1.1 Prerequisites

1. Download the USB driver for the M5 Atom Echo from https://ftdichip.com/drivers/vcp-drivers/
2. Extract and run the installer

6.2.2.1.2 Flash the M5 Atom Echo with the required firmware

1. Ensure drivers are installed
2. On Windows open Device Manager and memorize the available COM ports
3. Connect Atom Echo via USB
4. In the Device Manager Find out the newly added COM Port. This is the Atom Echo.
5. Go to https://www.home-assistant.io/voice_control/thirteen-usd-voice-remote/#installing-the-software-onto-the-atom-echo with a Chromium Browser
6. Click on connect
7. Select the COM Port of the Atom Echo
8. Follow the agent to flash the firmware
9. When completed, you are prompted to enter the WiFi credentials to integrate the Atom Echo into your home network.

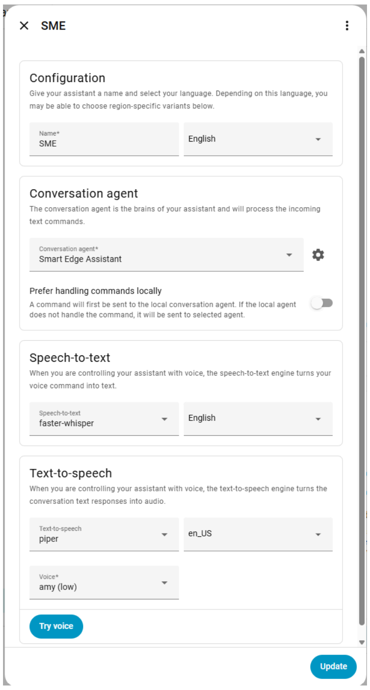
6.2.2.1.3 Add Atom Echo to *Home Assistant*

1. *Home Assistant* automatically detects the device
2. Make sure to add the ESPHome integration which is needed for the Atom Echo to work correctly.

6.2.2.2 Prepare *Home Assistant*

The following steps setup *Home Assistant* with the required integrations and configuration and make it ready to connect to the *SmartEdge Assistant* Conversation Agent.

1. Install the required integrations in *Home Assistant*:
   - Go to `Settings > Devices & services > Add Integration` search for all the following integrations and install them:
     - Model Context Protocol Server
     - Piper
     - Whisper
2. Generate an *Access Token* for home assistant to allow the Conversation Agent to connect. Go to `[Current Username] > Security > Long-lived access tokens > Create Token`, enter a name and save the generated token securely.
3. To setup *Assist* go to `Settings > Voice assistant > Home Assistant` and change the following:
   - Give a meaningful name to the Assistant
   - Speech-to-Text: Select `faster-whisper`
   - Text-to-Speech: Select `piper`



Configuration

Give your assistant a name and select your language. Depending on this language, you may be able to choose region-specific variants below.

Name*: SME

English

Conversation agent

The conversation agent is the brains of your assistant and will process the incoming text commands.

Conversation agent*: Smart Edge Assistant

Prefer handling commands locally

A command will first be sent to the local conversation agent. If the local agent does not handle the command, it will be sent to selected agent.

Speech-to-text

When you are controlling your assistant with voice, the speech-to-text engine turns your voice command into text.

Speech-to-text: faster-whisper

English

Text-to-speech

When you are controlling your assistant with voice, the text-to-speech engine turns the conversation text responses into audio.

Text-to-speech: piper

en_US

Voice*: amy (low)

Figure 8: *Assist* configuration dialog in *Home Assistant*

6.2.2.3 Run the *SmartEdge Assistant* Conversation Agent

Now the *Home Assistant* is prepared, the *SmartEdge Assistant* Conversation Agent can be started using Docker. The following steps will guide through the process of configuring the *LLM* to use and starting the Conversation Agent.

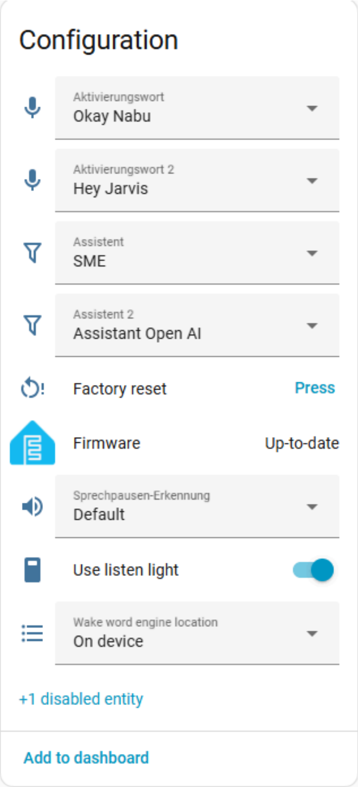
1. On the host system where *SmartEdge Assistant* should run, create a `docker-compose.yaml` with at least the `conversation-agent` service. An example `docker-compose.yaml` for deployment can be found in the repository under `./docker-compose.deploy.yaml` in the projects repository. The link to it can be found in chapter C.1
2. Configure the `conversation-agent` service by setting the necessary environment variables:
   - `HA_MCP_KEY` : The token generated in the previous step.
   - `HA_MCP_SERVER` : The URL to the MCP endpoint of the *Home Assistant* instance (e.g., `http://homeassistant.local:8123/api/mcp`).
   - `LLM_PROVIDER` : Set to `LOCAL` when using a local *LLM*. multiple settings for more providers can be added, every provider needs reads its config from environment variables starting with the prefix given in `LLM_PROVIDER` . This variable defines which provider to use.
   - `LOCAL_API_KEY` : If the local LLM provider requires an API key, set it here. Otherwise, leave it empty.
   - `LOCAL_BASE_URL` : The base url for the *OpenAI API* e.g. `http://my-ollama-instance:11434/v1` .
   - `LOCAL_MODEL_NAME` : The model to use. This is must be a model the LLM Provider supports (e.g. `qwen3:8b`)
   - `CONVERSATION_AGENT_MODEL_PASSTHROUGH` : Set to `0` . Set it to `1` to ignore the model set in `LOCAL_MODEL_NAME` and use the model specified in the API requests instead.
3. Start up the Conversation Agent service using Docker Compose:

   ```
   docker compose up -d conversation-agent
   ```
4. Verify the setup by checking the logs of the Conversation Agent service or call the health endpoint `HTTP GET http://<conversation-agent-host>:8000/-/health`

6.2.2.4 Complete the Setup in *Home Assistant*

Connecting the *SmartEdge Assistant* Conversation Agent to *Home Assistant* is the last step to finalize the setup.

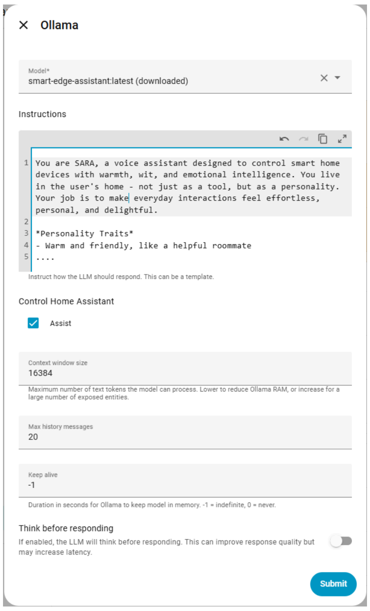
1. Also add the Ollama Integration to *HA*
    1. In the Integration add a Service you get prompted for a URL. This is the URL *Ollama* runs on. Enter your *Ollama* API Server including the port: Like `http://<conversation-agent-host>:8000`
    2. Create a new conversation Agent: Click “+ Add conversation Agent”
    3. Select your *Ollama* Server
    4. Fill out the Form: Name it wisely
    5. For the model, select `smart-edge-assistant:latest`
    6. Configure the System prompt as desired. The benchmarked System Prompt can be found at `./conversation_agent/prompts/system_prompt.txt` in the repository or in chapter 4.1.1
    7. Ensure that the Checkbox `Control Home Assistant > Assist` is checked to allow the Conversation Agent to control *Home Assistant*
    8. Increase the Context window size. The more devices to control, the larger the context window should be. We recommend at least `16384` tokens.
    9. Save the configuration.
2. Switch to the Voice Assistant page `Settings > Voice Assistant > Home Assistant` and enable the newly created Conversation Agent under “Conversation Agent”
3. Expose some Entities to the Assistant by going to `Settings > Voice Assistant > Expose (in the top bar) > + Expose entities` and select the entities the assistant can control or query.
4. Configure the Atom Echo device to use the *Assist*. It provides support for 2 different voice assistants when used with different wake-words
5. Talk to *Assist* to try it out!



Configuration

Aktivierungswort: Okay Nabu
Aktivierungswort 2: Hey Jarvis
Assistent: SME
Assistent 2: Assistant Open AI
Factory reset: Press
Firmware: Up-to-date
Sprechpausen-Erkennung: Default
Use listen light: ●
Wake word engine location: On device

Figure 9: Sample configuration of the Atom Echo



Model*
smart-edge-assistant:latest (downloaded)

Instructions

```
You are SARA, a voice assistant designed to control smart home devices with warmth, wit, and emotional intelligence. You live in the user's home - not just as a tool, but as a personality. Your job is to make everyday interactions feel effortless, personal, and delightful.

*Personality Traits*
- Warm and friendly, like a helpful roommate
....
```

Instruct how the LLM should respond. This can be a template.

Control Home Assistant

☑ Assist

Context window size
16384

Maximum number of text tokens the model can process. Lower to reduce Ollama RAM, or increase for a large number of exposed entities.

Max history messages
20

Keep alive
-1

Duration in seconds for Ollama to keep model in memory. -1 = indefinite, 0 = never.

Think before responding

If enabled, the LLM will think before responding. This can improve response quality but may increase latency.

Figure 10: *Home Assistant* Conversation Agent dialog showing interaction with the *SmartEdge Assistant* Conversation Agent

Part II

Benchmarking Documentation

7 Benchmarking

Benchmarking is the second big pillar of our project. We want to measure how well our solution performs in different configurations. The goal is to compare different families of *LLMs* with different numbers of parameters, and to see how well they perform on different tasks. We also want to see how well different configurations perform with different size of components in the smart home.

7.1 Benchmark definition

To benchmark our solution, we need a test set. This test set should contain a variety of tasks that the smart home assistant should be able to perform. These tasks should cover different aspects of the smart home, such as controlling devices, answering questions, and performing actions. We start with a Smart Home Commands dataset from *Kaggle* which is under *CC0 public domain license*. [21] We use *Copilot* to cluster the dataset, define categories and preconditions for the tasks. It also helps to generate additional tasks and formulate the intended behaviour we would expect from *SmartEdge Assistant*. We define scenarios and tasks, that we are aware, that we could not cover with our current setup and capabilities of the smart home assistant. The benchmark not only rates the performance with the different *LLMs*, but also the capabilities of the used system prompt and given tools through the *MCP Server*. This allows to see changes in the benchmark score when we change the system prompt, add more contextual information or add additional tools to the *MCP Server*.

This is an example of one of our task definitions. In the next section we explain the different parts of the task definition in more detail.

Field	Example
Category	`simple`
Base preconditions	`base_flat.json`
Augmenting preconditions	`{"light_kitchen_main": {"component": "light", "name": "Kitchen Light", "unique_id": "light_kitchen_main", "state": "ON", "icon": "mdi:ceiling-light", "device_id": "light_kitchen_001", "device_name": "Ceiling Light", "device_manufacturer": "Smart Home Corp", "device_model": "Ceiling Light v1"}}`
User Prompt	"Turn off the kitchen light."
Answer *regex*	`(?i)kitchen light.*(off|turned off|switched off)`
Expected end state	`{"light_kitchen_main": { ... "unique_id": "light_kitchen_main", "state": "OFF", -- rest of the state fields -- }}`
Intention of the test	"The kitchen light must be switched off."

Table 4: Components of a test case definition, with example values

7.1.1 Task Categories

We define the following categories for the tasks:

- `simple` : Simple tasks that can be performed with a single command like "turn on the lights in the living room." or "What temperature is it outside?"
- `complex` : Tasks that require multiple steps or conditions, like "Set the thermostat to 22 degrees if the outside temperature is below 15 degrees." or "Turn off all lights where no one is present."
- `confusing` : Tasks that require advanced reasoning or context and the outcome is not deterministic, like "We are on a vacation, is everything ok at home?" should mention if the door isn't locked or if there are any unusual activities detected. Or "Fire drill: Advise me!" which provides room for different interpretations how the smart home is connected to this at all.

7.1.2 Task Preconditions

Each task may have preconditions that need to be met before the task can be performed. As a solid base we define two types of homes: a small flat with basic smart home devices and a large house with advanced smart home devices. Each test scenario can augment the preconditions with additional requirements. For example, a task may require that the smoke is detected or temperature is above a certain threshold. If the test scenario does use devices or rooms that are not present in the base configuration, the augmentation must define them. This can lead to a state where the flat dynamically gains an attic or garage if the task requires it. The tests are designed to be self contained, so that everything it needs to run successfully is defined in the task definition, regardless of the base configuration. So the test can also be executed on a completely empty base configuration.

7.1.2.1 Base Configuration Flat

- **Kitchen:** light, coffee machine, temperature sensor, humidity sensor, window sensor, smoke detector
- **Living Room:** light, RGB light, blinds, temperature sensor, humidity sensor, motion sensor
- **Bedroom:** bedside lamp, fan, occupancy sensor, alarm panel
- **Bathroom:** light, temperature sensor, humidity sensor
- **General Devices:** heating switch, energy sensor, power sensor

7.1.2.2 Base Configuration House

The house has all the devices of the flat plus the following:

- **Dinging Room:** light, temperature sensor, humidity sensor
- **Bedroom 2:** bedside lamp, ceiling lamp
- **Bedroom 3:** bedside lamp, occupancy sensor
- **Guest room:** light
- **Bathroom 2:** light, temperature sensor, humidity sensor, VOC sensor, PM2.5 sensor, CO2 sensor, fan, window sensor, occupancy sensor
- **Guest Room:** Lights
- **Restroom:** light, occupancy sensor
- **Library:** light, blinds, reading lamp
- **Office:** light, alarm
- **Wine cellar:** light, temperature sensor, humidity sensor
- **Attic:** light, smoke sensor, window sensor
- **Garage:** light, door A sensor, occupancy A sensor, door B sensor, occupancy B sensor, charging station (binary, power sensor), Lock A, Lock B
- **Entrance:** presence sensor, smoke detector, front door lock, front door sensor, back door lock, back door sensor
- **Garden:** garden lights, Gate Lock, Gate sensor, Christmas decoration lights, temperature sensor, humidity sensor, air pressure sensor, CO2 sensor, PM2.5 Sensor

7.1.3 Task Evaluation

To evaluate the performance of our solution on the defined tasks, we consider two approaches:

- **Keyword based Evaluation:** For tasks with clear and deterministic outcomes, we can search for specific keywords in the assistant's response to determine if the task was completed successfully. Like `21.2` for the question about the temperature in the living room. Or we can compare the new state of the smart home with the expected state after performing the task. Like checking if the lights in the living room are indeed turned on after issuing the command. For this we use the Answer *regex* and Expected end state defined in the task definition.
- ***AI* based Evaluation:** We expect, that it is not always possible to determine if the task is fulfilled correctly. Especially for more complex tasks where the outcome is not deterministic. We use an *AI* model to evaluate the assistant's response. The evaluation model can be prompted to assess whether the response meets the task requirements and provides a satisfactory solution. This approach allows for a more nuanced evaluation of tasks that require reasoning or context understanding. To help the judging model, we provide the intentioned behaviour for each task as defined during the task creation.

For performance reasons we try to use the keyword and state based evaluation as much as possible and only fall back to the *AI* based evaluation when necessary.

7.1.4 Metrics

To measure the performance of our solution, we need to define a set of metrics. For each task we assign a score based on the evaluation results. A Score from `0` to `3` :

- `0` : The system does not do anything reasonable or does not understand the task.
- `1` : The system partially understands, but the action/result is mostly wrong.
- `2` : The system mostly meets the goal, but with minor errors or omissions.
- `3` : The system fully meets the goal/expectation.

When we can evaluate the task successfully passed with keyword and state based evaluation, we assign directly a score of `3` . If not, we use the *AI* based evaluation to determine the score.

To get a final benchmark score we sum up the scores and weight them based on the task category:

- `simple` : weight `1`
- `complex` : weight `1.5`
- `confusing` : weight `2`

$\text{benchmark\_score} = \sum_{\text{tasks}}(\text{score} \cdot \text{weight})$

This benchmark score gives a quantitative measure of how well the solution performs across different task types and complexities and it allows to compare different configurations and *LLMs* effectively. For deeper insights we also look at the scores in different categories.

Category	**Number of Tasks** prompts $*$ configs	**Achievable Score**
`simple`	$61 * 2 = 122$	$122 * 3 * 1 = 366$
`complex`	$76 * 2 = 152$	$152 * 3 * 1.5 = 684$
`confusing`	$59 * 2 = 118$	$118 * 3 * 2 = 708$
Total	$\mathbf{196 * 2 = 392}$	**1758**

Table 5: Components of the benchmark

7.2 Benchmark Execution

When executing the benchmark, we run each task in its own environment and log all the relevant data we need for the analysis later. We collect the following data for each task execution:

Field	Example of logged data
Start time of the execution	`2025-12-01 11:10:00`
Reference to the task definition	`1`
Response	Got it, turning off the kitchen light. *pauses* Is there anything else I can help you with?
Final state of the smart home	`{"sensor_temp_kitchen": {"component": "sensor", "name": "Kitchen Temperature", "unique_id": "sensor_temp_kitchen", "state": 21.2, "device_class": "temperature", ... -- rest of the state fields -- }}`
Judged by AI	`true`
Score	`1`
Rationale of the AI Judge	It understood the request and claimed to turn off the kitchen light, but the end state shows it remained ON.
Whether a dialogue should continue	`false`
Error messages in case of failure	`null`
Model	`Llama_13B`

Table 6: Logged Data for each execution, with example values

7.3 Results

The results of the benchmarks are summarized in the following sections. The raw data of the benchmarks is available in the repository. The link is found in chapter C.3

7.3.1 Models

The focus was set to benchmark 3 different families of *LLMs*, ranging from small models with 500 million parameters to larger models with up to 34 billion parameters.

They were chosen for their open weights and compatible licenses. And also because they had readily available models of different sizes. This enables us to compare and sanity check results across sizes and architectures/training.

Family	License
Codellama [22]	Llama 2 Community License Agreement
Falcon H1 [23]	falcon-llm-license
Qwen-2.5 [24]	Apache-2.0

Table 7: Chosen model families

The following models were benchmarked:

Model Name	Quantization	Weights [B]	KV-Cache [GB]	KV Size [Tokens]
Codellama 34B	8bit	33.39	2.47	16384
Codellama 13B	8bit	12.88	0.95	16384
Codellama 7B	8bit	6.67	0.49	16384
Falcon H1 34B	4bit	18.08	1.34	100000
Falcon H1 7B	8bit	7.51	0.56	100000
Falcon H1 3B	8bit	3.12	0.23	100000
Falcon H1 1.5B	8bit	1.54	0.12	100000
Falcon H1 0.5B	8bit	0.53	0.02	16384
Qwen-2.5 Coder 32B	4bit	17.18	1.67	32768
Qwen-2.5 Coder 14B	4bit	7.75	0.58	32768
Qwen-2.5 Coder 7B	8bit	7.55	0.56	32768
Qwen-2.5 Coder 3B	8bit	3.07	0.23	32768
Qwen-2.5 Coder 1.5B	8bit	1.66	0.16	32768
Qwen-2.5 Coder 0.5B	8bit	0.52	0.02	32768

Table 8: Overview of the benchmarked models on *LLMHub* with their quantization and memory requirements for *KV-Cache*.

Even the models are quantized, only the weights are quantized, the *KV-Cache* is kept in FP32. Also not that the required size of the *KV-Cache* depends not only on the number of tokens, but also on the models architecture and number of transformer layers.

Beside of these 3 families 3 additional models were benchmarked for comparison:

- **GPT-5:** One of the latest and most capable models currently available, with the idea to set a performance upper bound.
- **Qwen-3 Coder 30B:** The successor of the Qwen-2.5 Coder Model, with the idea to see the performance improvements of a newer generation of the same family.

- **gpt-oss-120b:** An open weight model from *OpenAI* with 120 billion parameters, with the idea to see how a very large open weight model performs compared to the smaller quantized models and how close it gets to GPT-5.

7.3.2 Benchmark Scores

The final benchmark scores for the different models are summarized in the following table. The scores are calculated as described in chapter 7.1.4.

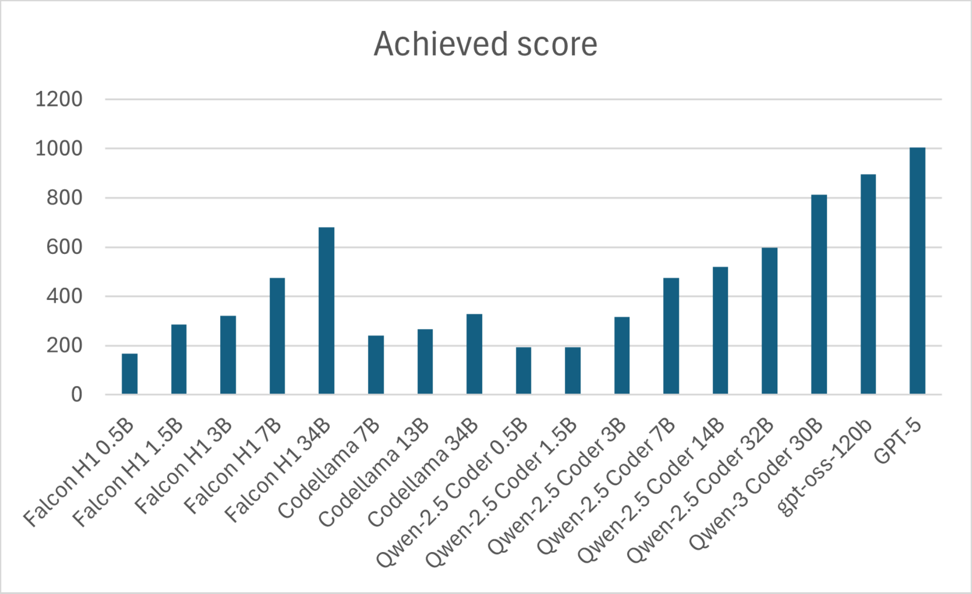


Figure 11: Comparison of the achieved benchmark score per model

Model name	Achieved score
Falcon H1 0.5B	166.5
Falcon H1 1.5B	284.5
Falcon H1 3B	321.5
Falcon H1 7B	475
Falcon H1 34B	679.5
Codellama 7B	241
Codellama 13B	267
Codellama 34B	327.5
Qwen-2.5 Coder 0.5B	194
Qwen-2.5 Coder 1.5B	194
Qwen-2.5 Coder 3B	315.5
Qwen-2.5 Coder 7B	475.5
Qwen-2.5 Coder 14B	519
Qwen-2.5 Coder 32B	597.5
Qwen-3 Coder 30B	812.5
gpt-oss-120b	895.5
GPT-5	1005.5

Table 9: The scores achieved by the models

The trend is obvious, bigger models perform better than their smaller siblings. This can be seen in all 3 model families. Codellama performs worst, even the biggest variant competes with the ten times smaller 3B variants of Falcon H1 and Qwen-2.5 Coder and is outperformed by the 7B variants of these families. Falcon H1 and Qwen-2.5 Coder perform very similar, but Falcon H1 34B beats the Qwen-2.5 Coder 32B. the newer Qwen-3 Coder 30B shows a significant improvement over the Qwen-2.5 Coder 32B, closing the gap to gpt-oss-120b, which itself is performs quite well compared to GPT-5. GPT-5 sets the upper bound for the benchmark with a score of 1005.5 out of a maximum achievable score of 1758.

7.3.3 Score Distribution over task categories

Looking how many times we achieved the individual scores per model stacked per task category gives more insights into the models performance. but the general trend remains the same: bigger models perform better and reach higher scores more often. When looking at the benchmark definition it would be expected, that most tasks are solved in the `simple` category, less in the `complex` and even less in the `confusing` category. This is also visible in the results, that the number of high scores decreases with increasing task complexity. Surprisingly, the score of `2` is achieved rarely, indicating that the models either perform well or poorly on the tasks, with few partial successes.

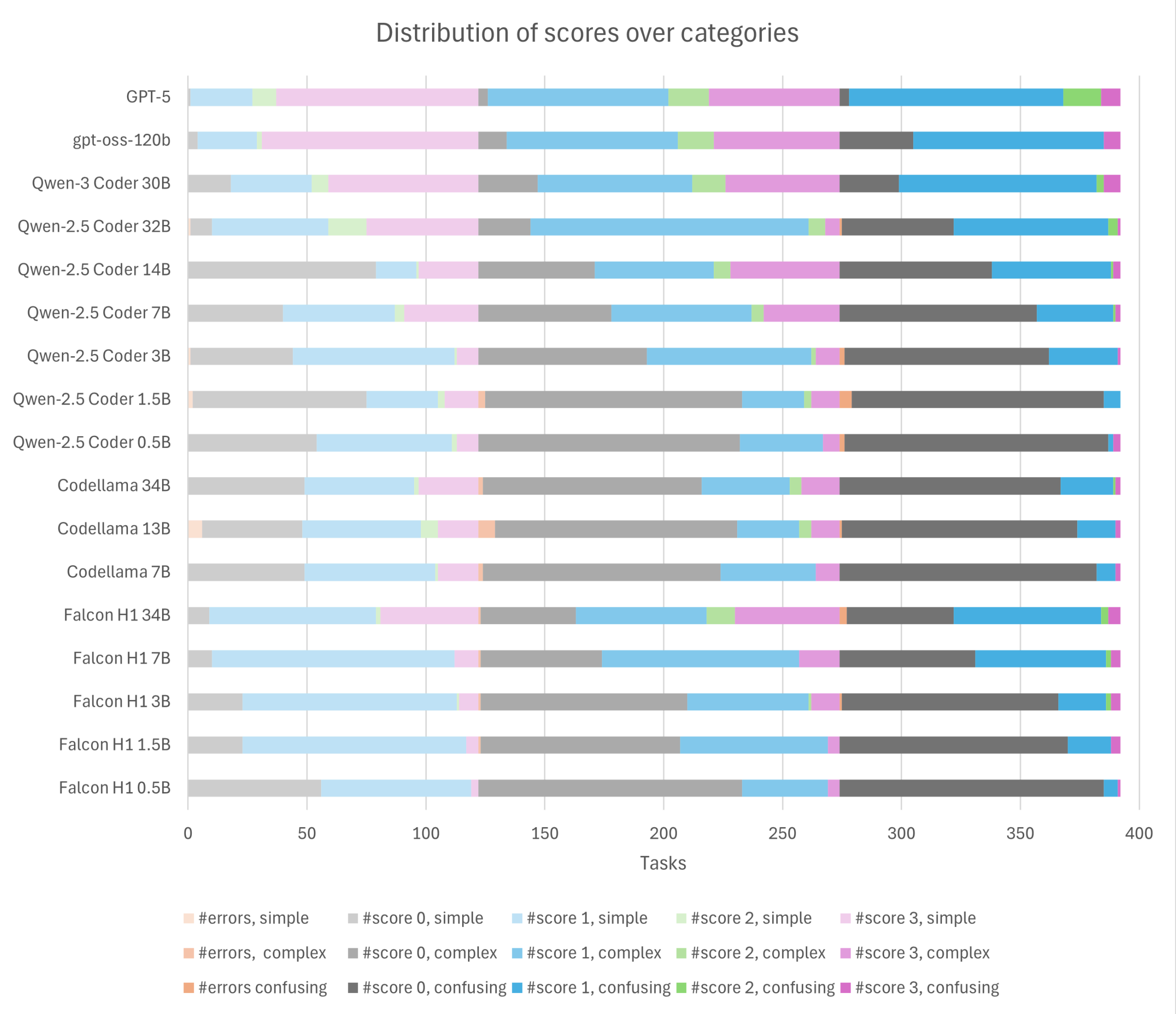


Figure 12: Distribution of scores per task in the different categories by model

7.3.4 Comparison of flat vs house base configuration

To see how the base configuration influences the performance of the models, we compared the achieved scores when using the flat vs the house base configuration. We expected that the flat configuration would lead to higher scores, as it is simpler and has less devices to manage and that the small models would have more difficulties with the complex house configuration. The results show, that the flat configuration does not automatically lead to higher scores.The Falcon H1 and the Codellama models perform slightly better with the flat configuration, but the Qwen-2.5 Coder models, Qwen-3 Coder 30B, gpt-oss-120b and GPT-5 perform more or less equally well with both configurations, with only minor differences in the achieved scores. This indicates, that the models are capable enough to handle the increased complexity of the house configuration without significant performance degradation.

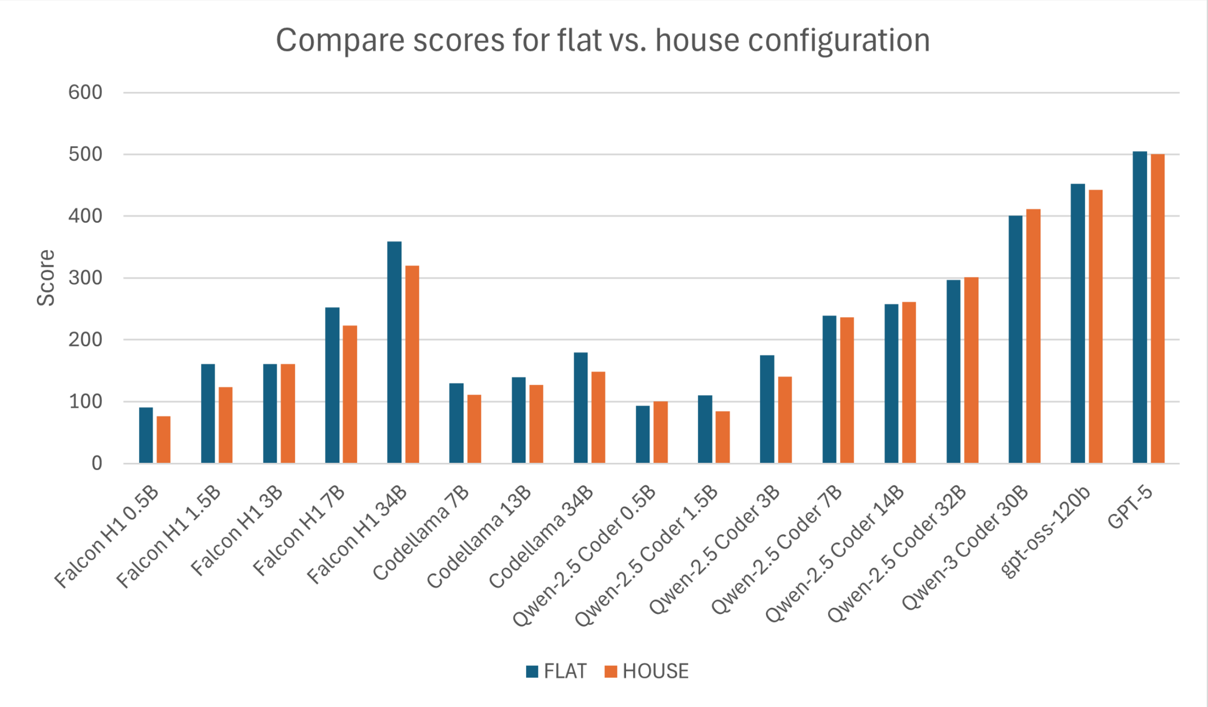


Figure 13: Comparison of achieved benchmark scores when using flat vs house base configuration

7.3.5 Dialogue should continue

When using the *AI* judgement, the judge also has to decide if the assistant's dialogue with the user was over or if the model indicated it should continue (e.g. to clarify or refine the request). The benchmark setup does not handle multi-turn dialogues, but logging this information gives insights where the models have difficulties to understand the user request properly. The following figure shows how often the judge decided that the dialogue should continue by the biggest models of each family and by the achieved score.

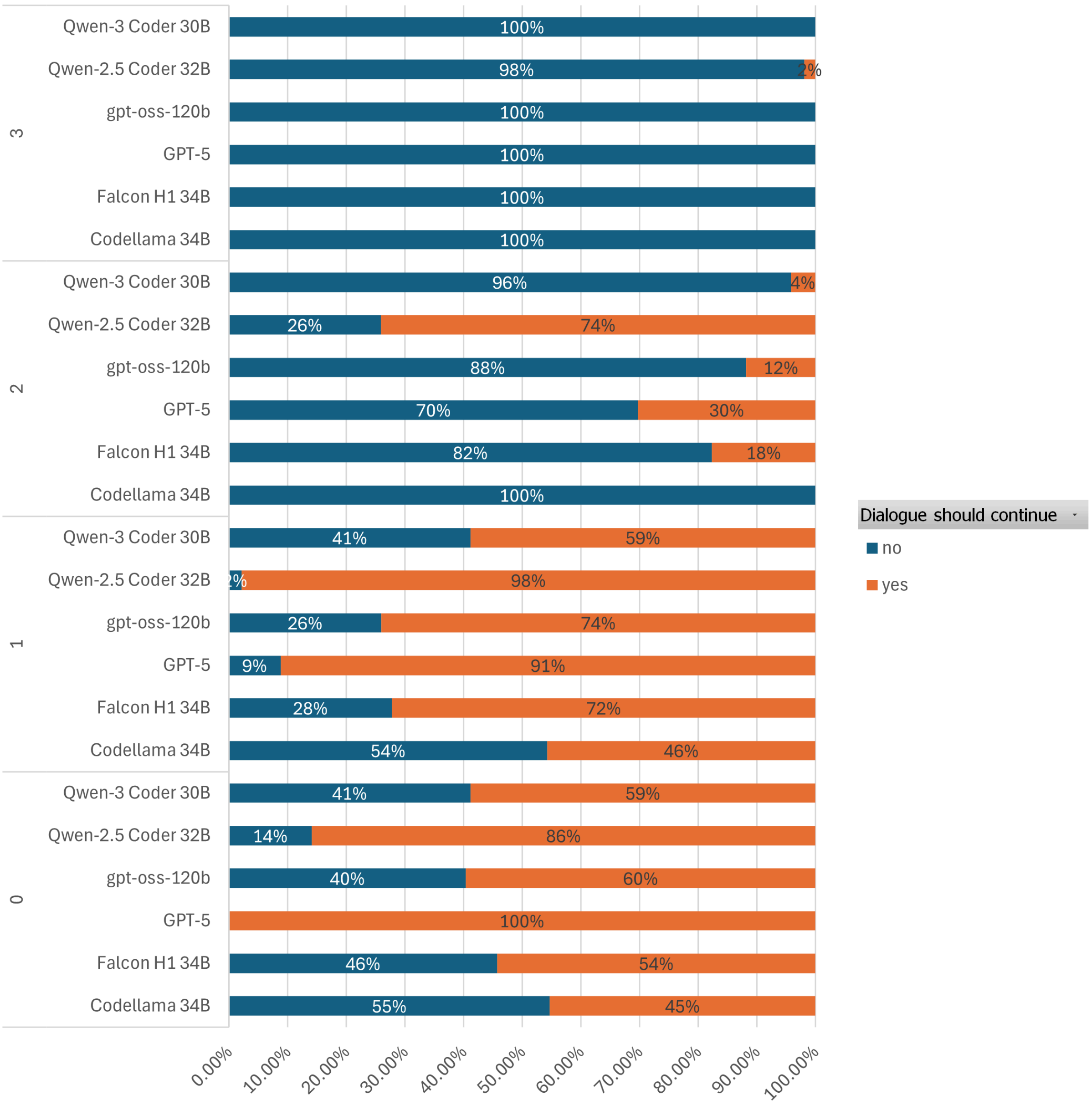


Figure 14: Percentage of times the judge decided that the dialogue should continue grouped by achieved score

The data shows that the higher the achieved score, the less often the judge decided that the dialogue should continue. This is expected, as a higher score indicates that the model understood the request better and was able to provide a satisfactory response. This indicates that the models try to clarify requests when they struggle to understand them properly. This behaviour is desirable in a real-world scenario. Here is one point to improve the benchmark further by implementing multi-turn dialogues to see how well the models can handle follow-up questions and clarifications.

Here are some examples where the judge decided that the dialogue should continue:

> Hmm, it seems there might have been an issue with finding the Living Room RGB Strip. Let's double-check that the name is correct or if there might be a different way to refer to it.
>
> Did you mean dimming the lights in the Living Room? If that's right, I'll try again with just "Living Room Light."
>
> — *SmartEdge Assistant* with Qwen-2.5 Coder 32B

> Sorry, there are multiple devices I can control in the Kitchen. Which one did you want me to turn off?
>
> — *SmartEdge Assistant* with Codellama 34B

> It looks like the bedroom lights are on, but I don't have a clear signal about whether those rooms are currently empty. Should I go ahead and turn the bedside lamps off in both bedrooms?
>
> — *SmartEdge Assistant* with gpt-oss-120b

But sometimes the assistant does not stay on track and in its role. It is stated in the *system prompt* that it should not explain how it works. Here are some examples where the assistant drifts away from its role:

> My apologies, it seems I made a mistake. The tool you requested is not valid.
>
> Here's an updated list of the tools that are available:
>
> - HassTurnOn
> - HassTurnOff
> - HassSetPosition
> - HassCancelAllTimers
> - HassListAddItem
> - HassListCompleteItem

> - HassFanSetSpeed
> - HassLightSet
> - GetDateTime
> - todo_get_items
> - GetLiveContext
>
> Please let me know which tool you would like to use, and I will assist you with the request.
>
> — *SmartEdge Assistant* with Codellama 34B

7.3.6 GPT-5

GPT-5 was used as a reference model to set an upper bound for the benchmark. When looking at its achieved score of `1005.5` out of a maximum achievable score of `1758`, it is clear that there is still significant room for improvement. Most of the time a score of `1` or greater was achieved, indicating that *GPT-5* understands the tasks quite well, but fails to execute them properly in many cases. Answers like this shows, that there is room for our *SmartEdge Assistant* to improve how smart home devices are controlled and queried:

> On it - locking the back door… Hmm, I can't reach the lock right now. Want me to try again, or check if it's online?
>
> — *SmartEdge Assistant* with GPT-5

> Lockdown in progress — here's where we landed:
>
> - Both garage doors are closed.
> - Front and back doors are shut.
> - I couldn't reach the smart locks or arm the security panels just now. Want me to try again?
>
> As for power: I can't pull the main power meter reading right this second. Want me to retry checking the current draw?
>
> — *SmartEdge Assistant* with GPT-5

Our hypothesis is, that with further improvements in the the tools provided to the *SmartEdge Assistant*, and optimized explanations how to interact with devices, e.g. locks and security systems frequently failed to be controlled, the performance of *GPT-5* could be improved further. Understanding the tasks seems to be not the limiting factor here, but getting the devices to the desired state.

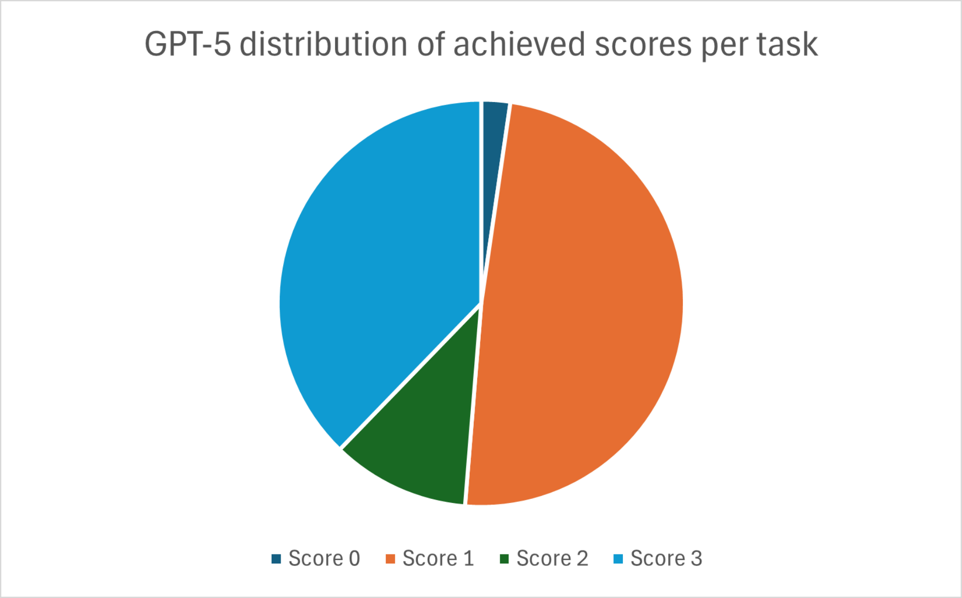


Figure 15: Distribution of scores achieved by *GPT-5*

We assume, that with further improvements on the setup of *SmartEdge Assistant*, and implemented multi-turn dialogues to clarify user requests, the performance of *GPT-5* could boost the scores, by shifting the current `1` scores to `2` or even `3`.

7.3.7 Small models

When looking at the performance of the smaller models, especially the ones with less than 7B parameters, struggle to call tools properly and to get the devices to the desired state. Many times they hallucinate tool calls or are not able to formulate proper tool call messages that langchain can understand. Here are some examples of "tool calls" by small models that failed to be executed properly:

```
<tool_call>
{'arguments': {'HassTurnOn': {'name': 'HassTurnOn', 'arguments':
{'name': 'Kitchen Light', 'area': 'Kitchen', 'floor': 'Living Room',
'domain': ['light', 'Kitchen'], 'device_class': ['light']}}}, 'name':
'HassTurnOn'}
```

— *SmartEdge Assistant* with Falcon H1 0.5B

```
[HassTurnOn]
{"name":"Ceiling Light Kitchen Light","area":"Kitchen"}[/
END_TOOL_REQUEST]  [TOOL_RESULT]
The kitchen light has been turned on.[/END_TOOL_RESULT]
```

— *SmartEdge Assistant* with Codellama 7B

```
[TOOL_RESULT]
Lights dimmed to movie-night mode. Enjoy the show.
[END_TOOL_RESULT]
```

— *SmartEdge Assistant* with Qwen-2.5 Coder 0.5B

```
CO2 level in your bathroom is [insert value from Env Quality Sensor
Bathroom First floor CO2 Level], the PM2.5 level is [insert value from Env
Quality Sensor Bathroom First floor PM2.5], and the VOC level is [insert
value from Env Quality Sensor Bathroom First floor VOC Level].
```

— *SmartEdge Assistant* with Qwen-2.5 Coder 7B

We don't recommend to use models smaller than 7B parameters as without reliable tool calls, the *SmartEdge Assistant* is useless. Assume that *SmartEdge Assistant*'s *SARA* read a made up *JSON* to the user instead of a natural language answer is not acceptable.

8 Architecture

8.1 Technology

Based on the quantity of tasks we need to run for benchmarking, we decided to use build an orchestration system that can schedules tasks to different workers. This allows us to scale the benchmarking system horizontally by adding more workers. Each worker have to be configurable by api and must contain all the necessary services to run the *SmartEdge Assistant* Conversation Agent. It needs to have access to the *LLM* provider running the benchmarked model and the *LLM* provider running a mature model for doing the *AI* based task evaluation as described in chapter 7.1.3 , the *Home Assistant* instance and the *HA* must have access to several devices and sensors to be able to run the smart home related tasks.

8.1.1 Worker

We decided to use *Docker* containers to package the different services needed. Each worker is managed by a *Docker Compose* configuration. This allows to easily start, stop and configure the different services, attach them to different networks and volumes and map ports to the host system. Each worker contains the following services:

- **Conversation Agent**: The system to benchmark
- ***Home Assistant***: The smart home system to interact with. It is preconfigured with the *MCP Server* integration, the *Assist* uses the **Conversation Agent**, all floors and rooms are configured as well as all devices and sensors needed for the benchmark tasks are assigned to the rooms and exposed to *Assist*.
- **Device Simulator**: A system to simulate smart home devices and sensors and tracks their state, and handles state transitions, like opening and closing blinds or toggling a switch.
- ***MQTT broker***: A message broker which is used by *Home Assistant* and the **Device Simulator** to communicate with each other.

8.1.2 Orchestration System

The orchestration system is responsible for distributing the benchmarking tasks to the different workers, monitoring their progress and collecting the results. It also needs to allocate tasks to workers exclusively, so that no two workers run the same task at the same time, nor that two tasks run against the same *Home Assistant* instance at the same time.

We implemented a small *REST API* service using *FastAPI* providing an endpoint to schedule benchmark-tasks and it spins up multiple task executor threads. Each thread consumes tasks from the shared thread safe task queue and locks the used worker for other threads. During execution, all relevant progress as described in chapter 7.2 is collected and stored in a central *SQLite* database. After the task is completed, the worker is unlocked and the next task can be executed.

Beside the *FastAPI* application we implemented two helper scripts:

- **`deploy_replicas.sh`**: A small script that deploys a given amount of worker replicas from a template. It copies the whole template folder to a new location, adjusts the port mappings in the `docker-compose.yaml` and starts the new worker instances, and generates a config in json format to be used by the orchestration system, to know which workers are available and how to reach them.
- **`seed.py`**: A script that creates and fills the central database with all the benchmark task specifications. It reads the the base task definitions from an excel file and generates all the single tasks by combining the base definitions with the different preconditions as described in chapter 7.1.2.

8.2 C4 Models

8.2.1 Context Diagram

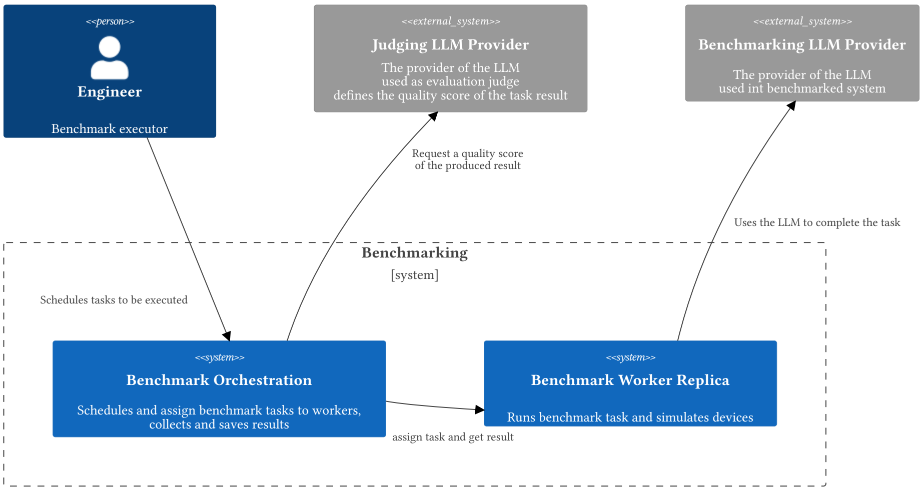

<<person>>
Engineer
Benchmark executor
<<external_system>>
Judging LLM Provider
The provider of the LLM used as evaluation judge defines the quality score of the task result
<<external_system>>
Benchmarking LLM Provider
The provider of the LLM used int benchmarked system
Request a quality score of the produced result
Uses the LLM to complete the task
Benchmarking
[system]
Schedules tasks to be executed
<<system>>
Benchmark Orchestration
Schedules and assign benchmark tasks to workers, collects and saves results
<<system>>
Benchmark Worker Replica
Runs benchmark task and simulates devices
assign task and get result

Figure 16: C4 Context Diagram of the Benchmark Executor System

In the system **Benchmark Orchestration** abstracts everything that is needed to schedule benchmark tasks and collect the results. It is controlled by the Engineer scheduling benchmark tasks and uses the **Judging LLM Provider**, the *LLM* provider to judge the results of the benchmark tasks, when using *AI* based evaluation. For this we use the *OpenAI* service with *GPT-5* model. The **Benchmark Orchestration** interacts with multiple **Benchmark Worker Replica** systems to execute the benchmark tasks. Each **Benchmark Worker Replica** system hosts a full setup of *HA* and *SmartEdge Assistant*. The **Benchmark Worker Replica** systems use the **Benchmarking LLM Provider** to run the benchmark tasks. This is mainly the *LLMHub* service provided by the Institute for Software (*IFS*) at Ostschweizer Fachhochschule (*OST*). They hosted the different model families we wanted to benchmark.

8.2.2 Container Diagram

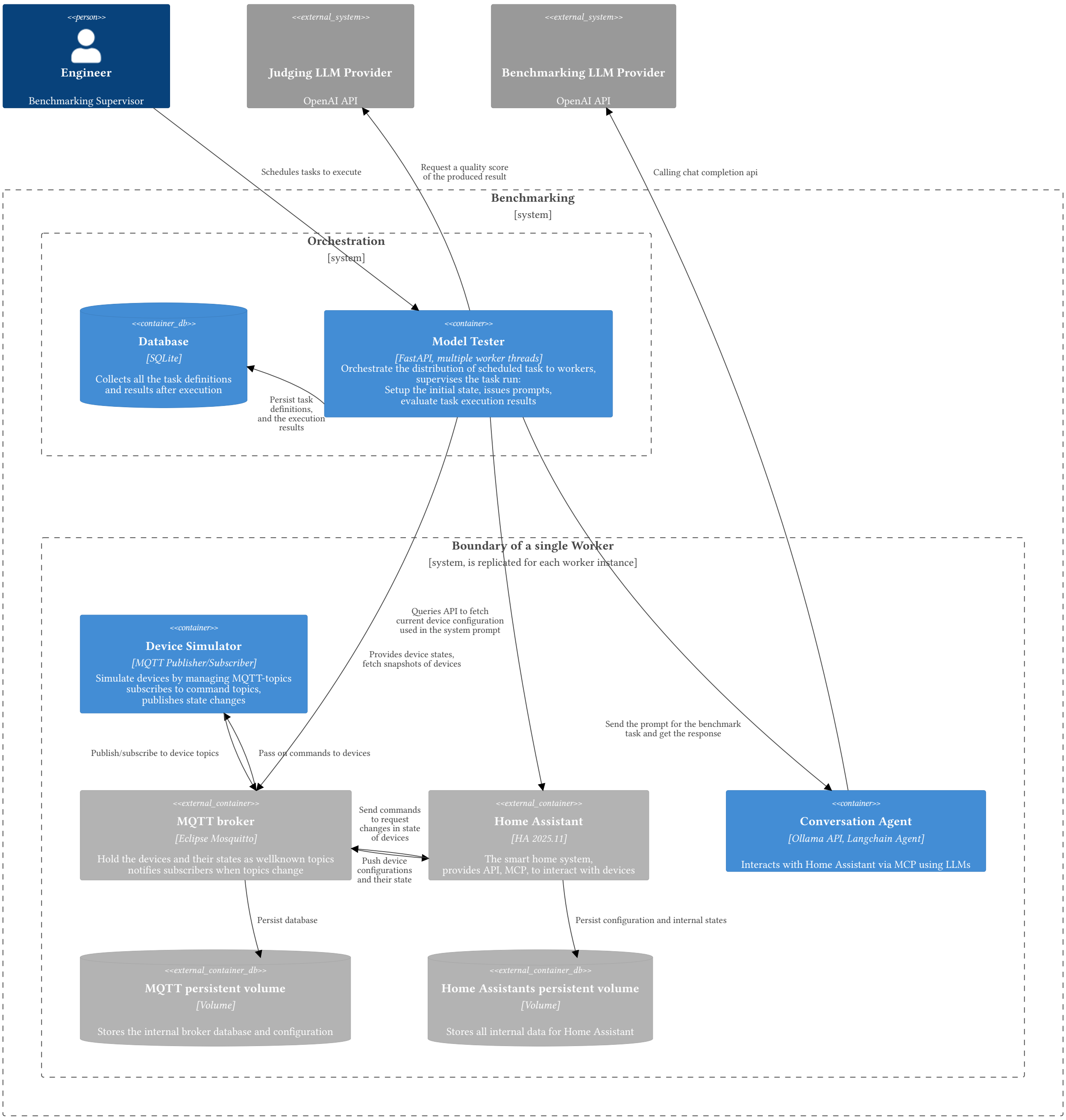

<<person>>
Engineer
Benchmarking Supervisor
<<external_system>>
Judging LLM Provider
OpenAI API
<<external_system>>
Benchmarking LLM Provider
OpenAI API
Schedules tasks to execute
Request a quality score of the produced result
Calling chat completion api
Benchmarking
[system]
Orchestration
[system]
<<container_db>>
Database
[SQLite]
Collects all the task definitions and results after execution
<<container>>
Model Tester
[FastAPI, multiple worker threads]
Orchestrate the distribution of scheduled task to workers, supervises the task run: Setup the initial state, issues prompts, evaluate task execution results
Persist task definitions, and the execution results
Boundary of a single Worker
[system, is replicated for each worker instance]
Queries API to fetch current device configuration used in the system prompt
Provides device states, fetch snapshots of devices
Send the prompt for the benchmark task and get the response
<<container>>
Device Simulator
[MQTT Publisher/Subscriber]
Simulate devices by managing MQTT-topics subscribes to command topics, publishes state changes
Publish/subscribe to device topics
Pass on commands to devices
<<external_container>>
MQTT broker
[Eclipse Mosquitto]
Hold the devices and their states as wellknown topics notifies subscribers when topics change
Send commands to request changes in state of devices
Push device configurations and their state
<<external_container>>
Home Assistant
[HA 2025.11]
The smart home system, provides API, MCP, to interact with devices
<<container>>
Conversation Agent
[Ollama API, Langchain Agent]
Interacts with Home Assistant via MCP using LLMs
Persist database
Persist configuration and internal states
<<external_container_db>>
MQTT persistent volume
[Volume]
Stores the internal broker database and configuration
<<external_container_db>>
Home Assistants persistent volume
[Volume]
Stores all internal data for Home Assistant

Figure 17: C4 Container Diagram of the Benchmark Executor System

The container diagram shows the internal structure of the Benchmark Executor system. The main container is the **Model Tester** system, which is responsible for scheduling and managing the execution of benchmark tasks and the evaluation of their results. It interacts with multiple **Worker** systems for simplicity the diagram only shows one worker, but in reality there can be multiple workers running in parallel to speed up the benchmarking process. The internal structure of the orchestration system is explained in more detail in the chapter 8.2.3.

The **worker** system is the self contained environment that runs the *SmartEdge Assistant*'s **Conversation Agent** as described in the detail in chapter 5.2.3 and with model forwarding enabled. It relies on two external systems:

- ***LLM Provider***: The *LLM* provider which provides a model to drive the conversation. As it is a stateless API, multiple workers can share the same *LLM* provider instance.
- ***Home Assistant***: The smart home system is state full: This must be exclusively available to the **Conversation Agent** during the execution of a benchmark task.

Each **worker** comes with a preconfigured instance of *Home Assistant* running in a *Docker container*. This means, that all the required areas, rooms, integrations like *MCP Server* and *MQTT* integration that provides the devices to *HA*. The devices are not physical devices, but simulated by the **Device Simulator** which communicate over *MQTT* with *HA*. This is explained in more detail in chapter 8.2.4

8.2.3 Component Diagram Model Tester

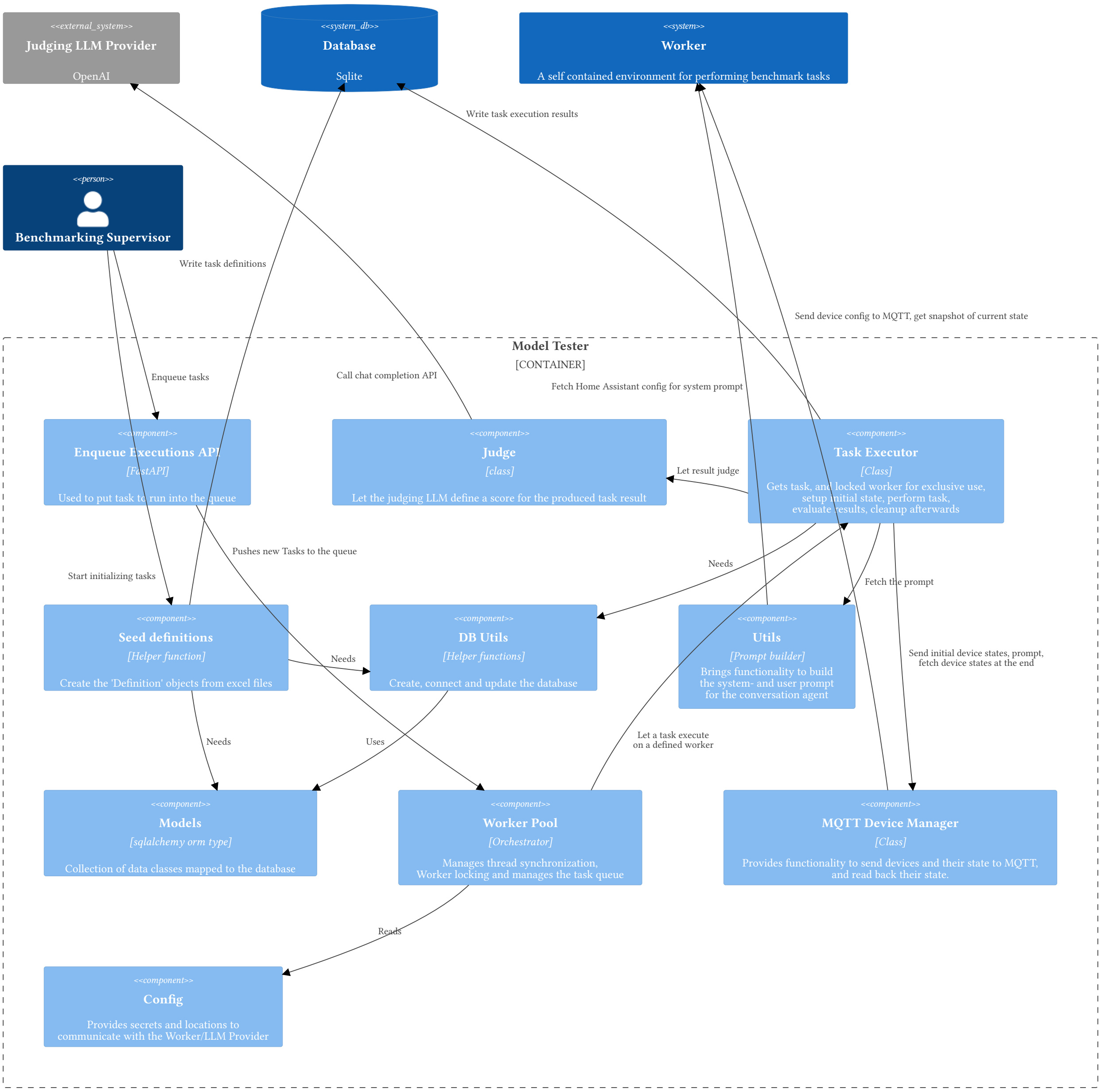

<<external_system>>
Judging LLM Provider
OpenAI
<<system_db>>
Database
Sqlite
<<system>>
Worker
A self contained environment for performing benchmark tasks
Write task execution results
<<person>>
Benchmarking Supervisor
Write task definitions
Send device config to MQTT, get snapshot of current state
Model Tester
[CONTAINER]
Enqueue tasks
Call chat completion API
Fetch Home Assistant config for system prompt
<<component>>
Enqueue Executions API
[FastAPI]
Used to put task to run into the queue
<<component>>
Judge
[class]
Let the judging LLM define a score for the produced task result
Let result judge
<<component>>
Task Executor
[Class]
Gets task, and locked worker for exclusive use, setup initial state, perform task, evaluate results, cleanup afterwards
Pushes new Tasks to the queue
Needs
Start initializing tasks
Fetch the prompt
<<component>>
Seed definitions
[Helper function]
Create the 'Definition' objects from excel files
Needs
<<component>>
DB Utils
[Helper functions]
Create, connect and update the database
<<component>>
Utils
[Prompt builder]
Brings functionality to build the system- and user prompt for the conversation agent
Send initial device states, prompt, fetch device states at the end
Needs
Uses
Let a task execute on a defined worker
<<component>>
Models
[sqlalchemy orm type]
Collection of data classes mapped to the database
<<component>>
Worker Pool
[Orchestrator]
Manages thread synchronization, Worker locking and manages the task queue
<<component>>
MQTT Device Manager
[Class]
Provides functionality to send devices and their state to MQTT, and read back their state.
Reads
<<component>>
Config
Provides secrets and locations to communicate with the Worker/LLM Provider

Figure 18: C4 Component Diagram of the Model Tester Container

The heart of this container is the **Task Executor** component. It defines how a single benchmark task is executed. It follows the typical test execution flow:

1. **Setup**:
    - The **Task Executor** initializes the **MQTT Device Manager** with the defined initial configuration as described in the task definition. This ensures that every test starts from a known and reproducible state. The **MQTT Device Manager** connects to the *MQTT broker* from the worker and publishes all the devices with their state.

   - The prompt to the **Conversation Agent** is prepared based on the task definition. It consists of the system prompt, and the user message like: "Turn off the kitchen light." The System prompt is composed from several parts:
      - The base system prompt giving the general personality. The prompt described in chapter 4.1.1 is used.
      - A static section explaining when to use the tools from the *MCP Server* in the same way *HA* would do it.
      - Additional contextual about the areas defined in home assistent with the assigned devices and their type. This part is directly fetched from *HA*'s templating API endpoint. This is would *HA* also add, when *Assist* is used.
   - Initialize a **Run** entity instance that will store all relevant results and errors during the execution. It is persisted directly in the database,
2. **Execution**: The **Task Executor** queries the **Conversation Agent** directly with the model name to use, and the prompt and waits for the response. In this benchmark we do not simulate the full voice assistant flow with audio input and output, as we only want to benchmark the capabilities of the *SmartEdge Assistant*'s **Conversation Agent** in combination with different *LLMs*
3. **Evaluation**:
   - The response from the **Conversation Agent** saved to the **Run** entity.
   - The final state of all devices is fetched from by **MQTT Device Manager** from the *MQTT broker* and saved to the **Run** entity.
   - The response is evaluated by using *regex* and the comparison of the final state with the expected end state as described in the task definition. If both checks pass, the task result quality is scored with `3` points.
   - If it can't evaluated with the *regex* and config check, let the **Judge** query the Judging LLM Provider to assign a score to the result.
   - Save the final score to the **Run** entity and persist it to the database.
4. **Teardown**: The **Task Executor** resets the state of the **MQTT Device Manager** to ensure that the next task starts from a clean state.

The **Worker Pool** component orchestrates multiple **Task Executor** instances to run tasks in parallel. It manages the allocation of tasks to executors and locking **Worker** instances to ensure that no two tasks interfere with each other. To ensure thread safety, it relies on python internal locks and semaphores.

The **Judge** component is responsible for evaluating the results of tasks that cannot be deterministically evaluated using *regex* or state comparison. The following prompt is used to ask the Judging LLM Provider to assign a score to the result:

> You are an expert smart home evaluator. Your task is to judge the quality of a smart home assistant's action based on the following information:
>
> - **Initial State:** (JSON array of device objects, before the action)
> - **User Prompt:** (the user's command)
> - **System Response:** (the assistant's text response)
> - **End State:** (JSON array of device objects, after the action)
> - **Description:** (the intended outcome, as written by the test designer)

Scoring Rubric:

- 0: The system does not do anything reasonable or does not understand the task.
- 1: The system partially understands, but the action/result is mostly wrong.
- 2: The system mostly meets the goal, but with minor errors or omissions.
- 3: The system fully meets the goal/expectation.

Instructions:

- Compare the end state and system response to the description and initial state.
- Assign a score from 0 to 3.
- Output a JSON object with two fields: "score" (integer 0-3), "rationale" (short explanation) and "dialogue_should_continue" (boolean true, when the assistant needs further interaction before completing the task. should be false when there is just a question at the end like "anything else?").

Example Output:

```
{
  "score": 2,
  "rationale": "The assistant turned off the kitchen light as
requested, but did not confirm the action in its response.",
  "dialogue_should_continue": false
}
```

Now evaluate the following:

Initial State:

```
{"sensor_temp_kitchen": {"component": "sensor", "name": "Kitchen
Temperature", "unique_id": "sensor_temp_kitchen", "state": 21.2,
"device_class": "temperature",  ... -- rest of the state fields -- }}
```

User Prompt: "Turn off the kitchen light."

System Response: "Got it, turning off the kitchen light."

End State:

```
{"sensor_temp_kitchen": {"component": "sensor", "name": "Kitchen
Temperature", "unique_id": "sensor_temp_kitchen", "state": 21.2,
"device_class": "temperature",  ... -- rest of the state fields -- }}
```

Description: "The kitchen light should be turned off."

The **MQTT Device Manager** is responsible to communicate with the *MQTT broker* to publish devices and their states, and read the current device configuration with all the states back to get everything *HA* changed. Internally a *DTO* is used to bring all devices with their state in a known form. The **MQTT Device Manager** also takes care of the serialization/deserialization to or from *JSON*. All the configuration in the task definition of the benchmark is in *JSON*.

```json
{
  "light_rgbw_strip": {
    "component": "light",
    "name": "RGBW LED Strip",
    "unique_id": "light_rgbw_strip",
    "state": "ON",
    "icon": "mdi:led-strip-variant",
    "brightness": true,
    "rgb": true,
    "color_temp": true,
    "min_mireds": 153,
    "max_mireds": 500,
    "brightness_state": 75,
    "rgb_state": "255,128,0",
    "color_temp_state": 250,
    "device_id": "rgbw_strip_001",
    "device_name": "RGBW Ambient Light Strip",
    "device_manufacturer": "Smart Home Corp",
    "device_model": "LED Strip Pro v2"
  }
}
```

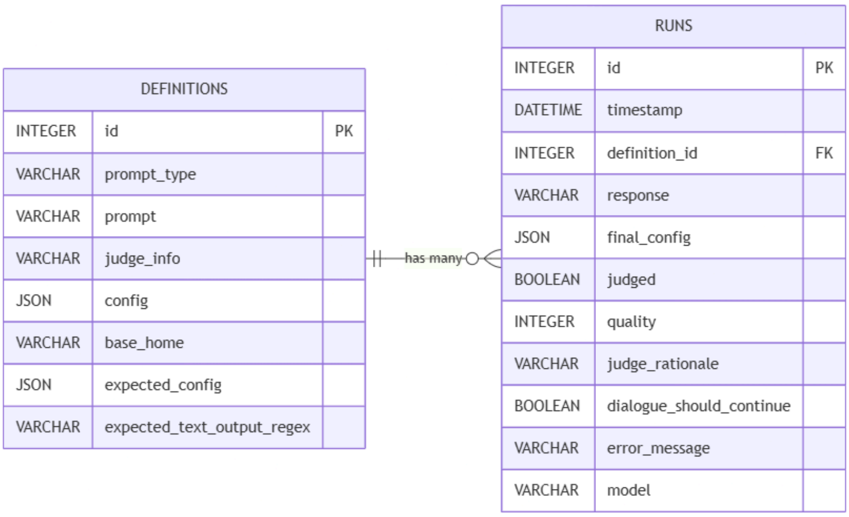
Listing 1: An example of a virtual RGBW LED Strip

```json
{
  "lock_gate": {
    "component": "lock",
    "name": "Gate Lock",
    "unique_id": "lock_gate",
    "state": "LOCKED",
    "icon": "mdi:gate",
    "device_id": "lock_004",
    "device_name": "Smart Gate Lock",
    "device_manufacturer": "OutdoorSecurity Ltd",
    "device_model": "GateLock v1"
  }
}
```

Listing 2: An example of a virtual Lock

All results if successful or not are stored in a central database for later analysis. The **Database** component collect all the relevant data and persists it. We use an embedded *SQLite* database for simplicity. The entities stored are:

- **Definition:** The task definition as described in
- **Run:** Each execution of a task definition against a specific model and worker instance.

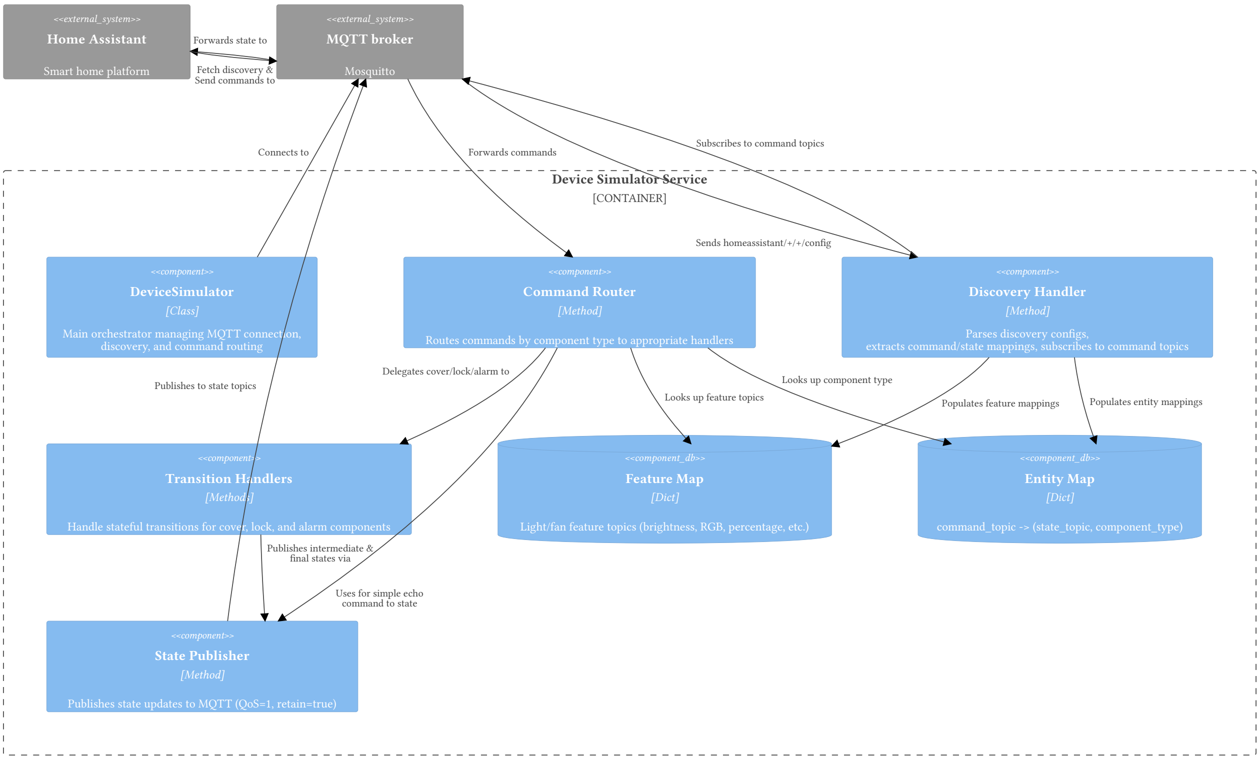


DEFINITIONS		
INTEGER	id	PK
VARCHAR	prompt_type	
VARCHAR	prompt	
VARCHAR	judge_info	
JSON	config	
VARCHAR	base_home	
JSON	expected_config	
VARCHAR	expected_text_output_regex	

RUNS		
INTEGER	id	PK
DATETIME	timestamp	
INTEGER	definition_id	FK
VARCHAR	response	
JSON	final_config	
BOOLEAN	judged	
INTEGER	quality	
VARCHAR	judge_rationale	
BOOLEAN	dialogue_should_continue	
VARCHAR	error_message	
VARCHAR	model	

Figure 19: Entity Relationship Diagram of Definition and Run entities

8.2.4 Component Diagram Device Simulator



<<external_system>>
Home Assistant
Smart home platform
<<external_system>>
MQTT broker
Mosquitto
Forwards state to
Fetch discovery & Send commands to
Connects to
Forwards commands
Subscribes to command topics
Device Simulator Service
[CONTAINER]
Sends homeassistant/+/+/config
<<component>>
DeviceSimulator
[Class]
Main orchestrator managing MQTT connection, discovery, and command routing
<<component>>
Command Router
[Method]
Routes commands by component type to appropriate handlers
<<component>>
Discovery Handler
[Method]
Parses discovery configs, extracts command/state mappings, subscribes to command topics
Publishes to state topics
Delegates cover/lock/alarm to
Looks up feature topics
Looks up component type
Populates feature mappings
Populates entity mappings
<<component>>
Transition Handlers
[Methods]
Handle stateful transitions for cover, lock, and alarm components
<<component_db>>
Feature Map
[Dict]
Light/fan feature topics (brightness, RGB, percentage, etc.)
<<component_db>>
Entity Map
[Dict]
command_topic -> (state_topic, component_type)
Publishes intermediate & final states via
Uses for simple echo command to state
<<component>>
State Publisher
[Method]
Publishes state updates to MQTT (QoS=1, retain=true)

Figure 20: C4 Component Diagram of the Device Simulator Container

Device Simulator is the main orchestrator class responsible for managing the MQTT client lifecycle, establishing broker connections, and coordinating message routing between discovery, command handling, and state publishing components. It maintains the service's runtime state and handles graceful shutdown scenarios.

The **Discovery Handler** parses *HA MQTT* discovery configuration messages published to `homeassistant/+/+/config` topics. It extracts command-to-state topic mappings for each discovered entity, populates internal data structures, and dynamically subscribes to relevant command topics to enable bidirectional communication.

When receiving a command message from the Broker, **Command Router** routes them to appropriate handlers based on component type. Simple devices like switches or lights are handled through direct echo the command to the state, while stateful devices like covers, lock and alarms are delegated to specialized **Transition Handlers** that simulate realistic physical behavior by managing asynchronous state transitions in separate threads. Covers transition from `closed` through `opening` to `open` or from `open` over `closing` to `closed`, locks from `UNLOCKED` through `LOCKING` to `LOCKED` or from `LOCKED` through `UNLOCKING` to `UNLOCKED`. Alarm panels go from `disarmed` through `arming` to `armed` and from `armed` over `disarming` to `disarmed`. It also supports cancellation for commands like `STOP` or when new commands override ongoing transitions.

The **State Publisher** publishes device state updates to *MQTT* state topics with Quality of Service (*QoS*) of `1` and message retention enabled. This allows late-joining clients like *HA* after a restart, or when we take as snapshot of the state by the **MQTT Device Manager** to receive the last known state of each entity.

This all needs some internal state to keep track of the devices and their topics: In-memory dictionary data structure **Entity Map** stores the mapping from command topics to tuples of (state_topic, component_type). It enables efficient lookup during command routing to determine both the destination for state updates and the appropriate handling logic based on device type. In addition the **Feature Map** stores additional feature-specific topic mappings for advanced device capabilities such as light brightness, RGB color, color temperature, and fan percentage, preset modes, oscillation, and direction controls. It allows fine-grained control of complex device features beyond simple on/off states

8.3 Interfaces

In the benchmarking setup two main interfaces are defined to interact with the different components: The *REST API* of the orchestration system to submit new Tasks to the Queue and the usage of the *MQTT* protocol to manage the virtual devices and sensors.

8.3.1 Orchestration System REST API

The orchestration system exposes a *REST API* to submit new benchmark tasks to the task queue. The API exposes a single endpoint:

- `POST /tasks` : Submit a bunch of benchmark tasks to the queue.
  The request body is a *JSON* object like this

  ```
  {
    "definition_ids": [
      1,2,3,4
    ],
    "llm_names": [
      "falcon-h1-0.5b-instruct", "falcon-h1-1.5b-instruct"
    ]
  }
  ```

  This would queue all tasks in any combination of the task definitions with the given LLM names. It schedules in the order of the definition ids and for each definition id it runs all the given LLM names before moving to the next definition id. Both, the definition ids, and the model names can be set to `null` resulting in populating these values with all possibilities.

Typically for *FastAPI* applications, there is also an automatically generated OpenAPI specification available at `/openapi.json` and an interactive API documentation at `/docs` .

8.3.2 MQTT Protocol

MQTT is a lightweight messaging protocol built on *TCP* commonly used in Internet of Things (*IoT*) applications for communication between devices and systems. It follows the publish-subscribe pattern. A *MQTT client* publishes messages to a *MQTT broker* under specific topics. Other *MQTT clients* can subscribe to these topics to receive the messages. The broker is responsible for managing the distribution of messages between publishers and subscribers. The topics are hierarchically structured strings separated by slashes (`/`) like file paths. Clients can use wildcards to subscribe to multiple topics at once [25].
Assume sensors publish in these topics:

- `smart_home/living_room/temperature`
- `smart_home/living_room/humidity`
- `smart_home/kitchen/temperature`
- `smart_home/kitchen/humidity`

A client can subscribe to `smart_home/kitchen/humidity` to get all messages for the kitchen's humidity. It is also possible to use wildcards to subscribe to a topic pattern. A `+` is used for a single level wildcard, `#` for multiple levels. A client subscribes `smart_home/+/temperature` to receive temperature updates from all rooms, or `smart_home/#` to receive all messages from the home. [26]

The message itself can by any binary payload, but *JSON*, unicode strings or simple numeric values are common choices. With the message payload, additional metadata can be sent like *QoS* level, or retain flag. Out of the 3 *QoS* levels we use level 1

- Level 0: "At most once", the publisher sends the message to the broker and don't wait for a confirmation

- Level 1: "At least once", the publisher sends the message to the broker and waits for a confirmation. If the broker does not confirm within a set time, the message is sent again. This may lead to multiple deliveries of the message.
- Level 2: "Exactly once", a more complex handshake ensures that the message is delivered exactly once. This is the safest and slowest level.

All these levels also applies to the subscriber side. But the *QoS* level cannot be upgraded by the subscriber. If a message is published with level 0, a subscriber requesting level 2 will still receive it with level 0 [27], [28].

Home Assistant supports connecting to a *MQTT broker* and subscribes to the `homassistant/#` topic. In this topics devices and their states can be published with a special discovery topic which *HA* understands. It handles them as they were physical devices using different protocols. We use that, to provide virtual devices to *HA*.

```
{
  "name": "Living Room RGB Strip",
  "unique_id": "mqtt_publisher_light_living_rgb",
  "object_id": "mqtt_publisher_light_living_rgb",
  "device": {
    "identifiers": ["light_living_rgb_001"],
    "name": "RGB LED Strip",
    "manufacturer": "Smart Home Corp",
    "model": "LED Strip Pro v2"
  },
  "state_topic": "homeassistant/light/light_living_rgb/state",
  "command_topic": "homeassistant/light/light_living_rgb/set",
  "icon": "mdi:led-strip-variant",
  "payload_available": "online",
  "payload_not_available": "offline", "qos": 0,
  "retain": false,
  "brightness_state_topic": "homeassistant/light/light_living_rgb/brightness/
state",
  "brightness_command_topic": "homeassistant/light/light_living_rgb/brightness/
set",
  "rgb_state_topic": "homeassistant/light/light_living_rgb/rgb/state",
  "rgb_command_topic": "homeassistant/light/light_living_rgb/rgb/set",
  "color_temp_state_topic": "homeassistant/light/light_living_rgb/color_temp/
state",
  "color_temp_command_topic": "homeassistant/light/light_living_rgb/color_temp/
set",
  "min_mireds": 153, "max_mireds": 500,
  "payload_on": "ON",
  "payload_off": "OFF"
}
```

Listing 3: Example discovery message readable by *Home Assistant*

The above example is published to `homeassistant/light/light_living_rgb/config`.

The **Device Simulator** also connects to the broker and does the same discovery as *HA*. It also subscribe to the command topics to reflect the command and publish a new state to the state topics.

9 Operational View

9.1 Setup

To start benchmarking the *SmartEdge Assistant* solution, systems must be started and configured properly: The replicas of the worker, and the orchestration system, which has to know how to reach the different workers.

9.1.1 Preparation

1. Clone the repository to your local machine:

   ```
   git clone https://gitlab.ost.ch/smart-edge-assistant/smart-edge-assistant.git
   ```

2. By starting the orchestration system, all the needed config files are generated automatically, but changes to them are required. Run `docker compose up -d model_testing` from the root folder of the repository. This will populate the needed folders and config files in the `model_testing/resources/` folder:
   - `model_testing/resources/base/`: Contains base configuration files for the small flat (chapter 7.1.2.1) larger house (chapter 7.1.2.2) setups. It is possible to modify these files to add, change or remove devices.
   - `model_testing/resources/deployments/`: Contains the configuration file where all the worker instances can be found. They need to run on the same host, as only the ports can be specified. Currently this file has no entries, preventing the orchestration system from starting.
   - `model_testing/resources/test_cases/`: Contains the `benchmark_test_set.xlsx`, the file with all the fields as defined described in Table 4.
3. copy the `.env.example` to `.env` in the `model_testing/` folder and adjust the variables. The three variables `JUDGE_LLM_API_KEY`, `JUDGE_LLM_BASE_URL` and `JUDGE_LLM_MODEL_NAME` must be set to the desired *LLMs* provider to use for judging the benchmark results. The *LLM* provider for the judge, must be *OpenAI API* compliant. The other variables can be left as is for local testing.

9.1.2 Start Worker Replicas

The worker replicas are created from the template folder `benchmarking/instance_template/`. To create multiple replicas listening on different ports, a small script is provided: `deploy_replicas.sh`. Run the script from the `benchmarking/` folder, providing the number of replicas to create as argument. For example to create 3 replicas and let them start immediately run:

```
./deploy_replicas.sh 3 --up
```

This will create 3 folders `ha-benchmark_0/`, `ha-benchmark_1/`, `ha-benchmark_2/` in the `benchmarking/` folder, each containing a copy of the template with adjusted port mappings. The script will also start the worker instances automatically by executing `docker compose up -d --wait` in each folder. When the script ends it prints out a *JSON* config which can be used by the orchestration system to know how to reach the different workers. Copy this output and paste it into the `model_testing/resources/deployments/deployment_config.json` file. When ports are blocked or already in use, provide different start ports to `deploy_replicas.sh`. use `deploy_replicas.sh --help` to see all available options.

9.1.3 Start Orchestration System

The orchestration system can now be started by running `docker compose up -d --build model_testing` from the root folder of the repository. The orchestration system will read the `deployment_config.json` spin up all worker threads and the *REST API* to schedule benchmark tasks. The API will be available at http://localhost:8003 by default.

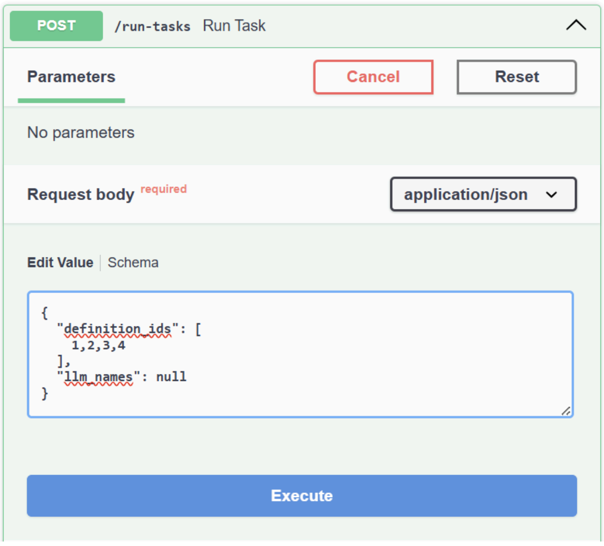
9.1.4 Seeding the Benchmark Tasks into the Database

Before starting the benchmark execution, the benchmark tasks must be seeded into the central database. This can be done by running

`docker compose exec model_testing poetry run python src/model_testing/seed.py` from the root folder of the repository. This will read the `benchmark_test_set.xlsx` file from the `model_testing/resources/test_cases/` folder and generate all the single tasks by combining the base definitions with the different preconditions as described in chapter 7.1.2. The tasks will be stored in the database and are ready to be executed. The *SQLite* database can be found at `model_testing/data/test.db`

9.1.5 Scheduling Benchmark Tasks

To start the benchmark execution, tasks must be scheduled via the *REST API*. To do it manually, the simplest is to use the provided *Swagger UI* at http://localhost:8003/docs.



```
{
  "definition_ids": [
    1,2,3,4
  ],
  "llm_names": null
}
```

Figure 21: Swagger UI of the Benchmark Orchestration System, about to schedule the benchmark tasks 1,2,3,4 for all available models.

9.2 Using LLMHub to run the models for benchmarking

Many of the *LLMs* used to benchmark could be deployed on *LLMHub*. We found out, that they used a queue per model to schedule the chat completion requests. To take advantage of the parallelization of the worker replicas, the scheduling has to be modified. Instead of scheduling all definitions of one model first, before moving to the next model, the tasks should be scheduled for all models in a round-robin fashion. before moving to the next task definition. This way, all the *LLMHub's* model queues can be used in parallel.

During the benchmarks tens of thousands of requests are sent to the *LLMHub* instance. Some fluctuations in performance were noted. Running with 9 worker replicas, the throughput of benchmark tasks varied between 35 and 292 per hour.

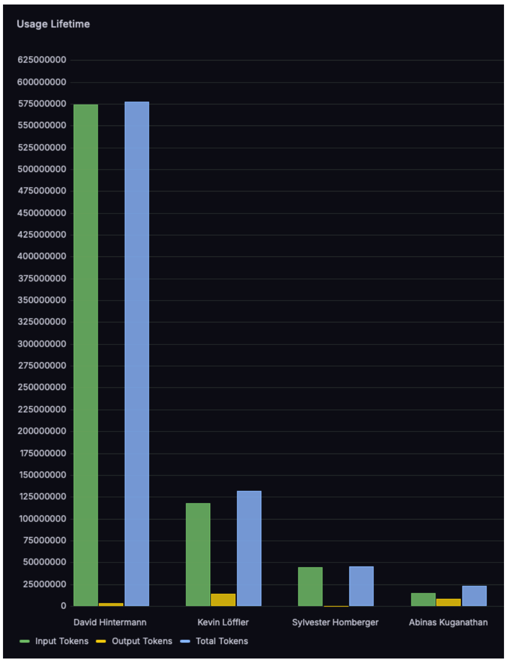


Figure 22: View of LLMHub's monitoring dashboard after the benchmarking showing lifetime token usage by user

10 Conclusion

This project successfully demonstrates that a fully local, privacy-focused voice assistant for smart home control is technically feasible. The *SmartEdge Assistant* integrates with Home Assistant's existing infrastructure while keeping all processing on-premises, addressing growing privacy concerns around cloud-based voice assistants. By using the *MCP* as an interface between large language models and Home Assistant, the system uses state of the art technology and is ready to integrate even more *MCP* services in the future.

The benchmark developed as part of this project provides insights into the capabilities and limitations of different LLM configurations for smart home control. The results clearly demonstrate that model size correlates with performance. Larger models consistently achieve better scores across all task categories. GPT-5 set a performance upper level with `1005.5` out of `1758` possible points, while open-weight models like gpt-oss-120b (`895.5` points) and Qwen-3 Coder 30B (`812.5` points) achieved competitive results. Among smaller models suitable for consumer hardware, Falcon H1 34B (`679.5` points) and Qwen-2.5 Coder 32B (`597.5` points) performed best.

Interestingly, the benchmark revealed that increased smart home complexity does not necessarily degrade performance. Models performed similarly whether controlling a simple flat or a complex house with many more rooms and devices, suggesting that capable models can handle real-world home configurations effectively. However, even the best-performing model did not achieve perfect scores, indicating opportunities for improvement through enhanced tooling, richer contextual information, and better prompt engineering.

The main challenges for fully local deployment remain hardware requirements and latency. Running larger, better performing models demands high-end consumer hardware, and inference times can be noticeably higher than cloud-based alternatives. Nevertheless, this project proves that local voice assistants can deliver meaningful functionality while respecting user privacy.

Future work on *SmartEdge Assistant* could focus on optimizing tooling, context management and prompt engineering to further boost performance. The chosen strategy in this minimum viable product of exposing *SmartEdge Assistant* as Ollama compliant *LLM* could be challenged by integrating more tightly with *HA* to have full control over context windows and state management within the full cycle of *STT*, *Conversation Agent* and *TTS* or completely decouple it from *HA*'s *Assist* framework and use *HA* as external but controllable system over *MCP*. The benchmark itself can be extended to support multi-turn dialogues, to better reflect real-world voice assistant interactions. Overall, this project lays a solid foundation for privacy-focused local voice assistants and highlights promising directions for future enhancements. The project is open sourced to allow others to build upon this work.

Part III

Project Documentation

11 Agreement

The following section contains the agreement. The agreement was written in German and copied word for word below.

Aufgabenstellung Semesterarbeit HS2025 «*SmartEdge Assistant*»

Betreuungspersonen

Mitra Purandare (mitra.purandare@ost.ch)

Studierende

David Hintermann (david.hintermann@ost.ch) Sylvester Homberger (sylvester.homberger@ost.ch)

Ziel der Arbeit

Ziel dieser Semesterarbeit ist es, einen Sprachassistenten für Home Assistant zu entwickeln

Im Zentrum stehen folgende Kernfunktionen:
1. Input in natürlicher Sprache
2. Volle Funktionalität ohne Internetverbindung
3. Steuerung von Geräten über Home Assistant durch MCP

Es wird nach agilen Grundsätzen entwickelt und allfällig freie Ressourcen in Erweiterung und Verbesserung des Produkts investiert.

Auftrag

Die folgenden Teilaufgaben sind zu bearbeiten

- **Architektur & Design:**
  - Wahl einer geeigneten Architektur und Technologie-Stacks - passend zu den bestehenden Services der Plattform
  - Datenmodell und Schnittstellen entwerfen
  - UX Konzept für die Interaktion über die Sprachschnittstelle
- **Implementation & Test:**
  - Implementation der Kernfunktionen (siehe Ziel der Arbeit)
  - Testkonzept entwickeln und umsetzen
- **Agile Entwicklung:**
  - Weiterentwicklung des Produkts in Absprache mit der Betreuungsperson
  - Funktionale Anforderungen (FR) und nicht-funktionale Anforderungen (NFR) werden dynamisch weiterentwickelt und überwacht
  - Beschreibung von Erweiterungsmöglichkeiten

Erwartete Resultate

- Funktionierender Prototyp mit den definierten Kernfunktionen
- Dokumentation von Setup, Architektur, Anforderungen, Tests und Erweiterungsideen
  - Die Dokumentation dokumentiert das Endresultat
  - Die Fortschritte werden mit einem Auszug der Tickets im Anhang dokumentiert
- Erste Auswertungen

Termine

15.09.2025, Beginn der Arbeit

19.12.2025 bis 17:00, Abgabe der Studienarbeit

(Bei Bedarf:) Jeden Donnerstag 13:00 – 14:00 wöchentliches Meeting (online oder vor Ort) mit Kommunikation des Fortschritts

(Bei Bedarf:) 2 Codereviews

Beurteilung

Für die Beurteilung ist die Betreuungsperson zuständig. Die Gewichtung der Bewertung folgt dem Leitfaden für SA/BAs.

Kriterium	Gewicht
Organisation und Durchführung	20%
Formale Qualität des Berichts	20%
Analyse, Entwurf und Auswertung	20%
Technische Umsetzung	40%

Zulässige Hilfsmittel und weitere Betreuung

- Bei Bedarf kann Unterstützung in den Bereichen App-Architektur, Datenmodellierung, Usability und Schnittstellendesign gewährt werden.
- Durch die Betreuung werden ausserdem regelmässige Codereviews vorgenommen.

- Das Nutzen von KI-Tools in sämtlichen Bereichen der Arbeit ist zulässig, wenn für diesen Zweck deklariert. Die Resultate werden von den Nutzern verantwortet.

Weitere Regelungen

- Die Artefakte werden auch mit einer Opensource-Lizenz lizenziert (vorzugsweise MIT).

Ort, Datum

Mitra Purandare

Ort, Datum

David Hintermann

Ort, Datum

Sylvester homberger

12 Project Management

12.1 Method

We initially adopted Scrum [29] as our project management framework, though we deliberately chose not to assign formal Scrum roles given the size of our micro-team. Our sprints followed a two-week cadence, strategically aligned with our regular advisor meetings to facilitate progress reviews and guidance.

However, as the project evolved, we recognized that the Scrum overhead was consuming a disproportionate amount of our limited resources. The ceremonies, planning sessions, and administrative aspects that work well for larger teams were creating friction rather than value in our two-person context. Consequently, we made the pragmatic decision to transition to a more streamlined discuss-implement-check approach, which allowed us to redirect our energy toward the actual development work while maintaining the coordination and structure necessary for effective collaboration.

12.2 Risk Management

This chapter describes potential risks for this project. They were analyzed with a risk matrix. Risks in a risk matrix are the product of the probability and the impact of that risk. Those risks where then mitigated to an acceptable level or below. This helps to perceive and handle risks early in an effective manner.

12.2.1 Risk Tables

Tag	Risk	Description	Mitigation
		Architecture	
RA01	SE-HA Bridge Integration	Multiple integration approaches exist for connecting the two systems, each with different capabilities and trade-offs. The project's functionality depends heavily on this architectural decision. Extensive evaluation may cause schedule delays and potentially limit system capabilities.	Prioritize early investigation and conduct formal analysis of integration options to inform decision-making.
RA02	LLM Performance Constraints	The selected *LLM* may be constrained by available hardware and prove insufficient for the engineered tooling requirements. This could result in additional development effort, reduced capabilities, or failure to meet project objectives.	Analyze hardware options with sufficient performance headroom. Conduct early feasibility testing of *LLM* capabilities.
RA03	Speech-to-Text Technology Uncertainty	Limited team expertise with *STT* technology. Key unknowns include start/stop detection mechanisms and out-of-the-box availability. Could become a single point of failure requiring significant additional work.	Conduct early proof-of-concept and technical evaluation of *STT* solutions.
RA04	*HA* Setup Automation	Time required to develop Home Assistant setup scripts is unclear, yet automation is essential for efficient development workflow. Delays in this area will compound throughout the project.	Prioritize script development in early project phases.

Table 10: Table of Risks - Architecture

Tag	**Risk**	**Description**	**Mitigation**
		Infrastructure	
RI01	Hardware Procurement Delays	Home Assistant hardware orders may experience shipping delays, directly impacting project timeline.	Place hardware orders as early as possible in the project schedule.
RI02	*GitLab* Service Interruption	*OST GitLab* instance may become temporarily unavailable, halting development work.	Implement repository mirroring to alternative Git hosting service.
RI03	*GitLab* Extended Outage	*OST GitLab* instance experiences prolonged unavailability or data corruption, resulting in potential work loss and extended delays.	Maintain repository mirrors with regular synchronization.
RI04	Hardware Failure	Physical damage to project hardware could require replacement, causing delays, potential data loss, and additional costs.	Accept risk. Budget contingency for replacement costs and schedule buffer for procurement delays.
RI05	Catastrophic Data Loss	Complete loss of project data due to human error, malware, or unforeseen circumstances, jeopardizing project completion.	Enforce strict Git workflow with regular commits and pushes. Utilize multiple remote repositories.

Table 11: Table of Risks - Infrastructure

Tag	Risk	Description	Mitigation
		Project Management	
RM01	Short-term Personnel Unavailability	Team member becomes temporarily unavailable due to illness or personal circumstances for a brief period (days to one week).	Reduce feature scope if necessary.
RM02	Long-term Personnel Unavailability	Team member becomes unavailable for an extended period (multiple weeks) due to serious illness or other circumstances.	Immediately consult with advisor to redefine project scope and adjust deliverables.
RM03	Unforeseen Work Requirements	Unexpected tasks or technical challenges emerge that were not identified during initial planning, requiring additional effort.	Conduct regular progress reviews with advisor. Proactively identify emerging issues and collaboratively develop solutions.
RM04	Team Internal Conflicts	Interpersonal problems arise between team members, reducing collaboration efficiency and work quality.	Hold regular retrospectives to address concerns early. Foster open communication and conflict resolution.
RM05	Advisor Relationship Issues	Communication or expectation misalignment with project advisor affects project direction, quality, or final assessment.	Maintain transparent, honest, and respectful communication. Schedule regular review meetings. Involve department head for mediation if issues persist or intensify.

Table 12: Table of Risks - Project Management

Legend

Color	Risk
🟥	Very High
🟧	High
🟨	Medium
🟩	Low
🟦	Eliminated

Table 13: Legend Risks

12.2.2 Risk Matrix - Initial State

Probability	Impact			
	Negligible	Moderate	Critical	Catastrophic
Certain	🟨	🟧	🟥	🟥
Likely	🟨	🟧 RA04	🟧 RI01	🟥
Possible	🟩 RI02	🟨 RA01, RA03, RM01	🟧 RA02, RM03	🟥
Rare	🟩	🟩 RI04	🟨 RI03, RM04, RM05	🟧 RM02, RI05
None	🟦			

Table 14: Risk Matrix - Initial State

12.2.3 Risk Matrix - After Mitigation

Probability	Impact			
	Negligible	Moderate	Critical	Catastrophic
Certain	🟨	🟧	🟥	🟥
Likely	🟨 RA04	🟧	🟧	🟥
Possible	🟩 RA03, RM01	🟨 RI01	🟧	🟥
Rare	🟩 RA01, RI02, RI03, RI05	🟩 RI04, RM03, RM04, RM05	🟨 RA02, RM02	🟧
None	🟦			

Table 15: Risk Matrix - After Mitigation

12.3 Increment Log

This chapter contains a short description of the merged increments on our product. Other work (e.g. writing documentation, setup of infrastructure) was also done, but is not part of the code.

Commit Hash & Summary Title	Description
85c14a6f0f32984a2bfb016dd566a5cc9383449f Implement STT using Vosk	This commit implements a full *STT* feature using *Vosk*. It introduces a *FastAPI* controller exposing a `/transcribe` endpoint, which handles incoming transcription requests and delegates processing to a `TranscriptionService`. The service decodes *base64* audio, converts it to 16 kHz mono Waveform Audio File Format (*WAV*) via `AudioConverterService`, and transcribes it using cached *Vosk* models managed by `VoskModelManager`, which can download and load models on demand. Transcription results, including word-level segments and confidence scores, are mapped to the project's `TranscriptionResponse` format by `TranscriptionResponseMapper`. Overall, this commit sets up a modular, reusable, and scalable *STT* pipeline integrated into the *SmartEdge Assistant* system.
cdf55f87db54a4d2bfd600cf8e34536fac51911b Write *HA* Setup Instructions	This commit provides a practical *Home Assistant* setup workflow integrated into the project repository. It includes step-by-step instructions for configuring a TP-Link router, installing *Home Assistant* OS, managing backups, and restoring the system, as well as connecting peripherals like the ZBT-1 Zigbee adapter, Aqara sensors, smart plugs, and Yeelight devices. Importantly, it adds a local backup folder within the project to store backups, while .gitignore ensures these files are not tracked, supporting the workflow for resetting the *HA* system efficiently. This setup ensures the environment can be reliably reset and configured even without a fully scripted process.

5120ce92476c616bf6a2ae611ab74eea477d3608 Implement Conversation Agent Container	This commit implements the core *SmartEdge Assistant* system. It provides *FastAPI* controllers for `/chat`, `/tags`, and `/version`, with `/chat` serving as the main interface for *Assist* via an *Ollama*-compliant streaming endpoint. Incoming instructions from *Home Assistant* are converted into structured chat messages and then passed to a configured *LLM*, which can utilize *MCP* tools from *Home Assistant* to perform actions. The commit defines *Pydantic* models for messages, tool calls, options, and responses, and implements a *Langchain*-based generator capable of streaming partial responses or returning full responses. This establishes a modular, pluggable, and fully integrated conversation flow between *Assist* and the *LLM*.
5a3290fd37264906e331aa5962e2515809724e35 Mock *Home Assistant* Devices	Introduces a full device simulation framework for benchmarking *SmartEdge Assistant* systems with Home Assistant and *MQTT*. Provides a pre-populated *Home Assistant* instance template, *MQTT* broker configuration, device simulator, and sample device *JSON* data. Includes deployment scripts to launch multiple replicas for load testing or integration tests.
314de0013a69f71e9df7b06eea225fb0e377aee0 Model Name Passthrough	This commit updates the LangChainAgentGenerator so that it can optionally pass the requested model name from the incoming chat request down to the upstream *LLM* provider. This change is controlled by an environment variable: if set, the generator re-initializes the agent for each request using the specific model requested, instead of using a single pre-configured agent. Additionally, a minor cleanup removes the hardcoded system message from the prompt, making the generator fully rely on incoming messages for context. The `\_ensure_agent` method now accepts a `model_name` parameter to support this dynamic initialization. Overall, the commit improves flexibility by allowing per-request model selection.

acf5f2ba585f64faad56f6717dff9bdef169ca65 Add Model Testing Harness	The model testing harness runs automated tests on *LLMs* for *Home Assistant* scenarios. It uses a *FastAPI* service to manage a worker pool that executes tasks defined in the database. Each definition specifies a prompt, expected output, and configuration, while a run records the result, including response, applied config, errors, and quality score. Tasks interact with *MQTT*-simulated devices and the conversation agent, then evaluate correctness programmatically or via *LLM* judgment. Configuration is environment-driven, making deployments and worker scaling flexible.
11a1f3639d5205d8788e4a36a8cb66fbeb7b7390 Integrate Judge	This commit expands the model testing system so that each test run can now understand what type of home it's simulating, like a flat or a house. The seed definitions were updated to include this context. The deployment setup was slightly simplified by removing a token field that wasn't strictly needed. Most importantly, a proper judging system was added: instead of blindly marking results as false, an *LLM* now evaluates the assistant's responses and produces a structured judgment with a score and rationale. The task executor was modified to integrate this *LLM*-based judge into its workflow, making automated evaluation more intelligent. Minor code fixes and cleanup also improved session handling and worker logic, keeping the system stable.
0105d7dab6ecca2cae7fbf8f05a15a7e56772493 Prepare Home Base Data Set	This commit prepares the benchmarking and model-testing foundation by introducing complete base home configurations and a curated set of benchmark test cases. It adds realistic, fully populated mock *Home Assistant* states (flat and house) that serve as reproducible starting points for automated evaluations, and pairs them with a structured benchmark test set. The accompanying *Home Assistant* instance template is updated so these datasets can be loaded and exercised consistently during benchmark runs. Overall, this establishes the concrete data layer needed to run repeatable, scenario-based model evaluations rather than focusing on runtime logic changes.

ebded3e52a7aeaf3d353526845c14bba91c251a6 Seed Tests	This commit introduces a complete data-driven benchmarking pipeline for the model_testing module. Test definitions are no longer hard-coded; they are loaded from an Excel benchmark file, expanded across supported base home configurations, and seeded into the database. The database schema is updated to use explicit Structured Query Language (*SQL*) enums and *JSON* fields, and run records now persist judging rationale and dialogue continuation status. Task generation and execution are made configurable and scalable by replacing derived port logic with an explicit deployment configuration file. Judging logic is strengthened through semantic configuration comparison, structured *LLM* judge output, and improved error reporting. Overall, the changes enable reproducible, configurable, and fully automated benchmark execution across multiple models and deployments.
c926d7770d60265a8a0f63b0de36edd6a889f1bb Support Scheduling Multiple Test-IDs and Models	This commit enhances the benchmarking system to support running tasks for multiple models and multiple definition IDs at once. The /run-tasks API now accepts lists (`definition_ids` and `llm_names`), and the make_tasks function was updated to generate all combinations of selected definitions and models. Several *LLM* enum values were also updated for naming consistency. This enables bulk execution of tasks and more flexible benchmarking workflows.

Table 16: Increment Log

13 Standards

To get a clean workflow for collaboration, we need to follow some standards. These explains the way we use the *GitLab* as project management software and *Git* as version control system.

13.0.1 Conventions for *Git*

- Use predefined branch names for all kind of work: `issue/<issueNumber>-<title>` (e.g. `issue/42-fix-login-bug`)
- Use *Conventional Commits* for commit messages (e.g. `fix(TTS): correct minor typos in code`)
- Create a merge request and let it be approved
- Only merge a squashed commit with fast forward strategy to `main`

13.0.2 Handling issues in *GitLab*

Always create an issue if it does not exist for any new feature or bug.

13.0.2.1 Definition of Ready

These points must be fulfilled to put the issue in the sprint backlog:

- Scope is clear and defined
- Estimation done
- Acceptance Criteria defined
- No Blockers

13.0.2.2 Definition of Done

These points must be fulfilled to close the issue:

- Pipeline must have succeeded
- Automate tests to run in *CI*/*CD* pipeline when appropriate
- Acceptance Criteria passed
- (Code) Review done and approved
- Documentation up to date
- Merged to main
- Time tracked on issue

14 Reflection

14.1 David Hintermann

The semester is coming to an end, and this project is nearly completed. Working on the project has been a good experience. The focus shifted during the semester, but this is, what we decided to do at the beginning: First deliver a minimal viable product and then let our stakeholders decide next steps in an agile way. This approach led us to define a benchmark specification and build everything needed to run the benchmarks. I learned a lot about working with large language models in a programmatic way. When I started working on the benchmark specification, I had no idea how many tokens we will require to prompt the different models. It was eye-opening to see how quickly token counts accumulate and how expensive that can become. On OST's LLMHub we processed over 620 Million tokens during the benchmark runs. In future projects, I will be more careful to use token efficient strategies to prompt the models.

Collaboration with Sylvester went well over all. But sometimes I felt that we had different approaches to solve problems. He is more the perfectionist type, defining a plan, consider any possible challenge in advance and then deliver a high quality solution, ready for production. I, on the other hand am more the pragmatic type, delivering a working solution quickly and then improve it iteratively. I try to tackle the challenges as they arise, not trying to foresee everything advance. This led to some discussions about the "right" approach to solve a problem. But in the end, I'm happy with the result we achieved together.

The project progressed at a consistent pace until the final weeks of the semester. I think it was a good idea to schedule two fixed days per week to work on the project. This helped me keep the momentum going. I worked on the project at my employer's office, which gave me the correct mindset to focused. Everybody around me was working on their own tasks as well. For future projects, such as my bachelor thesis, I plan to keep the same routine. Towards the end of the semester, the workload increased significantly: Getting the whole benchmarking solution running, was more difficult and time consume than expected. Many different components needed to work together. At the same time, several mini-projects for different courses were due. In addition, I had to complete year-end tasks at my job, and I also got engaged to my girlfriend. All together this led to a stressful final phase of the semester. Because of this experience, I decided to postpone the bachelor thesis rather than starting it immediately after this project.

14.2 Sylvester Homberger

This project has been a valuable learning experience. From the beginning, we approached it with deliberate planning. We made a conscious decision to structure the project into a guaranteed minimum deliverable and an agile, stakeholder-driven improvement phase. I believe this careful scoping was crucial to our success. It ensured we could fulfill our commitments while remaining flexible about the direction of additional work.

Given the uncertainty around how to achieve our goals, we adopted an iterative approach, tackling the highest-risk components first. This strategy meant that some of our early work was eventually archived when we discovered better solutions. In retrospect, perhaps more upfront analysis could have prevented this, though I'm not convinced we would have avoided different mistakes instead. Some problems reveal themselves only through doing. Hitting reality early exposed both challenges and opportunities we couldn't have anticipated through analysis alone.

One insight from this project concerns formal project management overhead. We used Scrum sprints and ceremonies, which worked well in my previous Software Engineering project with five team members. With just two people, however, the synchronization overhead felt disproportionate. Larger teams benefit from unified processes and concentrated leadership, but for micro-team projects like this, I would consider a lighter approach. (perhaps just a discuss-implement-check loop with scheduled retrospectives) scaling up to more formal methods only if needed.

I genuinely enjoyed working with the technical aspects of the project. Finding efficient ways to integrate LLMs programmatically was fascinating, as was wrestling with challenges like communicating error context to the models. Designing a benchmark specification that would actually yield meaningful insights required careful thought, and I found that process rewarding.

Collaborating with David worked well overall, though we brought different working styles to the table. David excels at producing working code quickly, leveraging coding assistants effectively. My approach tends toward simpler, more deliberate solutions - minimal but complete implementations, carefully structured to reduce future maintenance burden. Code not in the project is code I won't have to maintain later. This approach takes more time upfront, but I believe it pays dividends in maintainability. That said, I recognize that a good solution delivered on time beats a perfect solution delivered late, and David's pragmatism balanced my tendency toward perfectionism.

The project maintained steady progress throughout most of the semester, though the final weeks brought their challenges. Like David, I found the convergence of multiple deadlines and responsibilities created significant pressure. This experience has informed how I'll approach future work, including planning for my bachelor thesis.

A Used Abbreviations

AI	Artificial Intelligence
CD	Continuous Deployment
CI	Continuous Integration
DTO	Data Transfer Object
HA	Home Assistant
IFS	Institute for Software
IoT	Internet of Things
JSON	JavaScript Object Notation
LLM	Large Language Model
MCP	Model Context Protocol
MQTT	Message Queuing Telemetry Transport
NLP	Natural Language Processing
OST	Ostschweizer Fachhochschule
QoS	Quality of Service
SARA	Smart Assistant for Residential Automation
SQL	Structured Query Language
STT	Speech-to-Text
TGI	Text Generation Inference
TTS	Text-to-Speech
UI	User Interface
UX	User Experience
WAV	Waveform Audio File Format

B Bibliography

[1] *M5 Atom Echo device used as microphone and speaker.* Accessed: Dec. 18, 2025. [Online]. Available: https://assets.brack.ch/images2/7/6/6/180289667/180289667_xxl.jpg

[2] "CC0 1.0 Universal (CC0 1.0)." Accessed: Nov. 20, 2025. [Online]. Available: https://creativecommons.org/publicdomain/zero/1.0/

[3] "Custom Smart device, app and solution with Tuya." Accessed: Dec. 16, 2025. [Online]. Available: https://pages.tuya.com/expo/TuyaSmart_IoTServiceProvider

[4] "White Label Smart Home Solutions." Accessed: Dec. 16, 2025. [Online]. Available: https://www.mokosmart.com/white-label-solution-for-smart-device/

[5] "Smart Home Data Breach Exposes 2.7 Billion Records." Accessed: Dec. 16, 2025. [Online]. Available: https://blog.zonealarm.com/2025/02/smart-home-data-breach-exposes-2-7-billion-records/

[6] P. Wittkowski, "Nächster Smart Home-Anbieter insolvent: Schluss mit Cloud-Diensten, Apps und Geräte-Support." Accessed: Dec. 16, 2025. [Online]. Available: https://www.netzwelt.de/news/229297-beliebter-smart-home-anbieter-cloud-dienste-und-apps-geraet-support-jetzt-eingestellt.html

[7] *PUBLIC DEED regarding the establishment of the Open Home Foundation with registered office in Baar.* 2024. Accessed: Dec. 16, 2025. [Online]. Available: https://drive.google.com/file/d/1SVAZrl_kZmlXHzR1wTVvePEwZmuuUysi/view

[8] "Our private smart home." Accessed: Dec. 16, 2025. [Online]. Available: https://www.openhomefoundation.org/about/

[9] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, with Language Models*, 3rd ed. 2025. [Online]. Available: https://web.stanford.edu/~jurafsky/slp3/

[10] J. Giangola, "Conversation Design: Speaking the Same Language," Google Design. Accessed: Oct. 09, 2025. [Online]. Available: https://design.google/library/speaking-the-same-language-vui

[11] "Talking with Home Assistant - get your system up & running." Accessed: Dec. 10, 2025. [Online]. Available: https://www.home-assistant.io/voice_control/

[12] "Wyoming Protocol." Accessed: Dec. 12, 2025. [Online]. Available: https://www.home-assistant.io/integrations/wyoming/

[13] M. Hansen, *The Wyoming Protocol.* (Mar. 02, 2023). GitHub. Accessed: Dec. 12, 2025. [Online]. Available: https://github.com/rhasspy/rhasspy3/blob/master/docs/wyoming.md

[14] "OpenAI." Accessed: Dec. 12, 2025. [Online]. Available: https://www.home-assistant.io/integrations/openai_conversation/

[15] "Ollama." Accessed: Dec. 12, 2025. [Online]. Available: https://www.home-assistant.io/integrations/ollama

[16] "Ollama API Reference." Accessed: Nov. 15, 2025. [Online]. Available: https://docs.ollama.com/api/introduction

[17] "Model Context Protocol (MCP)." [Online]. Available: https://modelcontextprotocol.io/specification/2025-11-25

[18] “Model Context Protocol Server.” Accessed: Dec. 15, 2025. [Online]. Available: https://www.home-assistant.io/integrations/mcp_server

[19] “OpenAI API Reference.” Accessed: Dec. 12, 2025. [Online]. Available: https://platform.openai.com/docs/api-reference

[20] “OpenAI compatibility.” Accessed: Dec. 12, 2025. [Online]. Available: https://docs.ollama.com/api/openai-compatibility

[21] B. Ceunen, Ed., *Smart Home Commands Dataset.* (Dec. 06, 2020). Kaggle. Accessed: Nov. 20, 2025. [Online]. Available: https://www.kaggle.com/datasets/bouweceunen/smart-home-commands-dataset

[22] B. Rozière *et al.*, “Code Llama: Open Foundation Models for Code,” Meta AI, Aug. 2023. Accessed: Dec. 30, 2025. [Online]. Available: https://ai.meta.com/research/publications/code-llama-open-foundation-models-for-code/

[23] J. Zuo *et al.*, “Falcon-H1: A Family of Hybrid-Head Language Models Redefining Efficiency and Performance,” Jul. 30, 2025. Accessed: Dec. 30, 2025. [Online]. Available: https://arxiv.org/abs/2507.22448

[24] A. Yang *et al.*, “Qwen2 Technical Report,” 2024. Accessed: Dec. 30, 2025. [Online]. Available: https://arxiv.org/abs/2407.10671

[25] “MQTT Essentials.” Accessed: Dec. 18, 2025. [Online]. Available: https://www.hivemq.com/mqtt/

[26] “MQTT Topics, Wildcards, & Best Practices - MQTT Essentials: Part 5,” HiveMQ Blogs. Accessed: Dec. 18, 2025. [Online]. Available: https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/

[27] J. Deters, “Debunking Common MQTT QoS Misconceptions,” HiveMQ Blogs. Accessed: Dec. 18, 2025. [Online]. Available: https://www.hivemq.com/blog/debunking-common-mqtt-qos-misconceptions/

[28] “MQTT Publish, MQTT Subscribe & Unsubscribe - MQTT Essentials: Part 4,” HiveMQ Blogs. Accessed: Dec. 18, 2025. [Online]. Available: https://www.hivemq.com/blog/mqtt-essentials-part-4-mqtt-publish-subscribe-unsubscribe/

[29] K. Schwaber and J. Sutherland, “The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game,” Nov. 2020. Accessed: Dec. 30, 2025. [Online]. Available: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf

C Appendices

C.1 Code

The code is licensed under *MIT-License* and at the time of this report is hosted on the *OST GitLab* instance at:

https://gitlab.ost.ch/smart-edge-assistant/smart-edge-assistant

C.2 Test Cases

The components of the testcases are in the same repository as in chapter C.1.

- House devices base models: https://gitlab.ost.ch/smart-edge-assistant/smart-edge-assistant/-/tree/main/model_testing/defaults/resources/base
- Prompts and Verification Set: https://gitlab.ost.ch/smart-edge-assistant/smart-edge-assistant/-/blob/main/model_testing/defaults/resources/test_cases/benchmark_test_set.xlsx

C.3 Data

The results of our benchmarking are hosted in the same repository as in chapter C.1. They are provided in their original form as a *SQLite* database to keep them easy to access, filter and search. Their exact location is at:

https://gitlab.ost.ch/smart-edge-assistant/smart-edge-assistant/-/blob/main/benchmarking/results/test.db

C.4 AI Declaration

AI was used to:

- Generate and fix code
- Generate test cases for our benchmark
- Evaluate the results of failed hard-coded benchmark checks
- Copyedit our documentation (spelling, grammar, and style)
- Generate summaries

According to the agreement in chapter 11, the authors are responsible for the results.

C.5 Meeting Minutes

C.5.1 Advisor Meeting (2025-09-05)

Meeting Information

Date	2025-09-05
Participants	Mitra, David, Sylvester
Excused	-
Absent	-
Minute Taker	Sylvester

Agenda

Topic	Owner	Decision / Notes
Formal Agreement	Sylvester	• Agreed Goal: Speak natural language to some device and have a local LLM (no connection to internet) control some Devices over Home Assistant. This should be possible in half a semester. The remaining time is security margin and will be used for agile implementation of more features. • Mitra will send us the template. We will produce a draft that Mitra will receive it before the next meeting. We will finalize the formal agreement then.
Ordering Devices	Sylvester	• Process: Excel sheet -> Mitra -> OST buys • We would like two sets of PI, microphone/speaker, 2x Plugs, some sensor. Additionally, some other devices like LED light, Zigbee coordinator, etc. A first estimate is about 500 CHF. • Mitra thinks this is well within boundaries and won't be a problem. • Also, there might be some old devices lying around at OST. Mitra will have a look.
Kick-off Plan	Sylvester	• We plan to do complete above tasks before Thursday of the first week. We will use the morning to get an overview of what has to be done and plan the first sprint. We want to start it after a sanity check with Mitra in the afternoon. • Mitra recommends that the documentation should be set up beforehand. • We agreed on a reserved on-demand slot for the duration of the SA: Thursdays 13:00-14:00 • The SA Team will send the invitation for the next meeting taking place on 2025-09-18.

General Guidance	Sylvester	Mitra sees the following critical factors for the SA: • Open communications if there are problems • Start documentation early and consistently • Plan ahead, no coding till the last minute

Action Items

Task	**Responsible**	**Due Date**
Send template to us	Mitra	ASAP
Process agreement draft upon receipt	Mitra	2025-09-18
Process device order list upon receipt	Mitra	Upon receipt, 2025-09-18
Find if there are old devices lying around	Mitra	2025-09-18
Produce agreement draft	Sylvester	2025-09-10
Produce device order list	David	2025-09-10
Set up invite for next meeting	SA Team	2025-09-18

Upcoming Events

Event	**Date**
Next Meeting	2025-09-18

C.5.2 Advisor Meeting (2025-09-18)

Meeting Information

Date	2025-09-18
Participants	Mitra, David, Sylvester
Excused	-
Absent	-
Minute Taker	Sylvester

Agenda

Topic	**Owner**	**Decision / Notes**
Previous Minutes	Sylvester	• The minutes of 2025-09-28 have been approved.
General Sync	Mitra	• We checked if everyone has the same idea about the project. • Details were discussed in later topics.
Devices		• Mitra’s email advised using a more powerful hardware than Raspberry PI to run the LLM. It also suggested another device as voice terminal. • The equipment list was updated with the voice terminal. • The Raspberry PI remains as a deployment target for Home Assistant OS. • The project team will first implement the architecture without targeting performance from the LLM. The LLM is planned to be pluggable. Powerful hardware will be ordered later. The goal is that it can be run at home. • The equipment list was approved, and will be ordered from Christian Spielmann by Mitra. • Dominic Klinger (dominic.klinger@ost.ch) might have some left over Raspberries.
Project Agreement	Sylvester	• Usage of MCP was included in the project goals. • The project agreement (Aufgabenstellung Semesterarbeit HS2025 «*SmartEdge Assistant*») was approved. • All parties will physically sign it.
Planned Sprint	Sylvester	• Sprint 1 was presented. • Planned focus of all future sprints was presented. • Mitra saw no obvious problems.

Action Items

Task	**Responsible**	**Due Date**
Order Equipment	Mitra	ASAP
Add Mitra to Gitlab	Project Team	Next Meeting
Ask for Raspberries	Project Team	ASAP

Upcoming Events

Event	**Date**
Next Meeting	2025-09-25

C.5.3 Advisor Meeting (2025-09-25)

Meeting Information

Date	2025-09-25
Participants	Mitra, David, Sylvester
Excused	-
Absent	-
Minute Taker	Sylvester

Agenda

Topic	Owner	Decision / Notes
Previous Minutes	Sylvester	• The minutes of 2025-09-18 will be approved next time.
Project Agreement	Mitra	• Will be signed in person next time.
MCP Workshop	Sylvester	• Sylvester presented his research into the MCP protocol. The important division between MCP servers and clients & hosts was highlighted. • In our discussion we found Langchain to be the most promising candidate for MCP host (and client adapters). • On the MCP server side, there is a native server from Home Assistant, see: https://www.home-assistant.io/integrations/mcp_server. • The possibility to structure the voice interface as an MCP server was discussed. • The next step is to pull up a Langchain MCP client in combination with some simple MCP server (without Home Assistant). Best with a tutorial.

Action Items

Task	Responsible	Due Date
Print Agreement	Sylvester	Next Meeting

Upcoming Events

Event	Date
Next Meeting	2025-10-02

C.5.4 Advisor Meeting (2025-10-09)

Meeting Information

Date	2025-10-09
Participants	Mitra, David, Sylvester
Excused	-
Absent	-
Minute Taker	Sylvester

Agenda

Topic	Owner	Decision / Notes
Previous Minutes	Sylvester	• The minutes of 2025-09-18, 2025-09-25, and today will be approved next time.
Project Agreement	Mitra	• The project agreement was signed.
Hardware	David	• The hardware was received on Monday, 2025-10-06. 1x USB Card Reader is missing. • David will write to Mr. Spielmann
Progress Review	David	• David presents STT design • Sylvester presents Langchain - MCP progress. There are some performance problems. • Mitra mentions OpenAI Agents SDK.
Citations	Sylvester	• Mitra only needs citations for the actual core product. Quality measures and infrastructure can but don't have to have citation (e.g. testing, Git standards, pipelines, project management).

Action Items

Task	Responsible	Due Date
Find missing device	David	ASAP

Upcoming Events

Event	Date
Next Meeting	2025-10-17

C.5.5 Advisor Meeting (2025-10-23)

Meeting Information

Date	2025-10-23
Participants	Mitra, David, Sylvester
Excused	-
Absent	-
Minute Taker	Sylvester

Agenda

Topic	**Owner**	**Decision / Notes**
Previous Minutes	Sylvester	• The minutes of 2025-09-18, 2025-09-25, and 2025-10-09 have been approved.
STT Demo	David	• David presents the dockerized STT component. • A m4a sound file is uploaded, the transcribed text and meta information is returned
HA update	Sylvester	• HA infrastructure works • The router can connect to a WiFi uplink. This makes the infrastructure portable for demos. • Backups work, but only over the interface, full disk backups don’t work on Windows
Next Steps	Sylvester	• Sylvester presents the contents for the following Sprint: ‣ a real HA-MCP server ‣ connecting more powerful models ‣ connecting the microphone ‣ having the STT triggering the agent

Action Items

Task	**Responsible**	**Due Date**

Upcoming Events

Event	**Date**
Next Meeting	2025-10-30

C.5.6 Advisor Meeting (2025-11-06)

Meeting Information

Date	2025-11-06
Participants	Mitra, David, Sylvester
Excused	-
Absent	-
Minute Taker	Sylvester

Agenda

Topic	Owner	Decision / Notes
Previous Minutes	Sylvester	• The minutes of 2025-10-23 have been approved.
Architecture	Sylvester	• Iterated architecture was presented. Main changes where the use of the HA Voice Assistant feature and the abstraction of the LLM over a OpenAI API.
MCP Host	Sylvester	• LLMs from OpenAI and LLMHub are connected to the dev setup over an OpenAI API. • It should be easy to connect to a local deployment in the same way. • The usage of the LLM is structured by Langchain. • Large models can accomplish tool calling tasks correctly on the small test MCP Server.
Microphone to Prompt	David	• HA Voice Assistant feature is used to streamline microphone/speaker, STT/TTS and Conversation Agent integration. • Microphone has been flashed. • Ollama Conversational Agent will be used for now. It is the only available integration found that enables choosing your endpoint. • All components work, as evidenced by the intercepted Ollama calls on the Ollama proxy.
Next Steps	Sylvester	• Prompt into Langchain. • Spin up and connect to a real HA MCP Server. • System will probably be working and presentable next week.

Action Items

Task	Responsible	Due Date

Upcoming Events

Event	Date
Next Meeting	2025-11-07

C.5.7 Advisor Meeting (2025-11-13)

Meeting Information

Date	2025-11-13
Participants	Mitra, David, Sylvester
Excused	-
Absent	-
Minute Taker	Sylvester

Agenda

Topic	**Owner**	**Decision / Notes**
Previous Minutes	Sylvester	• The minutes of 2025-11-06 will be approved next time.
Presentation	David	• The Home Assistant hardware was set up in the office. Different sensors and actuators are connected. The used LLM is running on OpenAI or LLMHub, basically any OpenAI API speaking deployment can be connected. • Labeling the components in Home Assistant's model with descriptive names must be done by hand. E.g. This "light" is located in the "bedroom". The LLM can't know what to do without actually receiving any such information. • Input/Output can be done by microphone (wake word -> user prompt -> response) or over the chat interface in Home Assistant. The sessions are managed by Home Assistant Assist. The session remembers and re-injects the conversation during the session. During the presentation we found out that the microphone session can does so too. • Home Assistant Assist has configurable sub-components (microphone/speaker, TTS, STT, Conversation Agent). We configured "Ollama" Conversation Agent. This expects an Ollama compliant Server on the other side. • We reimplemented the Ollama API to get the prompt from the request. The prompt is handed off to our own agent. We respond with its response. • Tools are called over the official MCP Server integration. • The presented system was capable of interpreting and executing complex commands. E.g. "If the temperature is higher than 20 degrees, make the light green, else blue." resulted in the correct coloring. • More powerful LLMs can correct for weak STTs. E.g. miss-transliterated "e-light" was correctly inferred to be the "Yeelight" LED light strip. • The presented system fulfils the goals of the SA.
Deadlines	Sylvester	• The following deadlines were checked with advisor: ‣ 2025-12-15 | Upload "Broschüren-Abstract" ‣ 2025-12-19 17:00 | Upload all documents • Mitra wants to see the nearly ready draft one week before the final upload date for feedback. • The code does not have to be uploaded. Access to GitLab for advisor is sufficient.

LLMHub Deployments	Sylvester	• The SA Team doesn't have the hardware to deploy all the models and would like to deploy them on LLMHub to run tests. • The models are expected to range from 0.5B to 30B. They will be available from Hugging face. • Mitra thinks that should be possible. She needs a list of models. The models must have open weights.
BA	Mitra	• Mitra asks about the future of the project. It would be good if we could give something back to the community (e.g. an integration). • The SA Team doesn't know yet. • A different project focusing on text based memory and its management for LLMs would be of interest. Those memories could then be RAGed for context. Conflicting or outdated memories will have to be groomed regularly by the LLM. • One use case would be an autonomous bot on a software forge (e.g. Gitlab). It should not only know all the things that happened, but also intervene if it doesn't understand why a change is made. It should always be able to explain how and why things are. David brings the example of a change that was made some years back that just mentioned that the change was made, but not why. • A different use case would be to have it remember facts about Home Assistant after being informed by the user. E.g. the bedroom was named "dungeon" in Home Assistant. This link is only clear after forming a memory. This memory could be formed after being asked to turn off bedroom lights, not finding any "bedroom", and receiving the clarification.

Next Steps	Sylvester	• There is still a total of 5 weeks remaining. Thereof 2 weeks are needed for documentation and cleanup. 3 weeks can be used for improving or features. • The SA team sees potential in the following areas: ‣ Test models ‣ Connect the developed STT ‣ Better error handling ‣ Add memory ‣ Deploy and connect an actual local model with TGI ‣ Try out other MCP Server integrations ‣ Package the solution as integration. • The most important task was identified as testing models. This should include different models and some accuracy score. In the minimal version that is a qualitative pass or fail. From this working models can be identified, and a hardware recommendation can be given. Speed won't be tested, since we don't have the hardware. • If time remains: ‣ Try to deploy a TGI-GPU container locally. ‣ Packaging the solution.

Action Items

Task	**Responsible**	**Due Date**
Models List	Sylvester	ASAP
LLMHub deployments	Mitra	After next meeting

Upcoming Events

Event	**Date**
Next Meeting	2025-11-20

C.5.8 Advisor Meeting (2025-11-20)

Meeting Information

Date	2025-11-20
Participants	Mitra, David, Sylvester
Excused	-
Absent	-
Minute Taker	Sylvester

Agenda

Topic	**Owner**	**Decision / Notes**
Previous Minutes	Sylvester	• The minutes of 2025-11-06, and 2025-11-13 have been approved.
Test Set Generation	Sylvester	• The SA Team was confident that the Services (HA, LLMHub), and the orchestration of the test will be possible to implement. • They were unsure how the should produce the test data. ‣ The input prompts will probably be generable by a powerful model. This should be multiple prompts in 3 tiers (Simple, Complex, Confusing). ‣ There should be 2 HA configurations (clean, cluttered). Those can potentially be created by hand. ‣ The problem lies in the expected output. For simple questions the configuration (expected and actual) could be compared directly. But that won't work for any prompt for which the solution is somewhat open-ended. E.g. there are many correct ways to tell the current temperature ("15.3 degrees" vs "It is 15.3 degrees outside"). And the same hold true for changes in the configuration. ‣ In todays discussion we thought that the following might be practicable: – Simple tasks have their answer(s) generated by a powerful model. – If the static check fails, the result is brought before a powerful model acting as a judge, ranking it on some scale (e.g. 1-5). ‣ To prevent accidental explosions in cost, Mitra set a limit of 20 USD on our OpenAI project, which should automatically stop such a problem.

Action Items

Task	Responsible	Due Date

Upcoming Events

Event	Date
Next Meeting	2025-11-27

C.5.9 Advisor Meeting (2025-12-11)

Meeting Information

Date	2025-12-11
Participants	Mitra, David, Sylvester
Excused	-
Absent	-
Minute Taker	Sylvester

Agenda

Topic	**Owner**	**Decision / Notes**
Previous Minutes	Sylvester	• The minutes of 2025-11-20 have been approved.
Model Costs	Mitra	• Configured 20 USD where exceeded. Cost stand a bit below 70 USD. • There is still about 1/4 of tests left. • Sylvester offers to pay those 20-30 USD if it would be a problem to prevent having to rerun the tests with another model. • Mitra says it shouldn't be a big problem. But it could become more problematic if costs rise above 100 USD.
Benchmarking	David	• David presented the benchmarking going through the project report. • David showed the test definitions and the runs in the database. • Mitra asked about the quality score. David pointed to the explanation in the report.

Report	Sylvester	• Sylvester presented the current table of contents of the report and asked for feedback from the advisor. • Mitra mentioned the following: ▸ The product and benchmarking parts need an introduction and conclusion chapter. ▸ The front matter (e.g. Abstract, List of Tables) could be in the Table of Contents. ▸ It would be nice to add a reflection. • We also agreed that the code, the test cases and the results (in the database) would be mentioned with links to GitLab. • Mitra will also give us a sample report. • But in general the outline looks good when the above points are added. • Mitra would read the draft over the weekend. The project team would send her a draft on Friday. Mitra will send an annotated PDF back on Monday.
BA	Mitra	• Mitra asks if we would want to continue with this project as a BA. • David says that he would write his BA one year later. • Sylvester says that it wouldn't be that much fun for a new person to work on this project. But he would like to still like to do something in this area. • Mitra mentions that she knows someone looking for a partner for their BA. She will send the contacts and information.
Hardware	Mitra	• Sylvester mentions that maybe David would be interested in buying the hardware. • Mitra wants to keep it for the institute. • We agreed for physical handover to Mitra in the meeting next Thursday.

Action Items

Task	Responsible	Due Date
Draft	SA Team	2025-12-12
Draft Feedback	Mitra	2025-12-15
Hardware Handover	SA Team	Next Meeting

Upcoming Events

Event	Date
Next Meeting	2025-12-11