University of Sheffield

Speaking to the Edge: Conversational Control of Smart Home IoT with LLM AI and Deterministic Gatekeeping

Mary Jayne Hewitt

Supervisor: Hamish Cunningham

A thesis submitted in partial fulfilment of the requirements
for the degree of *Doctor of Philosophy* in Computer Science
in the
School of Computer Science

May 14, 2026

Declaration

All sentences or passages quoted in this document from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure.

Name: Mary Hewitt

Date: 14th May 2026

Acknowledgements

I would like to thank my supervisor, Hamish Cunningham, for the invaluable guidance, support and advice, and the opportunities provided throughout this work, for which I am very grateful. I would also like to thank Heidi Christensen for constructive feedback and suggestions, and the friends I met along the way for their kindness and support.

Thank you to Billy, for your continual optimism, patience, and cooking, cheering me along the way. To my mum, for always being there for me with boundless love and care, and to my dad, for your encouragement, belief in me, and interest in my work. I am thankful to all my friends and family for the happy times and uplifting words that brightened this journey.

This work is supported by the Centre for Doctoral Training in Speech and Language Technologies (SLT) and their Applications funded by the UK Research and Innovation grant EP/S023062/1.

Abstract

Smart homes inject additional control layers into domestic objects, often using sensor-actuator-microcontroller devices which, when networked, have come to be known as the Internet of Things (IoT). Voice control has increasingly been applied as a user interface here, with systems like Amazon Echo streaming spoken command audio out to cloud-based processing for analysis. Control statements are returned to effect home device state changes, with a spoken output generated and delivered over a smart speaker or similar.

Previous versions of this technology have suffered the brittleness problem, where "expert" or "knowledge-based" systems perform acceptably in controlled conditions but fail rapidly given command diversity. The advent of Large Language Models (LLMs) and transformer architectures in speech and language processing has seen flexible, powerful reasoning capabilities become widely available, and brought about the possibility of robust conversational control of smart homes (and other environments), constituting a new wave of Artificial Intelligence (AI) technology.

The current state of the art, then, is conversational control via cloud-based services that use the new AI. For smart home control, there are two significant problems with this: the negative security and privacy implications of cloud-based data processing from the home; the unreliability of remote services given older domestic systems that set a high standard in this respect. This motivates the question: what happens if the cloud is replaced with locally-computed conversational control?

This thesis addresses that question via six contributions: (1) a taxonomic classification of smart home technologies; (2) a dataset of smart home commands, capturing diverse interaction scenarios; (3) a comparative ASR evaluation under smart home conditions; (4) an analysis of local LLM reasoning for smart home control; (5) a deterministic safety mechanism, validating LLM-generated control outputs; (6) an integrated and evaluated locally-operated smart home assistant, demonstrating its feasibility as a flexible, privacy-preserving alternative to cloud-based assistants.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Smart homes are no longer a futuristic vision but a present-day reality, where voice control, Artificial Intelligence (AI) and Internet of Things (IoT) devices seamlessly combine to offer users convenient control over their environments. The current offerings from leading technology corporations – such as Amazon's Echo, Google's Home, and Apple's HomeKit – rely on cloud servers to harvest and analyse user interaction data. While cloud infrastructures provide vast compute power for natural language processing and inference (Pais et al., 2022; Pallas et al., 2020; Tablan et al., 2013; Chard et al., 2011), the remote transmission and processing of user data is inherently linked to privacy risks, raising concerns about disingenuous end user agreements, data theft, and state abuse of citizens' privacy (Turton, 2022; Al-Mhabis and Cunningham, 2017). These increasingly well-documented concerns contribute to decreased trust in corporations and increased hesitation among users to adopt voice control and smart home technologies. Instances of remote service interruption or cancellation rendering smart home equipment unusable have also become well known (e.g., BBC News (2020)).

Locally computed conversational control is, then, a fundamental requirement for ensuring privacy, security and reliability. Local processing keeps sensitive user data in the home, eliminates network-related latency, and ensures system reliability in the face of connectivity issues. The advantages of local voice control also extend beyond smart homes to assistive care, wearables, robotics and Industry 4.0, with privacy-preserving, cloud-free voice assistants gaining significant research attention (Murshed et al., 2021). To address these issues, the central question the thesis seeks to answer is: *Can a cloud-comparable voice assistant for smart homes be achieved locally?*

Technological advances in recent years have made it feasible to run high-performing Automatic Speech Recognition (ASR) systems and Large Language Models (LLMs) on consumer-grade hardware, giving rise to new possibilities for locally-computed conversational smart home control. This thesis addresses three prerequisites of effective, safe and local conversational IoT control: accurate speech transcription, flexible and robust command interpretation, and verifiable safety measures to handle errors and prevent unintended consequences.

While cloud-based voice assistants deploy vast computational power, several limitations remain: limited context awareness and conversational repair; brittleness when interpreting abstract or imprecise commands; poor adaptability to new domains. In contrast to previous rigid solutions like template matching and slot filling, LLMs can improve flexibility, trained on massive datasets with knowledge and generative capabilities to map freely expressed and

ambiguous commands to device actions, generate structured outputs alongside explanations, and engage in dialogue-based repair when outputs are inaccurate.

Cloud giants have begun to incorporate LLMs in their systems in recent years, addressing the brittleness and rigidity of existing approaches, although this has introduced errors (Gilbert, 2025). Prior systems were generally predictable and reliable, using deterministic approaches used to map predefined user intents to device actions. In contrast, LLMs rely on probabilistic language generation, enabling adaptation to nuances in language and context for more flexible conversational interactions (Todericiu, 2025). However, this flexibility also introduces unpredictability and errors that pose safety challenges for physical control applications. In particular, hallucinations are a fundamental flaw in LLMs, where models may generate incorrect or invented outputs that are not grounded in user intent, provided context or truth (Banerjee et al., 2025). To address these challenges, the thesis proposes rule-based validation methods, referred to as deterministic gatekeeping, to verify LLM-generated device control plans against predefined rules and a structured home model before execution. This intends to ensure the safe operation of LLMs in the presence of hallucination and faulting, without impeding the ability to respond to diverse user requests.

The thesis presents the design, implementation and evaluation of a locally run voice assistant system for smart home environments, including ASR transcription, LLM reasoning, and a deterministic gatekeeper, with performance and scope benchmarked against leading commercial cloud-based systems. To support the work, a novel smart home commands dataset is developed and applied to evaluate and compare ASR models, LLMs and end-to-end system performance on the task of smart home device control. The dataset incorporates varied commands and recording conditions representative of smart homes to provide a challenging benchmark, addressing limitations in existing datasets and evaluations that often lack command diversity and realistic recording variability. Benchmarking evaluations give insights into the performance and robustness of locally run ASR models and LLMs for smart home control tasks. Informed by these findings, the implemented system combines models into an end-to-end smart home voice assistant, that goes beyond existing solutions that often remain cloud-dependent, constrained to rigid approaches for command understanding, or exploratory without implementation. Additionally, the gatekeeper mechanism implements a flexible validation framework for the safe and reliable integration of LLMs with IoT devices.

The overarching contributions of this work lie in the design, implementation and evaluation of a local LLM-based voice assistant for smart home IoT, that ensures privacy and incorporates robust safety measures to mitigate LLM output errors. The smart home commands dataset developed in the work is a central contribution, providing a resource for benchmarking ASR and LLM models for smart home voice control and supporting both model and system level evaluations. By developing a task-specific evaluation dataset and demonstrating high-performance, open-source, private and verifiably safe voice control technologies in a smart home context, the thesis offers a foundation for continued exploration and development in locally operated, LLM-driven voice assistants across diverse IoT domains.

1.1 Research Questions

The thesis explores the technological and practical dimensions of building and evaluating a local, offline and safe voice assistant for smart homes that uses open-source ASR and LLM models. The presentation is structured around four research questions, each of which corresponds to a key challenge that is addressed in subsequent chapters:

1. **How does the performance of locally run ASR models compare to cloud-based solutions for smart home commands?** This question investigates the accuracy, latency and hardware requirements of local ASR models in comparison to cloud-based systems. This involves designing and recording a dataset of spoken smart home commands (Chapter 3), and evaluating ASR performance under varying conditions (e.g., background noise, device types, speaker distances) (Chapter 4).

2. **How effectively can LLMs be adapted for reasoning and flexible command interpretation in smart home IoT environments?** This evaluates LLM responses to diverse command types (e.g., specific, direct, goal-oriented) in terms of generating valid and executable action plans and useful natural language responses. It involves benchmarking open-source LLM performance using the annotated dataset of smart home commands (Chapter 3), comparing performance across homes of varying complexity, and evaluating prompt techniques to minimise errors (Chapter 5).

3. **How to ensure safe and reliable execution of LLM-generated outputs?** This explores the design and integration of a deterministic gatekeeper to validate LLM-generated outputs before actions are executed on IoT devices. It evaluates the effectiveness of the gatekeeper in preventing unsafe or unintended actions caused by LLM errors and hallucinations (Chapter 6).

4. **How does the end-to-end local voice assistant system perform in real-world smart home scenarios?** This assesses the end-to-end performance, reliability and safety of the voice assistant system under real-world use using the dataset of recorded and annotated smart home commands (Chapter 3). It evaluates the interaction between the ASR, LLM reasoning and gatekeeper components in supporting natural and flexible voice control (Chapter 6).

1.2 Aims and Objectives

The key aims of the thesis and the specific steps taken to address the core research questions are outlined:

1. **Taxonomy of Smart Home Voice Control Functionalities**: Investigate the voice control capabilities of commercial smart home systems and classify the types of supported devices. This informs the functional scope for the local voice assistant developed.

2. **Dataset of Smart Home Commands**: Record a dataset of spoken smart home commands, covering a range of command types in realistic smart home conditions. Annotate commands with transcript and expected actions to support both component-level (ASR, LLM) and end-to-end evaluation.

3. **Evaluation of ASR Models under Smart Home Conditions**: Measure the performance of local and cloud-based ASR models across varying conditions (noise types, device types, distances) in terms of accuracy, latency and computational requirements.

4. **Assessment of LLM Reasoning for Command Interpretation**: Explore the capabilities of local LLMs to interpret smart home commands and generate structured control plans across a range of intent types and command phrasings.

5. **Deterministic Gatekeeping Mechanism for LLM Output Safety**: Design a rule-based, deterministic gatekeeper to validate LLM device update outputs. Evaluate its effectiveness in preventing hallucinated or unsafe device updates.

6. **End-to-End System Implementation and Evaluation**: Integrate the ASR, LLM and gatekeeping components into a local voice assistant for IoT device control. Evaluate and assess the end-to-end performance and safety from a real-world use perspective.

1.3 Thesis Structure

This thesis is organised into several chapters that systematically address the research questions and objectives outlined. Each chapter builds upon the previous ones, forming the exploration, implementation and evaluation of a private, offline end-to-end voice interface for smart home systems, with three central challenges addressed: accurate transcription, flexible command interpretation, safety and error handling.

- **Chapter 2**: Defines a taxonomy of smart home voice control functionalities, and establishes foundational concepts for the system design. A review of related work, where gaps are identified helps position the research within the context of existing smart home and voice control technologies. Parts of this chapter have been published in Hewitt and Cunningham (2022, 2024).

- **Chapter 3**: Describes the design and creation of a smart home command dataset for evaluating smart home voice control systems. The dataset is used throughout as core to the evaluation framework.

- **Chapter 4**: Addresses the requirement for local, accurate speech transcription. Evaluates ASR models, comparing local and cloud-based solutions across different scenarios, distances and hardware configurations.

- **Chapter 5**: Explores the application of LLMs for reasoning and command interpretation, assessing their flexibility and performance across command types and intents.

- **Chapter 6**: Focuses on safety and error handling in LLM outputs, introducing a deterministic gatekeeper to block hallucinated or unsafe device updates. Details the end-to-end system integration and evaluates performance in a real-world usage scenario.

- **Chapter 7**: Concludes the thesis, with a summary of findings, contributions, and directions for future work.

Chapter 2

Background and Taxonomy of the Technology Space

2.1 Introduction

Commercial voice assistants such as Amazon's Alexa, Google's Assistant, and Apple's Siri have extensive capabilities for smart home device management and information seeking, yet their heavy reliance on cloud-based processing to enable these functionalities raises significant privacy, security and trust concerns, prompting interest in locally run alternatives (Murshed et al., 2021). Motivated by these concerns, the thesis considers: *Can a cloud-comparable voice assistant for smart homes be achieved locally?*

To lay the foundation for addressing this, the current state of voice controlled smart home systems are examined in this chapter, identifying the core capabilities and technical components required, and discussing methods for implementing and evaluating a local voice assistant. The chapter begins with an overview of smart home and voice control technologies, followed by a comprehensive analysis of commercial products prevalent in today's smart homes. Findings are presented as a taxonomy that classifies and characterises the devices and systems on the market[1] in terms of their hardware and capabilities. While originally developed some years ago to define the project scope, the discussion addresses subsequent market advancements and its generalisation to newer voice controlled smart home products. As such, the taxonomy provides a structured market overview and helps identify key functionalities to replicate in a local voice assistant.

Academic implementations of smart home voice control are then examined, covering both cloud-based and offline systems. Focus is placed on the types of voice interaction technologies and devices in use, and the evaluation methods used to measure performance. The review highlights implementation trends for smart home assistants, while exposing technical challenges and limitations that remain. Following on, the feasibility of building a cloud-free yet cloud-comparable voice assistant is considered, with a comparison of techniques across hardware platforms (MCU, CPU, GPU) and a review of applicable evaluation methods.

From the the survey and analysis of commercial systems, academic efforts and available technologies, four major technical requirements are identified that shape the thesis direction:

[1] Based on devices available for purchase at the time of writing (November 2022).

- **Robust transcription:** Smart home environments are challenging for ASR (variable setups, ambient noise, distant microphones). This motivates the question: *How does the performance of locally run ASR models compared to cloud-based solutions for smart home commands?*
- **Flexible command interpretation:** Natural and conversational interaction requires systems that can handle a range of command phrasings, intents and ambiguities. This leads to the exploration of LLMs and the question: *How effectively can LLMs be adapted for reasoning and flexible command interpretation in smart home IoT environments?*
- **Safety in LLM-generated actions:** LLMs offer flexibility but are error-prone with hallucination and faulting tendencies. This raises the question: *How can deterministic gatekeeping ensure safe and reliable execution of LLM-generated outputs?*
- **A structured evaluation framework:** An appropriate dataset and range of metrics are required for the thorough assessment of real-world performance to answer: *How does the end-to-end local voice assistant system perform in real-world smart home scenarios?*

Bridging insights from commercial and academic fields, the chapter establishes a foundation for addressing the research questions and the technical work that follows. It highlights the current landscape of smart home technologies, surveys efforts towards implementing and evaluating local voice control, and identifies the necessary technologies and capabilities required to replicate and improve upon existing voice assistants.

2.1.1 The IoT and Smart Homes

From the early 20th century the vision of a smart home has been present in popular culture, where life would become easier as domestic tasks were handled by automatic devices, like robot butlers, with little human interaction (Lamkin, 2018). The 21st century saw this vision actualised with consumer products for home automation, including smart plugs and smart thermostats (Serrenho and Bertoldi, 2019). At the technical level of today, smart home architectures integrate networked devices and sensors with voice assistants to replace or augment traditional controls like TV remotes, thermostats and light switches. User benefits arise from the convenient monitoring and control of domestic functions (Gram-Hanssen and Darby, 2018), alongside saving energy, enabling comfier, healthier living environments, ensuring safety and security (Nicholls et al., 2020), and overall improving quality of life (Sutar et al., 2024). Beyond this, smart home services have expanded to support entertainment streaming, online shopping and information seeking dialogues, with a range of solutions aiming to make life more efficient and automated (Ammari et al., 2019; Shafei and Tan, 2024).

Smart device adoption trends show media devices (like smart TVs, streaming devices) to be the most common type of smart devices in use across many world regions, with surveillance devices also very common (Kumar et al., 2019). Entertainment devices, like smart TVs, are most commonly purchased across European countries, followed by communication and control devices, and then home automation and security (Serrenho and Bertoldi, 2019). Domestic appliances are purchased in far lower quantities but this could be due to expense. An analysis of a dataset of popular IFTTT (If This Then That) smart home routines finds lighting devices to be most popular, followed by entertainment and climate control (Yu et al., 2021).

Underpinning smart homes is the IoT, which envisions everyday objects becoming part of the internet and seamlessly fusing digital and physical worlds (Coetzee and Eksteen, 2011). In actuality, the IoT connects and integrates networked microcontrollers, microprocessors, sensors and actuators embedded in non-traditional devices to support device management and data sharing (Cunningham et al., 2022; Dorsemaine et al., 2015). Non-traditional devices are distinguished by the low-power, low-cost computation that is found embedded in special purpose devices (fridges, smart speakers), contrasting general purpose devices (tablets, phones) that typically require much greater computation, memory and power. IoT applications span industry, logistics, robotics, healthcare, cities, and smart homes, with the biggest consumer-facing product being the latter, and many view the IoT as an important technology for improving living environments and quality of life (Wang et al., 2021).

Edge devices are closely related to IoT devices, but typically serve a distinct role in smart home systems. In this work, IoT devices are defined as low-power, storage and compute-constrained network-connected hardware that primarily perform sensing or actuation within the smart home environment. In contrast, edge devices function as local infrastructure, sitting between IoT devices and cloud-based services, and providing gateway and processing capabilities (Bhatia, 2023). Compared to IoT devices, edge devices typically have greater computational and storage capacity, enabling them to perform some local processing and decision-making that can reduce latency and cloud dependence, with a common example being wakeword detection that runs on smart home speakers.

2.1.2 Voice Technology in Smart Homes

The move away from traditional text and touch graphical user interfaces, towards a more user-centred experience where humans can interact with computers through natural expression, has been termed 'heads-up computing' and is intended to allow for more freedom in interaction and seamless support in daily activities (Zhao et al., 2023b). Voice interaction is a step in this direction, and advances in ASR technologies have given rise to voice controlled smart homes and devices (Poongothai et al., 2018) that now populate the market, e.g., Amazon Echo, Apple HomeKit and Google Home (Serrenho and Bertoldi, 2019).

The underlying technology to support two-way spoken communication with smart home systems generally entails keyword (or wake word) spotting (KWS) for system entry, ASR for transcribing user utterances, and natural language understanding (NLU) to interpret actions, however definitions vary. These systems have also been described in terms of three modules: ASR, NLU and Decision Making (Mishakova et al., 2019). Voice Assistants (VAs) build on these functionalities with a dialogue manager, natural language generator and speech synthesis (or Text-to-Speech (TTS)) for two-way spoken interaction (Huang et al., 2015). Additional speech processing techniques that voice assistants have been seen to use include speech enhancement, speech localization, and device arbitration to improve aspects of performance and user experience (Ciccarelli et al., 2022).

Voice assistants have been widely adopted in smart homes, with user motivations including social interaction, information searches and improving life efficiency (Choi and Drumwright, 2021). An analysis of voice command requests shows music and information to be the most common type of query made to smart speakers, with single sentence commands being the most common structure, and stop being the most common individual command (Bentley et al., 2018). A similar study finds requests for audio and media control to be most common,

followed by requests to control smart home devices (Malkin et al., 2019). For older adults in long-term care, voice assistant interactions can alleviate boredom, loneliness and stress, with common usages entailing asking personal questions, seeking advice on sensitive matters, and playing games (Oewel et al., 2023).

While voice assistants are common in the present day, they are not well-liked by all. Infrequent users, in particular, can have frustration with the limitations of voice-based interaction, and would prefer a more conversational system, however, human-likeness can also cause unrealistic expectations about system capabilities (Cowan et al., 2017). Mismatched functional expectations, poor function discoverability, and inaccurate speech recognition are commonly identified usability issues with voice assistants (Cambre and Kulkarni, 2019). Furthermore, the inability to handle complex tasks, create routines, provide explanations about device capabilities and incorporate personalisation also constrain usability (Gallo et al., 2023). A study of Google Home identifies the inability to handle multiple commands in a single turn, and inaccurate command recognition, especially in noisy environments (Pyae and Joelsson, 2018). Collectively, these issues contribute to misunderstandings and user frustration with the interaction. The emergence of LLMs enables more intelligent and human-like conversation, where misunderstandings can be resolved through dialogue, and more complex tasks can be achieved, however challenges remain due to variability and inconsistency in responses. The extent to which voice assistants should be human-like with apparent agency and social intelligence or tool-like with subservient behaviour also varies between people and their preferences, meaning overly conversational interaction styles may not be likeable to all users (Cambre et al., 2020).

A more recent review highlights the benefits offered by LLMs for advancing voice interaction, while noting the variability in their effectiveness in dynamic conversational scenarios (Patil et al., 2024). Directions for improving the performance and user acceptance of voice assistants are also outlined, and relate to the limited robustness of ASR systems to diverse accents, speech patterns and background noise, as well as the privacy and ethical concerns.

2.1.3 System Architectures in Smart Homes

Smart home systems can either be centralised or distributed. A centralised gateway architecture is generally optimal for supporting multiple resource-constrained devices that smart homes entail (Lin and Bergmann, 2016), where the gateway typically serves to coordinate devices, and connect the local infrastructure to the internet (Samuel, 2016).

Market leaders typically provide a central hub for smart homes, where compatible smart devices can be purchased to connect with the hub and establish a smart home network, e.g., in product descriptions you can see ‘works with X’ (Serrenho and Bertoldi, 2019). Smart devices generally entail low-power MCU or CPU-based IoT devices with sensors and actuators attached, with hubs typically CPU-based edge devices with more power and processing capabilities. Smart home hubs typically serve as the voice interface, connecting to cloud-based voice assistants such as Amazon Alexa or Google Assistant to interpret and respond to user commands. Following command interpretation, the hub coordinates and controls connected smart devices. Cloud-based computation is the predominant method for processing speech and device data, enabling the voice interaction between the user and smart home environment.

Wireless technologies allow the flexibility to add and remove components (e.g., smart devices) to the smart home network, enabling scalability and expansion (Viani et al., 2013). Communication types commonly used in smart homes include Wi-Fi, Infrared, Radio Fre-

quency, and Bluetooth (Katuk et al., 2018), plus Global System Mobile (GSM), Z-Wave, ZigBee and 6LoWPAN (Serrenho and Bertoldi, 2019). Wired communication is also an option, using cables to connect devices to networks, e.g., Ethernet (Arriany and Musbah, 2016).

Device interoperability remains challenging for smart home devices given the variety of communication standards and protocols used for inter-device communication, and efforts to standardise are ongoing (Serrenho and Bertoldi, 2019). Matter, for example, is a universal smart home standard developed through industry collaboration to improve interoperability of smart home devices (Madadi-Barough et al., 2025). In terms of installation, the onus is on the user to configure, connect and coordinate components to form the smart home system (Serrenho and Bertoldi, 2019), posing challenges for non-technical users. Further challenges faced by smart home users include privacy, security, and flexibility (Yuneela and Sharma, 2022), with the next section discussing specific smart home privacy and security concerns.

2.1.4 Trust, Privacy and Security Concerns

Trust, privacy and security issues are negatively associated with the adoption of IoT, smart home and voice assistant technologies in both domestic and healthcare settings (Zhong et al., 2022; Philip et al., 2021; McLean and Osei-Frimpong, 2019; Lau et al., 2018; Worthy et al., 2016; Brush et al., 2011). In this section, the privacy and security issues relating to cloud computing, IoT devices and diminished user trust are outlined.

Cloud Computing Privacy and Security

The integration of IoT devices and cloud environments raises complex security and privacy threats (Thushari, 2022). Cloud environments, used to store and analyse data from IoT devices and provide voice assistant services, are inherently not secure. Standardised security and privacy solutions are notably lacking across cloud infrastructures, accessing data through the internet is insecure, and service providers must be trusted to keep data safe and shielded from breaches and illegal access (Almuseelem, 2023). Particularly where security is of concern, companies are moving back to privately owned server infrastructures instead of paying for cloud services (McManus, 2024). It is also noteworthy that cloud providers rely on data centres that are sensitive to extreme heat, with heatwaves causing cooling system failures and outages (Morris, 2024), drawing attention to data and service availability concerns.

Edge computing places computing and storage resources in close proximity to end users on devices or sensors (Satyanarayanan, 2017) and it is deemed vital towards ensuring the security and privacy of a network (Ding et al., 2022). In contrast to cloud-based systems that suffer from power hungry components, high latency, need for internet connection, and privacy and security concerns (Pinto et al., 2020), edge computing brings advantages in terms of energy savings, bandwidth savings, low-latency, privacy protection, trustworthiness and reliability (Varghese et al., 2021). Processing spoken interactions locally on edge devices would contribute to socially acceptable and respectful smart assistants (Seymour, 2018), and recommendations have emphasised the adoption of local data processing methods for voice assistants to protect users' rights and abide by GDPR principles (Hernández Acosta and Reinhardt, 2022). While edge computing has traditionally been limited to domain-specific and relatively small models, recent advances in LLMs have enabled more general-purpose capabilities at the edge (Chen et al., 2025). Though not as high-performing as their cloud-based

counterparts, smaller models are increasingly able to run on CPU-based devices, supporting a broader range of applications while retaining the latency, privacy, and security benefits associated with edge computing. This marks a shift away from task-specific edge intelligence towards more flexible systems with broader capabilities, including complex reasoning, code generation, mathematical problem solving, and robotic control.

IoT Device Security and Vulnerability

Smart home IoT devices capture huge amounts of private and sensitive data about people that could be used against them if stolen, making security concerns prevalent (Serrenho and Bertoldi, 2019; Tukur et al., 2019). Sensor data is able to reveal behavioural and temporal activities in the home (Luqman and Faridi, 2018), giving insight into users' sleeping patterns, exercise routines and medical information (Gao et al., 2018). A study examines smart home devices' data-collecting abilities to assess the privacy risk of the system based on the information the user exposes (Sturgess et al., 2018). Voice data captured by microphones is particularly sensitive given the range of personal information that can be inferred, including a person's gender, personality, age, health condition, socioeconomic status, geographical origin, emotional state or intoxication level (Kröger et al., 2020).

Despite the sensitivity of the information IoT devices collect, they are not secure by nature. IoT devices generally transmit messages in an open medium, allowing for eavesdropping, and are challenging to secure given that conventional security practices are not suitable for such low-power devices (Luqman and Faridi, 2018; Sicari et al., 2015). With IoT device manufacturers often failing to implement robust security mechanisms (Tukur et al., 2019), and towards addressing such issues, the UK has introduced new laws to mandate cybersecurity standards for IoT devices to protect consumers against cyber attacks, partly motivated by a study that found a smart home could face over 12,000 hacking attempts per week (Daws, 2024).

Authentication and authorisation mechanisms are also notably lacking in current systems (Gao et al., 2018; Shafei and Tan, 2024), making it possible for anyone to interact with your voice assistant and perform any task. Commands can be embedded in media to interact with your system, and hidden or inaudible voice commands can be injected without you realising (Cheng and Roedig, 2022). Hidden voice attacks, including ultrasonic and human unintelligible voice commands, have been found to be recognised by the majority of commercial voice controlled systems (Heartfield et al., 2018). In one case, a Google smart speaker was able to recognise the phrase 'OK Google' with 95% accuracy, compared to human transcriber's 22%. An overview of cybersecurity risks that arise from the absence of authentication schemes in smart speakers are described by Sudharsan et al. (2022).

User Trust and Transparency Issues

Lack of trust in the organisations that distribute smart home devices is common given the limited transparency (Worthy et al., 2016) and secrecy that surrounds corporate practices, where companies can capture conversations without users being aware of what, how or why data is recorded (Nicholls et al., 2020). Consumers commonly have a perceived lack of control over the data collected by smart home products, and specific concerns have been expressed regarding audio and video access, household profiling, government access, and data breaches

(Haney et al., 2020; Li et al., 2021; Brush et al., 2011; Marikyan et al., 2019).

Events in the news draw attention to concerns surrounding trust in organisations. Amazon workers review audio clips to help improve the Alexa voice assistant (Day et al., 2019), while Google contractors have been paid to transcribe audio clips collected by Google's assistant (Vincent, 2019). Amazon submitted a patent for algorithms that use trigger words, such as like and love, to build a profile of users' likes and dislikes for targeted advertising purposes (BBC News, 2018). Amazon's Ring doorbell can give data to the police without needing your knowledge or consent (Morrison, 2022), and in one case Amazon handed over audio data from a device that was operating when an alleged murder took place (BBC News, 2017). Fraudulent legal requests have led some technology companies to provide sensitive information about their customers that has been used for harassment and extortion (Turton, 2022). The company Ring was charged for allowing employees and contractors access to users' private video streams, and for not implementing basic privacy and security protections, allowing hackers to control users' accounts, cameras and videos (Federal Trade Commission, 2023). Wyze security cameras also had an issue following an outage, where, when the cameras came back online, users could see thumbnails from other people's feeds in the app and view the feed (The Washington Post, 2024). A security analysis of the Samsung SmartThings framework found it was possible to steal lock pin-codes and cause fake fire alarms through exploiting design flaws and vulnerabilities of over-privileged third-party apps (Fernandes et al., 2016). More recently, Google has settled a lawsuit alleging that private conversations were recorded via users' phones and used for targeted advertising (BBC News, 2026b), with Apple reaching a similar settlement in relation to its Siri voice assistant (BBC News, 2026a).

There is scepticism over the honesty of companies given their business models relying on customer data collection (Lau et al., 2018). The poor sales of the Facebook Portal devices were given as an example of how knowledge of bad practices can impact adoption (Tabassum et al., 2019). Even when companies employ privacy mechanisms, such as mute buttons and on-device operation, this is not sufficient given it is unverifiable and therefore requires trust in the vendor (Seymour et al., 2022). Specifically regarding private conversational data from smart devices and its impact on younger people, Shouli et al. (2025) highlight a lack of transparency in how manufacturers collect and use data, alongside limited user awareness of always-on microphones and their privacy implications. The review reinforces concerns around the handling of voice data, noting it is frequently exploited for advertising, behavioural profiling and third-party resale, and advocating for stronger regulation and more transparent, ethical design practices to build user trust. However, such measures alone may be insufficient, and effective privacy-preserving solutions should aim to minimise reliance on vendor trust altogether, for example through open-source approaches and local data processing.

2.1.5 Open-Source for Trust, Privacy and Security

Open-source initiatives address concerns around trust, privacy and security in many technologies by promoting transparency, innovation, community collaboration, and offering users control over their data and systems.

Transparency, Data Control, and Customisation

Open-source solutions address concerns about transparency by providing access to the source code, allowing users and experts to examine how data is collected, stored and processed, and ensure compliance with specific privacy, legal or ethical data handling requirements. This auditing and examination of code, often by communities, allows for the faster identification and repair of security vulnerabilities that further promotes trust. In contrast, proprietary systems are notably constrained by slower update cycles and bug fixes, and code cannot be audited or modified.

The control over personal data that open-source systems offer is unmatched by proprietary, now often cloud-reliant, systems. Open-source systems allow users to process data locally on their own devices, giving total control, and reducing cloud-related privacy risks like data breaches, unauthorised access, and misuse. With local processing and open code, users can make informed decisions about what data is collected and how it is used.

Customisation is a major advantage of open-source projects, where systems can be tailored to specific use cases, beyond just privacy preferences. For instance, unlike commercial smart homes, open-source solutions can allow for adjusting the wake word detection mechanism, disabling internet-dependent features for complete offline use, or modifying functionality to work with other systems. The freedom to adapt a system in whichever way is desired encourages innovation and supports the diverse requirements of users and communities. While these are not typically plug-and-play systems and therefore inaccessible to most users, they remain of research interest and offer valuable opportunities.

IoT Device Security Through Open-Source and Open Hardware

IoT devices in smart homes are prone to vulnerabilities due to diverse hardware ecosystems and inconsistent security practices. Several open-source projects seek to address these issues, including the Open Connectivity Foundation (OCF)[2] and Home Assistant[3], that provide robust security protocols, including device authentication and encrypted data transfer.

Open hardware designs, like the Raspberry Pi[4], provide transparency about the physical components of a device so that users can both inspect and verify hardware functionality, and also modify and build devices for specific applications (Sas and Neustaedter, 2017). Open hardware, like that of Adafruit[5] and Arduino[6], has brought about a DIY culture of 'maker' communities interested in building personalised devices and understanding how these work, rather than adopting the unfixable black boxes common to consumer culture (Kuznetsov and Paulos, 2010; Hertz, 2011). Such open hardware supports quick prototyping, allowing for devices to be built and programmed for specific use cases, and will be explored further explored and used in the system implementation (Chapter 6) to proxy smart home products.

[2] https://openconnectivity.org/foundation/
[3] https://www.home-assistant.io/docs/configuration/securing/
[4] https://www.raspberrypi.org/
[5] https://www.adafruit.com/
[6] https://www.arduino.cc/

Challenges for Open-Source

Despite the numerous advantages, the adoption and longevity of open-source solutions comes with some notable challenges. Many open-source platforms require significant technical expertise that can deter non-technical users. Open-source projects also suffer from resource constraints (e.g., funding) that can restrict their sustainability, scalability and innovation. Despite the idea that with many people examining the code, bugs and vulnerabilities will be identified, the reality is that security issues can slip through and go unnoticed for years[7]. The Open Source Technology Improvement Fund (OSTIF)[8] has been set-up in part in response to such issues, along with the GitHub Secure Open Source Fund[9].

Open AI systems, like Meta's LLaMa release, have generated significant discourse regarding their implications for security and innovation. By offering some transparency and broadening access to these technologies, open LLMs contribute to the democratisation of AI, albeit with risks. Advocates argue that open models promote transparency and competition, and closing this would stifle innovation based on speculative risks (Brooks, 2024). However, critics worry that open-sourcing AI is dangerous given the opportunity for malicious misuse, like generating mass misinformation or facilitating the production of weapons (Harris, 2024).

Nevertheless, open LLMs continue to be released, and integrating open-source components into voice controlled IoT ecosystems positively contributes towards privacy, trust and security. Following the taxonomy, relevant open-source frameworks, models and software for voice control of smart home devices are examined, with a comparison of their relative capabilities.

2.2 Market Overview and Taxonomy of Smart Home Products

2.2.1 Existing Categorisations and Classifications

The importance of having clear terminology relating to IoT devices has been stressed (Haller, 2010), but there exist a variety of diverse smart home devices that are somewhat difficult to categorise. Previous efforts to classify and categorise aspects of smart home systems and IoT devices are analysed, in order to inform the design of the taxonomy that is intended to cover a wide range of currently available smart home devices.

Suh and Ko (2008) define the role of sensors and actuators in networks, where sensors gather home environment information and actuators control home devices, while both have computation and communication capabilities to wirelessly connect to the home network. Additional components defined include control components that manage the actuators, decision components that select services based on the sensor data, and service components that are the software that provide the user benefits. Likewise, Sun et al. (2013) define smart homes as having sensing agents (e.g., temperature sensor), action agents (e.g., door lock), administration and decision agents (e.g., smart speaker), and database agents (e.g., knowledge bases). Studies that classify devices based on their role in a network are useful towards the taxonomy design, however there can be crossover between device categories.

[7] https://project.linuxfoundation.org/hubfs/LF%20Research/Harvard%20Census%20II%20of%20Free%20and%20Open%20Source%20Software%20-%20Report.pdf

[8] https://ostif.org/

[9] https://github.blog/news-insights/company-news/announcing-github-secure-open-source-fund/

For further insights into how to classify and categorise smart home products, the ISADN specification characterises devices as having Identity, Sensors, Actuators, Network connectivity and Decision making abilities and was proposed to accommodate devices with multiple functionalities (López et al., 2011). The terminology is specific, and helps towards differentiating the nature of hardware devices based on their abilities. Another work classes the services that smart homes offer as comfort, security and healthcare, and categorises the hardware that enable these services as sensors, physiological devices and multimedia devices, with examples given for each category (Alam et al., 2012). A review of both literature and commercial products at the intersection of human-computer interaction (HCI) and IoT categorises devices as either person-centric if they gather data about the human body, or home-centric if they gather data about the environment (Koreshoff et al., 2013). Products are further categorised into functional categories (e.g., fitness tracking, home appliance) and detailed in terms of inputs, sensor types, user interaction and outputs to reveal trends, e.g., person-centric devices most commonly contain accelerometers. The papers somewhat mirror the taxonomy aims, and therefore aspects of the ISADN specification will be considered, as well as the services devices provide, and their inputs, outputs and user interaction mechanisms.

Commercial products are focused on in the following works. A smart home device classification considers the room each resides in, and details the manufacturers and smart features of devices, e.g., LED lights by Philips allow for preferred lighting modes (Katuk et al., 2018). While consideration for manufacturers and features is useful, categorising based on rooms is restrictive given homes have a range of layouts and devices can span multiple rooms. Fourteen categories of smart devices have been presented (Kumar et al., 2019): computer, network node (e.g., home router), mobile device (e.g., iPhone or Android), wearable (e.g., Fitbit, Apple Watch), game console (e.g., XBox), home automation (e.g., Nest Thermostat), storage (e.g., home NAS), surveillance (e.g., IP camera), work appliance (e.g., printer or scanner), home appliance (e.g., smart fridge), generic IoT (e.g., toothbrush), vehicle (e.g., Tesla), media/TV (e.g., Roku), home voice assistant (e.g., Alexa). The vendors and popularity of each device category are discussed, yet there is limited detail on device characteristics.

Smart home system components are classed as user interfaces, smart hardware and software platforms in another study that details hardware in terms of the interface, communication and interaction types, and the major market players (Serrenho and Bertoldi, 2019). Similarly, home automation systems are deemed to have four components in another work (Gunge and Yalagi, 2016): user interface, mode of transmission (wired or wireless), a central controller (a hardware interface) and electronic devices (compatible with the transmission mode and connected to the central controller).

User benefits provided are commonly used to differentiate smart home systems and devices. Smart home benefits have been classed as energy saving, support for elderly and disabled and security and safety (Holroyd et al., 2010). Smart home device applications have been categorised as healthcare, better life, security, energy efficiency (De Silva et al., 2012), as well as convenience and entertainment (Arriany and Musbah, 2016). While it is useful to consider user benefits and applications, detail on the devices that support these is lacking.

A review of AI-controlled smart homes covers key components such as lighting, security, and climate control, along with their benefits, features, connectivity and integration (Sutar et al., 2024). Smart home hubs, such as Google Home, Amazon Alexa and Apple HomePod, are also compared in terms of their hardware specifications (RAM, storage, processor, con-

nectivity), security and third-party support, with consideration for system cost, subscription fees and ease of installation. While useful, the work provides limited insight on specific IoT device specifications and the broader IoT smart home ecosystem.

Smart home voice assistants can be distinguished as being manually activated, speech activated, or always on (Hernández Acosta and Reinhardt, 2022). Voice assistant systems can also be characterised by the types of speech act it can understand and respond to, e.g., speech acts can inquire about information, control devices and request services (Huang et al., 2015). Response styles of popular commercial assistants have been categorised as either minimal, keyword and full sentence, by analysing responses to frequently used commands (Haas et al., 2022). A taxonomy of voice assistants reviews their application and usage, and the techniques used in their implementation (Islas-Cota et al., 2022). More recently, virtual assistants can be categorised as deterministic or probabilistic systems (Todericiu, 2025). Deterministic systems refer to the previous generation of VAs that use rule-based logic to produce predictable outputs but struggle with complex or ambiguous commands. Probabilistic systems address this limitation by using statistical, typically transformer-based, models to infer intent and generate responses, allowing for more flexible and context-aware interaction.

An enabling technology for voice assistants, ASR systems can be characterised in various ways, including speaker dependent vs. independent models, isolated vs. continuous speech, dictation vs. spontaneous speech styles, and vocabulary size (Peinl et al., 2020). Arriany and Musbah (2016) classify ASR systems as speaker-based or word-based. Speaker-based systems can either be speaker dependent or speaker independent, where speaker dependent systems use template matching and are trained on certain voices or words before use. Speaker independent systems perform feature analysis to analyse the input voice. Word-based systems are categorised by how the speaker says the words in the sentence, e.g., discrete words or connected words (continuous speech). Another consideration is language and regional support. ASR models are typically trained on a single language, but following the dawn of transformer architectures, ASR research increasingly aims towards multilingual support (Sharrab et al., 2025). Coverage and accuracy across languages and dialects remains uneven, however, limiting the use of ASR in global contexts, therefore adapting pretrained ASR models to underrepresented languages and dialects, and to individuals' accents are active research areas.

2.2.2 Taxonomy of Smart Home Voice Control Products

The taxonomy is informed by the study of commercial products and voice assistants (Google Assistant, Amazon Alexa, Apple Siri) and previously existing categorisations. Presented in Figure 2.1, three main categories are defined to comprise voice controlled smart home systems from a user interaction and control perspective: Voice Assistants (VA), Voice User Interface (VUI) devices, and IoT smart home devices. VAs handle the interaction, supporting various operations to manage home devices and aspects of personal life. VUI devices, which can be hubs or wearable devices, receive voice input, interface with cloud-based VAs, and coordinate IoT devices, that can be sensors or actuators.

Crossover between the hardware device categories can occur, therefore categories are further distinguished here. IoT devices are classed as sensors or actuators based on their primary function, although some combine both to achieve their functionality (e.g., smart thermostats). In some cases, a VUI could also be considered an IoT device, e.g., a smart speaker is a low-power device with audio output as the primary function. However, VUIs are distinguished

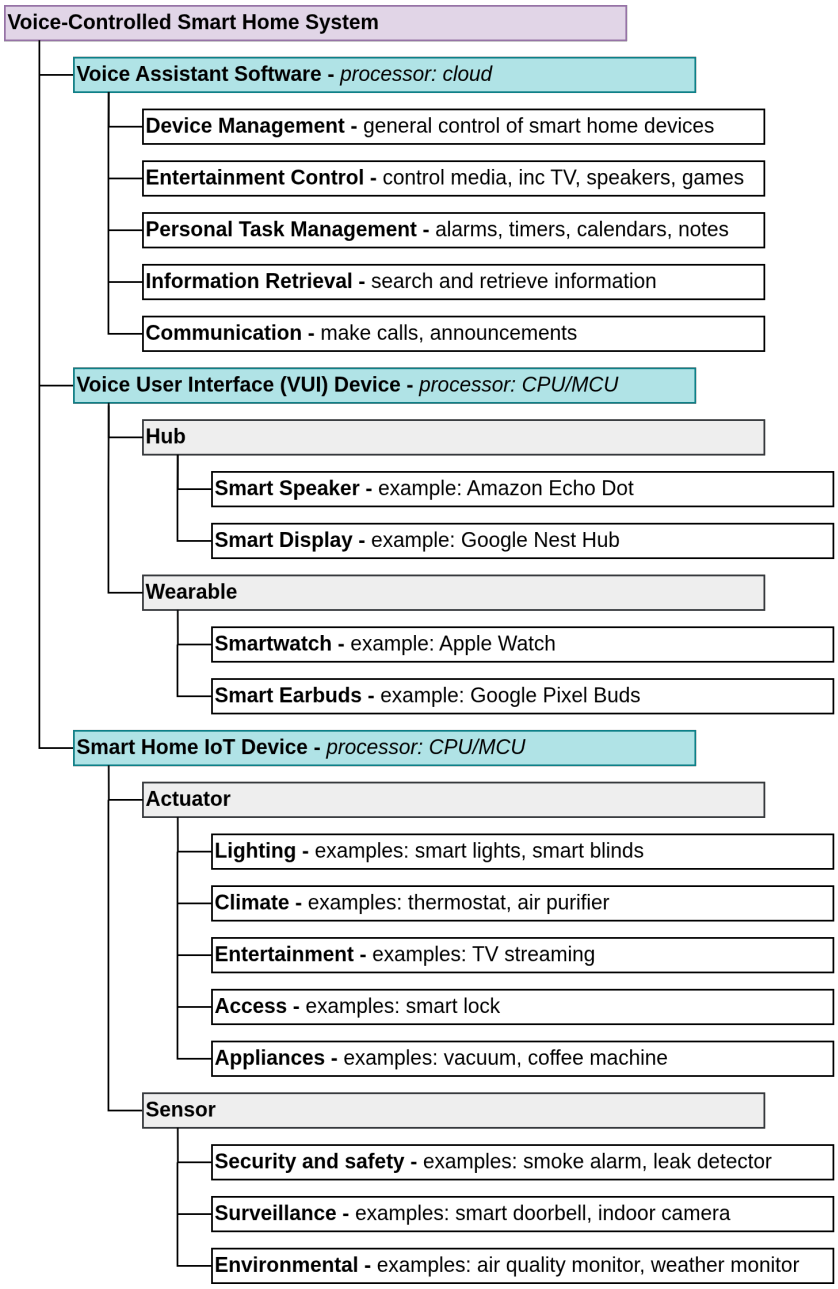

Figure 2.1: *Proposed taxonomy of smart home voice control products.*

by being microphone-equipped and able to interface with voice assistants and other devices, enabling interaction and control capabilities, e.g., to change music, set timers and make calls. The following subsections discuss the taxonomy in more detail and analyse the findings.

Outside of the taxonomy scope lies general purpose computing and networking devices that are not exclusive to smart homes, e.g., routers, mobiles and games consoles. Third-party services that provide additional functionality to commercial systems are also not considered.

Name	Capabilities	Command type	Response behaviour
Google Assistant	Control smart home devices; play and control media; search and retrieve information; manage alarms, timers, lists, calendars, tasks; play games; purchase items; make calls and announcements	Full sentence	Full sentence ('brief' mode available)
Amazon Alexa	Control smart home devices; play and control media; search and retrieve information; manage alarms, timers, lists, calendars, notes; play games; purchase items; make calls and announcements	Full sentence	Full sentence ('brief' mode available)
Apple Siri	Control smart home devices; play and control media; search and retrieve information; make calls, texts, announcements, payments; manage alarms, timers, lists, calendars, notes, reminders	Full sentence	Full sentence

Table 2.1: *Voice Assistant characteristics.*

Voice Assistants

The VAs from Amazon, Google and Apple are explored and compared given their significance, popularity and wide-spread use. Commercial VAs share similar capabilities, all able to complete the set of actions shown in Figure 2.1. The interface for each is a compatible VUI product that awaits a wake word before streaming user commands to cloud servers for processing to return an associated response. The available wake words, like Hey Siri, OK Google, Hey Google, Alexa, consist of three or more syllables. Each VA responds to full sentence command inputs, and provides full sentence spoken outputs, except Google Assistant and Amazon Alexa that offer brief modes for shorter responses. None of the assistants are open-source and rely on cloud-based processing, with third-party apps available to extend functionality. Combined, these shared characteristics indicate the limited transparency, flexibility and diversity among current VAs, and emphasise the need for alternatives that prioritise user privacy with on-device processing. Table 2.1 gives full details of the VA study.

The categorisation of VA capabilities is complete and covers the core functions, but additional smart home features can often be configured through apps or third-party integrations. For example, Apple HomeKit supports the creation of automations for controlling devices by triggers, and the creation of scenes to control multiple devices at once, but only via the app interface. The same applies to adding new devices and managing rooms in the home[10]. While Apple HomeKit is used as an example, the separation between apps handling configuration and voice assistants handling execution is common to all major smart home offerings.

Voice User Interfaces

Selected for inclusion are products that are either manufactured by, or are compatible with the voice assistants of, Google, Amazon and Apple. VUIs divide into two categories: hubs and wearables. Hubs are further divided into smart speaker and smart display devices, with products in each category identified. Wearables are further divided into smartwatches, smart earbuds and smart glasses, with named products in each class. The products in each category are detailed in terms of functionality, connectivity, compatibility, processor type, input and output mechanisms in order to draw insights about each product type. Table 2.2 gives an extract of the VUI devices considered, with full details of the study in Appendix A.

[10] `https://support.apple.com/en-gb/HT204893`

Category	Device type	Device name	Function	Connect.	MCU/ CPU	Inputs	Outputs
Hub	Smart speaker	Amazon Echo Dot	Amazon Alexa; play audio	Wi-Fi, Bluetooth	CPU	Microphone, buttons	Speaker, LEDs
Hub	Smart display	Google Nest Hub	Google Assistant; display media	Wi-Fi, Bluetooth, Thread	CPU	Microphone; touch screen; ambient light, motion, temperature sensors	Speaker, screen display
Wearable	Smart watch	Apple Watch S3	Apple Siri; monitor/track health; display media, information	Wi-Fi, Bluetooth	CPU	Microphone; barometric altimeter; optical heart-rate sensor; accelerometer; gyroscope; ambient light sensor; force touch; GPS	Speaker; screen display
Wearable	Smart buds	Google Pixel Buds	Google Assistant; listen to media; make calls	Bluetooth	MCU	Microphone; capacitive touch; IR proximity sensor; accelerometer	Speaker

Table 2.2: *Voice-user-interface devices in smart homes.*

At a hardware level, most products are CPU-based, with the exception of some wearable devices like smart earbuds. Wi-Fi and/or Bluetooth are common across all devices, with the addition of Thread in some products and NFC in smart watches for contactless payments. Microphones and speakers are embedded in all devices to enable two-way VA interaction. In terms of output types, smart displays and smart watches incorporate touch screens, and smart speakers often have LED lights. Sensors are particularly common in wearable devices to collect various types of information and expand device capabilities, with common examples being accelerometers, optical heart rate and motion detectors.

The characterisation of VUI devices supports the identification of low-cost, programmable alternatives that can replicate their functionality in a privacy-preserving system.

IoT Devices

Smart home IoT devices are broadly classed as either sensors or actuators. Sensors and actuators are further categorised according to the type of service they enable in the smart home, which can also be seen as their perceived role in the smart home from a user perspective. Sensor devices are differentiated by the purpose for which they gather information (security and safety, surveillance, environment), and actuator devices divided into the aspect of the home they control (lighting, climate, entertainment, access, other appliances). For each high level category, examples of product types that fit within each category are detailed. For each product type, commercially available devices that are compatible with the VUIs and VAs described are identified. Each device is detailed in terms of its functionality, connectivity, compatibility, processor type, input mechanisms and output mechanisms/actions controlled. Table 2.3 and Table 2.4 include a selection of the actuator and sensor devices considered in the study. See Appendix B for a detailed list of products for each category under actuator devices. See Appendix C for a detailed list of products for each category of sensor devices.

IoT devices typically use MCUs, except for complex actuators like robot vacuums (e.g., iRobot Roomba), TV streaming devices (e.g., Chromecast), and door cameras (e.g., Ring Doorbell). Connectivity is mostly standardised, with Wi-Fi and/or Bluetooth across all devices, and some supporting Thread or Zigbee. Actuators have diverse input methods including

Category	Device type	Device name	Function	Connect.	Compat.	MCU/ CPU	Inputs	Outputs
Lighting	Smart lights	Philips Hue Smart Bulbs	Control light state and brightness	Bluetooth, Zigbee	Google Assistant; Amazon Alexa; Apple Siri	MCU	App; voice assistant	LED light
Lighting	Smart blinds	WEONESEE Smart Blinds Motor	Control blinds remotely; schedule blinds	Wi-Fi	Amazon Alexa; Google Assistant	MCU	App; voice assistant; remote	Raise and lower blinds
Entertainment	TV streaming	Google Chromecast	Stream media to the TV, control via Google Assistant and mobile devices	Wi-Fi, HDMI	Google Assistant	CPU	Voice assistant, mobile device	Streams media to TV
Climate	Humidifiers & purifiers	Dyson Humidify+Cool Smart Air Purifier	Purifiers, humidifiers and controls airflow, senses air quality and humidity	Wi-Fi	Amazon Alexa; Apple Siri; Google Assistant	MCU	Voice assistant, app, remote control; particulate matter, VOC, temperature, humidity sensors	Fan, purified and humidified air
Climate	Heating	Google Nest Thermostat E	Control heating remotely	Wi-Fi, BLE	Google Assistant	CPU	Voice assistant, app control; temperature, humidity, proximity, occupancy, ambient light sensors	LCD screen; temperature adjustment
Access	Smart lock	Yale Smart Door Lock	Control home access, enable keyless entry, track activity	Bluetooth	Google Assistant, Amazon Alexa, Apple Siri	MCU	Voice assistant, app control	Actuate door lock
Appliances	Smart plug	Philips Hue Smart Plug	Control plug state	Bluetooth	Amazon Alexa; Google Assistant	MCU	Voice assistant, app	Switch plug state
Appliances	Vacuum	Samsung JetBot robot vacuum	Clean floors	Wi-Fi	Google Assistant; Amazon Alexa; Samsung Bixby	CPU	Voice assistant, app control; LiDAR, cliff sensors	Brushes clean floor, suction airflow
Appliances	Coffee maker	Smart Coffee Maker	Control and schedule coffee maker	Wi-Fi	Google Assistant; Amazon Alexa; Apple Siri	MCU	Voice assistant, app, buttons	Makes coffee, LED display

Table 2.3: *Actuator devices in smart homes.*

touch controls, voice commands via assistants, and mobile app interfaces. Output mechanisms include LED indicators, displays, alarms, notifications, physical actuations (e.g., door locks, lights) and spoken responses. Sensor devices are used to monitor specific aspects of the home, such as the environment (temperature sensor) and security (window contact sensor). The reviewed products include sensors for detecting humidity, particulate matter, water, proximity, light, occupancy, contact, smoke, motion, carbon monoxide, carbon dioxide, volatile organic compounds (VOCs), accelerometers, magnetometers and cameras. Sensors are also incorporated into actuators and VUI devices for automation.

The taxonomisation of smart home IoT devices helps understand their functional roles and underlying hardware, helping to identify programmable devices that can replicate these products in a local, open-source system.

Category	Device type	Device name	Function	Connect.	Compat.	MCU/ CPU	Inputs	Outputs
Security & safety	Door and window monitor	Ring Alarm Contact Sensor	Detects opening & closing of windows & doors	Z-Wave	Amazon Alexa	MCU	Contact sensor	Alexa device & phone notifications
Security & safety	Smoke alarm	Google Nest Protect	Detect smoke & carbon monoxide	Wi-Fi, Bluetooth	Google Home	CPU	Ambient light, humidity, temperature, split spectrum smoke sensors; microphone; accelerometer	Speaker; LED; phone notifications
Security & safety	Leak detector	Eve Water Guard	Detect leaks	Bluetooth, Thread	Apple HomeKit	MCU	Water leak sensor	Speaker; LED; phone notifications
Surveillance	Smart doorbell	Ring Video Doorbell	Detect motion; monitor home; two-way talk	Wi-Fi	Amazon Alexa	CPU	Camera; microphone; motion sensor; button	Speaker; LED; home device and phone notifications
Surveillance	Indoor camera	Google Nest Cam	Monitor home; detect people and animals	Wi-Fi, BLE	Google Home	CPU	Camera; microphone; motion sensor	Speaker; phone notifications
Environment	Air quality monitor	Amazon Air Quality Monitor	Monitor and track air quality	Wi-Fi, BLE	Amazon Alexa	MCU	Temperature, CO, humidity, VOCs, particulate-matter sensors	Home device and app notifications and monitoring
Environment	Weather monitor	Eve Weather	Provides real-time weather data and forecast	Bluetooth, Thread	Apple HomeKit	MCU	Temperature, humidity, barometric-pressure sensors	E-ink display, home device and app monitoring

Table 2.4: *Sensor devices in smart homes.*

Taxonomy Extensions and Generalisation

The taxonomy presented in the work is informed by smart home ecosystems up to 2022, covering the core functional components of commercial voice controlled smart home systems, but a consideration in the design of the taxonomy is the ability to expand to incorporate future developments in the smart home domain beyond the time from which it was proposed.

In the years following the taxonomy creation, the market has seen incremental improvements, but limited fundamental change in smart home products and ecosystems. New devices are typically refinements or recombinations of existing device categories rather than entirely new functional classes, e.g., the inclusion of additional sensors and capabilities into existing devices. IoT products not included in the taxonomy, but that have become more common on the market, include pet-related devices such as automated feeders and smart cat flaps, and voice-enabled smart glasses. The taxonomic categories can accommodate such products under actuator and VUI wearable device categories, indicating flexibility to incorporate new product categories and functions.

Devices are also increasingly incorporating intelligence features. Person identification is now common in smart doorbells, and voice interfaces are common on smart TVs and remotes. While such features do not alter the device categorisation, the taxonomy could extend to characterise intelligence features of products as these become prevalent, without altering the underlying structure. However, boundaries between VUI hubs and VUI-enabled devices, and sensing and actuation devices, may become increasingly blurred as device complexity

increases. For example, voice controlled TV devices begin to blur this boundary, but remain better characterised as hybrid IoT-VUI devices as they typically lack integration with the broader smart home. For now, the categories and distribution of roles generally persists, but future extensions could include hybrid categories and define cross-device interactions further.

Therefore, while smart home voice assistants and ecosystems are expected to become more expansive, complex and intelligent, it is not expected that systems and devices will fundamentally divert from the taxonomy categories. Instead evolution is more likely to occur in advancements to cloud-based voice assistant capabilities and variations on existing forms of IoT devices to incorporate additional functionalities and intelligence. This view of incremental over fundamental change is consistent with research on the future of smart homes, where findings from expert opinions suggest a shift towards more pro-active and context-aware interactive systems, with user preference learning, and also the increased prevalence of health-related devices and functionalities (FakhrHosseini et al., 2025). The taxonomy is considered generalisable as it is grounded in functional device roles and categories rather than specific products and implementations. Most emerging devices and features can be mapped onto existing categories, while the taxonomy can also be extended to incorporate new categories (e.g., health monitors, pet-related actuators) and characteristics (e.g., intelligence) to track smart home trends.

2.3 Academic Solutions for Smart Home Voice Control

2.3.1 Online Implementations

To gain insights into academic efforts, this section reviews cloud-reliant smart home implementations, examining the device types, control mechanisms, and evaluation methods used.

A Raspberry Pi based home automation system uses Google's Python Speech-to-Text API for voice control, integrating NLP to identify keywords and actions from transcribed speech (Baby et al., 2017). The system controls devices like lights, fans and air conditioning via relays connected to the Raspberry Pi, with additional sensors integrated to trigger lights by motion and fans by temperature. System evaluation checks system functionality, including whether voice commands trigger the corresponding action execution. The implementation is relatively simple, with scope to add further intelligence for more complex interactions.

Similarly, Putthapipat et al. (2018) use Google Cloud API for voice recognition, Wit.ai for intent identification, and Google Translate for text-to-speech conversion in a home automation system. A Raspberry Pi serves as the central control unit, with a speaker and microphone attached, allowing for flexibility. However, the system requires stable internet connection, has high energy consumption, and speech recognition varies with microphone quality.

Rani et al. (2017) use a mobile device as the central controller for voice controlled appliances in an IoT-based home automation system, with cloud-based NLP for command interpretation. The setup includes fans, lights, coffee machines and door alarms connected to Wi-Fi via MCUs. While the proof of concept is outlined, details on the NLP software used and system evaluation are not provided. Likewise, an Android app allows remote control of appliances via buttons or simple voice commands, like 'turn on' and 'turn off' followed by the appliance name in another implementation (Kodali and Mahesh, 2017). Here, an ESP8266 MCU acts as the central controller with connected appliances including lights and

fans. Experimentation results show the system works as expected with both voice and button inputs.

Poongothai et al. (2018) implements a remote voice controlled IoT lab using Google Assistant, with an Android smartphone app serving as the interface for voice and text-based commands. The system allows monitoring and control of lights, fans, a projector, and an air-conditioner, all connected to Wi-Fi via NodeMCU. Testing involved verifying sensor readings, ensuring voice commands function correctly, and measuring device energy consumption.

A mobile device uses Google's DialogFlow API for voice control of appliances in another study, with a Raspberry Pi as the central server, and ESP8266 and Arduino MCUs enabling remote control of devices (Kumar et al., 2021). IR sensors are used for object detection to control lights, and PIR sensors are used to turn fans on or off. While the experimental setup is described, evaluation methods are left out. Vishwakarma et al. (2019) detail a similar home automation system, where Google Assistant is used on a mobile device to allow the remote voice control of lights connected to a Wi-Fi enabled ESP8266 NodeMCU.

Oyucu (2021) use cloud-based speech recognition to enable a home automation system to recognise predefined commands, without performing semantic analysis. The command is interpreted as an action and a device to perform the action on. Experimentation shows a 2 second speech command takes 0.3 seconds to be transcribed and parsed.

A prototype Alexa smart speaker with authentication, composed of a Raspberry Pi, microphone, speaker and camera, is proposed by Sudharsan et al. (2022). OpenCV is used for face recognition to authenticate users, the Alexa Voice Service SDK enables voice interaction, and Snowboy wake word detection is used to prevent accidental activation of the smart speaker. Wake word detection accuracy is measured based on false alarms per hour and miss detection rates, but no additional evaluation methods are provided.

Dasgupta et al. (2022) presents a home automation system using an ESP32 Wi-Fi module as the gateway, that connects to temperature, flame and humidity sensors and a lightbulb via a relay. Data is transmitted to cloud platforms, with an Android app interface used to notify the user of fires, provide voice control functionality and display sensor readings. There is no detail on the voice control implementation, and methods of testing are limited to checking the basic system functionality, e.g., voice commands trigger correct action, fire triggers fire alert.

Cloud-based LLMs like ChatGPT have more recently been applied to smart home control in experimental implementations (King et al., 2023a; Rivkin et al., 2023), albeit without voice control capabilities and with scope for accuracy improvements and further evaluation.

With the central aim to achieve a similar smart home implementation, but without reliance on the cloud, methods for local speech and language processing on-device are detailed next.

2.3.2 Offline and Local Implementations

Literature on offline implementations of voice controlled smart home systems are outlined in this section. The term offline is used in the sense of not relying on third-party, typically cloud-based services, rather than meaning the system is disconnected from any network. Snips was one of the first implementations focusing on local voice control for privacy, but was acquired by Sonos[11]. Sonos offers on-device voice control for media devices, however the ecosystem and command set is narrow, and the system is no longer open-source (Coucke et al., 2022).

[11] `https://snips.ai/`

The Snips Voice Platform used a custom version of a Kaldi training recipe for model training, and the LibriSpeech dataset for evaluation, measuring WER, speed and memory usage (Coucke et al., 2018). Given voice assistant use in often noisy, far-field conditions and in rooms with reverberation, and that in-domain data is valuable for training, the authors also used data augmentation to simulate noisy and reverberant conditions.

Kaldi is widely used, with one study finding it to be the most accurate ASR solution amongst Mozilla DeepSpeech, PaddleSpeech and Facebook's wav2letter (Peinl et al., 2020). Models were compared on Raspberry Pi 3 and Nvidia Jetson devices, and tested using a custom dataset and LibriSpeech (Panayotov et al., 2015), with runtime performance and accuracy (WER) measured. Pinto et al. (2020) also implements a Kaldi ASR system with a vocabulary size of 200K words on a mobile device containing a CPU and a low-power GPU.

Arriany and Musbah (2016) use Windows 7 speech recognition software to control a smart home via a mobile device. The setup consists of a laptop as the user interface, a PC as the home server, and an Arduino Uno to process and route commands to devices. Performance evaluation measures word recognition accuracy in different noise environments, and with various microphone qualities, as well as the execution time and general system functionality.

Several local voice control systems only recognise a predefined command set. Isolated word recognition smart home voice control is implemented using voice activity detection (VAD), feature extraction, and pattern matching against a template library, where the template with the highest similarity to the captured features is selected as the recognition result (Bai, 2022). Voice commands like "help" and "turn on the living room light" were played to the system for testing, and repeated 20 times each. The average accuracy reported is 83.75%, with microphone quality and pronunciation clarity affecting performance. Similarly, command classification is implemented on a Raspberry Pi 3 using the Tensorflow Keras Library, with an ARM MCU to perform feature extraction, and achieves 87.8% accuracy and 1.136 second latency on an 8 command recognition task (Zonios and Tenentes, 2021). The evaluation dataset consisted of spectrograms for 905 voice samples, each 4 seconds long and recorded by a male and female on both headset and mobile phone microphones.

MCU-based voice recognition modules are popular to provide offline voice control for a limited set of appliances. A voice controlled system to manage devices in home and office environments uses the EasyVR 2.0 voice recognition module trained to recognise a specific set of commands (Ali et al., 2017). Experiments involved repeating commands 30 times, with the success rate measured across different age groups, genders, and in the presence of noise or physical obstacles. Both noise and obstacles were found to negatively affect command recognition performance. Elsokah et al. (2020) also uses the EasyVR Shield 2.0 for voice command recognition, enabling control of lighting, radio, TV, music player, and air conditioning connected to Arduino MCUs and a relay module. Experiments tested command recognition in varying noise and weather conditions with participants of different genders and ages, achieving an average success rate of 96%, with system cost and response speed also considered.

Munir et al. (2019) develop a speaker-dependent offline smart home using an Arduino v3 module for speech recognition, with a Raspberry Pi for face recognition using the OpenCV library to control access. Accepted commands were limited to switching appliances on and off, achieving 90% accuracy for speech recognition and 96% for face recognition with low latency, though specific evaluation methods are not provided. The v3 module has been used in further studies for voice control of appliances like fans, air-conditioning and lighting (connected to a

central Arduino MCU) (Shehab, 2019; Shehab et al., 2020). While these studies demonstrate command recognition on tightly constrained devices, the Arduino v3 module is limited in that it can only store 80 commands and must be trained for a single person.

Julius and PocketSphinx are two lightweight and open-source ASR libraries with no external dependencies that can run on CPU devices. A comparison of these systems found Julius to perform best in terms of word recognition accuracy, when each model was run on a Raspberry Pi 3 (Vojtas et al., 2018). PocketSphinx has been used as the ASR component in the design and development of a prototype cloud-less voice assistant system, where the best result shows 93% intent recognition accuracy (Polyakov et al., 2018). Festival is used for text output and the intent recognition is implemented as a decision tree algorithm. However, the command set is relatively small and the NLU component is not robust or flexible for diverse user inputs given it relies on a predefined grammar and a restricted set of keywords. Another study implements PocketSphinx on a mobile device to perform speech recognition, where recognised commands are transmitted to a Raspberry Pi smart home controller (Bhagath et al., 2021). When tested with live speech, the system achieved 80% accuracy for switching lights on/off with a delay time of 1 second for command recognition and execution.

In a comparison of voice assistants for embedded smart home systems, Erić et al. (2017) highlight the open-source platform Jasper (Li et al., 2019), that is modular in design, written in Python, and supports an offline voice enabled gateway architecture (when using offline ASR and TTS). Vosk and Sopare have also been evaluated for offline ASR control of IoT devices, achieving accuracy of 84% and 95% respectively (Setiawan and Yusuf, 2022). Additionally, Vosk has been demonstrated to perform better than DeepSpeech (Fuad et al., 2024).

Rhasspy[12] is an open-source, offline voice interface for smart homes that was found to balance accuracy, trustworthiness, security and on-device intelligence in a comparison of voice assistant solutions, where Alexa and Mycroft were also considered (Jesse et al., 2021). Commands are specified in a template language and Rhasspy produces JSON events that can trigger action in home automation software such as Node-RED and Home Assistant. Rhasspy can run on Raspberry Pis, and when applied to voice controlled lighting, fast response times were seen, although ASR accuracy was suboptimal and could be improved with a better language model and a noise-filtering microphone (Dallmer-Zerbe and Haase, 2021).

In terms of varied evaluation techniques, Venkatraman et al. (2021) implements a secure, personalised voice recognition system for smart home device control and evaluates it by asking the system incorrect questions that sound similar to those desired. In replicating commercial surveillance systems, Beugin et al. (2022) evaluates a privacy-preserving Raspberry Pi-based camera system against privacy requirements, while maintaining commercial system features, with consideration for ease of use, resolution, frame rate, and latency. The goals of this work mirror our own, in balancing the advantages of cloud-based solutions with data privacy.

More recent work has explored integrating LLMs into voice assistants, moving beyond rigid command-based systems. Motivated by privacy, reliability, and accessibility in rural settings, Umesh et al. (2025) combines offline Whisper ASR, lightweight (<3B) LLMs, and TTS to enable conversational interactions on resource-constrained devices (2GB RAM). While not integrated with IoT devices, the approach achieves 75–90% ASR accuracy and 2.5 second latency on queries relating to agriculture, health and general knowledge, highlighting both strong LLM reasoning performance and the remaining limitations for offline voice assistants.

[12] `https://rhasspy.readthedocs.io/en/latest/`

2.4 Feasibility of a Cloud-Free, Cloud-Comparable Voice Assistant

2.4.1 Hardware and Programmable Devices

The taxonomy of smart home systems helps identify a representative set of devices that can proxy commercial products. For each product type, it is desirable to identify an open hardware IoT solution for use in a prototype system. A review of commercial products highlights the potential for using DIY technologies and kits, like Arduino, to quickly and cheaply construct prototypes that can support the envisionment of IoT technologies (Koreshoff et al., 2013).

Arduino and Raspberry Pi devices frequently appear in academic smart home implementations, often as the central controller. An analysis of such implementations is presented by Gunge and Yalagi (2016), highlighting their capabilities in smart home setups. Sensors for smart home applications that work with Arduino and Raspberry Pi devices are compared in Gazis and Katsiri (2021), while Omidvarborna et al. (2021) compare air quality monitoring systems, and Demanega et al. (2021) provide technical details on low-cost air quality sensors.

ESP32 devices are well-suited for prototyping smart home systems given their versatility, affordability and low-power consumption. The ESP32-S3 chip is capable of connecting to and controlling several home devices from lighting to TVs, while being cost-effective, low-power and having high-performance wireless communication (Litayem, 2024). Modern smart home projects are increasingly using ESP32-based platforms, with one study showing the energy-efficiency and low-latency response times of this device compared with other solutions, and noting the affordability and flexibility provided over commercial systems that restrict users to specific devices and ecosystems (Hardi et al., 2025). This demonstrates the ESP32 platform as comparable to commercial devices, with advantages in flexibility and privacy.

With support for a wide array of peripherals and sensors, including microphones, speakers, cameras, the ESP32 chip can be applied to replicate the functionality of most commercial smart home products. However, more complex actuators, like robot vacuums and coffee makers, require more extensive hardware integration and programming that makes these much harder to replicate. The ESP32 ecosystem also includes wearable interfaces, like smart watches, and touch-screen devices to support various user interaction mechanisms. The smart home implementation will use ESP32-S3 devices to simulate commercial smart home products, demonstrate system transferability to real-world device control, and allow for end-to-end system performance evaluation.

2.4.2 Open-Source Tools and Technologies

Towards the design of a privacy-preserving smart home voice assistant, this section examines the four technical components required to enable the interaction, and replicate commercial functionalities, in a fully local setup: KWS, ASR, NLU and TTS.

The investigation explores open-source models and libraries that can operate on various devices, including those comparable with smart home hardware (MCUs, CPUs), and GPUs. Table 2.5 summarises the technologies and tools identified for each key component – KWS, ASR, NLU, TTS – mapped to their compatible hardware platforms.

[13] `https://www.tensorflow.org/lite/microcontrollers`

[14] `https://github.com/espressif/esp-sr/tree/master/model/wakenet_model`

Component	Processor	Technologies
KWS	MCU	TensorFlow Lite[13] ; WakeNet[14]
	CPU	Porcupine[15] ; Snowboy[16]; PocketSphinx[17] ; MyCroft Precise[18]
ASR	MCU	ESP MultiNet[19] ; Arduino v3 speech recognition[20]
	CPU	Kaldi[21] ; PocketSphinx[17] ; DeepSpeech[22] ; Julius[23] ; Whisper-small[24] ; PaddleSpeech[25] ; wav2letter[26]
	GPU	Whisper-large[24] ; ESPNet[27]
NLU	MCU	Simple rule-based pattern matching.
	CPU	FuzzyWuzzy[28] ; Flair[29] ; SnipsNLU[30] ; RASA NLU[31] ; Home Assistant conversation[32]
	GPU	LLaMa models [33] ; OLMo models[34] ; Qwen models[35] ; Gemma models[36] ; Dolly[37] ; Mistral[38]
TTS	CPU	PaddleSpeech[25] ; Coqui[39] ; Flite[40]
	GPU	ESPNet[27] ; eSpeak[41] ; ; Tortoise[42] ; SpeechT5[43]

Table 2.5: *Comparison of voice-assistant technologies across hardware types.*

The analysis not only demonstrates the feasibility of implementing local voice control functionality across diverse hardware configurations, but also highlights specific approaches that could support cloud-comparable performance in a local and private setup. CPUs provide a balance between flexibility and cost, while MCUs are suitable for basic applications with a small vocabulary and rigid grammar. GPUs are deemed pivotal towards achieving a cloud-

[15] https://github.com/Picovoice/Porcupine
[16] https://github.com/seasalt-ai/snowboy
[17] https://github.com/cmusphinx/pocketsphinx
[18] https://github.com/MycroftAI/mycroft-precise
[19] https://github.com/espressif/esp-sr/tree/master/model/multinet_model
[20] https://github.com/elechouse/VoiceRecognitionV3
[21] https://github.com/kaldi-asr/kaldi
[22] https://deepspeech.readthedocs.io/en/r0.9/
[23] https://github.com/julius-speech/julius
[24] https://github.com/openai/whisper
[25] https://github.com/PaddlePaddle/PaddleSpeech
[26] https://github.com/flashlight/wav2letter
[27] https://espnet.github.io/espnet/
[28] https://github.com/seatgeek/fuzzywuzzy?tab=readme-ov-file
[29] https://github.com/flairNLP/flair
[30] https://github.com/snipsco/snips-nlu
[31] https://github.com/RasaHQ/rasa
[32] https://www.home-assistant.io/integrations/conversation/
[33] https://llama.meta.com/llama3/
[34] https://allenai.org/olmo
[35] https://github.com/QwenLM/Qwen
[36] https://deepmind.google/models/gemma/
[37] https://github.com/databrickslabs/dolly
[38] https://docs.mistral.ai/getting-started/models/models_overview/
[39] https://github.com/coqui-ai/TTS
[40] https://github.com/festvox/flite
[41] https://github.com/espeak-ng/espeak-ng
[42] https://github.com/neonbjb/tortoise-tts
[43] https://github.com/microsoft/SpeechT5

comparable VA locally, with the ability to run advanced Whisper models for accurate ASR and pass outputs to LLMs to make use of their extensive reasoning capabilities for action planning and response generation. The adaptable nature and contextual reasoning abilities of LLMs can support a robust and flexible home assistant, able to handle conflict resolution, creative commands, ambiguity, and user preferences, giving rise to a number of opportunities.

Rhasspy[44] and Willow[45] are two notable open-source and modular frameworks that facilitate the integration of the components required to build a voice assistant system. Willow is designed for GPUs, enabling integration between ASR models and LLMs, while Rhasspy focuses on CPU operation. Each framework supports integration of various tools given in Table 2.5. Where resources are constrained, or the application is small and limited, Rhasspy is most suitable. However, for a more versatile and high performance setup, Willow is preferred given its support for recent and advanced models.

In contrast to the previously rigid smart home voice assistants like Siri, Alexa and Google Assistant, LLM powered chatbots provide flexible conversational abilities, responding to any request rather than refusing those out of domain[46]. This signifies the outdated-nature of intent and slot filling NLU systems, and promotes the move towards generative architectures for voice assistants (further discussed in Chapter 5.1). Simply integrating LLMs with HomeAssistant can produce a personalised and likeable voice interactive home assistant[47] that can eloquently answer any question, while Siri and Alexa are thrown by simple requests that are out of scope (Coward, 2024). LLMs can also use conversational context to resolve ambiguities, such where selecting the 'third' film on a TV can be confused for the 'third' channel (Santos et al., 2020). However, LLMs are imperfect and can often select the wrong device for operation, or mistake the intent that could negatively impact user experience, and in the worst case, safety, where turning on the heating could be mistaken for the oven (Li et al., 2025).

This section establishes a technological background and foundation for the selection and integration of open-source solutions into a local, voice controlled smart home prototype that aligns with the thesis goals of privacy, security and flexibility. Using Willow as a basis for integrating locally run ASR and LLM models with ESP front-end devices is a promising avenue to achieve a local, flexible and intelligent voice assistant. The next section examines LLM hallucination and associated safety concerns, and an evaluation framework is then considered.

2.4.3 Hallucination in LLMs

Hallucination in LLMs can be defined as the generation of seemingly plausible but non-factual content (Huang et al., 2025), where outputs may be inconsistent with the user input or with the training data (Bang et al., 2025). Huang et al. (2025) distinguish between factuality hallucination, where generated content does not align with real-world facts, and faithfulness hallucination, where outputs do not align with user input or lack self-consistency (e.g., deviation from instruction, context, or inconsistent logic). A further survey defines hallucination types as input-conflicting, context-conflicting and fact-conflicting, and identifies causes to include model's knowledge boundaries and overestimation of their capabilities, as well as noisy

[44] https://rhasspy.readthedocs.io/en/latest/

[45] https://heywillow.io/

[46] https://www.macstories.net/linked/the-new-york-times-declares-that-voice-assistants-have-lost-the-ai-race/

[47] https://johnthenerd.com/blog/local-llm-assistant/

training data (Zhang et al., 2023). Issues in the training data, such as the presence of misinformation and fake news (e.g., from social media), can lead to factual hallucinations, as well as societal biases (e.g., relating to gender), and hallucinations also commonly occur when inputs fall outside the model's training data and knowledge base.

Various benchmarks measure the hallucination tendencies of LLMs. For example, Bang et al. (2025) release a benchmark to measure extrinsic hallucination, where generation is not consistent with training data. FaithEval is a benchmark measuring intrinsic hallucination, where outputs are input-conflicting or lack self-consistency (Ming et al., 2024). Results show all models struggle to remain faithful to given context, and that larger models do not necessarily perform better. TruthFulQA examines LLM factuality, finding LLMs mimic popular misconceptions and larger models are less truthful, with fine-tuning techniques suggested to mitigate this (Lin et al., 2022). HALoGEN is another benchmark that classifies output errors based on whether they stem from incorrect recollection of training data, incorrect knowledge in the training data, or fabrication of knowledge (Ravichander et al., 2025). Relevant to the structured plan generation required for smart home device control, hallucination errors relating to code generation are considered in another study (Zhang et al., 2025).

Mitigation techniques for hallucination include applying RAG to provide external knowledge databases, and using fine-tuning approaches to improve response accuracy on domain-specific queries in question-answering systems (Li et al., 2024). Model editing has also been explored, where additional up-to-date knowledge is added through editing model parameters (Wang et al., 2024b). Using model certainty outputs as an estimate of hallucination presence, and chain-of-verification prompting techniques can help guide LLMs to correct their own mistakes to reduce hallucination and inconsistencies (Dhuliawala et al., 2023). Such techniques are also detailed in a survey that differentiates between prompt engineering (e.g., RAG and chain of verification), and model development techniques (e.g., using knowledge graphs or supervised fine-tuning) for hallucination mitigation (Tonmoy et al., 2024).

LLM hallucination is generally seen as an unsolvable problem though, given the transformer-based predictive model architectures in use (Huang et al., 2023). Banerjee et al. (2025) consider hallucinations an inherent limitation of LLMs, arising from the impossibility of a fully complete training dataset and the non-zero probability of errors in generation. Improvements in model architecture and training data may reduce hallucinations, but cannot remove them entirely. Addressing the non-determinism of LLMs and how this limits their use in operational environments, a framework has been proposed that decouples model outputs from system flow logic, where the LLM handles subtasks but does not control outcomes (Qiu et al., 2025). Similarly, LLM outputs for IoT control are validated through a framework that checks the validity, completeness and executability of the output in another work, including whether an operation is supported or whether a device exists (Torkamani and Zarin, 2025).

Hallucination negatively impacts user trust and presents challenges for the safe application of LLMs, particularly when given control of devices in smart homes. LLM hallucination tendencies are seen in Chapter 5, while Chapter 6 implements a safety framework combining LLM outputs with a deterministic, rule-based gatekeeper to mitigate hallucination and ensure reliable operation.

2.4.4 Towards an Evaluation Framework

The evaluation framework is intended to assess each component of the voice assistant system, as well as the end-to-end performance in a real-world setup. The aspects to quantify and assess include: accuracy of ASR transcription in smart homes, accuracy of LLM outputs for device control, safety of LLM outputs for device control, and end-to-end system performance.

ASR performance evaluation

Speech recognition systems are generally assessed according to accuracy and speed, with ASR accuracy typically measured in terms of word error rate (WER). Consideration for vocabulary size, latency and computational efficiency (CPU and memory footprint) are also common. Saade et al. (2018) use end-to-end spoken language understanding (SLU) metrics (e.g., F1 score) to assess the performance of SLU models on two domains (lights and speaker control), arguing these are a better proxy for user experience than word error rates (WER). Likewise, Mehrabani et al. (2015) argue that semantic tags are the most important system output given they determine whether a smart home task is recognised correctly.

Speech datasets are required to benchmark and compare ASR models, as in studies where the Librispeech (Panayotov et al., 2015) dataset is used to compare ASR models running on Raspberry Pi and Nvidia Jetson devices (Peinl et al., 2020; Gondi and Pratap, 2021). The maximum effective distance (between the microphone and the person speaking) for speech recognition varies significantly between platforms, which is an important consideration for smart home systems where users are at varying distances from the voice input device (Heartfield et al., 2018). A dataset of one speaker recorded at varying distances away from the microphone has been used to evaluate the WER of open-source voice recognition systems, Julius and PocketSphinx, running on a Raspberry Pi 3, where Julius performed better in terms of WER (Vojtas et al., 2018). Noise is prevalent in smart homes and can affect the accuracy of ASR (Ali et al., 2017) so it is important to test models under varying conditions as seen in several academic implementations (Elsokah et al., 2020; Arriany and Musbah, 2016).

Aspects of ASR robustness are specifically evaluated in several papers. Off-the-shelf ASR systems, including Google Cloud STT, Amazon Transcript and Kaldi, have been shown to perform worse on non-American accents, particularly for non-native English speakers (Tadimeti et al., 2022). Similarly, ASR-FAIRBENCH benchmarks ASR model variation across demographic groups with the aim to promote inclusivity, given that variations in accent, gender, age and linguistic background leads to unequal ASR performance (Rai et al., 2025). Speech Robust Bench, a benchmark proposed to standardise ASR robustness evaluations, includes challenging real-world ASR scenarios, such as noisy, multi-speaker environments, far-field recognition, and diverse speaker accents, and finds Whisper-large to be most robust (Shah et al., 2024). Likewise, Kuhn et al. (2024) find Whisper outperforms commercial services in many cases in a review of ASR performance across vendors, speakers and languages. Streaming APIs were also found to have higher error rates compared to batch transcription, where accuracy may be traded for lower latency. The paper also highlights that accuracy alone is not always a reliable indicator of real-world usefulness or alignment with human judgement of transcription quality, therefore qualitative evaluation alongside metrics is required.

While it is sought to evaluate the WER of ASR models under realistic smart home conditions (i.e., varying microphones, background noise, speaker positioning), there are no large-

vocabulary English datasets specifically designed to assess ASR performance and robustness in the context of smart home command recognition. The gap presents an obstacle to benchmarking systems in this domain. This is addressed in the following chapter (Chapter 3), by developing a dataset to support the evaluation of models on smart home voice control.

NLU performance evaluation

Intent and entity recognition accuracy are widely used metrics for evaluating NLU in voice assistant systems (Chen et al., 2018). The intent and entity recognition accuracy are compared across each system function (e.g., asking the weather, dialling a contact) in one voice assistant evaluation (Chen et al., 2018). For the Snips benchmark, the primary evaluation metric is the average F1-score of intent classification and slot filling, using chat-bot, web apps and ask Ubuntu datasets. Inference times of the NLU pipeline were also compared across different devices, including an iPhone 6s and a Raspberry Pi 3 (Rouchon, 2017; Coucke et al., 2018).

LLMs for smart home control are evaluated using a set of immediate commands of varying ambiguity, with binary quality scores assigned to outputs (King et al., 2023a). Results show promise for the use of LLMs in this domain, however improving system reliability and robustness is noted as future work. A further work evaluates LLM outputs across a set of smart home tasks (e.g., temperature, lighting, security control) in terms of token cost, latency, annotator-rated response quality and categorisation of common failure modes (King et al., 2023b). Both persistent and immediate commands are experimented with, and LLM device selection accuracy is quantified in terms of false positives, false negatives and relevance. A benchmark for evaluating LLMs on smart home control tasks, HomeBench, has also been proposed that specifically includes a mixture of valid and invalid instructions that target single and multiple devices (Li et al., 2025). The task defines a structured home environment with devices, states and methods, where the LLM selects actions based on user input. Evaluations quantify LLM performance across valid instructions, as well as invalid instructions to measure whether models correctly refuse commands where no action is possible.

More generally, however, LLM evaluation is a complex challenge with mounting research on the topic due to their applicability across diverse domains. A holistic evaluation of LLMs measures accuracy, calibration, robustness, fairness, bias, toxicity, and efficiency (Liang et al., 2022), indicating the wide characteristics of these models. Quantitative benchmarks are common, for example Bang et al. (2023) detail an evaluation framework for interactive LLMs that compares popular systems like ChatGPT on common NLP tasks, and (Liu et al., 2023b) benchmark LLM-augmented autonomous agents effectiveness on real-world tasks. The Hugging Face Open LLM Leaderboard[48] provides an ongoing comparison of model performance on various benchmarks, helping to keep pace with the regular release of newly fine-tuned or trained LLMs. The lmsys chatbot arena[49] also ranks LLMs using crowdsourced user feedback, and the OpenAssistant dataset provides annotated prompts and responses with quality ratings, supporting research in aligning LLM outputs to human preferences (Köpf et al., 2023). The outlined works illustrate the ongoing research into robust evaluation strategies to compare aspects of LLM performance and behaviour across various applications.

Taking inspiration from these evaluation approaches, the spoken commands dataset developed in the next chapter will be annotated to assess LLM reasoning performance across a

[48] `https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard`
[49] `https://lmsys.org/blog/2023-05-03-arena/`

range of command types. Labels for expected device selection and actions will be included to develop an approach for evaluating the accuracy of LLMs in a smart home control context.

End-to-end performance evaluation

The evaluation of voice assistant systems generally requires a multifaceted approach looking at performance, usability, and efficacy. Intent recognition and task completion rates are commonly used as a basis to represent the number of commands accurately interpreted and executed, however user satisfaction is an important consideration, entailing usability, latency, and the extent of natural language understanding (Todericiu, 2025). User feedback from real-world interaction is therefore valuable to gain end-user insights of system usability.

Vacher et al. (2015) explores both qualitative and quantitative methods to evaluate smart home voice interaction, using scenarios to simulate real-world tasks to allow for a structured evaluation of how effectively users could interact with the system to complete tasks. The evaluation revealed user dissatisfaction with rigid grammars, a preference towards natural commands, and uncertainty due to the absence of user feedback mechanisms. Task-based evaluation is common to assess smart home control systems. Huang et al. (2015) design scenarios requiring users to give commands to complete predefined tasks in order to analyse user experience (number of turns taken), task success rates and overall system function. A set of voice commands is used to compare Google Assistant, Amazon Alexa and Apple Siri in terms of command execution success rate and response time in another study, with Google Assistant performing best in both, with 95% success rate and 1 second response time (Isyanto et al., 2020). Similarly, Mun et al. (2020) apply a voice command dataset to evaluate the response times of Amazon Echo and Google Home smart speakers, with analysis considering where delays in performance occur, such as in device startup or voice understanding.

Voice assistant user experience can be evaluated from a HCI perspective across scales of usability, value, adaptability, desirability, anthropomorphism and personality (Faruk et al., 2024). This research highlights that the interaction should not allow for confusion or misunderstandings, and that the voice assistant should ask questions, clarify doubts, and positively reward the user to give confidence. A three-dimension usability scale for voice assistants includes usability, referring to the users perception that the voice assistant will do the instructed task, affective, referring to the satisfaction of the user after using the assistant, and recognisability, meaning the functions and responses should be easily understood (Zwakman et al., 2021). Pyae and Joelsson (2018) evaluate the usability of the Google Home smart speaker, using a survey to gather feedback on user's experience with the product in terms of the perceived usefulness and issues encountered, with responses coded to identify common patterns and themes. The ISO 9241-11 framework has also been considered as a standardised framework for assessing voice assistant usability (Dutsinma et al., 2022).

In the context of this research, creating a dataset of commands, with target devices and expected device updates, will enable thorough evaluation and comparison against existing cloud-based systems. The accuracy of local ASR in smart home environments can be measured, and LLM outputs can be assessed in terms of precision and correctness in generating valid device updates. End-to-end performance will lastly be evaluated, covering task success rate, latency, safety and testing the integrated system functions as expected.

2.5 Summary

The chapter explores the feasibility of implementing a locally processed, voice controlled smart home system, addressing concerns around privacy and security by keeping data on devices in the home. Through a taxonomic classification of smart home technologies on the market, terminology is clarified and the types of capabilities required for a local voice assistant are identified. The taxonomy informs system design that will seek to replicate the identified voice assistant functionalities in a locally run setup.

Examining both commercial and academic approaches to voice controlled smart home systems, highlighting various architectures, implementations and evaluation methods, gives a holistic reflection of current smart home technologies. The analysis of open-source technologies and hardware devices indicate Whisper models for ASR, combined with effectively grounded and bounded open LLMs for reasoning, as promising for a local, flexible and high-performance voice assistant. This setup could provide accurate ASR transcription, and flexible command interpretation with LLM capabilities that could improve upon the brittleness of previous voice assistant implementations (detailed in Chapter 5.1) and allow for conversational repair behaviour. The accuracy and performance of the ASR and LLM components need to be evaluated in a smart home context, and LLMs require a mechanism for safe and reliable error handling given their faulting and hallucination tendencies. Therefore, these are posed as key questions to address in the development of the end-to-end voice assistant system.

The lack of standardised evaluation framework and the absence of a dedicated smart home commands dataset to evaluate ASR and NLU components in this domain is a notable research gap. A robust evaluation approach is required to compare voice technologies for smart home systems across commercial and research realms and evaluate system performance to reliably address the research questions. The next chapter details the development of the smart home command dataset that will be used in the evaluation and benchmarking of system components.

The chapter has begun to answer the research questions laid out in Chapter 1 by identifying candidate open-source software and available hardware for a local voice assistant, comparing local ASR and NLU models that can support this, and outlining methods and requirements for the thorough evaluation of voice control technologies. With this, the chapter establishes a foundation for the design, development and evaluation of a local, LLM-driven smart home voice assistant, where identified approaches will be systematically addressed in subsequent chapters.

Chapter 3

Smart Home Commands Dataset

3.1 Introduction

This chapter investigates the design, recording and analysis of a dataset of spoken smart home commands, with the aim of forming a robust evaluation framework for the voice assistant system. The development and reproducibility of voice control systems for smart environments is limited by the availability of data in the domain of smart home voice interaction (Mishakova et al., 2019). Creating a dataset of spoken smart home commands is therefore important for evaluating locally run voice technologies in the context of smart home control.

Smart home command recognition is uniquely challenging as microphones are typically placed far from the speaker, microphone quality varies, and various types of noise are present that can interfere with accurate recognition. The aim is to create a dataset to evaluate ASR, NLU and end-to-end performance under realistic smart home conditions, that is varying device placement, device types and environmental noise.

The dataset will include recordings of typical smart home command sentences, with noise types played through speakers and microphones positioned at varying distances to simulate real-world conditions. The dataset will consist of speech from UK-based adults, and will incorporate a variety of command types applicable to voice control in the home. This setup and resulting dataset will allow for evaluating and comparing ASR performance in smart home environments across hardware configurations (cloud vs local). In addition, the dataset can be used to fine-tune ASR models specifically for smart home command recognition.

To allow for robust evaluation of local ASR systems for a smart home voice assistant, the dataset is designed with the following requirements:

- **Smart Home Commands**: Capture typical voice commands used to control smart home devices.
- **Hardware Diversity**: Use diverse recording devices given varying hardware is common to smart home settings and microphone quality can affect ASR performance.
- **Accent Diversity**: Include speakers with different accents, including second-language speakers, to assess ASR performance across a diverse range of people who use smart home assistants.

- **Noise Variation**: Simulate typical household noise conditions, given noise can impact ASR performance.
- **Distance Variation**: Place microphones at various distances from the speaker, reflecting real-world usage in smart homes, to evaluate ASR performance across distances.

The chapter begins with an overview of existing datasets, that serve to inspire and influence the dataset design. Following this, a command grammar is constructed by adapting the existing VISH grammar (Noura et al., 2020) and tailoring it towards the intents and capabilities of voice assistants that will be replicated and that were identified in Chapter 2. The recording methodology is outlined, detailing the recording devices, the environment setup, the participants and metadata collection, the recording procedures, and the data post-processing steps. Finally, an overview of the resulting smart home command dataset is given, in terms of its structure and the analysis of the obtained data.

The work contributes to the central objectives of evaluating privacy-preserving methods for smart home interaction that rely on local processing, eliminating the need for cloud-based solutions. While the primary use of the dataset is for the ASR evaluation in Chapter 4, it will be re-used and adapted in Chapter 5 for the evaluation of LLM reasoning capabilities on smart home commands, and for the end-to-end system and safety testing in Chapter 6.

3.2 Existing Datasets

3.2.1 Speech Datasets

Standard speech corpora are widely used for training and evaluating ASR systems. These datasets are designed for general, large-vocabulary ASR transcription tasks. Commonly used corpora are listed, summarising the characteristics of each, with comments on their applicability to the thesis aims:

- **Librispeech**: A corpus containing 1000 hours of read speech derived from audiobooks, sampled at 16kHz (Panayotov et al., 2015). The data is high quality and clean, making it ideal for training and benchmarking ASR models. The focus on read, formal speech however, makes it less applicable to smart home interactions.
- **Mozilla Common Voice**: A crowdsourced dataset containing single sentence utterances sourced from Wikipedia articles that are read aloud by volunteers (Ardila et al., 2019). The dataset contains a broad demographic representation and a variety of recording devices that is desirable, however the sentence contents do not reflect the nature of smart home commands.
- **TedLium**: Comprising 452 hours of speech from TED talks (Hernandez et al., 2018), the data is expansive. The limitation to public speaking scenarios limits its applicability to the short, command-oriented utterances in smart home environments.
- **GigaSpeech**: Contains 10,000 hours of transcribed speech across multiple domains, including arts, science, sports (Chen et al., 2021). The dataset is vast, but is not command-oriented, and the contents are not relevant to smart home control.

- **Google Speech Commands**: Contains 65,000 utterances of 30 short words, each 1 second long and spoken by thousands of different speakers (Warden, 2018). While useful for keyword spotting tasks, it does not include full command sentences.

While these speech datasets are large and vast, they do not suffice for the task of benchmarking ASR models on command recognition in smart homes, that is required to thoroughly answer the research questions. The unique real-world conditions of smart homes, such as environmental factors, microphone positioning, noise present, are not adequately represented. The types of command structured sentences and sentence lengths are also not seen in the datasets, where a user is issuing an action in a conversational, spontaneous setting.

Smart home specific speech datasets include CHiME-Home, annotated domestic environment audio recordings designed for a noisy speech recognition challenge (Barker et al., 2018). Additionally, the DIRHA corpus includes 24 native English speakers speaking in a domestic environment, with multiple microphones recording and spoken content including read (articles) and spontaneous speech (on conversational topics), as well as keywords (Ravanelli et al., 2015). While typical home environmental noise and speech is present in these datasets, they do not adequately represent the types of speech that would allow for a robust evaluation of ASR model performance on smart home commands.

All datasets discussed solely focus on speech transcription, with no consideration for understanding the meaning of utterances. The absence of annotations for intents, entities, or actions means that following the transcription of an utterance, there is no existing way to then evaluate whether its intended meaning is correctly interpreted. This prevents the ability to evaluate ASR, NLU and end-to-end performance with the same dataset and assess the overall effectiveness of the voice assistant in real-world scenarios that is required to answer the research questions. However, in the field of spoken language understanding (SLU), where spoken inputs are directly mapped to outputs, datasets have been developed with structured annotations to allow for end-to-end evaluations. SLU datasets go beyond raw transcription by including annotations of intents, entities, and expected actions, with existing SLU datasets and their suitability for evaluating voice control in smart homes discussed in the next section.

3.2.2 Spoken Language Understanding Datasets

Several SLU and NLU datasets have been released that are more applicable to the evaluation of command and control systems, like our smart home control task. Such annotated datasets, that relate to the objective of creating a dataset of smart home commands, are introduced:

- **SNIPs**: The dataset, collected through crowdsourcing, consists of a few thousands text queries with annotated intents and slots, and was later paired with corresponding spoken utterances for each text query (Coucke et al., 2018; Saade et al., 2018). There are two datasets: SmartSpeaker (9 intents, 65k vocabulary size) that includes track navigation, volume control, playing, pausing and retrieving information, and SmartLights (6 intents, 400 word vocabulary) centred on lighting, brightness control and colour settings. A far-field dataset was created by playing the spoken utterances through a neutral speaker and recording them with a microphone array positioned 2 metres away. Data augmentation techniques were also applied to simulate noisy and reverberant conditions.

- **Fluent Speech Commands**: Contains 30k utterances crowdsourced from 97 speakers in the USA and Canada, recorded as 16kHz single-channel files, and consisting of control commands for smart home devices and virtual assistants (Lugosch et al., 2019). There are 248 unique sentences that link to 31 intents that incorporate action, object and location slots. Phrases were presented in a random order, and participants spoke each phrase twice. A CSV file describes each utterance in the dataset, including the transcript, speaker ID and the action, object and location contained in the utterance. A CSV file details demographic information about each speaker, such as their first language, current language used for work, gender, age range and speaking ability.

- **SLURP**: The SLURP dataset[1] contains approximately 72k audio recordings of user interactions with a home assistant, originally designed for an in-home robot assistant (Bastianelli et al., 2020). It spans 18 domains, with three levels of semantic information annotated: Scenario (18+ types), Action (46 types) and Entities (55 types). Commands were formulated by Amazon Mechanical Turk workers who were given 200 pre-defined prompts, and then were further annotated with scenario, action and entity. This was first released as a text-only NLU benchmark, and audio data was later collected in home and office-like environments, with 100+ participants reading prompts displayed on a tablet in 1 hour-long sessions. Speech was recorded using close-talk and distant microphone arrays, where participants varied their location in the room and switched between seating, standing and walking positions.

- **Timers and Such**: This dataset[2] contains spoken commands for common voice control use cases that involve numbers (Lugosch et al., 2019). The dataset includes four intents: SetTimer, SetAlarm, SimpleMath and UnitConversion. Prompts were randomly generated using a script[3] that produces variations of phrases for each of the four intents, with random numbers inserted. 89 recording sessions were conducted (but not necessarily 89 unique speakers), with recordings converted to single-channel 16kHz WAV files.

- **MASSIVE**: An Amazon-released, open-source speech dataset, aiming to encourage the development of third-party apps and services for the smart speaker Alexa. The dataset contains one million spoken samples across 51 languages (Quach, 2022). Each spoken command corresponds to a text transcript with annotated intent and slot labels. The dataset is extensive and useful for building multilingual AI models for voice assistants.

- **STOP**: The Spoken Task-Oriented semantic Parsing (STOP) dataset (released by Meta as part of a challenge) is intended for use in end-to-end SLU tasks and contains 236,000 audio files of 885 speakers (Tomasello et al., 2023). The dataset builds on top of the TOPv2 dataset (Chen et al., 2020) that provides text-only inputs for domains including: navigation, event, alarm, messaging, music, reminder, timer, weather. Two audio recordings were recorded by Amazon Mechanical Turk Workers for each utterance in the validation and test sets of TOPv2, and one for each utterance in the train set.

The highlighted SLU datasets provide valuable insights and inspire the design of the smart home command dataset. While not all are tailored to smart home commands and

[1] `https://github.com/pswietojanski/slurp`

[2] `https://zenodo.org/record/4623772`

[3] `https://gist.github.com/lorenlugosch/5df9e30227aa5c67ff51cd28271414f0`

environments, they incorporate relevant principles: diverse range of speakers, acoustic variety through different noise conditions and device placement, hardware variety using different recording devices, and coverage of a variety of command phrasing to reflect real-world, natural interaction, in addition to semantic annotations (e.g., intents, actions, entities). The features are fundamental for the evaluation and comparison of ASR and NLU components in a local smart home voice assistant and to address thesis objectives.

3.3 Command Grammar

3.3.1 Existing Command Grammars

This section focuses on the design of a smart home command grammar, that is the prerequisite for generating a structured and comprehensive dataset that covers all voice assistant capabilities identified and that are sought to be replicated. It establishes the structure, scope, diversity and vocabulary of the commands to be recorded. By reviewing existing grammars, an existing crowd-sourced smart home grammar is identified, and then further adapted to ensure it aligns with realistic voice assistant usage, captures a breadth of control scenarios, and represents the diversity in how commands are expressed.

There are some attempts to formalise and enumerate the commands associated with various smart home control tasks. Such grammars enable different devices and control scenarios to be systematically covered, reflect the variation in which commands can be uttered, and help to define the overall scope and format of the command grammar to record. The taxonomy work (Chapter 2) guides the set of intents, commands and device types sought to include, therefore grammars are reviewed in terms of the range of user intents, the variation in intent expression, and the format and structure.

- **HomeAssistant Intents**: The Home Assistant intents repository[4] contains text-only templates to define various types of smart home commands for use with the Home Assistant voice component[5]. The Intents include TurnOn, TurnOff, GetState, SetTemperature, MediaPause, to name a few, with several template sentences given for each intent. Ranges are set for numerical units, like brightness and temperature, in a common YAML file. Expansion rules are also defined to include synonyms and variation in the way things are said, with examples including `name: [the] {name}`, `what_is: (what's|whats|what is)`, `set:(set|make|change|turn)`. An example sentence can look like this: `<turn> on <name> (light[s]|[light] switch[es])`. There are folders listed with services that can be used with HomeAssistant and the actions that can be performed for each service, e.g., Spotify[6] includes actions browse_media, next_track, pause, play, play_media, previous_track, repeat_set, seek, select_source, shuffle_set, volume_set. While it is useful to see the range of intents and the clear structured format, it is a rigid approach, and moreover is not written to generate natural language smart home commands that is required for the recording setup.

[4] https://github.com/home-assistant/intents

[5] https://developers.home-assistant.io/docs/voice/overview/

[6] https://github.com/home-assistant/core/blob/dev/homeassistant/components/spotify/media_player.py

- **VISH grammar**: VISH was created for training NLU models to support goal-oriented and more flexible smart home dialogue systems (Noura et al., 2020). This aims towards enabling more flexible understanding of user utterances, where direct and indirect commands can be interpreted, aligning with the thesis goal of a flexible voice assistant. The grammar was created by collecting and analysing user utterances, and then processing these to categorise and identify grammatical similarities and placeholders. The data was further expanded to include synonyms and varying syntactic structures. The grammar is specified in extended Backus-Naur Form (eBNF) that defines rules to generate valid utterances for smart home contexts, and in total the VISH grammar is reported to produce 5 million unique utterances for smart home control. The VISH grammar has a variety of actions that cover an extensive range of smart home capabilities, and incorporates linguistic flexibility and diversity to cover variation in command expression.

- **Wit.ai**: Wit.ai is a cloud-based NLU service that has built in intents[7] (add_to_playlist, create_timer, decrease_volume), entities[8] (duration, location, volume) and traits [9] (on_off, sentiment). There are no sentences defined as these must be user generated, and the scope of built-in intents is very limited.

- **Rhasspy**: Rhasspy[10] has a method for formulating different commands that relate to the same intent for training SLU systems[11]. Commands are specified in a template language, e.g., `states = (on | off), turn (<states>){state} [the] light`, where words in square brackets are optional, and words separated by | in brackets give alternatives, with options to define other grammar components like number ranges, slot synonyms and lists.

The review of existing command grammars informs the approach taken towards creating a grammar to generate commands in the recording sessions. The recording session will involve participants reading out commands on a screen, therefore a grammar that generates commands relating to a range of smart home control tasks and that reflects variations in the way commands are phrased is required. Among the reviewed grammars, VISH meets the requirements, with a variety of intents, syntactic flexibility and vocabulary. The following section presents the analysis and adaptation of this dataset for use in the recording setup.

3.3.2 Grammar Development

The VISH grammar appears a solid basis for the command grammar to record for the smart home command dataset. The grammar is examined further, and then the scope of the dataset to include is selected based on the set of capabilities and devices identified in the taxonomy work (Chapter 2). Further modifications and refinements are then made so that the grammar generates natural-sounding smart home commands.

[7] `https://wit.ai/docs/built-in-intents/20210928/`
[8] `https://wit.ai/docs/built-in-entities/20210928/`
[9] `https://wit.ai/docs/built-in-traits/20210928/`
[10] `https://github.com/rhasspy/rhasspy`
[11] `https://rhasspy.readthedocs.io/en/latest/training/`

Initial Analysis

Insights from the initial analysis of the grammar for generating smart home commands include:

- **Formatting errors**: The grammar has formatting errors, causing errors when using a script to generate commands from templates. These include spelling errors for production rules meaning they cannot be found, and missing '||' from the end of lines.
- **Strange sentences**: Generating templates based on the raw grammar does not yield sentences that make sense to be spoken. Perhaps this is of little issue in NLU systems, but for ASR systems it is important that the sentences somewhat make sense for people to speak aloud. There are too many template sentences to correct each one, therefore a subset will be captured and amended to make sense as spoken commands.
- **Data augmentation**: The dataset is significantly augmented or padded out by adding the phrases 'automatically' and 'for me' to template sentences. These phrases appear an unrepresentative amount of times, especially automatically, that does not sound natural in smart home commands. There is very little syntactic variation at times, with commands differing by just one word, instead of setting the word as optional.

The grammar is then examined closely, looking at the intents, slot fillers and vocabulary covered:

- **Out-of-scope intents**: The taxonomy work guides the set of intents and commands required and therefore a set of devices and actions that represent the capabilities of commercial voice assistants is targeted. There are intents that are not useful, relevant to the task, or are too specific or niche to be included, are repetitions of intents covered elsewhere, or have very few associated template sentences. These include: 'DeviceCharge', 'DeviceFill', 'DeviceDrain', 'DeviceDock', 'DeviceFlash', 'DeviceUnflash', 'DeviceDisconnect', 'DeviceConnect', 'CookFood', 'DeviceBlend', 'SmartDrink', 'DevicePreheat', 'DimDevice', 'AdjustDimDevice', 'IncreaseColour', 'DecreaseColour', 'DecreaseWaterConsumption', 'DeviceStateMode', 'RecordMovie', 'AcceptRejectCall', 'MakeCall', 'DeviceGamePlay', 'DeviceOscilate'.
- **Out-of-scope slot fillers**: Some slot fillers that are not useful or in the scope of the recordings, by either being too specific or devices being out-of-scope, or the vocabulary in these slot fillers does not make sense. Slot fillers removed include: 'SmartGamingConsole', 'SmartRemote', 'SmartPhone', 'SmartKitchenAppliance', 'SmartCooker', 'SmartFridge', 'SmartToaster', 'SmartBoiler', 'SmartBlender', 'SmartTub', 'SmartShower', 'SmartTap', 'SmartGarden', 'SmartWatch', 'WaterVerb', 'AcceptRejectCall', 'ReadObjects', 'MusicName', 'MovieArtist', 'MusicArtist', 'NegAirCommand', 'NegAirVerb', 'PosBrightnessVerb', 'Oscillate'.
- **Obscure vocabulary**: There is obscure vocabulary that should be removed, either through not making sense or being potentially offensive.

Adapting the Grammar

The process of transforming the VISH grammar into a usable state for the task of generating smart home commands is described, addressing each point from the above grammar analysis:

- **Fix formatting errors**: Enable usable command generation with the grammar by fixing spelling errors for production rules, and adding missing '||' to the end of lines.
- **Fix data augmentation**: Remove all template sentences containing 'automatically' given this appears to be unnecessary and unrealistic padding.
- **Remove unwanted intents**: Select intents that are relevant and useful, and remove those that are not required to fulfil the outlined scope.
- **Remove slot fillers**:
  - Remove slots fillers/non-terminal symbols that are not used/unreachable after the removal of template sentences fixing data augmentation and removing unwanted intents.
  - Identify and remove remaining slot fillers/non-terminal symbols that are not required (e.g., are out-of-scope from the market analysis), and remove productions that use these slot fillers.
- **Remove unwanted vocabulary**: Remove specific vocabulary from the slot fillers that is inappropriate or obscure.
- **Select template subset**: Select 30 templates from each intent, distributed evenly over the annotated command types (device command, direct, indirect). Remove slot fillers that are no longer used in templates (resulted in 7 unused slot fillers).
- **Form sensible sentences**: Manually modify the grammar to form sensible sentences based on observations. Each template was tested, and modifications were made, including adding plural non-terminal symbols, e.g., SmartAudioPlural, SmartHeaterPlural, expanding vocabulary, and reincorporating any slot fillers that were no longer used, but where the vocabulary is relevant and desired in the dataset. The Home Assistant intents repository was also referred to for further vocabulary to include relating to smart home interaction.
- **Test resulting grammar**: Manually test and check the resulting grammar reliably generates smart home command sentences that sound realistic and natural.

Grammar Intent Hierarchy

Following the grammar adaptation, a set of intents are defined, with thirty template sentences relating to each intent, and a corresponding set of slot fillers for flexibility and variation. The intents are organised into a hierarchy to guide the random template selection in the recording setup. The hierarchical organisation classifies and groups actions and intents related to specific devices, actions and voice assistant capabilities. The resulting grammar classification is given in Figure 3.1, with four top level categories: General, Atmosphere, Entertainment, Security.

```
{
    "general": {
        "device_control": {
            "on_off": ["TurnDeviceOn", "TurnDeviceOff", "TurnAllDeviceOn", "
    ↪ TurnAllDeviceOff"],
            "start_stop": ["DeviceStart", "DeviceStop"],
            "pause_resume": ["DevicePause", "DeviceResume"]
        },
        "assistive_tasks": {
            "assist": ["SetAlarm", "DecreaseEnergyConsumption", "SetPreference", "
    ↪ DeviceBrew"]
        }
    },
    "atmosphere": {
        "set_lighting": {
            "set_brightness": ["AdjustBrightness", "IncreaseBrightness", "
    ↪ DecreaseBrightness", "AdjustDeviceColour"]
        },
        "set_temperature": {
            "set_temperature": ["AdjustTemperature", "IncreaseTemperature", "
    ↪ DecreaseTemperature"]
        },
        "set_climate": {
            "set_humidity": ["AdjustHumidity", "IncreaseHumidity", "DecreaseHumidity"],
            "set_airquality": ["AdjustAirQuality", "IncreaseAirQuality", "
    ↪ DecreaseAirQuality"]
        }
    },
    "security": {
        "control_door_window": {
            "open_close": ["DeviceOpen", "DeviceClose", "ShadesOpen", "ShadesClose"],
            "lock_unlock": ["DeviceLock", "DeviceUnlock", "DeviceLockAll", "
    ↪ DeviceUnlockAll"]
        },
        "control_surveillance": {
            "arm_disarm": ["DeviceArm", "DeviceDisarm"],
            "show_hide_cam": ["ShowCamera", "HideCamera"]
        }
    },
    "entertainment": {
        "set_entertainment": {
            "play_item": ["PlayMusic", "PlayMovie"],
            "adjust_channel": ["AdjustChannel", "IncreaseChannel", "DecreaseChannel"]
        },
        "set_volume": {
            "adjust_volume": ["AdjustVolume", "IncreaseVolume", "DecreaseVolume"],
            "mute_unmute": ["DeviceMute", "DeviceUnmute", "DeviceMuteAll", "
    ↪ DeviceUnmuteAll"]
        }
    }
}
```

Figure 3.1: *Smart home grammar intent classification*

Metric	Original VISH	Modified VISH	Description
Vocabulary size	2228	1267	Number of unique words used in the grammar.
Number of Templates	377262	1470	Number of sentence templates defining command structures.
Number of Intents	72	49	Number of distinct intents covered.
Number of Slot Fillers	339	209	Number of variable placeholders, for names, synonyms, values.
Average Command Length (in words)	11.12	7.85	Average number of words per generated command.
Estimated Unique Utterances	45576737463472518	867416675	Approximate number of unique commands that can be generated.

Table 3.1: *VISH grammar comparison, before and after modification.*

These categories group related actions together, that each have their own subset of actions with a set of specific intents.

The random generation of smart home commands will first randomly select from the top level categories: General, Atmosphere, Security, Entertainment. Given a category, it will select an action within that, and then a more specific action within that category, before randomly selecting a specific intent. Each intent has thirty associated templates, where a template for a given intent will be randomly selected, and then each template slot will be filled in with a random vocabulary selection for each slot name to form a sentence. The hierarchy will be used to select intents in the following way:

- **General** → `device_control` → `pause_resume` → `DevicePause`
- **Atmosphere** → `set_lighting` → `set_brightness` → `IncreaseBrightness`
- **Security** → `control_surveillance` → `show_hide_cam` → `ShowCamera`
- **Entertainment** → `set_entertainment` → `adjust_channel` → `DecreaseChannel`

3.3.3 Grammar Evaluation and Analysis

Grammar Analysis

Following the process of modifying and refining the VISH grammar to better align with the objectives of the dataset recording, the resulting grammar is analysed, exemplifying the types of commands it can now generate, and acknowledging its limitations and boundaries.

The analysis of the VISH grammar before and after modification focuses on metrics like vocabulary size, number of templates, actions, slot fillers, and estimated unique utterances to quantify the scope and flexibility of the grammar. The results of this analysis are given in Table 3.1, providing a comparison of the grammar before and after the modification steps taken in the previous section.

The number of templates is greatly reduced, with focus placed on generating natural-sounding smart home commands, rather than the jumbled commands seen in the original

Original VISH examples	**Modified VISH examples**
i would love to automatically ACTION:turn DEVICE:outlet STATE:out please ||| SCOPE:Device GOAL:DeviceCommand	i'd love to ACTION:wake at VAL:5 TIME:pm VAL:friday ||| SCOPE:Global GOAL:Indirect EFFECT:IncreaseValue PARAM:Brightness EFFECT:IncreaseValue PARAM:Noise
please automatically ACTION:close_up DEVICE:doors for me in my LOCATION:library ||| SCOPE:Device GOAL:DeviceCommand	move VAL:four programs forward ||| EFFECT:IncreaseValue SCOPE:Device GOAL:Direct PARAM:Channel
i command you to ACTION:manage this LOCATION:atmosphere DEVICE:timer to on VAL:40 TIME:seconds on an DEVICE:yogurt_maker ||| SCOPE:Device GOAL:DeviceCommand	could you ACTION:make the LOCATION:flat warmth VAL:22 UNIT:degree_celsius ||| EFFECT:FixedValue PARAM:Temperature SCOPE:Global GOAL:Indirect
please automatically ACTION:make DEVICE:entrance_doors in LOCATION:bedroom ACTION:unfold ||| SCOPE:Device GOAL:DeviceCommand	please ACTION:modify the LOCATION:downstairs warmth to VAL:23 UNIT:c please ||| EFFECT:FixedValue PARAM:Temperature SCOPE:Global GOAL:Direct
ACTION:modify a LOCATION:inside DEVICE:apple_tv VAL:85 UNIT:% please ||| PARAM:Brightness SCOPE:Device GOAL:DeviceCommand	i command you to ACTION:unlock the DEVICE:bay_window LOCATION:inside please ||| SCOPE:Device GOAL:DeviceCommand

Table 3.2: *Grammar generated sentence examples.*

grammar. The number of actions has been reduced in line with the device scope defined in the taxonomy, which is also the scope for the voice assistant evaluations. While the original grammar is somewhat larger, the vocabulary size remains large, covering a wide range of actions, and a more than adequate number of unique utterances can be generated.

The grammar analysis outputs two files, for each the original and modified VISH grammar:

- templates.csv: containing each template, with its intent, calculated number of variations, and an example generated sentence.
- slots.csv: containing each slot symbol, the associated terminals, and the total number of terminals the symbol produces.

The average utterance length is calculated by averaging the number of words in each example generated sentence in templates.csv. The slots.csv shows how many terminals a non-terminal symbol can map to (e.g., DeviceOnOff maps to 102 terminal symbols), indicating how such vast variation in utterances is captured in the dataset, and how millions of unique utterances can be produced given 1470 template sentences. The estimated unique utterances is calculated by adding the calculated number of variations for each template sentence from templates.csv.

To compare the commands generated by the VISH grammar in its original form, and after modification, examples of grammar-generated sentences are presented in Table 3.2. The examples highlight how the modified grammar generates much more natural sounding command sentences that reflect real-world smart home interactions.

Grammar Scope and Limitations

This section evaluates the coverage and variability of the dataset to assess how effectively the grammar represents smart home interaction scenarios, and how flexible the grammar structure is for modification and adaptation purposes. Firstly, the grammar has constraints in representing specific vocabulary types that are common in smart home interactions but are difficult to standardise due to their limitless nature:

- **Proper nouns**: Names of places, people, artists, songs, albums, TV shows, films, directors and actors are not extensively covered. These categories have high variability and are personal to users, making them challenging to cover in the predefined grammar. A limited selection of artists, songs, films and TV shows are included based on the contents of the original grammar, but this is not extended.
- **Product Names**: Commercial product names, such as Nest, Eve, Philips Hue, Chromecast, Fire TV, identified in the taxonomy work, are not consistently represented in the grammar. While some products are included, e.g., Fire TV and Chromecast, the grammar does not account for the full breadth of available smart home device product names.

The omission of these vocabulary types limits the grammar's ability to generate commands tied to media content or brand-specific devices.

Several command categories are either under represented or entirely omitted in the grammar, limiting its completeness in certain smart home scenarios:

- **Automations and Event-Driven Commands**: Commands relating to setting automated triggers and time-based events, such as *"lock the door at 8 PM"*, *"open the shades at sunrise"*, *"turn the fan off at noon"* are not fully covered.
- **Appliance-Specific Settings**: Commands relating to fine-grained settings of appliances like washing machines, dishwashers, pet feeders and robot vacuums are not included, with commands limited to turning these devices on and off.
- **Health Monitoring Commands**: Queries and commands relating to health metrics, such as those collected by wearable devices, like *"what's my heartrate?"* and *"how many steps did I take today?"*, are not included. These are not represented in the VISH grammar, and were not identified in current voice assistant systems, therefore lie outside the scope of the work.
- **Assistive and Productivity Tasks**: Commands for productivity and assistive tasks like calendar management, notes, lists, calls, texts, announcements and reminders, are absent due to their non-specific and extensive nature. Such examples include *"add milk to my shopping list"* and *"remind me to take my medication"*.
- **Information-Seeking Queries**: The grammar does not include internet-based information queries, such as *"what's the weather tomorrow?"* and *"what's the news today?"*, or sensor-based queries like *"how's the humidity?"*. These types of commands are more limitless, therefore are harder to define and require external data sources and information retrieval.

- **Compositional Commands**: Multi-step commands, where multiple actions are combined in a single utterance are not modelled in the grammar. Such examples include *"turn on the lights and play music"* and *"turn the heating on and dim the lights"*. These types of multiple actions can be inferred from the goal-oriented SetPreference commands, but compound commands are not specifically defined. This limits the ability to evaluate reasoning performance on explicitly multi-step commands.

The adapted VISH grammar covers a significant range of smart home control commands, and defines a structure for specifying smart home actions and sentence templates, with high potential for future expansion to address the limitations. It supports flexibility through the slot fillers, allowing for new device names, media content and synonyms to be added without requiring structural changes. However, it is less straightforward to add new intents. This requires manual definition of templates and slot fillers for the new actions. Future iterations of the grammar could potentially explore dynamic template generation, or use machine learning methods of augmentation to reduce the reliance on manual updates, allowing for a more cost-efficient approach to expand the grammar as smart home technologies evolve.

For the purpose of the smart home command scope identified in Chapter 2, the grammar captures a representative, diverse and functionally complete set of smart home control commands. This ensures the dataset aligns with the thesis objective to replicate and evaluate the local system implementation against existing smart home voice control functionalities. The next section details the recording methodology in which this command grammar is used.

3.4 Recording Methodology

3.4.1 Recording Devices and Hardware

The recording devices consist of wall-mounted microphones, a close-talk microphone and microphone-equipped devices. It was sought to collect a variety of makes and models given that microphone quality can affect speech recognition performance (Rouchon, 2017). The hardware selected for inclusion was determined by the availability of devices and microphones within the University, where varying qualities were sought. This aims to replicate the real-world variability of audio input devices used in smart home scenarios, and extends the dataset so that the robustness of ASR systems can be tested across high- and low-quality recordings.

Two sets of devices are included: dedicated microphones mounted in the room, and mobile, microphone-equipped devices. The room mounted microphones are all wired into B-16 interfaces, which connect to a sample synchronous AVB network linked to a Mac desktop. Table 3.3 gives details of these, along with the speaker that is also wired in to provide audio output for playing scenario noise. Audio recording is controlled by Python code that uses the sounddevice library. To ensure consistent gain calibration across the microphones, white and pink noise were played through speakers to test that no microphones were recording sound at disproportionate (higher or lower) levels. The close-talk microphone gain was particularly lowered to prevent clipping and distortion from loud inputs.

The specification for each mobile recording device, including the app used to record, the device OS, the number of microphone channels and sample rate are summarised in Table 3.4. Multiple of the same device (e.g., ESP-Box, Nokia G22) are included so that they can be placed at various distances from the speaker position to allow for evaluation of the impact of

Device	Code	Num	Details
the t.bone HeadmiKe-O & **Sirus Quad B 823 body-pack transmitter**	CT	1	Omnidirectional condenser headset Frequency band: 823-832 & 863-865 MHz; Frequency range: 40-18,000 Hz
Sennheiser MEB 114 B	SEN-1..4	4	Frequency range: 40-20,000 Hz; Direction: Half cardioid
Superlux E304 BK	T-, FL-, FR-, FRS-, SL-, SR-, BL-, BR-	30	Frequency range: 20-18,000 Hz; Direction: Half sphere
Genelec 8010 AP (audio output)	–	3	Frequency range: 74-20,000 Hz (+/-2.5 dB)

Table 3.3: *Details of microphones and speakers.*

distance on speech recognition accuracy. Two iPads are used, where one records audio, and the other uses an app to measure the dB level in the recording session. The microphones and devices incorporate a mix of single- and dual-channel microphones, where all record at 48kHz, except the ESP devices that record at 16kHz. The recording app is standardised across the Android and iPhone devices for consistency. The mobile devices use various versions of Android and iOS systems, where the version is dependent on the age of the device. While the ESP-Box devices have dual-channel microphones, they are set up to stream a single audio channel to a laptop, where recordings are locally saved.

The codes in Table 3.4 correspond to Figure 3.2 and Figure 3.3 in the following section, where the environmental setup is presented with diagrams of the specific device and microphone positioning within the recording room.

3.4.2 Environmental Setup

The environmental setup is designed to reflect realistic and diverse smart home conditions. This section describes the characteristics of the recording room, including its size, layout and the positioning of microphones, devices, and the speakers used to play scenario noise. Figure 3.2 illustrates the full room configuration, with the layout of furniture, microphones, devices and speakers in the environment. The participant sits on a chair in front of three tables pushed together in a meeting-like setup. Mobile and ESP-Box devices are placed around the table at various distances, with Figure 3.3 depicting the specific configuration of devices on the table. Specific measurements for all device, microphone and furniture positioning in the room are given to allow for reproducibility, and the labels in Figure 3.2 and Figure 3.3 correspond to the device codes in Table 3.4.

- The Superlux omnidirectional microphones are mounted on the wall at various locations in the room and placed on the table, with Figure 3.4 showing a diagram of the specific configuration of the microphones on the board. Sennheiser microphones are placed at varying locations on the table in front of the speaker.
- ESP devices are placed at increasing distances and varying angles from the speaker: 50cm, 150cm, 250cm, 450cm and 600cm. These devices are static and generally will not move from their position in the home, therefore it is typical that speech will be captured at a distance on this type of device.

Device	Code	OS	Year	Recording app	Mics	Sample rate
Samsung Galaxy S5	SG5	Android 6	2014	Audio Recorder (Dmytro Ponomarenko developer)	2	48kHz
Moto G4 Play	MGP6, MGP7	Android 6, Android 7	2016	Audio Recorder (Dmytro Ponomarenko developer)	2	48kHz
OnePlus 6T	OP6T	Android 11	2018	Audio Recorder (Dmytro Ponomarenko developer)	2	48kHz
Google Pixel 8	GP8	Android 14	2023	Audio Recorder (Dmytro Ponomarenko developer)	2	48kHz
Nokia G22	NG22+F, L1, L2, L3, R1, R2, R3	Android 12	2023	Audio Recorder (Dmytro Ponomarenko developer)	2	48kHz
iPhone 6s	I6S	iOS 15.8	2015	Audio Recorder-WAV, M4A (loop sessions developer)	1	48kHz
iPhone X	IX	iOS 16.7	2017	Audio Recorder-WAV, M4A (loop sessions developer)	1	48kHz
iPad Pro 3rd Gen 11 inch	IP1, IP2	iOS 17.2, iOS 17.5.1	2021	Audio Recorder-WAV, M4A (loop sessions developer); Decibel X: dB Sound Level Meter (thanh dinh developer)	2, 1	48kHz
ESP32-S3-Box-3B	ESP1..5	Willow firmware	2024	Modified Willow, Willow-Inference-Server and Willow-Application-Server to stream and store recordings (tovera inc developer)	1	16kHz

Table 3.4: *Mobile recording devices specification.*

- The mobile devices are predominantly placed on the table in front of the speaker given this is typical, and phones tend to be in close proximity users. Several NG22 devices are also placed around the table for evaluating the effect of distance on speech recognition.

For the speaker configuration to play the scenario noise, a TV sound system setup is mirrored, as though the user were sitting on a sofa and watching TV. An abundance of information is available on how to position TV speakers. Advice varies, but indications of the appropriate height and distance are given, e.g., the distance between the speakers should be 75% to 100% of the distance that you are from them, at least 2 feet away from walls and the floor[12], and at ear height when sat down[13]. Additional advice suggests the left and right speakers should be located along, but several inches away from, the wall at the same distance from your seating position and at about a 30-degree angle so that sound is more precisely directed to the main listening position[14]. Taking the advice onboard, the speaker setup sets the speakers the same width apart as the user is from the centre of the speakers. The user is positioned 2.6m away from the middle of the 'TV', each speaker is positioned 1.3m either side of the 'TV' middle, and away from the walls. The height of the speakers is 104cm, which is about ear height when sat down in the chair and greater than 2 feet from the floor, as suggested. A third speaker faces toward the back wall that will play the background activity sounds, while the other two speakers will play the music and TV audio.

[12] `https://www.aperionaudio.com/blogs/aperion-audio-blog/2-channel-stereo-speaker-placement`

[13] `https://www.dolby.com/about/support/guide/speaker-setup-guides/5.1.2-dolby-atmos-enabled-speaker-setup-guide`

[14] `https://www.enclaveaudio.com/blogs/news/2-1-vs-5-1-vs-7-1-surround-sound`

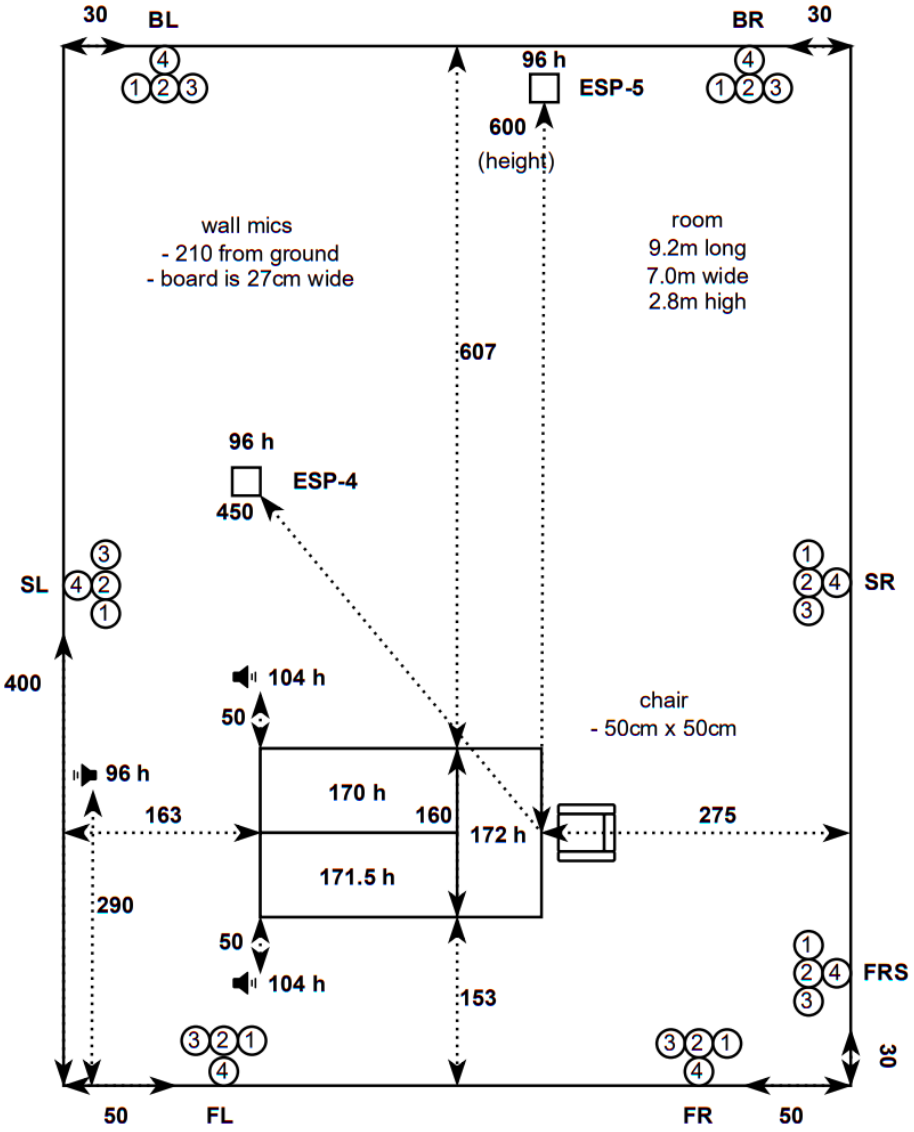

Figure 3.2: *Recording room microphone positioning (measurements in cm).*

For completeness, Figure 3.5, Figure 3.6, Figure 3.7 and Figure 3.8 give photos of the room and device layout, to bring the diagrams to life and show the actual environment. It can also be seen where soundproofing has been applied to the walls and how the boards hang from the wall soundproofing panels.

3.4.3 Scenarios Design and Definition

Scenarios are defined to simulate common noise types that occur in home environments. The four scenarios entail: Clean (no noise), Music, TV, Activity. The process of obtaining the audio to represent each scenario is detailed, along with the process of volume adjustment to reflect typical home conditions.

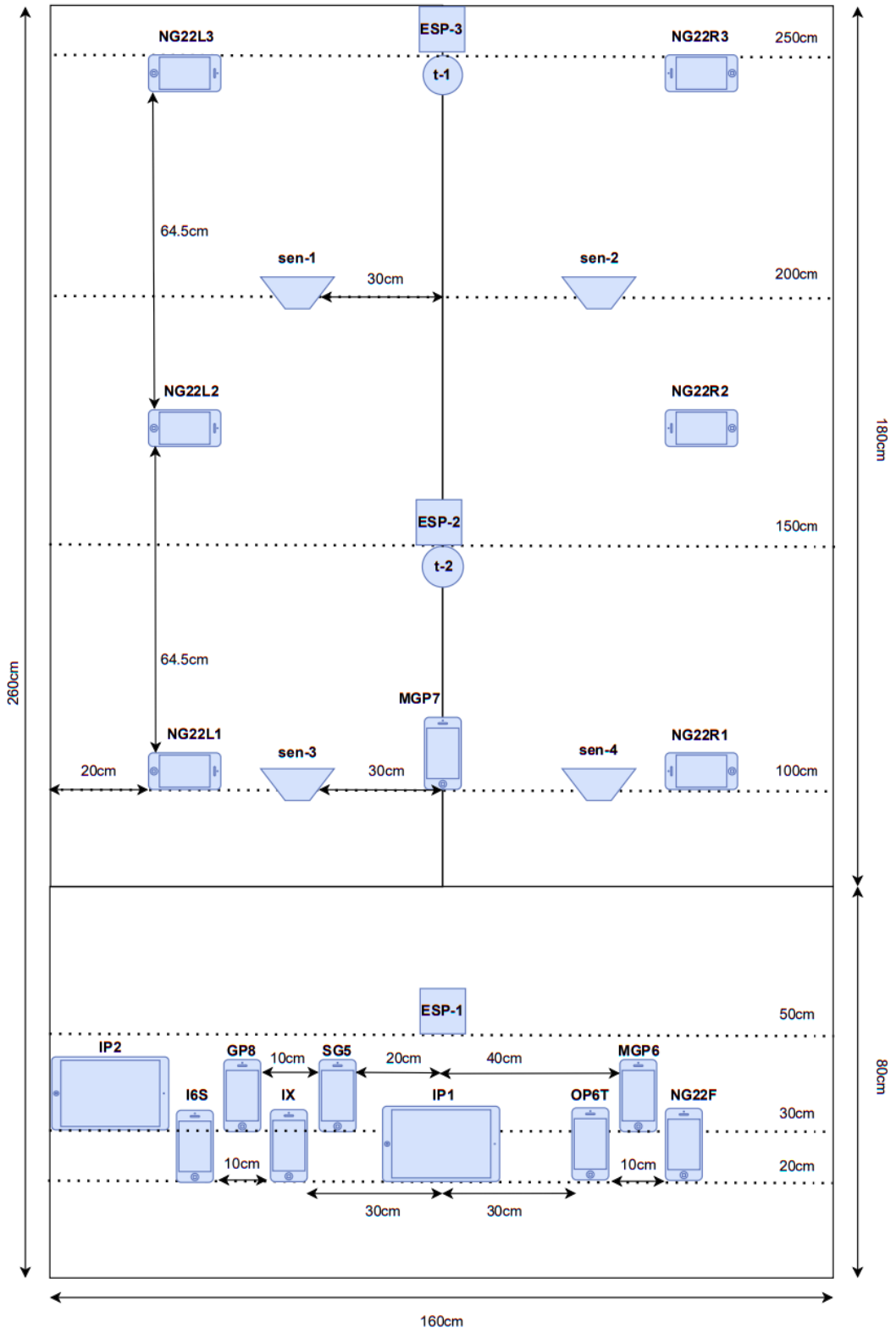

Figure 3.3: *Table device positioning.*

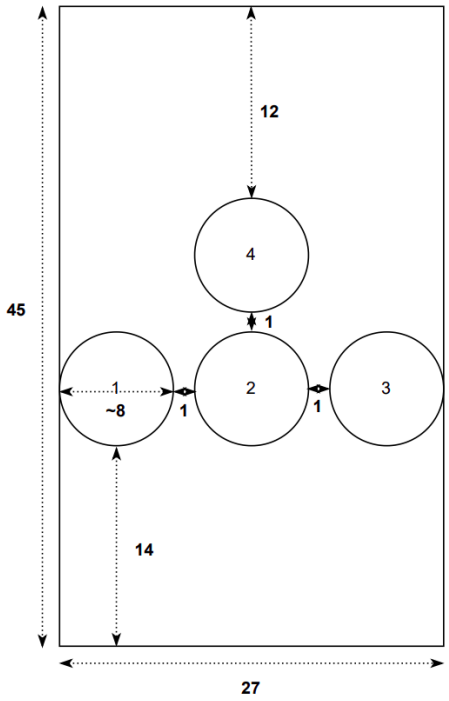

Figure 3.4: *Wall-hung microphone board setup (measurements in cm).*

1. **Music noise**: A set of six single minute music clips across a variety of music genres are selected. The music clips were sourced from Free PD music[15], where the selection includes *'Celtic Bar Brawl'*, *'Classical The Celebrated Minuet'*, *'Country Burning Trapezoid of Fire'*, *'Disco Gotta Keep on Moving'*, *'Nightlife Club Shenzhen Nightlife'*, *'Soft Rock Guitars Trip Up North'*. The first minute of each audio clip was taken and appended together to form the 6 minute clip to be played in the recording session.

2. **TV noise**: A stereo audio clip from the UK public domain film *'Dressed to Kill'* was chosen, released in 1946, sourced from the Public Domain Movie website[16] and contains British English dialogue.

3. **Activity noises**: To form the audio for activity noises, a subset of clips were selected from the ESC50 dataset, according to their applicability to smart homes: *'dog'*, *'cat'*, *'crying_baby'*, *'sneezing'*, *'keyboard_typing'*, *'door_wood_creaks'*, *'can_opening'*, *'washing_machine'*, *'vacuum_cleaner'*, *'clock_alarm'*, *'clock_tick'*, *'glass_breaking'*. 6 audio clips were randomly selected from each of these 12 categories. Each audio clip is 5 seconds long, and 72 audio clips were selected, therefore the resulting audio is 6 minutes long in accordance with the laid out requirements. The original audio IDs of the clips selected are recorded so that they can be found in the original ESC50 dataset.

The noise levels of the resulting three audio files (music, TV, activity) were adjusted to an appropriate level, typical of that in a home. To inform this process, the db-A level of each

[15] `https://freepd.com/misc.php`

[16] `https://publicdomainmovie.net/movie/dressed-to-kill-1`

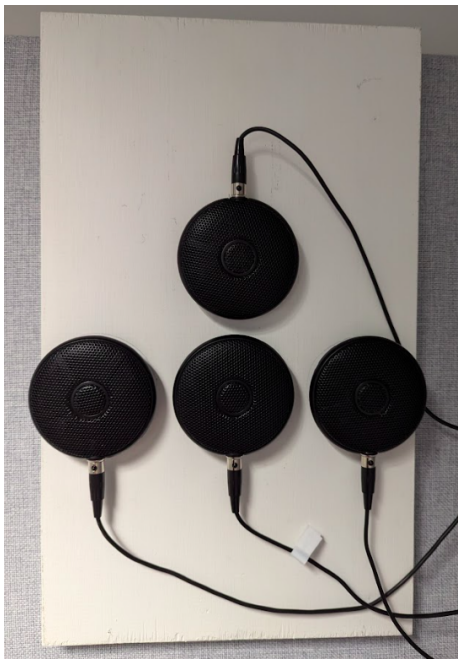

Figure 3.5: *Wall-hung microphone board setup photo.*

audio clip was measured using the Decibel X:dB Sound Level Meter app (by Thanh Dinh) from the Apple app store on an iPad Pro (3rd Gen 11inch 17.2 iOS). The resulting db-A measurements over each 6 minute audio clip are:

- **Music**: AVG 46.5 MAX 58
- **TV**: AVG 47.6 MAX 66.3
- **Activity**: AVG 39 MAX 57

A Python script is used to play the audio files sequentially in the recording session through the connected two speakers using the sounddevice[17] library.

3.4.4 Participants and Recording Procedure

Departmental ethics approval was sought to approve the study design and data management plan. Participants are limited to adults that currently reside in the UK. The recruitment process involved asking colleagues and groups within and outside of the University to participate in the study, providing them with an information sheet to detail what would happen in the study and how their data would be used. At the start of each recording session, participants were given consent forms and the opportunity to ask questions to inform their decision to participate in the study. Following agreement to participate, and the signing of the consent form, participants provided metadata, including age, gender, native language and self reported English language proficiency. Participants also generated a codeword, or participant

[17] `https://python-sounddevice.readthedocs.io/en/0.5.1/`

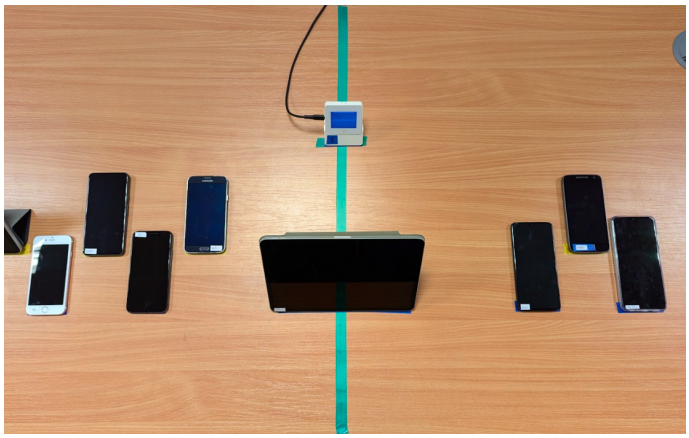

Figure 3.6: *Desk devices setup.*

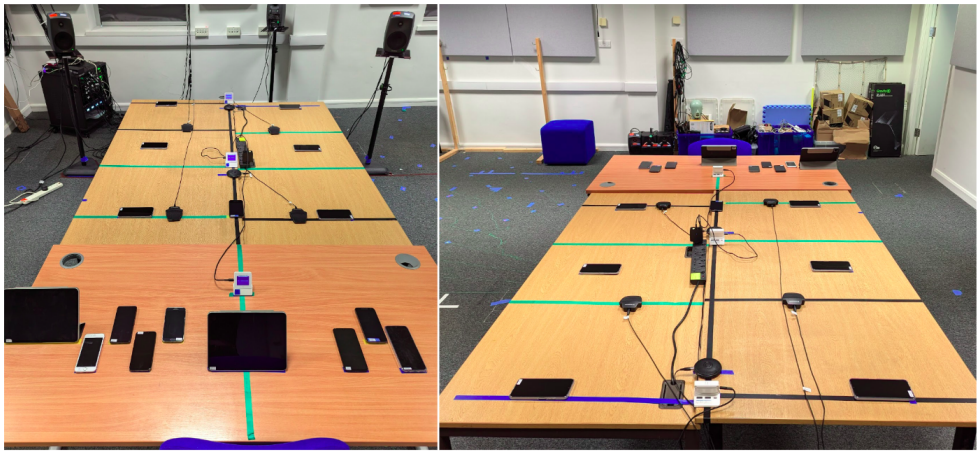

Figure 3.7: *Recording table setup.*

ID, that is created in such a way that they can recreate it to identify themselves, but no one else could identify them given the codename.

Each recording session lasted 27 minutes, with participants sitting in the chair, entering their codename on the tablet screen while awaiting the microphones to be turned on. Python scripts began the microphone and ESP-Box recordings, and each phone was manually set to record. A clap was done to align the audio, before the participant pressed start and began reading out commands displayed on the tablet screen in front of them. The total 27 minute recording session was split into blocks to capture different scenarios:

- **Clean** (no noise) - 6 minutes
- **Music** - 6 minutes
- **TV** - 6 minutes
- **Activity** - 6 minutes
- **Words** (no noise) - 3 minutes

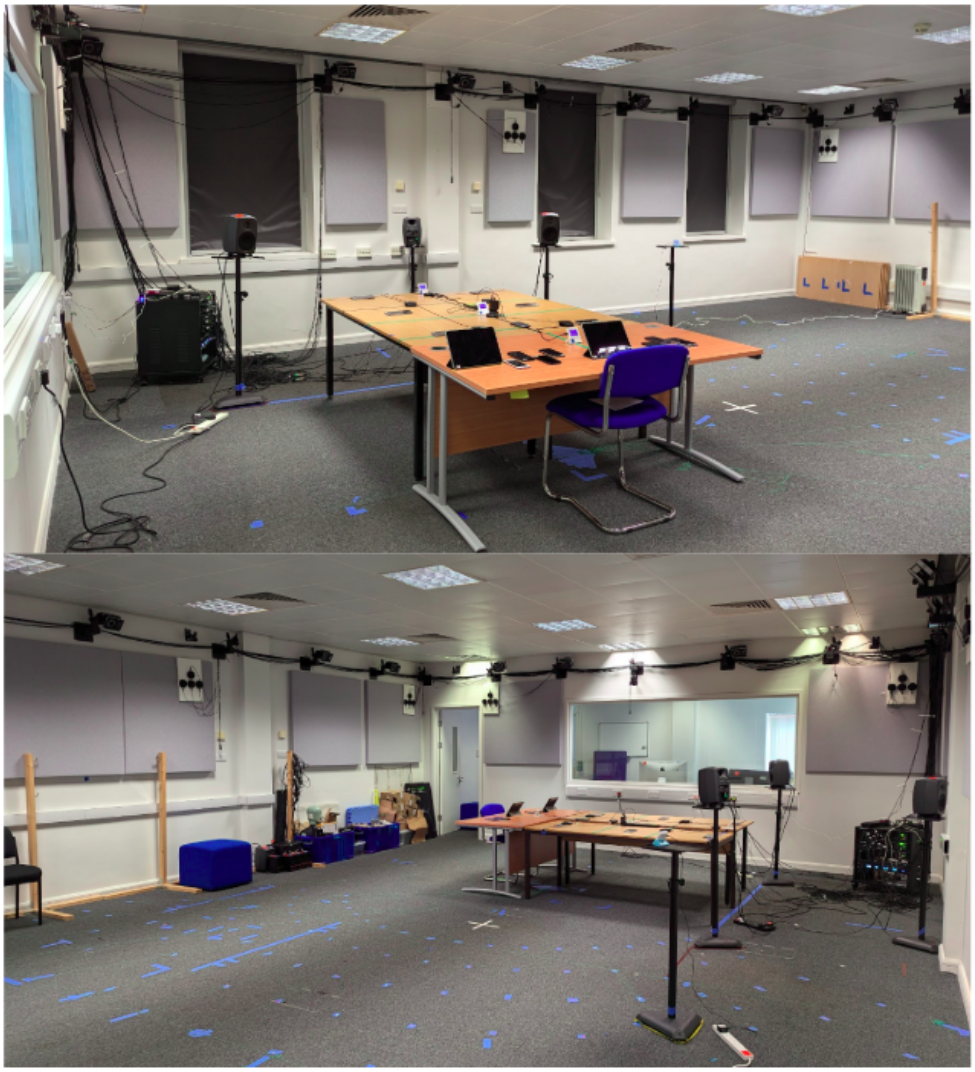

Figure 3.8: *Recording room setup.*

A beep was played to signify the next scenario, and the user then pressed 'next task' on the screen. This allowed for the relevant scenario to be mapped to the command in the transcript of the recording session. The process was monitored throughout to check all was well, in that the microphones were still recording, the participant was comfortable, the task had been changed at the appropriate time, and the speakers were functioning correctly for audio output. Figure 3.9 depicts the user interface that was displayed to the user, from which they read the commands and on which they pressed 'next task'.

The commands were generated based on the grammar created in Chapter 3.3. Wake words were randomly appended to half of the generated commands to represent various voice assistant trigger-command combinations in the dataset. The 'words' scenario consisted of a random selection of words from the grammar vocabulary set. Generated commands were written to a YAML file, along with timestamps for aligning the audio with the command transcripts in the post-processing stage. The command intent, the scenario and whether the

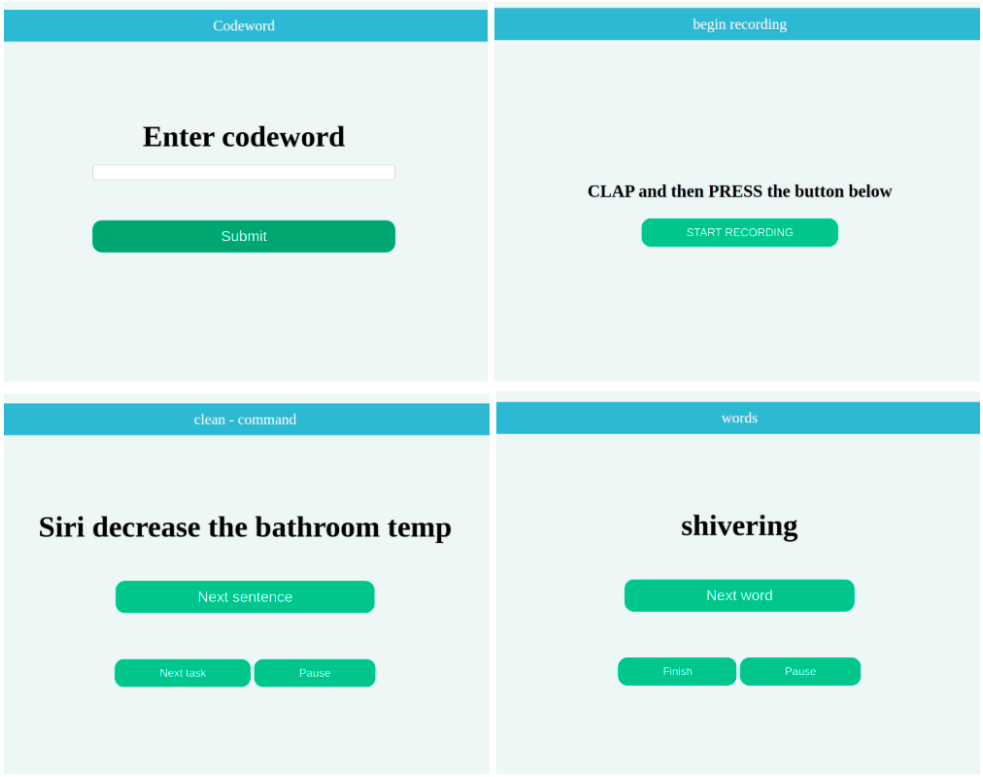

Figure 3.9: *Recording session command user interface.*

command contains a wake word were also recorded, along with the command annotation from the original grammar (which may not always be correct). These can be used as labels in the resulting dataset.

At the end of the recording session, the participant left the room and then the microphones were switched off, with the full process repeated for each participant. The post-processing techniques applied to the recorded data are detailed in the following section, where the steps taken to store, clean and organise the recordings to produce a structured, high-quality dataset ready for ASR evaluation are outlined.

3.5 Dataset Preparation and Analysis

3.5.1 Post-Processing and Data Preparation

Following the recording and collection of data, the data processing steps taken to form the dataset for ASR evaluation are explained. The process consists of storing the collected data under the participant codename, removing audio captured outside of the recording session (before the clap, and after the session is complete), and then aligning the transcripts with the audio and making corrections where the audio does not match the transcript.

The steps taken to initially store, process and annotate the audio towards forming the final dataset include:

1. **Store tracks**
    - Create [codename] folder on server.
    - Transfer audio files from ESP-Box, mobile device and microphones.
    - Transfer the YAML file detailing the transcripts.
    - Transfer text file with dB readings.
2. **Trim tracks**
    - Run Python script to identify the clap in each audio file, and remove audio beforehand to ensure consistency and alignment.
    - Manually trim audio after the last word spoken.
3. **Create Audacity files** *(version 3.6.1-alpha-20240809)*
    - Load the room microphone audio as individual tracks. (room-mics.aup3)
    - Load mobile device and ESP-Box audio as individual tracks. (device-mics.aup3)
4. **Add transcript, timestamp and dB labels**
    - Run Python script to convert the YAML data into text files that are formatted as Audacity labels (for timestamp, command transcript, scenario).
    - Trim the dB reading file so that it aligns with the clap captured in the corresponding device audio and convert it to an Audacity label file.
    - Import labels into Audacity, and listen to check time alignment, transcript and scenario label accuracy.
5. **Document recorded commands**
    - Format the participant YAML files into a single CSV to document all command prompts given to participants, and the corresponding intents and annotations (the-commands.csv).
    - Export the device-mics Audacity labels, and use these to create a CSV that documents all recorded commands, with the corresponding participant codename, transcript, scenario, dB readings, audio length, and a list of device IDs on which the commands were recorded (device-labels.csv).
    - Match the-commands.csv with the device-labels.csv (by transcript), and, where commands match, add the corresponding intent, goal type and annotations to the device-labels.csv.

Audacity is used to process the recordings given that it is free software that has longevity and stability, and works across all systems. The alignment of the audio matches for the mobile and ESP-Box devices, and the microphones in the room that are connected to the B-16 interfaces also are aligned, however, the two sets of audio do not align well. For this reason, there are two Audacity files per participant: room-mics.aup3 and device-mics.aup3.

#	b-16 (1) channels	#	b-16 (2) channels	#	b-16 (3) channels
1	sen-1	17	br-1	33	t-1
2	sen-2	18	br-2	34	t-2
3	sen-3	19	br-3	35	-
4	sen-4	20	br-4	36	-
5	fr-1	21	bl-1	37	-
6	fr-2	22	bl-2	38	-
7	fr-3	23	bl-3	39	-
8	fr-4	24	bl-4	40	-
9	frs-1	25	sl-1	41	-
10	frs-2	26	sl-2	42	-
11	frs-3	27	sl-3	43	-
12	frs-4	28	sl-4	44	-
13	sr-1	29	fl-1	45	-
14	sr-2	30	fl-2	46	close-talk
15	sr-3	31	fl-3	47	-
16	sr-4	32	fl-4	48	-

Table 3.5: *Microphone channels mapped to microphone device codes.*

The tendency for participants to sometimes misspeak or misread commands meant the transcripts were checked for such errors. This also caused the separation between the transcript correction phase, and the command annotation, given that speaking errors could cause the intent or annotation recorded to no longer align with the actual spoken transcript.

The room-mics.aup3 audio also skips on occasion, resulting in a different transcripts to device-mics.aup3 at times. The dB readings, taken at 0.2 second intervals on the iPad (IP2), are also only correct for the device microphone recordings due to the alignment issue. For this reason, device-mics.aup3 is the most clean and complete data, and the documentation of recorded commands uses this command set.

While the device-mics tracks are named with the codename given in Table 3.4, the room-mics audio tracks are named 1..N according to the device channels. Therefore, the mappings between the Audacity track name (device channel), and the microphone this represents in the recording setup is given in Table 3.5.

With the storage, alignment and documentation of the recorded audio complete, the dataset is accurate and distributable, but it is not yet in a suitable format for the ASR evaluations in Chapter 4. To address this, audio tracks are exported from Audacity and split into command-level audio files, named with an ID, and documented with the recording device name, command transcript, intent, scenario, wake word, dB level and annotation in a CSV.

The specific process taken to create the ASR evaluation dataset is detailed:

1. **Export tracks**: For each codename, export audio and label tracks from device-mics.aup3 into a [codename] folder, where each track is named with the device ID.

2. **Split audio**: Split each exported track into command segments according to the corresponding transcript label, storing each as [codename]_[deviceID]_[comamnd_ID].wav (using a Python script).

3. **Document audio**: Document each audio ID in a CSV, along with the participant codename, device ID and command ID, and the additional labels from device-labels.csv (e.g., annotations, dB levels, command length). This is called the-asr-dataset.csv.

The resulting dataset, following this segmentation and documentation process, includes the audio and documentation CSV files, for each the room microphones and device recordings:

- room-labels.csv: annotations and documentation for commands recorded for each participant on the dedicated room microphones.
- device-labels.csv: annotations and documentation for commands recorded for each participant on the mobile device microphones.
- the-asr-dataset.csv: annotations and documentation for command-level audio files, derived from the device-mics.aup3 exported tracks, and is essentially an expanded version of device-labels, designed for ASR evaluation.
- [codename] folder: stores all audio data for each participant
  - device-mics.aup3: Audacity project containing full audio and label tracks.
  - room-mics.aup3: Audacity project containing full audio and label tracks.
  - device-audio: contains the audio clips [codename]_[deviceID]_[commandID].wav
  - room-audio: contains the audio clips: [codename]_[deviceID]_[commandID].wav

Given the dataset post-processing and preparation for evaluations is complete, the following section presents a summary and analysis of the recorded dataset, with the scope and size discussed with respect to benchmarking ASR and LLM systems for smart home control.

3.5.2 Dataset Analysis and Insights

Table 3.6 gives an overview of the recorded dataset. The total number of recording sessions conducted was 34, with 34 unique participants, resulting in a total duration of 15.5 hours and around 14K recorded utterances. The vocabulary size recorded is slightly higher than that of the grammar, given some introduction of words in the mis-pronunciation of commands.

The proportion of the dataset attributed to each scenario is given in Table 3.7, alongside the associated noise variability metrics. Music is the loudest scenario, followed by TV and ambient activity noise. There is a difference of $\sim$7dB on average between the clean scenario, and the loudest music scenario, indicating suitable variation.

Participant demographic information is summarised in Table 3.8, showing a relatively even gender split, with an average age of 33 and around 20% non-native or bilingual speakers.

As mentioned, the commands recorded sometimes differ from the prompts given to the speaker, due to mis-reading, mis-pronouncing, and general speech errors and stutters. Therefore, where this occurred, and no command could be matched to the given command set, the annotations are missing. For the annotated portion of the dataset, for which over 90% of total commands have annotations, the distribution of recorded commands across command types and intents are presented in Table 3.9. There is a wide variety of commands across each category, meaning a broad selection of commands relating to each intent have been captured.

Metric	Value
Total recording sessions duration [hrs]	15.49
Total recorded utterances	13,537
Unique utterances recorded (incl. words)	11,225
Number of speakers	34
Avg utterances/speaker	398
Avg utterance length [s]	4.12
Avg utterance length [s] (exclude words)	4.89
Avg command length [num words] (exclude words)	8.17
Vocabulary Size	1,371
Number of Scenarios	4

Table 3.6: *Recorded smart home commands dataset overview.*

Scenario	Duration [hrs]	Avg [dB]	Min [dB]	Max [dB]	SNR vs Clean [dB]
Clean	3.67	40.21	24.2	72.5	0.00
Music	3.33	47.86	26.4	74.2	3.47
TV	3.37	46.85	25.5	74.0	3.04
Activity	3.39	43.52	24.7	72.9	2.24
Words	1.72	36.49	24.3	70.3	-1.51

Table 3.7: *Scenario overview.*

Table 3.10 presents a comparison between the recorded dataset, and the reviewed SLU datasets. The duration of the dataset is comparable with Fluent Speech Commands, much larger than SNIPs and Timers and Such, but much smaller than STOP. The number of speakers is lower than the other datasets, however this is primarily due to the lack of crowdsourcing of speakers, and instead running controlled recordings in one setting.

While this section analyses the dataset itself, the formal evaluation is conducted in Chapter 4 through its application to an ASR transcription task. This application-based evaluation tests whether the dataset meaningfully captures real-world challenges in voice controlled smart home environments by assessing how well current state-of-the-art ASR systems perform on it. In particular, it examines whether known challenges relating to distance, microphone variation, accents, overlapping speech, and background noise lead to measurable differences in ASR accuracy and robustness. If meaningful and significant variation is seen, then the dataset captures the challenges it set out to, and if not, these conditions were not captured adequately and the resource does not allow for the robust assessment of current ASR models.

Chapter 4 therefore evaluates and analyses both ASR system performance on the dataset and whether the dataset's variation in noise conditions, speakers, distance, devices, vocabulary, and command length exposes ASR limitations. This validates the dataset as an effective benchmark by demonstrating that it reveals performance limitations in both cloud-based and open-source ASR systems across a range of realistic smart home interaction scenarios.

Demographic	Value
Gender	16 male, 17 female, 1 non-binary
Age Range	MIN 29 MAX 59 AVG 33
Native Languages	English, Greek, Mandarin, Cantonese, Sindhi, Hungarian, Thai
Self-Reported English Proficiency	26 native, 3 bilingual, 5 advanced

Table 3.8: *Participant demographics.*

Goal		
DeviceCommand: 5313	Direct: 1790	Indirect: 2069
Intents		
PlayMusic: 290	TurnAllDeviceOff: 75	AdjustBrightness: 210
DecreaseAirQuality: 122	DevicePause: 184	DeviceUnlockAll: 146
AdjustVolume: 195	IncreaseHumidity: 143	DecreaseEnergyConsumption: 290
DeviceArm: 287	IncreaseVolume: 192	DeviceUnlock: 124
DecreaseTemperature: 255	AdjustDeviceColour: 191	ShadesClose: 146
DeviceDisarm: 288	DeviceMute: 181	DeviceUnmute: 157
SetAlarm: 257	TurnAllDeviceOn: 86	DeviceClose: 138
PlayMovie: 295	AdjustHumidity: 116	DeviceMuteAll: 129
DeviceBrew: 279	DeviceLock: 155	DeviceResume: 187
DeviceStop: 184	SetPreference: 294	IncreaseBrightness: 194
DeviceOpen: 151	AdjustAirQuality: 131	TurnDeviceOn: 94
DecreaseBrightness: 214	HideCamera: 276	DecreaseChannel: 169
DeviceUnmuteAll: 170	IncreaseChannel: 179	TurnDeviceOff: 90
DecreaseHumidity: 125	IncreaseTemperature: 227	DecreaseVolume: 204
DeviceLockAll: 161	AdjustTemperature: 248	IncreaseAirQuality: 123
ShowCamera: 289	ShadesOpen: 154	AdjustChannel: 189
DeviceStart: 188		

Table 3.9: *Intents and goals counts of the annotated recorded commands.*

3.5.3 Dataset Scope, Limitations and Future Use

The dataset is designed to capture aspects of variation that affect ASR performance in smart home environments. These include background noise conditions, speaker characteristics, recording devices, microphone distances, command length and intents. While speaker accent variation is captured, the dataset does not cover non-English speech or age-related variability such as elderly or children's speech. Extreme noise conditions beyond typical domestic environments are not included. Although overlapping speech is partially represented through background TV noise inclusion, conversational multi-speaker interactions are not present. The number of unique speakers is also relatively limited (34 participants), compared with larger-scale speech corpora. In addition, all recordings were conducted in a single room, meaning that variations in room acoustics are not captured. Some prior datasets also include dynamic speaker positions, e.g., movement during recording, which are not present here.

The dataset includes structured variation in intent and vocabulary, through generating commands based on templates defined in the command grammar. The recorded vocabulary size is approximately 1,4000 and around 11,000 unique utterances were recorded across 49

Dataset	**SNIPS**	**FSC**	**SLURP**	**STOP**	**Timers and Such**	**Smart Home Commands**
Task	SLU	SLU	SLU	SLU	SLU	ASR / NLU
Speakers	69	97	177	∼95	<89	34
Audio files	5,886	30,043	72,277	236,477	2,151	13,537
Duration [hrs]	5.5	19	58	218	2.5	15
Avg. length [s]	3.4	2.3	2.9	—	4.1	4.12

Table 3.10: *Comparison of the smart home commands dataset with existing similar datasets.*

intents relating to smart home device control. While this vocabulary is substantial for the smart home domain, it remains small relative to general-purpose ASR corpora and is therefore not intended for evaluating general purpose transcription systems. The dataset also excludes brand-specific terms and is constrained to a fixed set of smart home-related intents. As a result, it does not cover all possible phrasings or command types, such as multi-step instructions or highly device-specific configurations (e.g., fine-grained appliance settings).

The primary purpose of the dataset is evaluation rather than training. It is intended to benchmark ASR and LLM-based systems under conditions representative of smart home voice control, including ASR robustness to noise, speaker variation, and device-dependent recording conditions, as well as LLM handling of commands given diverse intents and ambiguity. As such, it provides a controlled benchmark for analysing model and system-level performance on smart home commands in a realistic but well-defined setting.

The dataset can support limited training and adaptation tasks, however. The audio data may support fine-tuning of lightweight ASR models (e.g., Whisper-small) for speaker adaptation, or to improve performance improvements on smart home commands, and the grammar and transcriptions may be used for intent classification experiments. While data augmentation techniques could be applied to extend the dataset, e.g., to simulate additional acoustic conditions, the dataset is fundamentally designed to evaluate voice-based smart home interaction. For ASR model training, substantially greater speaker diversity, vocabulary breadth, and recording variability would be required, and for which existing large-vocabulary datasets could be used. The scope and limitations reflect practical constraints in data collection and experimental design, where the dataset is intended to capture several factors affecting model performance in smart home interaction, relating to both ASR transcription and LLM reasoning, making it most suited for the evaluation of these systems.

The dataset will be made publicly available for non-commercial research use to support reproducibility and further research on voice interfaces for smart home and IoT control. In accordance with ethical approval, it will be released under the CC BY-NC 4.0 license[18], permitting sharing and adaptation provided appropriate credit is given, the licence is referenced and use remains non-commercial. Participants consented to their recordings being shared for research purposes, with each referenced using codename identifiers for anonymisation.

[18]Creative Commons Attribution-NonCommercial 4.0 International License

3.6 Summary

The chapter outlined the step-by-step process taken to design, record and prepare the dataset of spoken smart home commands in order to evaluate ASR systems under realistic conditions, and quantify the performance gap between cloud-based and locally run models.

A review of existing speech and SLU datasets highlighted the lack of data relating to smart home device scenarios, but identified useful dataset design approaches to ensure a range of intents, vocabularies and scenarios are represented in the recorded data. A review of command grammars, specifically, analysed existing frameworks for defining smart home commands, and identified the VISH grammar as useful for our task, which was then adapted to produce fluent spoken commands across a variety of intents, with diverse phrasing and vocabulary.

The recording methodology section outlined the hardware, environmental setup, and scenario design used to ensure data is recorded under diverse conditions that simulate real-world smart home use cases. The recording processes detailed how the data was collected, and the outlined post-processing steps described how the raw collected data was transformed into a structured dataset for ASR evaluation. Analysis of the data highlights the dataset size, with varied participant demographics, command diversity and noise.

The resulting dataset is intended to enable the robust evaluation of ASR systems in the context of smart home control and be representative of real-world smart home interactions. The next chapter details how the dataset is applied to answer the research question: *How does the performance of locally run ASR models compare to cloud-based solutions for smart home commands?* During this analysis the dataset aims will also be evaluated in terms of whether the challenges of ASR in smart homes are adequately captured and represented.

Chapter 4

Producing Transcriptions Locally: Exploring ASR for Smart Homes

4.1 Introduction: Automatic Speech Recognition

Automatic Speech Recognition (ASR) systems are fundamental to enable voice control of smart home devices and allow for this natural and hands-free interaction with the environment. Cloud-based ASR services have been preferred due to their heightened performance in terms of accuracy and speed that comes with vast computational power and training on massive datasets. However, the reliance on the cloud is associated with privacy concerns and depends on internet connectivity.

Relatively recent advancements in ASR models are beginning to close the gap in accuracy between local and cloud-based solutions. Highlighted in Chapter 2, models like OpenAI's Whisper can run on modern GPU hardware without the need for any internet connection, offering a promising alternative. The strengths and limitations of such local ASR technologies are investigated in relation to the task of smart home command recognition.

Using the dataset developed in Chapter 3, this chapter investigates how local ASR systems perform in comparison to cloud-based solutions within smart home environments. The dataset was designed specifically to support robust ASR evaluation, incorporating a range of command types and recording conditions. Here, an assessment of whether those dataset aims have been achieved is given through evaluating ASR performance across several aspects: command types, hardware diversity, accent variability, microphone distance, noise types.

Accurate speech recognition is necessary for the usability of voice assistants, where systems must handle brief spoken commands in the presence of ambient noise, varying accents and at various distances. The dataset includes realistic smart home scenarios involving household noise, multiple recording devices and varied speaker to microphone distances. The chapter presents a comparative evaluation of local and cloud-based ASR models, analysing transcription accuracy across these conditions, and providing insights into their suitability for local, smart home voice control systems.

Beginning with an overview of ASR systems, the technical advancement in architectures over time are highlighted, along with the relative performance and limitations of current ASR systems. Models to include in the evaluation are selected, and the benchmarking methodology and metrics are presented. The analysis compares ASR performance on the smart home

commands dataset, discussing model robustness to variation, before outlining methods that could potentially improve performance.

4.2 ASR Technology and Models

4.2.1 ASR Technology

ASR systems have evolved over the technological advancements made in the past decades. Early ASR systems used statistical models, such as Hidden Markov Models (HMMs) combined with Gaussian Mixture Models (GMM), to form HMM-GMM systems that were somewhat effective but very limited. With deep learning, Recurrent Neural Networks (RNNs) became popular, with models like DeepSpeech (Hannun et al., 2014) adopting this architecture to enable more accurate speech transcription. Modern architectures are transformer-based, with transformers able to overcome several challenges faced by RNN-based models, such as computational inefficiencies. Widely used transformer models include Wav2Vec2 (Baevski et al., 2020) and Whisper (Radford et al., 2022) that use self-attention mechanisms to efficiently process audio. Not only has the improvement in model architectures and training processes brought about these advancements, but the production of massive transcribed datasets and progress in GPUs (Hannun, 2021).

Despite the advancements and widespread adoption of ASR technologies, challenges remain, including robustness to linguistic and speaker diversity, interference from background noise, and the handling of domain specific terminology, where research is ongoing (Shrivastava et al., 2023). A review highlights the impact of transformers in improving the performance and efficiency of ASR models, alongside the challenges that remain for linguistic diversity, noise robustness in very noisy environments, and the need for real-world testing and dataset development for these systems to be effectively adapted and applied in healthcare, customer service or industrial environments (Sharrab et al., 2025). The Chapter 2 review also highlights challenges faced by local ASR technologies in real-world device control settings.

With the existing ASR research challenges in mind, the specific challenges relevant to the smart home command recognition task are outlined:

- **Background noise**: Smart homes are not controlled environments. Noise from appliances, activities, televisions, music, or conversations can interfere with ASR accuracy.
- **Hardware variability**: Smart home devices vary in terms of microphone quality, causing inconsistencies in the audio input.
- **Distance variability**: Smart home devices vary in terms of the positioning and placement. Users may issue commands from distances that may challenge ASR models.
- **Speaker variability**: Smart homes have multiple users. Differences in accents, speaking styles and clarity among users can be challenging for ASR systems.
- **Overlapping speech**: Multiple people speaking simultaneously can create challenges, where the model needs to identify and process a single command issued by a single user.

Addressing these challenges is important when benchmarking ASR system performance. The evaluations will measure performance robustness across a range of smart home scenarios,

not just for the clean and ideal use case. The dataset design reflects challenges of accent, distance, noise and hardware variability, and overlapping speech, to identify the relative performance, strengths and limitations of cloud-based ASR services and local ASR models. This enables their suitability to real-world smart home command transcription tasks to be assessed. The next section introduces available ASR models to inform the model selection.

4.2.2 ASR Models

Several pretrained ASR models available for inference and evaluation on the smart home command dataset are introduced and detailed in terms of features and architectures. Options for local speech transcription presented in Chapter 2 are focused on in further detail. The models detailed reflect the shift in architectures over the years, with inclusion of models that can run locally on CPU up to GPU hardware, alongside cloud-based transcription services available to compare local ASR performance with. The analysis informs the model selection for the ASR benchmarking task.

Commonly seen CPU-based ASR models:

- **PocketSphinx**[1]: An outdated ASR model, seen regularly in older literature on smart home voice control (see Chapter 2), that uses the HMM-GMM architecture, and therefore achieves low accuracy compared with newer models, particularly with regard to complex commands and noise.
- **VOSK**[2]: Built with Kaldi, uses HMM-DNN hybrid architecture, and is designed for resource-constrained environments, like embedded systems. This model is also fairly old, and performance is significantly lower than transformer-architectures, with similar limitations to PocketSphinx.
- **Whisper (small)**[3]: Transformer-based model, released by OpenAI, that significantly outperforms older models. It is capable of running on CPU-equipped devices such as the Raspberry Pi. This has much improved robustness to noise and speaker variability.

Popular GPU-based ASR models:

- **DeepSpeech**[4]: RNN-based pretrained model, released by Mozilla, that is less resource-intensive, but lower performance compared to transformer models.
- **Wav2Vec2**[5]: Transformer-based pretrained model, released by Meta, trained on extensive datasets, with high performance and robustness, and widely adopted.
- **Whisper (large)**[6]: Transformer-based model released by OpenAI, achieving high performance on ASR tasks. Whisper ASR supports prompting, where context can be provided to the model to better recognise speech, e.g., a spelling guide can give the names of people, products, etc., but these techniques are not deemed entirely reliable.

[1] `https://github.com/cmusphinx/pocketsphinx`
[2] `https://alphacephei.com/vosk/`
[3] `https://huggingface.co/openai/whisper-small`
[4] `https://github.com/mozilla/DeepSpeech`
[5] `https://huggingface.co/docs/transformers/en/model_doc/wav2vec2`
[6] `https://huggingface.co/openai/whisper-large-v3`

Cloud-based transcription services to compare with local ASR performance:

- **Amazon Transcribe**[7]: Paid cloud-based transcription service with widespread adoption. Supports real-time and batch transcription, but the network dependence introduces latency, the cost can be high, and there are privacy concerns regarding the remote data storage.
- **Google Cloud Speech-to-Text**[8]: Paid cloud-based transcription with performance and capabilities generally comparable to Amazon Transcribe, and limitations similar to Amazon Transcribe (latency, cost and privacy).

4.2.3 Model Selection

The goal of the evaluation is to quantify the performance gap between local and cloud-based systems, across noise conditions, distances, hardware, and speakers. The review of ASR models informs the selection of models to include in the evaluation. The selection criteria included that local models must be open-source, run on consumer-grade hardware, and use transformer-architectures, given this transition indicates the most recent iteration of ASR systems. The cloud-based models were selected from the most well-known, and widely used transcription services, that are consequently offered by the companies whose home voice assistants are in the scope to replicate. The models chosen for evaluation therefore include: Whisper-small, Whisper-large, Amazon Transcribe, Google Cloud Speech-to-Text (STT). A comparative overview of the selected ASR models is outlined in Table 4.1, where the key features of each are summarised.

This combination allows for the evaluation of locally run transformer models against commercial cloud transcription services, highlighting differences in accuracy, robustness and latency. The GPU-based Whisper model represents a high-resource, local benchmark, with the CPU-based Whisper-small selected to assess the feasibility of a lower resource deployment more representative of smart home devices. Although it is expected that Whisper-large will perform better in terms of accuracy, it is of interest to evaluate and compare this with the smaller model that has significantly reduced resource requirements to enable a more cost accessible deployment for home use cases, and may have better transcription speed. The performance gap between the two models is therefore compared, identifying the strengths and limitations of each, and how these compare to the cloud-based ASR systems.

While it is possible to fine-tune these models with the smart home command data collected, and the Whisper models also allow prompting, the benchmarking evaluations focus on the baseline performance of these ASR models with no adaptation. The experimental setup is detailed in the following section, further detailing model configuration, as well as the data preparation steps, and metrics used for benchmarking the ASR models on the smart home commands dataset.

[7] `https://aws.amazon.com/pm/transcribe/`
[8] `https://cloud.google.com/speech-to-text`

Model	Architecture	Hardware resource	Real-time processing	Fine-tuning	Strengths	Limitation
Whisper (Small)	Transformer	CPU	Limited	Yes	Lightweight, prompt support	Lower accuracy
Whisper (Large)	Transformer	GPU	Yes	Yes	High accuracy, prompt support	Higher resource
Amazon Transcribe	Proprietary	Cloud	Yes	No	Robustness, ease of use	Paid, no fine-tuning
Google STT	Proprietary	Cloud	Yes	No	Robustness, ease of use	Paid, no fine-tuning

Table 4.1: *Comparison of selected ASR models.*

4.3 Experimental Setup

4.3.1 Dataset Preparation

This dataset collected in Chapter 3 is prepared and applied to the ASR evaluation task. The segmented audio recordings of individual smart home commands, spoken by 34 participants, across varying noise scenarios and recorded on varying devices are used, and a CSV file centrally documents each audio segment, with an ID and details of the speaker, device, and transcript. The inputs and outputs for the benchmarking task therefore consist of audio segments and a CSV file:

- **Audio segments**: The audio command segments are used to feed into the ASR models, where audio tracks have been split into distinct utterances that correspond to individual command transcriptions. The segments are predominantly recorded at 48kHz, but for the ASR evaluation, audio is downsampled to 16kHz.
- **Documentation**: A CSV file details all audio segments, including the segment ID, speaker ID, device ID, scenario and transcript. The transcribed output of each audio segment, given each ASR model, is appended to the CSV during the inference process.

It is common to split data into training, validation and test sets when experimenting with training and fine-tuning approaches. However, for this task all of the data is used to evaluate ASR systems on all of the collected data.

The dataset structure organises the audio segments into appropriate folders according to each participant ID (or codename). The data used for the ASR evaluations looks like:

- the-asr-dataset.csv → for documenting audio clips and transcriptions
- codename folder → stores segmented and downsampled audio clips
  - device-audio-16kHz: [codename]_[deviceID]_[commandID].wav (naming system)
  - room-audio-16kHz: [codename]_[deviceID]_[commandID].wav (naming system)

The dataset is structured in a way to ensure easy and efficient file access for the transcription and evaluation processes. While the room audio is available, the ASR evaluation

uses only the device audio, as this part of the dataset best represents the types of devices found in smart homes. The CSV file (the-asr-dataset.csv) details all information relating to each individual audio file collected on the mobile device microphones, and is where the ASR inference results (transcription and latency) will be stored pending analysis.

4.3.2 Model Configuration

The technical setup used to run and evaluate the selected models is detailed, with information on the hardware and software setup and model configuration. The hardware setup includes a standard desktop machine equipped with an Nvidia GPU, enabling the evaluation of the lightweight Whisper-small model, and the resource intensive Whisper-large model. Cloud-based services, like Amazon Transcribe and Google Cloud STT, are accessed via APIs. The inference and evaluation takes place in a Python virtual environment, and uses libraries like transformers, soundfile, torch and whisper for audio file processing and model inference.

Hardware environment:

- **CPU**: AMD Ryzen 7 7700X 8-Core Processor, 2 threads per core (16 total threads)
- **Memory (RAM)**: 32GB. DDR5, 4800 MT/s
- **GPU**: NVIDIA GeForce RTX 4090 with 24GB VRAM
- **Storage**: 1 TB NVMe SSD
- **OS**: Ubuntu 22.04.5 LTS (jammy)

Model configuration:

- **Whisper-small**: Local, per-clip transcription (non-streaming), no prompting applied.
- **Whisper-large**: Local, per-clip transcription (non-streaming), no prompting applied.
- **Amazon Transcribe**: Cloud-based batch audio transcription using the standard model.
- **Google Cloud Speech to Text**: Cloud-based per-clip transcription using latest-long model.

The model configuration is defined in the Python scripts used to interface with each selected model to obtain the output transcript and latency. The standard model configuration is used for each ASR system. For example, Whisper-large and Whisper-small have prompting options, but these methods are not used in the evaluations. Google Cloud STT provides options for models to use for transcription. When experimenting with latest-short, designed for short audio utterances of a few seconds, a large proportion of commands returned no transcript, therefore latest-long is the model selected for the Google STT transcriptions. Amazon transcribe provides a standard model, but, unlike Google STT, file upload is very slow. For this reason, all audio files are uploaded to Amazon S3 buckets before the transcription process begins. Real-time streaming options are available for all models, but are not used in these evaluations. For the purpose of this evaluation, the focus is purely on quantifying local- and

cloud-based ASR model accuracy given smart home commands recorded in realistic and challenging environments. Real-time transcription metrics are considered and reported in Chapter 6, where the end-to-end system performance is evaluated and therefore latency considerations become more important.

4.3.3 Evaluation Methodology

This section details the evaluation methodology, including the metrics used to assess ASR model performance on the smart home commands dataset. The methodology is as follows:

1. Transcribe the audio files
   - Transcribe each audio file individually by providing it to the model, and awaiting the transcription result.
   - Record the transcription result and latency in the CSV file that documents the audio clips.
2. Compute metrics on output transcripts
   - Compute accuracy metrics by comparing (normalised) model output transcripts with the ground truth.
   - Compute average latency

After transcription is complete, and before the comparison of each ASR model transcription with the manually verified ground truth transcript, text normalisation is applied to the model transcript for consistency and fairness. These include lowercase conversion, punctuation removal, converting written numbers to numerical form, and mapping word variations like ‘home like’ to ‘homelike’. Python scripts apply the text normalisation (temporarily) to both the ground truth transcript and the model output transcript. Evaluation metrics are then computed by comparing the model output transcripts and the ground truth for each command. The following evaluation metrics are computed to allow for a detailed comparison of model performance:

- Word Error Rate (WER)
- Character Error Rate (CER)
- Match Error Rate (MER)
- Word Information Loss (WIL)
- Substitutions, insertions, deletions counts

Metrics are calculated using the Python package jiwer[9], that supports calculation of CER, MER, WIL, WER, substitutions, insertions, deletions, and provides visualisations of the calculations for each sentence. Examples of how the metrics are derived from comparing the ground truth transcript with the model output transcript are given in Figure 4.1.

---

[9] `https://github.com/jitsi/jiwer`

```
Word-Level Alignment:
REF: **** ** *********** ******** push up  the smart blind inside please
HYP: what an unfortunate accident push up this   mob blind inside please
        I  I           I        I            S     S

number of sentences: 1
substitutions=2 deletions=0 insertions=4 hits=5

mer=54.55%
wil=67.53%
wip=32.47%
wer=85.71%

Character-Level Alignment:
REF: ***************************push up th*e smart blind inside please
HYP: what an unfortunate accident push up this *mob* blind inside please
     IIIIIIIIIIIIIIIIIIIIIIIIIIII          IS D SSD

number of sentences: 1
substitutions=3 deletions=2 insertions=30 hits=32

cer=94.59%
```

Figure 4.1: *Visualisation of ASR metric calculation.*

4.4 Results and Discussion

4.4.1 Model Performance Comparison

Overall Performance

The performance results of the selected ASR models are presented and analysed in this section. Transcription accuracy across different scenarios and hardware configurations are compared, with specific observations and trends highlighted, towards identifying the strengths and limitations of each model for smart home command recognition. The overall performance of each ASR model is first compared across the audio collected on the ESP-Box device placed closest to the speaker in the recording setup (ESP1), with results given in Table 4.2.

Whisper-large performs the best overall (6.5% WER), with Amazon transcribe closely behind (6.6% WER). Google STT is worst performing (14.05% WER), with a number of audio clips returning 'no transcription result', but even without this, transcription accuracy is generally lower than other models. Whisper-small achieves 9.93% WER overall, outperforming Google STT by more than 4%. Commands are more accurately transcribed than individual words across all models, perhaps given the context of commands can be used to disambiguate similar sounding words. Amazon transcribe performs best on individual word transcription, but still WER is 24.65%.

CER is a relatively generous metric, consistently lower than WER, indicating portions of words are captured, where the word itself is not correct. For example, overall CER for Whisper-large is 3.79% compared with 6.5% WER. MER is very similar to WER (6.37% for Whisper-large), while WIL is a harsher metric (9.83% for Whisper-large), highly sensitive to insertions and consistently higher than WER. WIL is also particularly high for the individual word transcription, with 38.98% for Amazon Transcribe.

Table 4.3 details the substitution, insertion and deletion counts for each model comparative to the WER. This shows how common these errors are relative to each other, and how they

Model	Overall				Commands			Words		
	WER	CER	MER	WIL	WER	CER	WIL	WER	CER	WIL
wh-sml	9.93	5.75	9.62	15.05	8.80	5.19	13.31	36.88	17.83	49.78
wh-lrg	6.50	3.79	6.37	9.83	5.57	3.34	8.30	28.80	13.13	41.74
google	14.05	8.73	13.37	20.09	13.21	8.45	18.62	34.23	14.64	52.02
aws	6.60	3.83	6.47	10.08	5.85	3.56	8.74	24.65	9.66	38.98

Table 4.2: *WER, CER, MER, and WIL metrics in % for ESP1 (ESP-Box closest to speaker).*

Model	Overall				Commands				Words			
	WER	SUB	INS	DEL	WER	SUB	INS	DEL	WER	SUB	INS	DEL
wh-sml	9.94	4807	2578	667	8.80	3995	2197	660	36.88	812	381	7
wh-lrg	6.50	2969	1724	581	5.57	2303	1464	570	28.80	666	260	11
google	14.05	6163	4149	1083	13.21	5291	4051	939	34.23	872	98	144
aws	6.60	3100	1680	573	5.85	2451	1544	556	24.65	649	136	17

Table 4.3: *WER, substitution, insertion, and deletion counts for ESP1 (ESP-Box closest to speaker).*

correspond to the WER metric. Substitutions are the most common error type, where words are mistaken, with insertions also common and deletions the least common. It can be seen that substitutions account for 2303 errors in Whisper-large command transcriptions, with insertions accounting for 1464 errors. Extra words could be inserted due to background noise, or multi-syllable words being transcribed as individual words. Deletions are a much rarer error, accounting for 570 errors on command transcription. and only 11 errors on word transcriptions for Whisper-large, with similar counts for Amazon transcribe. Deletions are highest for Google STT given the 'no transcription result' tendency, with overall 1083 errors of this type. Transcription error types are further explored and exemplified in Chapter 4.4.2.

Device Analysis

To analyse the effect of microphone quality on recognition accuracy, model performance across audio collected from various mobile devices is compared. These devices were all placed on the table directly in front of the person speaking in the recording setup, therefore distance is very similar, but the recording device quality varies. Table 4.4 indicates how each model performs given the varying recording qualities, and Figure 4.2 depicts these figures.

There is ∼4% WER difference between the best and worst performing device for the Whisper-large model, where the iPhone 6s sees the best performance (5.34% WER overall), and the Samsung Galaxy 5 performs worst (9.18% WER overall). Apple devices generally see the best ASR performance, regardless of the age of the device, with the iPhone 6s and Samsung Galaxy 5 released within only a year of each other (2015 and 2014, respectively), yet achieving the best and worst performance for Whisper-large. Performance is worst for older Android phones, including Moto G4 play (mgp6, 2016), and newer Android phones sit in the middle, e.g., OnePlus 6t (op6t, 2018) and Nokia G22 (ng22f, 2023). A similar trend is observed for Whisper-small, although performance degrades more significantly for the older Android phones, indicating a lesser robustness to audio quality. Results clearly indicate performance varies by recording device, which may be due to information loss in low quality audio or a

cmd	ix			ip1			i6s			ng22f		
	ALL	CMD	WRD	ALL	CMD	WRD	ALL	CMD	WRD	ALL	CMD	WRD
wh-sml	6.69	5.77	28.55	6.79	5.57	36.03	7.04	6.06	30.77	7.02	6.11	29.01
wh-lrg	5.44	4.73	22.56	5.37	4.51	26.13	5.34	4.58	23.43	5.41	4.64	23.71

cmd	gp8			op6t			mgp6			sg5		
	ALL	CMD	WRD	ALL	CMD	WRD	ALL	CMD	WRD	ALL	CMD	WRD
wh-sml	8.14	7.10	33.06	8.34	7.17	36.42	10.35	8.84	46.59	12.40	10.76	51.98
wh-lrg	6.33	5.48	26.90	5.86	5.07	24.87	6.86	5.62	36.62	9.18	7.92	39.72

Table 4.4: *Microphone effect on WER for Whisper-large and Whisper-small.*

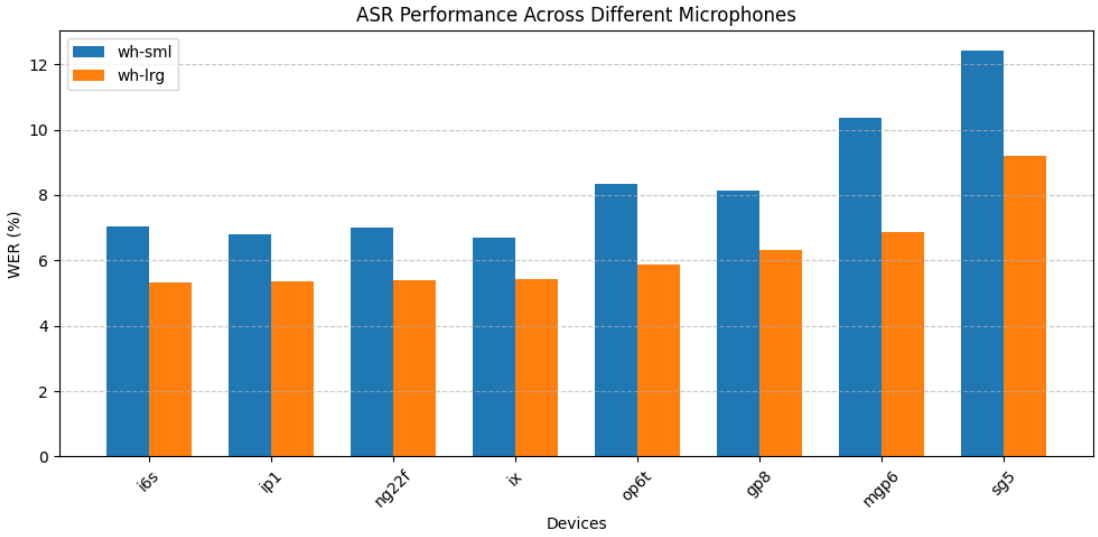

Figure 4.2: *Microphone effect on WER for Whisper-large and Whisper-small.*

lack of robustness of ASR models given lower quality audio.

The performance of the ESP-Box device is most comparable with the Google Pixel 8 (gp8) Android smartphone, with Whisper-large achieving a 6.5% overall WER on the ESP-Box audio and 6.33% WER on Google Pixel audio. While not the optimal microphone hardware in this comparison, the device is programmable, low-cost, with reasonable performance, and will therefore be used as the audio front-end in the voice assistant setup.

Distance Analysis

To measure the effect of microphone distance on recognition accuracy across the four models, the audio recorded on the ESP-Box devices is used. Distance was set to 50cm, 100cm, 250cm, 450cm and 600cm away from the speakers position for ESP1..5 respectively, with the resulting performance of each given in Table 4.5. There is a consistent and clear trend showing error rates increase across models as the microphone distance increases, as depicted in Figure 4.3, with high deletion rates observed that contribute to this. Overall WER rises from 6.5% at 50cm up to 28.44% at 6m for Whisper-large, with Whisper-small ranging more significantly from 9.93% up to 41.34%, again indicating its comparatively limited robustness.

Distance	Model	ALL	CMD	WRD
50cm	whisper-small	9.93	8.80	36.88
	whisper-large	**6.50**	**5.57**	**28.80**
	google stt	14.05	13.21	34.23
100cm	whisper-small	18.77	17.80	42.13
	whisper-large	**12.60**	**11.83**	**30.98**
	google stt	23.37	22.81	36.94
250cm	whisper-small	24.45	23.62	44.19
	whisper-large	**17.42**	**16.41**	**41.55**
	google stt	27.22	26.75	38.54
450cm	whisper-small	33.22	31.30	79.62
	whisper-large	**21.15**	**19.27**	66.68
	google stt	31.19	30.16	**56.12**
600cm	whisper-small	41.34	38.96	98.31
	whisper-large	**28.44**	**26.07**	85.16
	google stt	35.87	34.80	**61.52**

Table 4.5: *Distance effect on WER (%) for distributed ESP-Box-3 devices. Bold values indicate lowest WER per distance.*

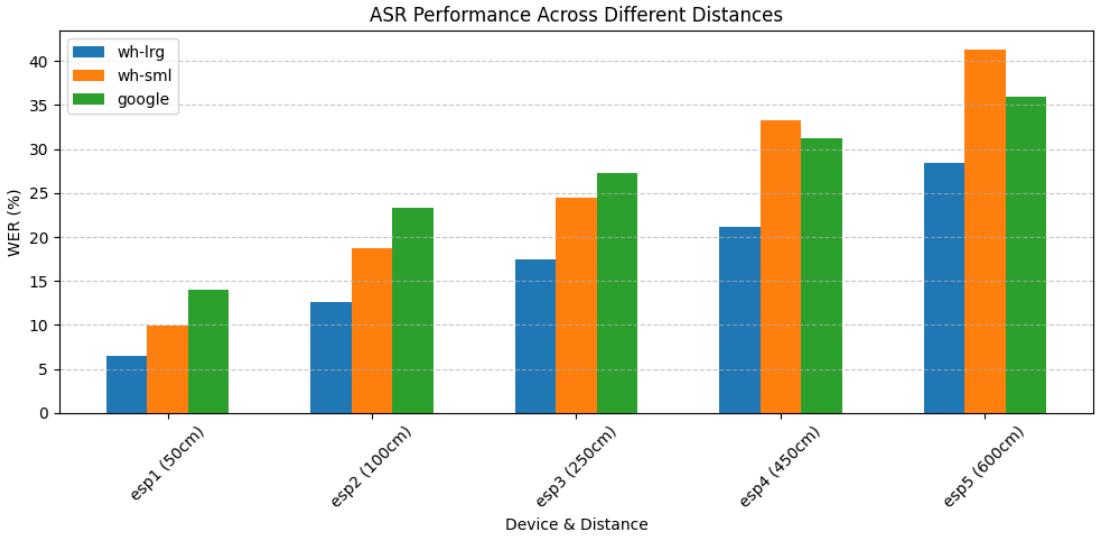

Figure 4.3: *Effect of distance on WER for distributed ESP-Box-3 devices.*

For distances over 250cm, the ASR system becomes less usable, with the WER rising to over 15% for Whisper-large, and over 20% for Whisper-small and Google Transcribe, however this distance is not a typical use case. While all models see several percent drops in accuracy as distance increases, Whisper-large performs best at an extreme distance overall. Individual word recognition is particularly affected by distance, where at 6m WER is 98% and 85% for Whisper-small and -large models respectively. It is only in the case where distance rises to 4.5m where Google STT outperforms Whisper-large, with a 60% WER.

Scenario	Model	WER	WIL	CER
Clean	whisper-small	4.34	7.28	1.95
	whisper-large	3.02	5.01	1.31
	google stt	7.27	12.07	3.37
	aws transcribe	**2.97**	4.97	1.32
Music	whisper-small	7.26	12.15	3.60
	whisper-large	**4.08**	6.84	1.88
	google stt	9.16	15.25	4.36
	aws transcribe	4.15	6.98	1.94
Activity	whisper-small	6.84	11.26	3.32
	whisper-large	**4.20**	6.95	2.09
	google stt	9.25	15.35	4.43
	aws transcribe	4.60	7.62	2.16
TV	whisper-small	16.78	21.92	11.87
	whisper-large	**10.97**	14.05	8.10
	google stt	27.11	29.95	21.57
	aws transcribe	11.64	15.04	8.79

Table 4.6: *Effect of scenario noise on performance (WER %). Bold values indicate lowest WER per scenario.*

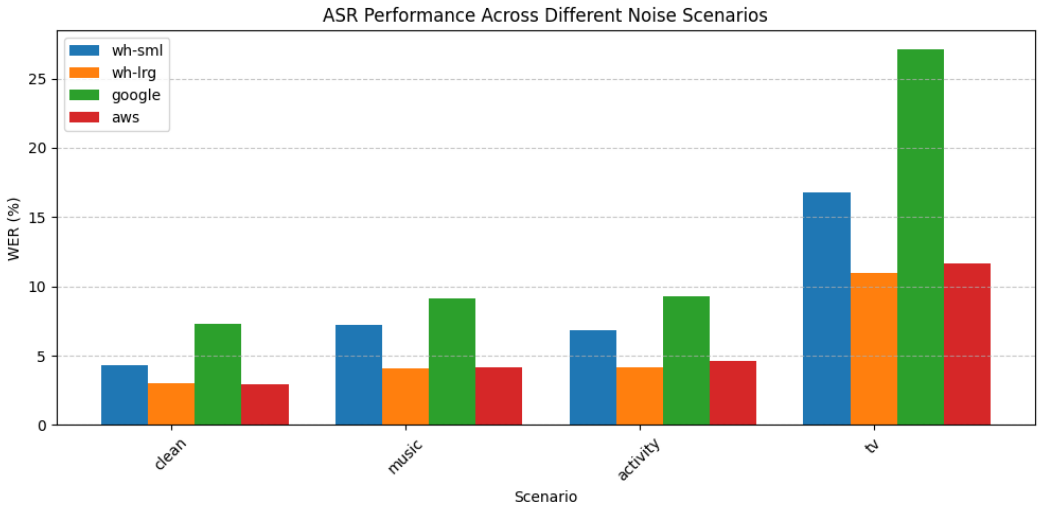

Figure 4.4: *Effect of scenario noise on WER across models.*

Noise Condition Analysis

Noise can interfere with speech recognition quality. Given the same recording device (ESP1), the performance of the ASR models in the 4 different scenarios (clean, music, activity, tv noise) are compared. Table 4.6 highlights the results of this analysis, showing how ASR performance is impacted by each noise type, compared with the clean scenario. Figure 4.4 depicts these metrics. It can be seen that Whisper-large and Amazon Transcribe perform similarly, both fairly unaffected by background music and activity noise with WER remaining around 4%, only 1% higher than the clean scenario. TV noise significantly degrades model performance however, where WER rises to around 11% for Whisper-large and Amazon transcribe. Google

Scenario	Model	SUB	INS	DEL
Clean	whisper-small	609	141	107
	whisper-large	404	73	120
	google stt	1025	214	196
	amazon transcribe	409	119	59
Music	whisper-small	990	221	142
	whisper-large	535	93	133
	google stt	1255	243	210
	amazon transcribe	554	147	72
Activity	whisper-small	952	247	164
	whisper-large	571	114	151
	google stt	1342	243	256
	amazon transcribe	634	161	121
TV	whisper-small	1444	1588	247
	whisper-large	793	1184	166
	google stt	1669	3351	277
	amazon transcribe	854	1117	304

Table 4.7: *Effect of scenario noise on substitutions, insertions, and deletions for each model.*

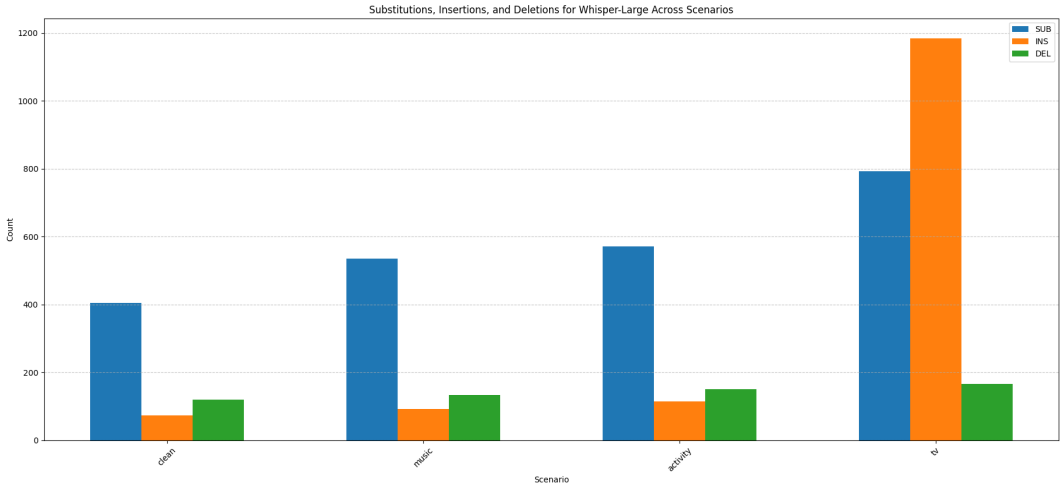

Figure 4.5: *Effect of scenario noise on substitutions, insertions and deletions for Whisper-large.*

is most affected by TV noise, with a 20% increase in WER from the clean scenario. Whisper-small is also less robust to noise compared with the best performing models, with a WER increase of 2-3% for activity and music noise above the clean scenario, and 12% for TV noise. This follows observed trends that Whisper-large and Amazon transcribe provide improved performance and robustness over other models.

The substitutions, insertions and deletion numbers behind the metrics are additionally inspected (see Table 4.7). It can be seen in Figure 4.5 that insertion rates dramatically increase in the presence of TV noise, and are particularly noticeable given insertion errors

are relatively low for all other scenarios. For example, insertion errors rise from 73 in the clean scenario to 1184 when TV audio is played for Whisper-large, with a similar increase for Amazon transcribe. Deletion errors remain low across scenarios, while substitution errors are most common across all scenarios, except for TV noise where insertion rates surpass this. Deletion counts remaining fairly low for TV noise suggests the command is most likely often correctly transcribed but with extra words captured from the TV chatter, and inspecting transcriptions shows this to often be the case.

The increase in insertions for TV noise aligns with intuition given that there is no specific speaker identification used, therefore all captured speech is transcribed. It could therefore be worthwhile incorporating speaker identification to improve ASR performance, enabling the system to focus on transcribing the voice that triggered the wake activation. However, it may also be possible for an LLM to disambiguate the user's intended command from TV chatter or background conversation, providing the user's speech is audible in noisy conditions.

Speaker Variability

ASR performance varies between speakers, given differing accents and pronunciations. Model performance for individual speakers in the dataset is therefore compared, keeping the recording device consistent (ESP1) to evaluate how speaker characteristics affect command transcription accuracy. Table 4.8 details the WER and brief demographic information of the ten best and worst performing speakers for each model. There is significant variability in WER between speakers, with a ∼17-18% range for Whisper-large and Amazon transcribe, and a ∼25% range for Whisper-small. WER can be less than 2% for some speakers, while almost 20% for other speakers given the best performing models (Whisper-large and Amazon transcribe). To better visualise this information, Figure 4.6 depicts the WER distribution for each model, showing how performance varies across all speakers. It follows the trend already seen in analysing variation to microphone, distance and noise, that Whisper-large offers enhanced robustness over Whisper-small, and that Amazon transcribe performs very similarly to Whisper-large.

Demographic factors that could underlie this variation are investigated. Depicted in Figure 4.7, English fluency, i.e. whether a speaker is native or not, has a significant impact across all models. Amazon handles non-native English accents best, with a ∼12.5% median for non-native speakers, while Whisper-large has a ∼10% greater median WER for non-native speakers compared with native speakers. Google appears most affected by non-native English speakers, with a very high median WER at ∼26%. It is noted that the non-native speakers have a small sample size, limiting this analysis, yet there is still a clear trend.

Examining the effect of gender on WER shows female gender appears to correlate with a slightly higher WER for Whisper-small and Google STT, but there is no trend for Whisper-large or Amazon transcribe (see Figure 4.8). While there is an equal gender split in the dataset, there is an unbalanced number of non-native speakers (4 women, 1 male), therefore this analysis considers native speakers only. Another demographic factor includes the age of participants, however this is not considered because initial analysis showed there to be no trend in this, or rather that perhaps the dataset does not capture adequate age range data for any trend to be identified.

#	whisper-small			whisper-large			amazon transcribe		
	code	demog	wer	code	demog	wer	code	demog	wer
1	er7ey20	5/F/N	3.02	er7ey20	5/F/N	1.77	er7ey20	5/F/N	1.99
2	on04ll18	2/F/N	3.2	on04ll18	2/F/N	1.93	on04ll18	2/F/N	2.35
3	am6el6	5/F/N	3.37	am6el6	5/F/N	2.31	am6el6	5/F/N	2.88
4	ey04am03	2/M/N	4.02	es05nd21	5/M/N	2.53	ey04am03	2/M/N	2.92
5	rn5my10	4/M/N	4.49	im05on26	4/F/N	3.2	ee5ll29	2/M/B	3.49
6	im05on26	4/F/N	4.69	ee5ll29	2/M/B	3.37	ns7er29	2/F/N	3.6
7	es05nd21	5/M/N	5.09	on4dy11	5/M/N	3.38	on4dy11	5/M/N	3.77
8	ee5ll29	2/M/B	5.52	fl8hn6	3/M/N	3.38	cl9rd11	2/M/N	3.88
9	cl9rd11	2/M/N	5.76	ey06rd15	2/M/N	3.4	rn5my10	4/M/N	4.12
10	ey06rd15	2/M/N	5.98	cl9rd11	2/M/N	3.63	an05er16	3/F/N	4.13
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
23	ey4am27	2/F/N	12.98	le5ew07	4/M/N	6.82	ey4am27	2/F/N	7.27
24	ni07is28	3/F/NN	13.56	on8am20	5/F/N	6.86	el7uy24	2/F/N	7.7
25	an5am23	2/F/N	13.77	an5am23	2/F/N	7.59	ni07is28	3/F/NN	7.98
26	en07mu06	2/F/N	14.07	ey4am27	2/F/N	7.96	an5am23	2/F/N	8.36
27	el7uy24	2/F/N	14.79	en07mu06	2/F/N	10.47	en07mu06	2/F/N	9.0
28	in08am11	2/M/N	19.52	in08am11	2/M/N	14.33	ug05in14	3/F/NN	11.81
29	ng06en01	2/F/NN	19.75	wi6ex17	3/M/N	16.29	in08am11	2/M/N	13.0
30	wi6ex17	3/M/N	21.83	ng06en01	2/F/NN	17.17	ng06en01	2/F/NN	16.8
31	ug05in14	3/F/NN	27.29	ug05in14	3/F/NN	17.25	wi6ex17	3/M/N	17.57
32	xu07an12	2/F/NN	28.49	xu07an12	2/F/NN	19.12	xu07an12	2/F/NN	20.57

Table 4.8: *Speaker variability effect on performance across models. (Demographic information is: age in decades / gender / native (N) or non-native (NN))*

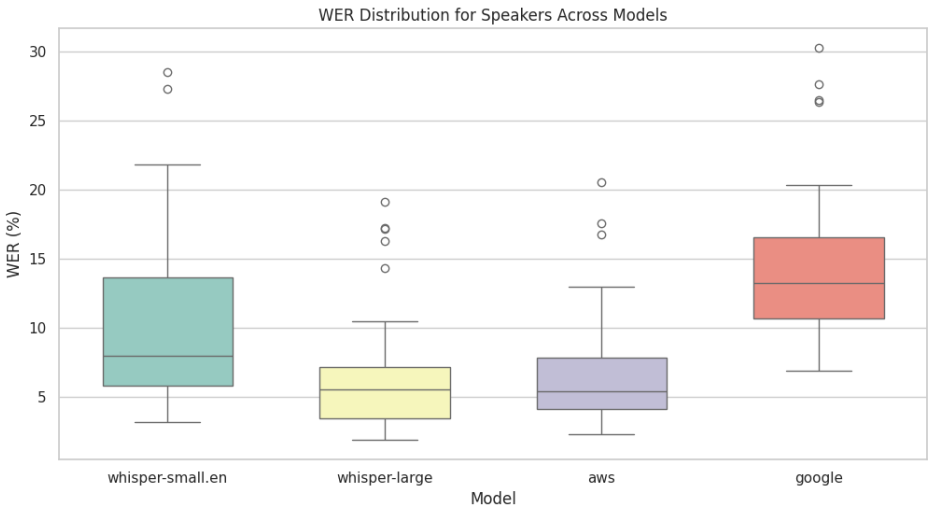

Figure 4.6: *WER distribution across speakers for each model.*

Latency

It is important that commands are transcribed in near real-time to allow for natural and efficient interaction, therefore the latency metrics for the average time taken to transcribe an individual utterance across each model are given in Table 4.9. Amazon is left out here, given the batch transcription process used, however it is noted that this inference took the longest to run. Whisper models provide very quick transcriptions, in a fraction of a second, with an average of 0.1 and 0.2 seconds per utterance transcription for the Whisper-small and large models respectively. Google averages a 1 second latency, which is still fairly quick, but

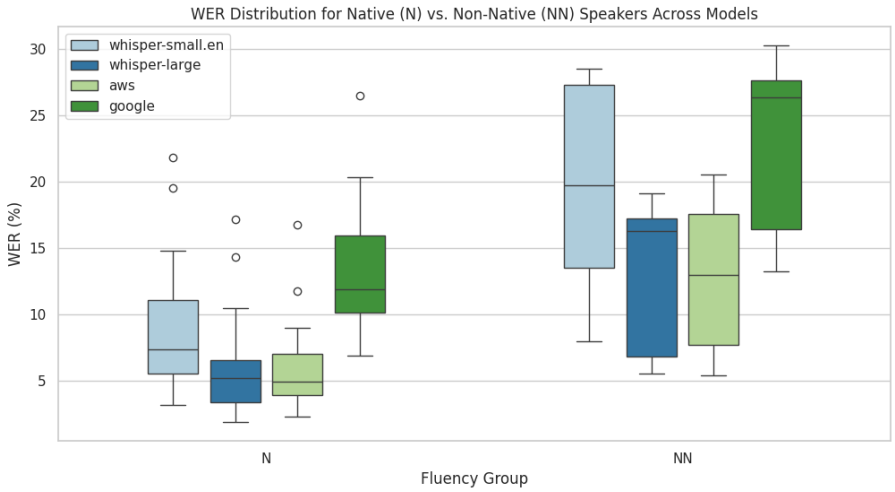

Figure 4.7: *WER distribution across native (N) and non-native (NN) speakers for each model.*

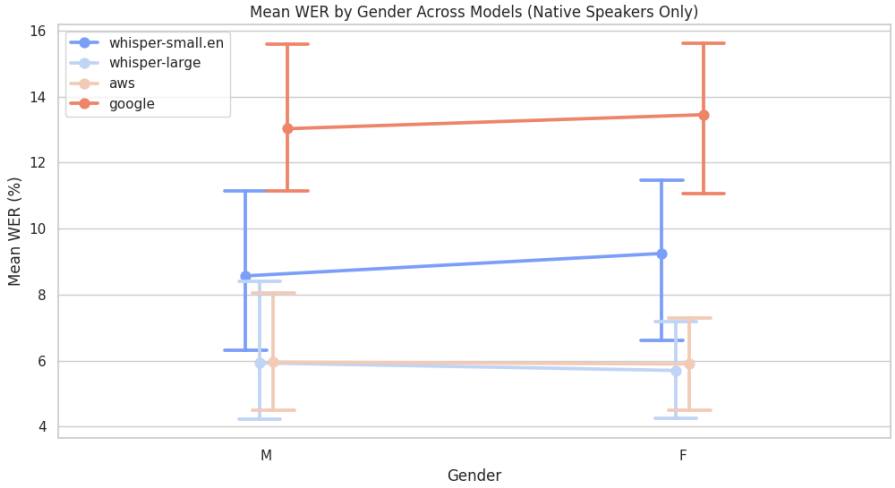

Figure 4.8: *Effect of gender on WER for each model.*

much slower than the Whisper models, indicating the latency advantages of using local, offline models. Google STT also shows less consistent and reliable response times, with transcription latency varying between 0.4 seconds and 6.7 seconds. For more accurate insights on the practical latency of each model, the audio could be played aloud, captured and streamed to models to more closely replicate real-world use. This audio playback to front-end devices is set up and used to evaluate the end-to-end system performance in Chapter 6, where Whisper-large is used as the ASR model.

Model	Min latency (s)	Max latency (s)	Average latency (s)
whisper-small	0.0557	1.4575	0.1024
whisper-large	0.1021	1.5480	0.1931
google stt	0.4113	6.6919	1.1111

Table 4.9: *Latency metrics (average time to transcribe a single command) across models.*

4.4.2 Transcription Error Analysis

Transcription errors are further analysed to reflect on ASR model performance in the context of smart home command recognition. The analysis focuses on the nature of errors, such as whether insertions are filler words, whether deletions are critical words, where substitutions occur, and how these affect downstream command understanding. All analysis is performed on the transcriptions obtained from the ESP-Box device placed closest to the user in the recording setup (ESP1), given this is the device proposed for use in the end-to-end voice assistant setup in Chapter 6. Initial example transcription errors are given in Table 4.10. At a glance, similar sounding words are substituted (i.e. trigger → tricker), and insertions and deletions appear to be fairly insignificant words (i.e. at, the, would).

Table 4.11 presents the twenty most common substitutions, deletions and insertions for each model, with counts indicating how often each occurs. These were computed using the Levenshtein distance from the Python RapidFuzz[10] library. It appears that very common, small words are most likely to be inserted or deleted, with 'I' being the most common insertion overall, and 'the' being the most common deletion. It is not critical if small words (e.g., I, the, to, you) are inserted as these do not convey much meaning, and likewise common deletion words (e.g., please, the, to) should rarely cause a command to be misunderstood in practice.

Substitution errors commonly occur for similar sounding words, for example all models commonly misrecognised 'unmute' as 'mute', and other examples include 'actuate' mistaken for 'activate', or 'disarm' and 'design'. Such errors can have a knock-on effect on downstream command understanding, potentially causing the opposite effect. Other substitution errors occur where the ASR models expand shortenings automatically (i.e. 'mins' → 'minutes'),

[10] `https://github.com/rapidfuzz`

Transcript	Hypothesis	Error	Error Type
trigger my front porch timekeeper	tricker my front porch timekeeper	trigger → tricker	SUB
hey siri lower the air quality in my attics	hey siri loan the air quality in my attics	lower → loan	SUB
make the temperature 77 fahrenheit for me	make the temperature at 77 fahrenheit for me	at	INS
i command upstairs album to unmute	i command the upstairs album to unmute	the	INS
alexa i would prefer the volume to be a bit more noisy	alexa i prefer the volume to be a bit more noisy	would	DEL
alexa turn the inside duskier for me please	alexa turn inside duskier for me please	the	DEL

Table 4.10: *Examples of transcription errors (taken from Whisper-large and Whisper-small outputs).*

Substitutions			
whisper-small	**whisper-large**	**google**	**aws**
(‘echo’, ‘ok’) - 49	(‘rooms’, ‘room’) - 36	(‘id’, ‘i’) - 63	(‘unmute’, ‘mute’) - 65
(‘rooms’, ‘room’) - 40	(‘esp’, ‘asp’) - 33	(‘echo’, ‘ecco’) - 58	(‘esp’, ‘sp’) - 52
(‘unmute’, ‘mute’) - 40	(‘i’, ‘id’) - 30	(‘rooms’, ‘room’) - 54	(‘i’, ‘id’) - 41
(‘id’, ‘i’) - 39	(‘esp’, ‘esb’) - 26	(‘unmute’, ‘mute’) - 49	(‘esp’, ‘esb’) - 34
(‘i’, ‘id’) - 31	(‘id’, ‘i’) - 22	(‘echo’, ‘ok’) - 48	(‘rooms’, ‘room’) - 27
(‘telly’, ‘tele’) - 29	(‘echo’, ‘ok’) - 20	(‘inside’, ‘insight’) - 43	(‘id’, ‘i’) - 26
(‘smart’, ‘small’) - 25	(‘degree’, ‘degrees’) - 18	(‘hi’, ‘isp’) - 41	(‘echo’, ‘ok’) - 25
(‘a’, ‘the’) - 24	(‘a’, ‘the’) - 18	(‘i’, ‘a’) - 40	(‘siri’, ‘sir’) - 24
(‘degree’, ‘degrees’) - 21	(‘unmute’, ‘mute’) - 17	(‘echo’, ‘ech’) - 35	(‘degree’, ‘degrees’) - 18
(‘siri’, ‘we’) - 18	(‘smart’, ‘small’) - 17	(‘this’, ‘this’) - 32	(‘louden’, ‘in’) - 17
(‘quieten’, ‘in’) - 17	(‘esp’, ‘sp’) - 16	(‘actuate’, ‘activate’) - 32	(‘siri’, ‘sorry’) - 16
(’mins’, ‘minutes’) - 16	(‘inside’, ‘insight’) - 13	(‘quieten’, ‘in’) - 30	(‘actuate’, ‘activate’) - 16
(‘esp’, ‘espy’) - 16	(‘echo’, ‘eco’) - 13	(‘adjust’, ‘just’) - 27	(‘a’, ‘the’) - 16
(‘esp’, ‘asp’) - 15	(‘disarm’, ‘design’) - 12	(‘the’, ‘a’) - 27	(‘esp’, ‘speed’) - 16
(‘hall’, ‘whole’) - 15	(‘hall’, ‘whole’) - 12	(‘i’, ‘are’) - 26	(‘hi’, ‘high’) - 15
(‘siri’, ‘series’) - 15	(‘secs’, ‘seconds’) - 11	(‘in’, ‘and’) - 24	(‘quieten’, ‘in’) - 14
(‘halt’, ‘hold’) - 15	(‘change’, ‘changed’) - 11	(‘disable’, ‘disabled’) - 24	(‘decibel’, ‘decibels’) - 12
(‘google’, ‘willy’) - 14	(‘mins’, ‘minutes’) - 11	(‘lock’, ‘look’) - 22	(‘esp’, ‘asp’) - 12
(‘siri’, ‘sir’) - 14	(‘the’, ‘a’) - 10	(‘command’, ‘on’) - 22	(‘temp’, ‘tent’) - 11
(‘siri’, ‘so’) - 13	(‘telly’, ‘tally’) - 10	(‘esp’, ‘asp’) - 22	(‘hall’, ‘whole’) - 11
Insertions			
whisper-small	**whisper-large**	**google**	**aws**
i - 113	i - 83	the - 187	i - 79
the - 90	the - 63	to - 116	degrees - 76
to - 75	you - 58	i - 89	the - 71
you - 71	to - 41	a - 78	on - 41
a - 56	a - 35	and - 74	you - 38
and - 48	of - 27	you - 72	to - 38
of - 43	this - 25	have - 64	a - 33
so - 33	not - 22	this - 61	and - 26
in - 33	in - 21	that - 58	of - 25
it - 32	and - 20	it - 56	so - 23
this - 31	for - 18	in - 55	loud - 22
we - 31	have - 18	not - 48	we - 21
for - 26	it - 17	we - 45	that - 21
ill - 25	we - 17	on - 41	in - 17
have - 24	be - 16	so - 41	quiet - 17
hd - 24	its - 16	all - 36	uh - 17
on - 22	im - 16	is - 36	like - 16
its - 22	so - 15	for - 36	have - 16
at - 21	dont - 14	be - 36	it - 16
quiet - 21	that - 14		its - 15
Deletions			
whisper-small	**whisper-large**	**google**	**aws**
the - 50	the - 45	the - 61	would - 45
would - 46	echo - 42	me - 61	the - 30
to - 43	would - 40	to - 47	please - 28
me - 34	to - 31	esp - 43	to - 25
a - 28	me - 19	would - 39	me - 16
please - 24	my - 18	o - 36	my - 16
my - 18	a - 17	clock - 36	a - 15
siri - 15	esp - 14	please - 35	us - 13
of - 15	of - 14	a - 32	temp - 13
us - 13	up - 13	us - 23	up - 13
in - 13	i - 12	my - 23	esp - 13
up - 12	us - 11	hi - 19	of - 11
for - 11	please - 9	up - 18	inside - 11
on - 11	lit - 9	in - 17	for - 10
sticks - 10	in - 8	i - 16	room - 10
room - 10	room - 8	it - 16	f - 9
you - 8	sticks - 7	you - 15	hey - 8
am - 7	for - 6	off - 14	siri - 7
is - 7	on - 6	for - 14	in - 7
i - 7	you - 5	f - 12	it - 6

Table 4.11: *Common substitutions, deletions and insertions for each model.*

AX: echo, each	**AT**: id, at, out, white, it	**A**: i, we, who, you, a, eye, o
FR: for, fire	**AN**: on, in, wine, any, an	**TSPL**: decibel, disable
H: hey, hi	**KL**: cool, glow	**NT**: need, not
M: me, my	**XT**: shade, shut	**AL**: ill, all, will
AK: ok, wake	**KM**: cam, gym	**T**: tea, too, to, toy
ANT: want, and	**AF**: of, off	**KT**: quiet, kit
AS: is, us	**P**: bay, by, be	**AM**: im, am
LK: lock, like	**S**: see, s	**LT**: light, lit
KN: can, con	**K**: c, go, wc	**MT**: mood, mute
TKRS: decrease, degrees	**PR**: power, per, brew	**TR**: door, dry

Table 4.12: *Clusters of common substitutions in the dataset.*

indicating the transcription is not always verbatim. Further reasons for substitution errors include the addition/removal of 's' (i.e. 'room' → 'rooms'), or differences in spellings (i.e. 'echo' → 'ecco'). Many of these substitution errors will not have a knock-on effect on command understanding as they convey the same meaning, however the confusion between 'unmute' and 'mute', or 'disarm' and 'design' would cause a direct impact on the command understanding stage. For further analysis and insight, the phonetic clusters in Table 4.12 shows the similar sounding words in the dataset that are commonly confused by the ASR models evaluated. Many common errors do not affect command meaning, while others could likely be resolved by the LLM through contextual understanding. However, some transcription errors alter command intent and may therefore cause misunderstandings.

The raw counts for errors do not consider how common the words are in the dataset, however. It is possible that word frequency corresponds to the chance of that word being inserted/deleted/substituted. To account for this, the error counts are normalised based on the frequency of each word in the ground truth. Figure 4.9 and 4.10 depict the results for Whisper-large and Amazon transcribe, respectively. Highly frequent words appear to have low error rates, as expected, but there are also a lot of very common word substitutions. This pattern of frequent words having low error rates most likely reflects the model training data and speaker familiarity with words, although the smart home command vocabulary contains very few rare words, especially in comparison to medical or specialist domains.

On a final note, it is interesting to look at cases where the model transcription is particularly wrong. Table 4.13 shows how insertions from TV background noise significantly impact the WER of transcribed commands. Even where the command itself is transcribed correctly, the extra insertions disguise the user command. For example, the intended command 'save my energy' is nestled in the transcription 'what are you going to do huh save my energy id like to deduce the message from the'. Likewise, Table 4.14 reveals the weaknesses of the Whisper models when transcribing individual words in a clean, quiet environment. While Google and Amazon models may recognise single words as multiple, the number of syllables recognised remains similar (i.e. vertigo → there you go), Whisper models can hallucinate entire streams of words when given single second audio. In the most extreme cases this results in a WER of 55, where transcribing the word 'porch' strangely produces a transcript talking about security cameras not being real and watching interviews. Also strange, is where Whisper-small outputs 'hd' numerous times in a row.

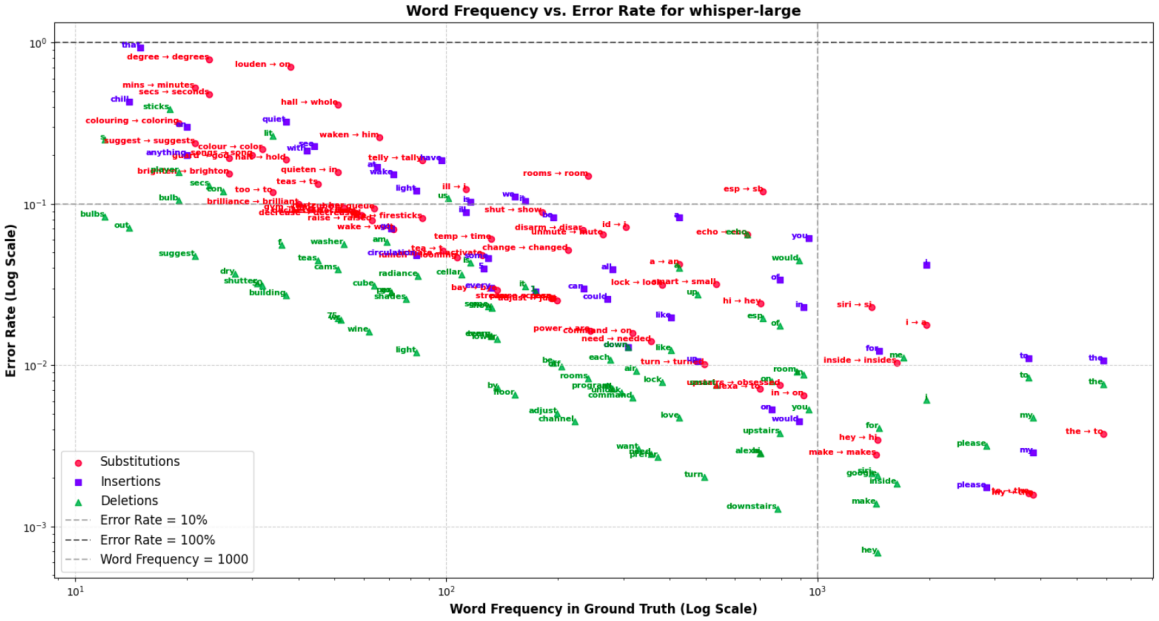

Figure 4.9: *Word frequency vs. WER for Whisper-large.*

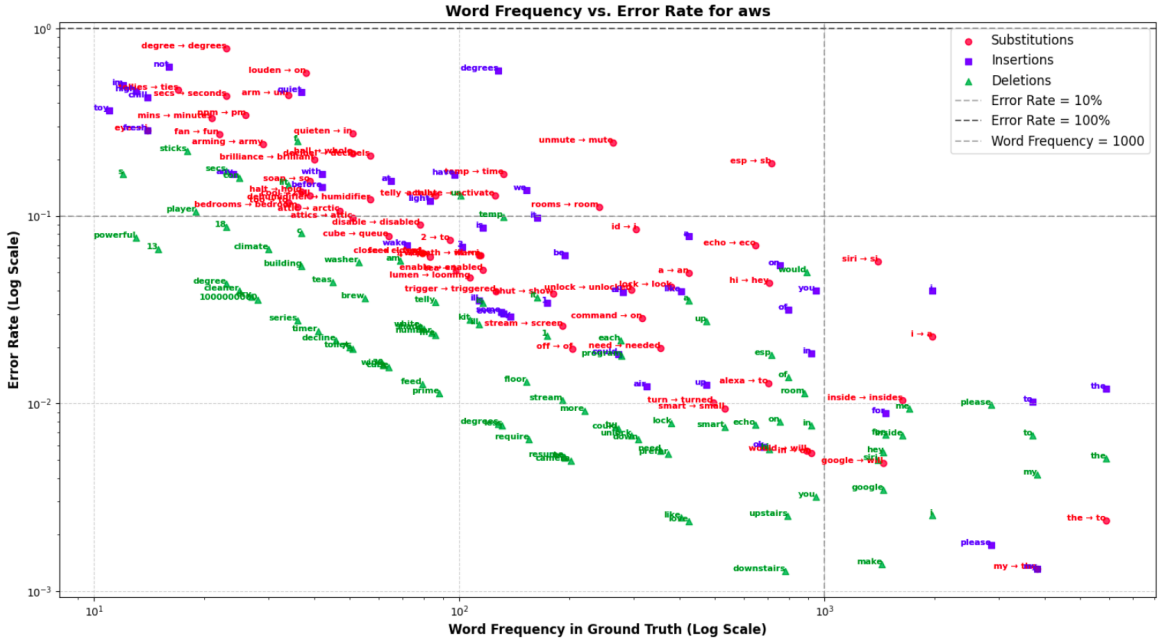

Figure 4.10: *Word frequency vs. WER for Amazon transcribe.*

4.4.3 Results Significance and Impact

The observed performance differences indicate several factors that have a meaningful impact on ASR accuracy and robustness. While formal statistical significance was not tested, practical significance of the results can be assessed through the consistency and scale of performance changes across the dataset. Smaller variations are less likely to cause noticeable real-world dif-

Scenario	Truth	Hypothesis	WER %
		model: whisper-small.en	
tv	im dehydrating	good cooks could flood england with 51000 im the hightray team the usual sense of the word	8
tv	make the downstairs bright	these little musical boxes having them so important make the downstairs bright dont forget theyre made in dartmoor prison you can smuggle stuff into it	5.25
tv	save my energy	what are you going to do huh save my energy id like to deduce the message from the	5
		model: whisper-large	
tv	make the downstairs bright	these little musical boxes having them so important make the downstairs bright they were made in dartmoor prison you can smuggle stuff into	4.75
tv	watch youtube tv	in shaking public confidence whats youtube tv we start everything after we arrested david	4
tv	im dehydrating	crooks could flood england with 5 pound notes im dehydrating	4
		model: google	
tv	im dehydrating	cooks could flood england with 5 pound trading to the	5
tv	please make 3 oolong teas	that all 3 musical boxes were made by the same 3 on low teeth 7 or 7 year terms of	3.8
tv	macchiato please	it must have to place i	3
		model: aws	
tv	im dehydrating	crooks could flood england with 5 usual sense of the word	5.5
tv	power down the sound please	we could have arrested them at tipets toy shop if we hadnt proof power down the sound please proof my dear fellow	3.4

Table 4.13: *Commands with the highest WER across models.*

ferences, therefore, practically significant differences in model performance and transcription accuracy across dataset conditions are highlighted.

Differences between ASR models was one of the significant findings, addressing the research question: *How does the performance of locally run ASR models compare to cloud-based solutions for smart home commands?*. The locally run Whisper-large model consistently achieves similar results to Amazon transcribe cloud-service, while able to run offline on a consumer-grade GPU. This is a positive result, and Whisper-large will be used in the real-world implementation. Whisper-small and Google STT were shown to perform comparatively worse, both in terms of accuracy and robustness to variation.

Environmental and recording conditions caused notable effects across all models. Performance was seen to consistently decrease with increased speaker-to-microphone distance, with significant drops beyond 2 meter distances. TV noise also caused interference with command recognition, where background chatter was seen to be captured and transcribed, while music and activity noise had little impact. Device variation contributed to relatively small performance fluctuations, although performance was seen to decrease by several percent for older phones. Accent-based variation in performance was also seen, with higher WER common for non-native English speakers, indicating limitations in training data and model generalisation.

Models perform well on command transcription and analysis showed many errors to be minor, such as substitution of similar words, or the insertion and deletion of minor or filler words, indicating many errors may not impact downstream command understanding. Single word transcription is seen to be challenging across ASR models however, with the absence of

Truth	**Hypothesis**	**WER %**
	model: whisper-small.en	
chillier	c staggering its been 5 years and i cannot comment on yet ok meaning anything 4000 more if you wanted see you im sorry its not really down	28
ill lit	intensity elite hit that like button and ill see you next time 3 hd hd hd hd hd hd hd hd hd hd hd hd hd hd hd hd hd hd hd hd hd hd hd hd hd hd hd	18.5
cellar	cdc brings out exciting adventures that we will be talking about next week sella	14
	model: whisper-large	
porch	but you know the security cameras are not real for this 0 so its not safe to think about this and im gonna go had a look during the show or something make sure you watch those interviews which are pretty 15 minutes long so do the introswich ok this was an authentic interesting interview	55
azure	ask sick people shout at 1 see what you can do if you took long enough for only 1 rule	20
gangster	my house claire told me to have gum quandasève	9
	model: google stt	
vertigo	there you go	3
docudrama	doc you drama	3
wc	to be c	3
	model: aws	
millionaire	maybe in a	3
notably	no of like	3
vertigo	there you go	3

Table 4.14: *Words with the highest WER across models.*

sentence contexts resulting in a much higher WER. Whisper models can also hallucinate on single word transcriptions, producing long outputs from one second audio.

The performance variation observed across recording conditions involving overlapping speech, speaker accents, microphone hardware, distance, and command length shows that the dataset captures ASR challenges common to both local and cloud-based systems. The consistency of performance trends across models, together with the substantial variation observed, indicates that the results reflect actual ASR robustness limitations rather than experimental noise or natural variation. The dataset can therefore be considered an effective benchmark for robustness evaluations across realistic smart home conditions and interactions. The findings additionally highlight the comparative performance of locally run ASR systems relative to cloud-based services. Next, techniques that could be applied to potentially improve the performance of locally run Whisper models are discussed.

4.4.4 Improving Performance

Improving ASR performance often relies on fine-tuning pre-trained models to reduce the significant cost of training a model from scratch. Fine-tuning is typically efficient and practical, and involves the use of domain-specific datasets, such as the smart home command recordings. Pretrained models like Whisper have robust generalisation capabilities given their training on vast and diverse datasets. However, there is still potential for fine-tuning to further enhance the precision of the models given audio data that contains the characteristics and nature of a specified use case, like smart homes.

Developers and researchers increasingly use on-device training strategies, partly driven by user privacy concerns and data protection regulations (Almeida et al., 2021), eliminating

```
"The audio contains short smart home voice commands spoken in a smart home environment.
    ↪ Wake words may precede commands, including "Hi ESP" and "Echo". The audio may
    ↪ contain background noise and other speech. Focus on transcribing only the user
    ↪ command relating to smart home IoT device control."
```

Figure 4.11: *Example prompt for Whisper.*

the need for user data to be transmitted or stored on external servers. Whisper supports fine-tuning on small datasets, with a recommended minimum of 5 hours of training data, making the smart home dataset applicable. Community tutorials and documentation outline the resource requirements and guidelines for fine-tuning Whisper models[11] [12] [13] [14].

Another possible direction is personalised speech recognition models, that aim to tailor ASR systems to individual users' speech patterns, accents and voice characteristics. Research suggests that on-device fine-tuning of lightweight, small ASR models to personalise these to users could be particularly effective in increasing robustness and performance (Hannun, 2021). For example, fine-tuning the small Whisper model on recordings from a specific speaker could result in substantial performance improvements, however it is unknown how effective around half an hour of data applied to this would be. The personalisation of speech models is most applicable in low-resource settings to maximise performance of smaller speech models.

Aside from fine-tuning, prompting the Whisper models is an option to optimise model performance on a given domain. Fine-tuning is not always feasible due to limited resources and time constraints, whereas prompting is highly accessible. Prompting uses contextual cues that are provided to the model during inference, and guides its transcription outputs without requiring modifications to the model weights. With respect to smart home commands, spelling guidance for uncommon words and domain specific terminology (e.g., device names) can be included to prime the model to recognise these correctly and an example prompt is given in Figure 4.11. A long prompt can cause increased latency, however a short prompt could guide the model to better understand the context and scenario, focus on a single user uttering a command, and give hints to prevent small errors like mistranscription of Hi ESP and Echo (less common words), or confusion between frequent substitutions.

The recommendations for future improvements that could be made to the Whisper models, indicates how these already high performing and effective models could be adapted to better handle the characteristics of smart home environments and commands.

4.5 Summary

The chapter evaluates ASR models for smart home command transcription, with comparison of cloud-based and locally run models. The chapter began with an overview of ASR models, tracing the evolution from HMM-GMM architectures to modern transformer-based systems. Models were reviewed in terms of their architecture, resource requirements, and popularity,

[11] https://github.com/vasistalodagala/whisper-finetune

[12] https://github.com/vasistalodagala/whisper-finetune

[13] https://alphacephei.com/nsh/2023/01/15/whisper-finetuning.html

[14] https://medium.com/@bofenghuang7/what-i-learned-from-whisper-fine-tuning-event-2a68dab1862

with a subset selected for inclusion in the smart home ASR performance comparison.

The experimental setup detailed data preparation steps, the model and hardware configurations, and the performance metrics used for model comparison on the smart home commands dataset. The results and discussion analysed model performance across varying noise conditions, hardware types, distances and speakers. The metrics were reported, compared, and common patterns of errors were identified. The strengths and limitations of each model were interpreted in the context of smart home control scenarios, where their suitability for real-world deployment is considered. Finally, ways to improve model performance were explored, outlining fine-tuning and prompting approaches for tailoring ASR models to specific domains.

The findings highlight the Whisper-large model as a reliable, local and flexible option for implementing accurate command transcription in smart home environments, balancing performance, privacy and adaptability. Given these results, the Whisper-large model will be integrated into the local voice assistant setup that will be evaluated in Chapter 6. From now on, ASR improvement is not the focus given the high performance of pretrained models that run on reasonable hardware constraints, it appears there is not much to improve with regard to this task in the context of the goal: a local, private smart home assistant.

In the following chapter, focus is shifted to explore how LLMs can process and interpret command transcriptions, applying their extensive reasoning capabilities to overcome the brittleness of previous voice assistants (see Chapter 5.1) and enable more flexible command understanding.

Chapter 5

Beyond Brittle Control: Evaluating LLMs in Smart Homes

5.1 Introduction

The previous chapters established the foundation for a locally run smart home voice control system, creating an evaluation dataset (Chapter 3) and demonstrating feasible local and robust speech recognition (Chapter 4). With accurate transcription established, the exploration continues to consider how transcribed natural language commands can be reliably interpreted and translated into executable device actions.

Traditionally voice assistants have relied on rigid rule-based systems that lack intelligent inference capabilities (i.e. what might be termed *common sense*) for interpreting compound, underspecified or goal-oriented commands. Users must often formulate their intents into precise and rather unnatural expressions, typically limited to one action per utterance. With this rigidity, and lack of conversational dialogue for clarification and repair, these systems can be brittle and often frustrating to use (Luger and Sellen, 2016). This brittleness has been well-documented in cloud-based systems such as Google Assistant and Amazon Alexa (Bentley et al., 2018; Porcheron et al., 2018; Luger and Sellen, 2016), and in local implementations like HomeAssistant[1]. Conversational repair is absent, command rejection occurs without explanation (often indicated by silence), and fixed templates and device naming conventions must be followed. These limitations not only reduce user satisfaction, but also hinder scalability across an increasingly diverse smart home ecosystem (as detailed in Chapter 2).

To address these limitations, LLMs have begun being integrated with smart home assistants, with Google replacing its assistant with Gemini[2] and Amazon integrating LLMs with Alexa[3]. It is not a straight-forward transition however, and Gemini has hallucination tendencies, struggles with basic commands, and does not appear as reliable and trustworthy as its Google Assistant predecessor according to Gilbert (2025). While there are evidently challenges in applying LLMs to control environments, these industry efforts indicate recognition of the limitations of traditional dialogue systems.

[1] `https://www.home-assistant.io/integrations/conversation/`

[2] `https://blog.google/products/gemini/google-assistant-gemini-mobile/`

[3] `https://www.aboutamazon.com/news/devices/new-alexa-tech-generative-artificial-intelligence`

With LLMs appearing a promising solution to the longstanding brittleness problem and limitations in smart home assistants, their integration as the reasoning component in a local home assistant system is considered in this chapter, bridging the gap between the ASR transcriptions and IoT device control. Unlike rule-based systems, LLMs are pretrained on vast corpora and can flexibly interpret user intent from natural language commands, reason about feasible device actions within a given home configuration, and produce structured outputs (e.g., control plans) alongside conversational responses, such as clarification questions or explanations of actions taken. The exploration focuses on how LLMs can support the flexible interpretation of a variety of commands, generate actionable device updates, and improve user experience through conversational clarification and repair capabilities.

Numerous LLMs are open and available for local deployment on consumer-grade hardware, however, there remain challenges to be addressed: hallucination and error tendencies that pose safety and reliability questions, computational constraints that require consideration for model size and latency. The investigation first compares various LLMs, assessing the accuracy of action plan generation and identifying failure modes. Models are benchmarked on consumer-grade hardware using annotated commands from Chapter 3's smart home commands dataset across multiple home configurations representing the device diversity in the taxonomy from Chapter 2. An evaluation of prompting techniques (e.g., few-shot, chain-of-thought) and structured approaches to reduce errors follows, assessing how well prompting strategies transfer across increasingly complex home configurations, and computational trade-offs. The evaluation framework combines validation of device selection and task completion accuracy, and manual response quality assessment, with consideration for computational cost.

This chapter addresses the key question: *How can prompting techniques be applied to make LLMs sufficiently reliable for local smart home implementation, ensuring robust performance across diverse home configurations and user intents?* The investigation contributes to the broader goal of creating a fully local smart home voice control system by demonstrating that LLMs can provide reliable and flexible natural language reasoning for device control tasks. The findings inform the model selection and prompt design for the practical, local implementation of a smart home voice assistant, and motivate the safety mechanisms developed in Chapter 6.

5.2 LLM Technology and Applications

5.2.1 GPTs, LLMs and the New AI

LLMs have emerged as a transformative technology in recent years. The public release of OpenAI's ChatGPT generated much excitement and brought LLMs out of research and into mainstream use. These systems have been trained on extensive datasets and demonstrate vast and diverse natural language understanding and generation capabilities. Fundamentally based on transformer architectures, LLMs are trained using massive computational and data resources to be general purpose reasoners (Kojima et al., 2022). Various pretrained models are continuously released that can be tuned to specific domains and tasks. Models have a context length that is defined as the number of tokens a model can use, and a higher context length means that the model has more memory, at the expense of potentially slower inference speed. Additional tunable sampling parameters include temperature, top p and top k.

Cloud-based LLM implementations exist, running massive models remotely, but come with the security and privacy concerns common to cloud systems, making them unsuitable for use

with sensitive, personal and confidential data, e.g., studies show personal information from training data can be exposed through ChatGPT queries (Nasr et al., 2023; Farrell, 2023).

Open LLMs have been released, albeit mostly with only the model weights themselves available, and not the source training data, and can be run locally on consumer-grade hardware without internet connection. A notable early example of an open LLM was Meta's release of LLaMa, and many have followed to contribute to the democratisation of this new form of AI technology. Users can experiment with this technology on their own GPU devices, with complete privacy and control over their data, using libraries like Ollama[4] and HuggingFace[5].

Relatively small (e.g., sub 10B parameter) open LLMs can be run on consumer GPU hardware, and have substantial capabilities for many applications, making them an economical and effective means of using this type of AI (Belcak et al., 2025). With much scope for adaptability and flexibility, locally run LLMs can run offline, with low latency. The main barriers to adoption include the high cost of GPU hardware for running the models and limited public awareness of this deployment option for LLMs.

For further insights about LLM capabilities and applications, a review discusses the evolution, architectures and challenges relating to LLMs (Raiaan et al., 2024), and Lu et al. (2023) compare the performance of LLMs in various domains. A survey paper additionally considers the potential of LLM-based agents and how these could function in a brain, perception and action framework transferable to various applications (Xi et al., 2023).

5.2.2 LLMs and Smart Homes

The emergence of LLM-based conversational interfaces revealed a significant technological gap in smart home voice control. Traditional assistants like Siri and Alexa, built on intent classification and slot-filling architectures, exhibit fundamental rigidity that becomes more so apparent when contrasted with modern LLM capabilities[6]. While legacy systems refuse simple requests that deviate from expected phrasing patterns and have fixed, limited personalities (Coward, 2024), LLMs demonstrate likeable and flexible conversational abilities, with personalised interaction styles[7] and confident responses to virtually any user query (Rey-Jouanchicot et al., 2024).

Smart home ecosystems like Amazon Echo and Google Home have more recently been updated to integrate LLMs[8] [9], but with drawbacks in perceived trustworthiness, and inconsistent and error-prone functionality (Gilbert, 2025). In addition, Amazon's Alexa+ incorporates LLMs for more natural conversations, with contextual understanding and interaction history allowing for a personalised experience, more proactive assistive capabilities, and voice-based automation routine creation, but this comes with a monthly subscription cost, and a host of security and privacy concerns given the new capabilities require more data sharing and third-party services (Lamkin, 2025).

[4] https://ollama.com/

[5] https://huggingface.co/

[6] https://www.macstories.net/linked/the-new-york-times-declares-that-voice-assistants-have-lost-the-ai-race/

[7] https://johnthenerd.com/blog/local-llm-assistant/

[8] https://blog.google/products/gemini/google-assistant-gemini-mobile/

[9] https://www.aboutamazon.com/news/devices/new-alexa-tech-generative-artificial-intelligence

LLMs can resolve ambiguities in commands that would be refused by traditional systems. For example, in a TV interaction, LLMs can use conversational context to distinguish a user requesting the 'third' film on the screen vs the 'third' channel on a TV (Santos et al., 2020). Such common sense knowledge becomes more important to manage device networks as they grow in number and capabilities (Lieberman and Espinosa, 2006). Calò and De Russis (2024) consider multimodal command disambiguation, exploring how context can be used to improve LLM response alignment with user intent using both text and visual cues. These works demonstrate how LLMs can transform smart home interaction from rigid command-response patterns to conversations with contextual reasoning.

LLMs have been applied to provide explanations in smart environments (Sadeghi et al., 2024). Explanations can build user understanding and trust in interactive systems (Kordts et al., 2021), and King et al. (2024) apply a fine-tuned sub-3b LLM running on a Raspberry Pi to explain the device capabilities of a lightbulb and thermostat, where devices are modelled as state machines. LLMs have also been shown to support natural language-based automation routine creation, with several studies prompting ChatGPT to create structured automation routines given user instructions (Giudici et al., 2024; Li et al., 2023).

Goal-oriented reasoning in smart homes is a research area moving away from the usual one-to-one correspondence between user instruction and physical controls (Noura et al., 2020; Palanca et al., 2018). The capabilities of various LLMs for goal-oriented smart home commands, such as 'I'm tired' and 'let's watch a film', are analysed in a study where the authors prompt LLMs to generate action plans given such commands (King et al., 2023a,b). LLMs are shown to reason creatively about commands, although there are errors and challenges to address, including hallucinated actions. Building on these works, this study presents a performance comparison of a wider range of LLMs over a larger command set, and investigates prompt techniques and design to minimise errors, with consideration for the feasibility of real-world deployment as a smart home voice assistant.

5.2.3 Open and Local LLMs

Open-source LLMs capable of running on consumer-grade hardware are explored to guide model selection for the performance evaluation of these for smart home reasoning. A comparison of relevant, popular and recently released models that can operate within the defined hardware constraints (NVIDIA RTX4090 with 24GB VRAM GPU) is presented in Table 5.1. Instruction-tuned models, where instruction following datasets have been used to improve performance (Cao et al., 2023), are expected to be most appropriate to the task, therefore these are highlighted. While this study focuses on general purpose and instruction-tuned LLMs, there are LLMs trained for specific purposes, such as Gorilla (Patil et al., 2023) and ToolLLM (Qin et al., 2023) that are trained for API and service integration.

Several frameworks support the local deployment and customisation of LLMs. Ollama[10] simplifies the process of downloading and running models locally, and Hugging Face[11] provides a library for model loading, inference and fine-tuning. HuggingFace is particularly significant given the trend of open-source model release on this platform, while Ollama is simple and easy to use.

[10] `https://github.com/ollama/ollama`
[11] `https://huggingface.co/`

Model	Size	Context	Source	License	Instruct-tuned	Release Date	Reference
LLaMa3.1	3B, 8B	128k	Meta	LLaMa 3 Community License	Optional	07-2024	Grattafiori et al. (2024)
Gemma3	12B	32k	Google	Gemma License	Optional	03-2025	Gemma Team et al. (2025)
Qwen2.5	3B, 14B	128k	Alibaba	Apache 2.0	Optional	12-2024	Qwen et al. (2024)
Phi-4	14B	16k	Microsoft	MIT	No	12-2024	Abdin et al. (2024))
DeepSeek-r1	14B	128k	DeepSeek	MIT	Yes	01-2025	DeepSeek-AI et al. (2025)
Hermes3	8B	128k	Nous Research	LLaMa 3 Community License	Yes	08-2024	Teknium et al. (2024)
Cogito	10B	128k	Deep Cogito	Apache 2.0	Yes	04-2025	Deep Cogito (2025)
Mistral	7B	32k	Mistral AI	Apache 2.0	Yes	09-2023	Mistral AI Team (2023)

Table 5.1: *Comparison of open LLMs*

Technique	Description	Expected Improvement	Applicability
Prompting	Design and refine input prompt text	Adapt reasoning behaviour, control output structure for specific tasks	All LLMs, fast, flexible, no training
Fine-tuning	Train on domain-specific data	Domain adaptation, improve accuracy and robustness on specialised tasks	Resource intensive, requires data and compute power

Table 5.2: *Techniques to improve LLM performance.*

5.2.4 Prompting Techniques

Common techniques for adapting LLMs include prompt refinement and fine-tuning. Prompt engineering involves refining the input text to flexibly adapt LLMs to output more accurate and useful responses for a given task. Fine-tuning consists of further training models on domain-specific datasets to improve accuracy, robustness and reliability on specialised tasks. However, this is resource intensive, requiring high-quality training data and compute power. These methods are compared in Table 5.2. Given the resource-intensive nature of fine-tuning, both with regard to data and compute resources, the study focuses on prompting methods that could improve LLM performance on the smart home command interpretation task.

A further LLM adaptation technique, Retrieval Augmented Generation (RAG), uses external knowledge bases to provide the LLM specific information, like expert knowledge, so that models can incorporate this context in the response without requiring the compute resources of fine-tuning. RAG works by retrieving documents relevant to a question using keyword search, and then the model generates answers given the additional context (Lewis et al., 2020). Potential benefits of this technique include reduced model hallucination, increased transparency and some traceability. Discussion on whether to fine-tune models or use RAG is offered by Alghisi et al. (2024).

Standard prompting methods typically include zero-shot inputs, where a task description is provided to the model, and few-shot inputs, where a small set of examples are used to guide model responses (Brown et al., 2020; Kojima et al., 2022). However, there is an abundance of literature on more advanced prompting techniques to better guide models, with several

Technique	Description	Reference
Chain-of-Thought	Mirrors how humans decompose problems. Reasoning is enhanced with a sequence of intermediate steps, e.g., "think step by step."	Wei et al. (2022)
Complex Chain-of-Thought	Longer reasoning chains perform better on multi-step reasoning tasks.	Fu et al. (2022)
Program-of-Thought	Generates Python programs to pass to an interpreter, reducing LLM responsibilities and improving numerical reasoning.	Chen et al. (2022)
Least-to-Most	Reduces complex problems to sub-problems to be solved sequentially, e.g., "let's break down this problem."	Zhou et al. (2022a)
ReAct	Prompts LLMs to generate reasoning traces and actions interleaved as Thought and Action.	Yao et al. (2022)
Self-Consistency	Self-consistency in CoT reasoning improves performance.	Wang et al. (2022)
Chain-of-Symbol	Uses symbolic representations during chained reasoning steps to express spatial or abstract relationships.	Hu et al. (2023)
Contrastive Chain-of-Thought	Enhances reasoning with prompts that include both positive and negative demonstrations.	Chia et al. (2023)
Chain-of-Verification	The LLM generates a response, verifies it via sub-queries, then corrects the initial output. Reduces hallucination.	Dhuliawala et al. (2023)
Plan-and-Solve	First generates a plan, then solves each subproblem step-by-step. Example: "Let's first understand the problem..."	Wang et al. (2023)
Analogical Reasoning	Prompts the LLM to solve similar example problems before answering the original one.	Yasunaga et al. (2023)
Synthetic Prompting	Uses LLMs to generate few-shot examples, selecting those with longer reasoning chains.	Shao et al. (2023)
Logical Thoughts	Guides reasoning with structured, logical step-by-step analysis.	Zhao et al. (2023c)
Verify-and-Edit	Improves factuality by post-editing CoT outputs, including when/how to edit reasoning chains.	Zhao et al. (2023a)
System 2 Attention	Regenerates relevant context by removing irrelevant segments before answering.	Weston & Weston and Sukhbaatar (2023)
Program-Aided-Language Models	Generates interleaved code and text as reasoning steps to enhance interpretability and accuracy.	Gao et al. (2022)
Metacognitive Prompting	A framework that mirrors human self-monitoring: understand, judge, assess, confirm, explain.	Wang and Zhao (2023)
Rephrase and Respond	Allows LLMs to rephrase potentially ambiguous inputs before responding.	Deng et al. (2023)
In-Context Explanation	Provides explanatory examples in-context to aid LLM reasoning. Inspired by human learning.	Lampinen et al. (2022)

Table 5.3: *Comparison of prompt techniques.*

review papers categorising and comparing prompt methods across a range of NLP tasks (White et al., 2023; Vatsal and Dubey, 2024; Sahoo et al., 2024). This prompt report looks to understand prompting techniques and analyses their use (Schulhoff et al., 2024), and prompt design is further introduced and discussed by Amatriain (2024). Following existing NLP task classifications, and to identify potentially useful prompting methods, the smart home control task is considered to span: language-based task completion, contextual question-answering, multi-hop reasoning, template pattern/structured output and common sense inference. Table 5.3 presents prompting techniques that appear relevant to the work, which are further explored in LLM evaluations.

Prompting techniques can significantly affect LLMs effectiveness across tasks, as demonstrated in a study exploring prompts for maths question answering (Battle and Gollapudi, 2024). While the pursuit of the perfect prompt structure has generated much discussion[12], along with the suggestion that prompting is best done by LLMs themselves[13], the reality appears to be more complex, with trade-offs and subtle effects.

Chain-of-thought reasoning is a popular technique that guides models to solve problems step-by-step. While effective, it typically requires multiple LLM inference turns (for each step), increasing latency and computational cost. However, another paper suggests the same effect can be achieved by simply adding 'think step by step' to prompts (Kojima et al., 2022),

[12] `https://medium.com/aimonks/chatgpt-cheat-sheet-drafting-the-perfect-prompt-part-1-5149c9b1d8ab`

[13] `https://spectrum.ieee.org/prompt-engineering-is-dead`

highlighting how minor phrasing changes can cause seemingly disproportionate effects.

LLMs sensitivity to prompts can arise from system biases and instabilities. For example, the order and type of examples selected in a few-shot prompt can cause output bias (Lu et al., 2021; Zhao et al., 2021). Zhao et al. (2021) explain how majority label bias, recency bias, and common token bias can cause LLMs to prefer certain answers given few-shot examples. Majority label and recency biases lead the model to predict training answers that appear frequently or near the end of the prompt, e.g., a prompt that ends with a negative training example may cause bias towards the negative class. Liu et al. (2023a) find that placing relevant information at the very beginning or end of prompts leads to better performance (primacy and recency bias, respectively). In addition, placing queries both before and after context information can significantly help simple retrieval tasks, like key-value extraction, but does not have the same impact on complex tasks. Instruction-tuned models are less sensitive to positional bias, but it is not eliminated.

Given the trial-and-error nature of prompting, tools have been developed to automate the process and find the optimal prompt for a given task (Khattab et al., 2023), with automatic prompt engineering proposed by Zhou et al. (2022b). Guidance languages additionally exist to control and constrain the LLM output structure[14], e.g., for generating JSON structures. For enhanced model explainability and interoperability, LLMCheckup (Wang et al., 2024a) is designed to generate explanations for this purpose.

With the need for accurate, reliable and consistent responses, these studies inform the prompt design to optimise performance above a zero-shot baseline, and reduce errors, in the evaluation of LLMs for smart home control.

5.2.5 Evaluation Techniques

Evaluating LLM performance on smart home command reasoning requires both quantitative and qualitative approaches. Beyond accuracy, LLM evaluations also consider calibration, robustness, fairness, bias, toxicity, and efficiency, highlighting some important dimensions of performance to consider for real-world use cases (Liang et al., 2022). LLM responses can be assessed in terms of fluency, coherency and consistency, where expert annotators use these factors to evaluate model responses to a corpus of questions in one study (Giudici et al., 2023).

Datasets and benchmarks are available for LLM evaluation on various tasks. The Hugging Face open LLM leaderboard keeps track of the latest models performance across tasks and domains[15]. The lmsys chatbot arena[16] uses crowdsourced user feedback to compare LLM performance, where users vote on their preferred answer. A framework for benchmarking interactive LLMs across NLP tasks provides metrics, datasets and performance results for ChatGPT (Bang et al., 2023). The reasoning and decision-making capabilities of LLM agents are benchmarked in another study (Liu et al., 2023b). The OpenAssistant dataset contains prompt and response pairs annotated with quality ratings to help align LLMs with human preferences (Köpf et al., 2023). There are no available benchmarks or datasets relating to IoT devices or smart home control tasks, therefore the dataset used in this study is derived from the smart home commands dataset from Chapter 3.

[14] https://github.com/guidance-ai/guidance

[15] https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard

[16] https://lmsys.org/blog/2023-05-03-arena/

The evaluations in this work will measure the accuracy and robustness of LLM device control outputs, alongside the quality of the overall response in terms of usefulness, and alignment with the device control output. Quantitative metrics will focus on accuracy of device selection and value assignment to devices in action control plans, with error type categorisation, and consideration for computational cost (latency, token cost). Qualitative evaluations will consider response quality, style and conversational repair capabilities.

5.3 Experimental Methodology

5.3.1 Problem Definition

The investigation focuses on how LLMs can be applied to interpret smart home commands and produce grounded, structured action plans. The experiments evaluate a selection of prompting methods across models, home configurations and command types, with the goal of maximising the accuracy of LLM output device updates. Prompts will be evaluated across smart home environments, intents and command types to assess robustness and reduce common failure modes (e.g., hallucination, false negatives, invalid value assignment).

Given a natural language user command and a structured JSON home model describing available devices and their attributes, the LLM task is to generate a JSON action plan that specifies the correct device(s), attribute(s), and value(s) to update, along with a concise natural language explanation of the changes.

Evaluation will measure device selection accuracy (correct device selected to update), execution accuracy (correct device, attribute, value updates), latency (response time), token cost (total output generation length), robustness (performance consistency across homes, intents, and command types), and error categorisation (e.g., false negatives, hallucinated attributes, invalid value type).

The subsequent sections detail the annotated dataset used for these experiments, the definition of the home models, the evaluation framework and the benchmark setup.

5.3.2 The Dataset

To evaluate how accurately LLMs interpret and execute smart home commands, a dataset of annotated natural language commands paired with expected device updates is constructed. The dataset is derived from the recorded smart home commands dataset (Chapter 3), and designed to reflect the diversity and ambiguity of real-world smart home interactions. It covers a range of intents across various smart home devices, with command types defined as:

- **Device**: The device is explicitly stated.
- **Direct**: A device property is stated, but not the device.
- **Indirect**: No direct mapping to devices or values, vague, ambiguous, goal-oriented.

The device commands check the capabilities of LLMs to handle well-specified commands, and replicate the brittle rule-based control systems. The direct and indirect command types allow for evaluating LLM reasoning capabilities in scenarios with ambiguity and incomplete information, requiring more complex inference.

Total commands	440
Annotated commands	390
Unique intents	39
Device commands	235
Direct commands	155
Indirect commands	50

Table 5.4: *LLM evaluation dataset overview.*

For each intent category in the smart home commands dataset (Chapter 3), five of each the device and direct commands are selected for the LLM evaluation dataset. Where no direct commands exist for an intent, purely device commands are selected. Indirect commands are taken from the DecreaseEnergyConsumption and SetPreference intents, the only purely indirect intent categories. Each of the device and direct commands are annotated with expected device updates, but indirect commands are not, given their open-ended nature. An overview of the resulting dataset for LLM evaluation is given in Table 5.4, with 440 total unique commands across 39 intent categories.

The annotation process, for assigning expected device updates to commands, uses the fixed device and attribute list in Table 5.5 as the scope. The device set covers the actuator devices in the smart home device taxonomy (Chapter 2), and the intents in the smart home commands dataset (Chapter 3). The annotation process is simple in that for each command, the expected device, attribute and value updates are annotated given this fixed list of options.

Table 5.6 illustrates the structure of the resulting dataset. Each entry consists of:

- **Command**: Natural language instruction *(e.g., make the living room brighter).*
- **Intent**: Action category *(e.g., IncreaseBrightness).*
- **Command Type**: Categorised as Device-specific, Direct, or Indirect.
- **Expected Devices**: Expected device from device list (see Table 5.5).
- **Expected Updates**: Expected attribute:value updates for expected devices.
- **Location**: Location mentioned in command (if present) *(e.g., kitchen).*
- **Expected Device Opts**: Expected variations of devices (based on location).

The expected device options column allows for location based device naming, where any location mentioned in a command is appended to the device name (e.g., living_room_light), so the evaluation can specify and account for location specific devices.

The dataset annotations are used in the quantitative results, evaluating whether the LLM correctly translates the command into an appropriate device update, producing JSON outputs. The evaluations will primarily benchmark LLM performance by comparing JSON output responses against expected device update annotations. Accuracy will be assessed at device, attribute and value assignment levels, comparing error types across intents and command types to identify strengths and weaknesses. The indirect commands are not annotated, and instead a quality score is assigned to responses to these in the qualitative evaluation.

Lighting	Climate	Entertainment	Security	Misc
lamp state brightness colour **light** state brightness colour	**air_con** state temperature **heating** state temperature **humidifier** state humidity **dehumidifier** state humidity **fan** state speed **purifier** state air_quality	**tablet** state brightness volume mode **tv** state brightness channel volume content service **smart_speaker** state channel volume content service mode **display** state brightness channel volume content service	**security_cam** state **door** state **lock** state **window** state **gate** state	**coffee_maker** state brew quantity **tea_maker** state brew quantity **robot_vacuum** state mode **dishwasher** state mode **washing_machine** state mode **blind** state **alarm_clock** state time day

Table 5.5: *Devices and attributes covered in the dataset, grouped by category.*

5.3.3 The Homes

For inference, LLMs will be provided with the dataset command set and a home model configuration, with action plans expected to be grounded in the home model. The simulated home environment design approach proposed by King et al. (2023b) is adopted, that includes three progressively more complex setups, with the most complex setup including lighting, climate control, entertainment, security and miscellaneous devices:

- **H1**: Limited to lighting devices, like overhead lights and lamps.
- **H2**: Adds climate control (heating, air conditioning) and entertainment (TV and speaker) devices.
- **H3**: Adds security (door locks, door cameras) and miscellaneous devices (robot vacuum, coffee maker, blinds).

This variety allows for testing the LLMs ability to generalise across home configurations, comparing the effect of complexity on performance and inference time. A comparison of the defined home models is given in Table 5.7.

The homes are designed with the devices and attributes defined in Table 5.5. The JSON structure of the H1 home is given in Figure 5.1, with the other home models, H2 and H3, given

Command	Type	Expected device	Expected value	Location	Device opts
i'd love to turn the luminosity level of my tv downstairs to 100	Device	[tv, display]	{state: on, brightness: 100}	downstairs	[downstairs_tv, downstairs_display]
siri turn my television to luminosity level 85	Device	[tv, display]	{state: on, brightness: 85}	general	[]
i would prefer to make the glow 30 percent	Direct	[tv, display, tablet, lamp, light]	{state: on, brightness: 30}	general	[]

Table 5.6: *Example annotations for AdjustBrightness intent.*

```
"bedroom_light": {"state": "on","brightness": "60","colour": "warm_white"},
"bathroom_light": {"state": "on","brightness": "60"},
"living_room_lamp": {"state": "off","brightness": "55","colour": "green"},
"living_room_light": {"state": "off"},
"kitchen_light": {"state": "off","brightness": "41"},
"hallway_light": {"state": "off"}
```

Figure 5.1: *Example smart home environment JSON definition (h1 home).*

	H1	H2	H3
Device categories	1	3	5
Device count	6	12	24

Table 5.7: *Home complexity comparison.*

in Appendix D. A flattened structure is intentionally used for efficient processing. Devices are descriptively named with their position in the home so the LLM can infer their semantics, and default settings are assigned to indicate value types. In the evaluation, the LLMs ability to adhere to the given home configuration and assign appropriate values to relevant devices is measured, with particular importance placed on grounding actions in the home model, i.e., not hallucinating any devices or attributes.

5.3.4 Evaluation Framework

For each prompt, home configuration and command, the LLM inference outputs are expected to consist of structured device updates and natural language explanations. The inference responses will be saved in a CSV file, and the evaluations thereafter consist of extracting and evaluating the LLM output JSON against the expected device update annotations, and analysing the corresponding natural language explanation.

The JSON updates are evaluated against the home model (grounding) and expected device update (relevancy) annotations. Building on the work of King et al. (2023b), device selection accuracy is separated from execution accuracy (correct device updates), with error codes assigned to incorrect outputs to identify common failures. There are three validation layers

that the evaluation follows:

1. **Device correctness**: Are updated devices in the expected device list and present in the home configuration?
2. **Attribute correctness**: Are updated attributes expected for the selected device and present for the device in the home configuration?
3. **Value correctness**: Are the values assigned to each correct device and attribute combination expected and correctly formatted?

While conversationally expressed above in simpler terms, device correctness corresponds to device selection accuracy, and the attribute correctness and value correctness are combined into an execution correctness, or execution accuracy score, with each further defined below.

Device selection accuracy counts a response as correct if either:

- the predicted device matches an expected device (or device option), and is in the home configuration.
- there is no predicted device and no expected device (or device option) is in the home configuration.

Else the response is not counted as correct. The **device selection accuracy** reported is therefore:

$$\text{device accuracy} = \frac{\#\text{ correct device selections (including correct no device cases)}}{\#\text{ total responses}}$$

Execution accuracy counts a response as correct if either:

- an expected and correct device is selected and the updated attribute and value pairs match the expected update for that device, and are valid given the home configuration.
- there is no predicted device and no expected device exists in the home configuration (no device option case).

Else, it is not counted. The overall **execution accuracy** reported does not distinguish between attribute and value assignment errors, and is therefore defined as:

$$\text{execution accuracy} = \frac{\#\text{ correct device updates (including correct no update cases)}}{\#\text{ total responses}}$$

The full set of defined metrics used to measure the accuracy and computational cost of LLM responses, in order to evaluate their relative performance on smart home control tasks, include:

- **Device Selection Accuracy** = The percentage of responses that target a relevant device (or correctly target no device, if no relevant device exists). (dev)
- **Execution Accuracy** = The percentage of responses that target a relevant device (or not), and update relevant (expected) attributes with correct values. (exec)

#	**Column Name**
01	transcript
02	command_intent
03	command_type
04	expected_devices
05	expected_values
06	full_llm_response
07	token_cost
08	latency
09	json_update
10	is_valid_device
11	is_valid_execution
12	error_type

Figure 5.2: *LLM evaluation CSV, with commands, annotations, inference outputs, evaluation outputs*

- **Latency** = Time taken for the LLM to generate a response. (lat)
- **Token Cost** = The number of output tokens generated per response. (toke)

The evaluation additionally defines error types to understand each model's limitations, and the variation across models, in interpreting smart home commands, and generating action plans for device control. The error types considered include:

- Device - False Positive (dev_fp): Device selected from the home, where no relevant device is available.
- Device - Wrong Choice (dev_wc): Irrelevant device selected from the home, where there is a relevant device available.
- Device - False Negative (dev_fn): No device selected from the home, where there is a relevant device available.
- Device - Hallucinated (dev_hal): The model updated a device that does not exist in the home model.
- Attribute - Wrong Choice (att_wc): Wrong choice of attribute updated.
- Attribute - Hallucinated (att_hal): Attribute is not in the home model.
- Value - Incorrect (val_inc): Value assigned to (correct) attribute is not correct.

The resulting CSV file from the evaluation stage is depicted in Figure 5.2, where it can be seen that the JSON update from the LLM is compared against expected devices and expected values columns, and marked with whether the device selection is valid and whether the device update is valid, with a column for the error type.

The LLM evaluation process is somewhat complex, therefore Figure 5.3 further demonstrates how the metrics and error types are calculated, outlining the logic used to evaluate each LLM response on the annotated smart home commands dataset.

```
1) Load data and normalise
  - Home model JSON: the ground-truth set of supported devices and their attributes.
  - Expected devices: targets (‘device’, ‘device_opts’) relevant to the command.
  - Expected values: attribute updates relevant to the command (device-specific or shared).
  - Device update JSON: the device update extracted from the LLM ‘full_response’.

2) Validate device update -> returns is_valid_device, device_error
  Matching rule: An updated device is valid if it matches an expected device exactly,
  or by suffix (e.g., ‘bedroom_light’ -> ‘light’).

  If JSON empty:
    - If no expected devices exist in the home -> VALID
    - If expected devices exist in the home and include required attributes -> DEV_FN

  If JSON non-empty, check grounding and relevance:
    - If update mentions devices that do not exist in the home model -> DEV_HAL
    - If no expected devices exist in the home (considering exact or suffix) -> DEV_FP
    - If updated devices exist in the home, but none match any expected device -> DEV_WC
    - If an updated device both exists in the home AND matches an expected device -> VALID

3) Validate value execution  -> returns is_valid_execution, value_error
  Scope: Evaluate only devices that were VALID in step 2.
  For each VALID device, build a set of applicable options for the device update:
    - Supported attributes = keys in home model[device]
    - Build expected device updates (alts) to check based on the supported attributes:
        1) shared update options (for any expected device)
        2) exact device name update options
        3) device suffix update options (e.g., expectation for ‘light’ apply to ‘hall_light’)

  Matching rule: An alt is applicable if all keys are present in the JSON update
  For each valid device update AND applicable alt option:
    - Compare attribute key values, using.
        - Times: ‘<07:30’, ‘>=07:30’.
        - Relative changes: ‘increase’ / ‘decrease’ (numeric).
        - Absolute changes: ‘+5’, ‘-3’ (new = old +/- diff).
        - Thresholds: ‘>0’.
        - Exact string/number match.
    - If any applicable alt matches the device update (keys and values match) -> VALID.

  If no applicable alt matched the device update:
    - If values didn’t match the expectation -> VAL_INC.
    - If a device attribute is updated that is not present in the home model -> ATT_HAL.
    - Else -> ATT_WC.
```

Figure 5.3: *Logic of the LLM evaluation.*

Metrics relating to information retrieval problems, such as F1, precision, recall and relevance scores have been considered in similar works (King et al., 2023b), but are not considered in this evaluation that instead focuses on the accuracy of device updates, and errors that prevent accurate and correct device updates. Errors not measured include where extra devices or attributes are targeted. The responses will be manually examined in the qualitative evaluation, where it will be identified if this type of error occurs.

CPU	AMD Ryzen 7 7700X, 8-core, 16 threads
RAM	32GB DDR5, 4800 MT/s
Storage	1 TB NVMe SSD (SOLIDIGM SSDPFKNU010TZ)
GPU	NVIDIA GeForce RTX 4090, 24GB VRAM
OS	Ubuntu 22.04.5 LTS (Jammy)

Table 5.8: *Hardware specification for LLM benchmarking.*

```
"role": "system",
"content": "You are an AI that controls a smart home. You receive user commands and
    ↪ assign settings to devices in response.\n\nHome Model: { JSON }\nCommand: [ user
    ↪ command ]\n\nIf there are devices relevant to the user command, respond with a
    ↪ JSON block containing only the devices and attributes to update from the home
    ↪ model, and a one sentence explanation\n\nIf there are no devices relevant to the
    ↪ user command, respond with:\n{}\n\n"
```

Figure 5.4: *Baseline zero-shot prompt (prompt-zero).*

5.3.5 Benchmark Setup

The LLM benchmarking task and experiments are all conducted on a consumer-grade machine to simulate a feasible setup for local inference within home environments, without limiting the compute power too much so that models of different sizes, up to 14B parameters, can be compared. Table 5.8 details the full hardware specifications. The configuration allows for the comparison of locally run LLMs reasoning capabilities and computational efficiency, grounding the results in the feasibility of a privacy-preserving, local smart home assistant.

Model inference is performed using the Ollama library[17], which provides an interface for running a wide variety of open LLMs locally. For selected models, prompts are designed and compared, with the same prompt evaluated across all selected models. The sampling parameters top_p and top_k are fixed to 0.9 and 40 in the evaluations, however varying the temperature value (0.1, 0.4, 0.8, 1.0) is experimented with. Each command in the dataset is run once per model, prompt and home configuration, with no retries. Outputs are logged in the output CSV (Table 5.2), with the LLM output, token cost, and inference latency mapped to each command. Prompts are iteratively refined and adapted to minimise error types, with Python scripts used for evaluation and results analysis.

5.4 Results and Analysis

5.4.1 Baseline Zero-Shot Results

Baseline Performance Across Models and Homes

The evaluation begins with a performance comparison between base and instruction-tuned LLMs using the short, zero-shot task description prompt in Figure 5.4. This establishes a baseline for accuracy, efficiency, and error types across homes, command types and mod-

[17] https://ollama.com/

Model	Avg				H1		H2		H3	
	Dev	**Exec**	**Toke**	**Lat**	**Dev**	**Exec**	**Dev**	**Exec**	**Dev**	**Exec**
llama3.2:3b	78.21	72.31	32.65	0.15	92.05	91.03	74.36	64.36	68.21	61.54
qwen2.5:3b	83.59	81.11	24.29	0.13	99.23	99.23	77.18	75.13	74.36	68.97
llama3.1:8b	85.73	75.13	36.76	0.28	82.31	81.54	87.95	75.38	86.92	68.46
llama3.1:8b instruct	86.49	79.06	33.70	0.38	92.82	92.31	86.92	78.46	79.74	66.41
mistral:7b	54.87	45.3	64.78	0.41	12.82	12.05	60.51	51.03	91.28	72.82
wizardlm2:7b	61.71	51.11	80.41	0.51	42.05	41.03	61.03	48.21	82.05	64.10
hermes3:8b	82.39	69.83	54.89	0.37	86.41	85.38	74.10	58.46	86.67	65.64
yi:9b	76.24	67.61	39.59	0.34	72.56	71.54	79.23	66.67	76.92	64.62
gemma3:12b	**92.05**	**87.52**	21.36	0.29	98.46	98.46	89.74	84.62	87.95	79.49
phi4:14b	87.95	80.17	45.83	0.56	81.28	81.03	88.97	79.74	93.59	79.74
cogito:14b	**92.22**	**85.22**	39.04	0.52	94.62	94.62	92.56	82.82	89.49	78.21
qwen2.5:14b	**91.88**	**87.61**	39.56	0.53	98.72	98.72	91.54	86.67	85.38	77.44
qwen2.5:14b instruct	**92.05**	**87.69**	39.17	0.53	98.97	98.97	91.79	86.41	85.38	77.69
qwen3:14b	**91.11**	**87.86**	332.8	4.04	98.72	98.72	90.26	87.18	84.36	77.69

Table 5.9: *Prompt-zero results across homes. Values in %, except token count and latency in seconds. Bold values indicate best performing models.*

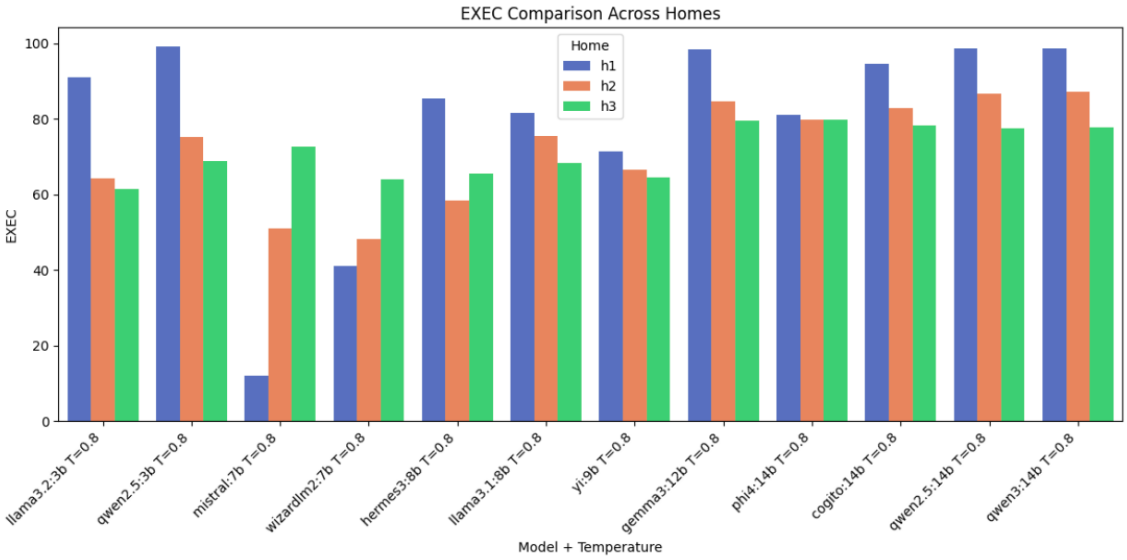

Figure 5.5: *Prompt-zero execution accuracy %, model comparison across homes.*

els. Comparing baseline model performance on the smart home control task informs model selection for the continued exploration of techniques to improve results.

Table 5.9 summarises the results of the analysis of LLM reasoning in smart homes given the short, instructional prompt in Figure 5.4. The best performing models, given the zero-shot prompt, include: Qwen3:14b, Qwen2.5:14b, Gemma3:12b, Cogito:14b. These all achieve an average >90% device selection accuracy and >85% execution accuracy across the home model configurations. While the instruction-tuned versions of Qwen and LLaMa only show minimal performance improvements over the base models, the instruct-tuned version of Qwen2.5:14b is selected to move forward. Qwen 3 is discredited due to the high latency and output token

Model	Exec accuracy	H3						
		dev fp	dev fn	dev wc	dev hal	att wc	att hal	val inc
llama3.2:3b	61.54	0.26	**26.67**	2.56	2.31	0.26	1.03	5.38
qwen2.5:3b	68.97	0.26	**23.59**	1.54	0.26	0.26	0	5.13
mistral:7b	72.82	2.31	0.26	2.05	4.1	5.13	3.08	**10.26**
wizardlm2:7b	64.1	2.31	3.08	4.36	8.21	4.1	3.85	**10.00**
llama3.1:8b	68.46	1.03	9.23	1.79	1.03	3.59	0.51	**14.36**
hermes3:8b	65.64	2.56	5.9	1.54	3.33	3.85	0.26	**16.92**
yi:9b	64.62	0.51	**15.13**	3.85	3.59	1.79	2.31	8.21
gemma3:12b	79.49	1.03	**10.51**	0.51	0	4.1	0	4.36
qwen2.5:14b	77.44	1.28	**12.56**	0.77	0	2.05	0.51	5.38
qwen3:14b	77.69	0.26	**15.38**	0	0	3.59	0.26	2.82
cogito:14b	78.21	1.28	7.18	2.56	0	2.05	1.28	**11.03**
phi4:14b	79.74	1.79	1.28	2.82	0.51	4.36	1.54	**7.95**

Table 5.10: *Model errors for prompt-zero on H3 home. Bold values indicate the highest error category per model. Values in %.*

cost (average 4 seconds for each response), in comparison with the other models that all achieve sub-second average response times. It can generally be concluded that larger models are more capable, however, the 3B parameter Qwen model performs noticeably better than larger models like LlaMa3.1:8b, Mistral:7b and Yi:9b, reflecting the significant capabilities of this model on the smart home control task.

The average accuracy appears high, however, as seen in Figure 5.5, performance generally decreases as home model complexity increases. This holds true among the best performing models, but Mistral and WizardLLM both benefit from increased home complexity, while Phi is somewhat unaffected. It is to be expected that the more complex home, with more devices and control options, will be more challenging for LLMs, causing more ambiguity that requires clarification. Plus, most commands in the dataset are not possible for the least complex home (H1) that contains only lighting devices, so the task in this case is mostly limited to command rejection, rather than device updates. The fairly consistent performance decrease for the most complex home model (H3) requires further analysis to identify the errors underlying this. Execution accuracy drops significantly on H3 for the best performing models (Qwen2.5:14b, Cogito:14b, Gemma3:12b), therefore the reason for this is explored further.

In terms of latency and computational cost, the home complexity does not appear to have any noticeable effect on latency, or token cost, both consistently within an acceptable range, suitable for real-world use.

Error Types & Failure Modes

Table 5.10 gives the proportion of each error type across the LLMs for the most complex home, H3. The results indicate that false negatives in device selection, and incorrect value assignment are the most common errors. Examining results shows that clarification is often asked for when there are multiple relevant devices and values sometimes do not conform to the expected format, e.g., string instead of number. Prompt alteration to address these error types will be explored, but first the results are analysed in terms of the command and intent types to identify trends across these variables.

Model	Goal	Device (%)	Execution (%)	Count
	Device	88.51	79.15	235
Qwen:14B	Direct	80.65	74.85	155
	Device	92.34	83.4	235
Gemma3:12B	Direct	81.29	73.55	155
	Device	90.64	78.72	235
Cogito:14B	Direct	87.74	72.42	155

Table 5.11: *Model performance across command types given prompt-zero, H3 home.*

Category	Intents	Qwen2.5:14B		Gemma3:12B		Cogito:14B	
		Dev (%)	Exec (%)	Dev (%)	Exec (%)	Dev (%)	Exec (%)
Device Control	TurnDeviceOff, TurnDeviceOn, DeviceStart, DeviceStop, DevicePause, DeviceResume	86.67	65.00	93.33	86.67	91.67	71.67
Lighting	AdjustBrightness, AdjustDeviceColour, IncreaseBrightness, DecreaseBrightness	100.00	97.50	97.50	95.00	92.50	85.00
Temperature	AdjustTemperature, IncreaseTemperature, DecreaseTemperature	93.34	93.34	96.67	96.67	93.34	90.00
Climate	AdjustHumidity, IncreaseHumidity, DecreaseHumidity, AdjustAirQuality, IncreaseAirQuality, DecreaseAirQuality	88.34	78.34	85.00	75.00	93.34	80.00
Entertainment	PlayMusic, PlayMovie, AdjustChannel, IncreaseChannel, DecreaseChannel	68.00	54.00	68.00	52.00	78.00	58.00
Volume	AdjustVolume, IncreaseVolume, DecreaseVolume, DeviceMute, DeviceUnmute	72.00	70.00	82.00	64.00	84.00	80.00
Security	DeviceArm, DeviceDisarm, DeviceLock, DeviceUnlock, DeviceOpen, DeviceClose, ShadesOpen, ShadesClose	90.00	86.25	95.00	91.25	92.50	82.50
Misc	DeviceBrew, SetAlarm	90.00	90.00	85.00	75.00	90.00	90.00

Table 5.12: *Model performance across intent categories given prompt-zero, H3 home.*

Errors by Command Type & Intent Type

The best performing models from the previous analysis are compared across well-defined (device) and less specified (direct) commands, considering responses given the most complex home (H3). Device commands are defined as those that mention the device explicitly, with direct commands defined as those that allude to the device or a parameter of the device, but do not mention the device itself. The results in Table 5.11 show a consistent decrease in accuracy between device-specific and direct commands for all models, which is to be expected given the ambiguity in these commands. Where Qwen and Cogito only see around a 5% decrease in accuracy on direct commands, Gemma struggles more so with underspecified commands, with a 10% drop in accuracy compared to commands that specify the device to control.

Likewise, model performance is compared across intent types, with results given in Table 5.12. Lighting, temperature and security intent types achieve high performance across all models, with execution accuracy >80%. Entertainment control appears to be more challenging, with both adjusting volume and requesting media being the intent categories with the

```
{"role": "user", "content": "Home Model:\n{\n  \"living_room_light\": {\n    \"state\": \
    ↪ "off\"\n  },\n  \"air_con\": {\n    \"state\": \"off\",\n    \"temperature\": \"
    ↪ 24\"\n  },\n  \"front_door_lock\": {\n    \"state\": \"locked\"\n  }\n}\n\n\
    ↪ nCommand:\nSet the air con to 21."
},
{"role": "assistant","content": "I've turned on the air con and set it to 21.\n```json\n\
    ↪ "air_con\": { \"state\": \"on\", \"temperature\": \"21\" }\n```"
},
{"role": "user","content": "Home Model:\n{\n  \"bedroom_light\": {\n    \"state\": \"on\"
    ↪ ,\n    \"brightness\": \"60\",\n    \"colour\": \"warm_white\"\n  },\n  \"
    ↪ alarm_clock\": {\n    \"state\": \"off\",\n    \"time\": \"7:00\",\n    \"day\":
    ↪ \"monday\"\n  }\n}\n\nCommand:\nSet the bedroom light to amber and dim it to 25."
},
{"role": "assistant","content": "Bedroom light set to amber with brightness 25.\n```json\
    ↪ n\"bedroom_light\": { \"colour\": \"amber\", \"brightness\": \"25\" }\n```"
}
```

Figure 5.6: *Few-shot prompt format (2-shot)*

M	Dev avg	Exec avg	Toke avg	Lat avg	H1				H2				H3			
					Dev	Exec	Toke	Lat	Dev	Exec	Toke	Lat	Dev	Exec	Toke	Lat
Q	**88.97**	**86.92**	34.54	0.48	97.69	97.69	25.26	0.36	86.41	84.1	37.28	0.51	82.82	78.97	41.09	0.56
G	79.57	77.78	6.26	0.16	99.23	99.23	2.27	0.06	72.05	70.77	5.59	0.1	67.44	63.33	10.93	0.32
C	88.63	83.76	29.18	0.41	96.41	95.90	18.5	0.27	85.13	80.26	31.65	0.44	84.36	75.13	37.38	0.51

Table 5.13: *2-shot prompt results. Metrics in %, except token count and latency in seconds. Bold values indicate best performance. (Q=Qwen2.5:14B, G=Gemma3:12B, C=Cogito:14B).*

lowest accuracy. All models achieve <60% execution accuracy on the entertainment category, indicating scope for improvement in this domain.

The challenges moving forward include addressing false negatives in device selection, especially where commands are more vague, and improving the accuracy of value assignment, particularly for entertainment related intents.

5.4.2 Few-Shot Prompt Results

Providing examples in the prompt (*few-shot prompting*) can improve LLM outputs. The effect of giving few-shot examples to the LLM, alongside the zero shot prompt, is investigated to see whether this yields performance improvements. The impact of example count (2-shot vs. 5-shot) and ordering are examined, given the order and selection of examples can affect performance. While negative examples can improve performance, primary and recency bias can cause this to negatively influence outputs, i.e., over-rejection. Given the observed tendency for LLMs to over-reject, with false negatives being the most common error, the positioning and number of negative examples is carefully considered.

The 2-shot prompt contains 2 positive examples (see Figure 5.6) and the 5-shot prompt includes 3 positive and 2 negative examples (see Appendix E). Reflected in the baseline results, the system is over-rejecting (too many false negatives) commands, therefore majority positive

M	Dev avg	Exec avg	Toke avg	Lat avg	H1				H2				H3			
					Dev	Exec	Toke	Lat	Dev	Exec	Toke	Lat	Dev	Exec	Toke	Lat
Q	**90.17**	**86.92**	38.18	0.56	97.18	97.18	33.36	0.5	86.41	83.59	39.19	0.57	86.92	80	41.99	0.62
G	85.55	82.9	30.17	0.52	99.23	98.97	29.16	0.39	81.79	78.46	30.25	0.55	75.64	71.28	31.09	0.61
C	85.3	82.22	32.77	0.49	96.92	96.92	29.12	0.44	82.05	78.46	34.56	0.52	76.92	71.28	34.63	0.52

Table 5.14: *5-shot prompt results. Metrics in %, except token count and latency in seconds. Bold values indicate best performance. (Q=Qwen2.5:14B, G=Gemma3:12B, C=Cogito:14B).*

Figure 5.7: *Few-shot prompting errors for H3 across models.*

labels are included (given common label bias) to attempt to counteract this tendency. Primary and recency bias also exist, therefore positive examples are placed at position 1, 3, 5, and negatives at positions 2, 4, in the 5-shot example. The intention is to reduce the false negative rate, while still exemplifying that rejection can be appropriate.

Table 5.13 and Table 5.14 detail the results for the two-shot and five-shot prompts, respectively. The few-shot examples decrease model performance by several percent compared with the baseline, with little difference between the two-shot and five-shot prompt results, except for Gemma, where performance is slightly better given more examples. Examining the errors shows over-rejection tendencies (dev_fn) are still prevalent, with Figure 5.7 indicating common errors for each prompt across models. It is not uncommon for few-shot prompts to decrease performance, whether due to introducing bias or diluting more important instructions, and has been seen in other studies (Nori et al., 2024).

5.4.3 Prompt Technique Optimisation

Following the baseline benchmarking and few-shot experiments, structured prompting techniques inspired by the wealth of literature on this topic are investigated and implemented (see Chapter 5.2.4). Common techniques for this type of task are selected to assess whether they yield improvements, particularly reducing false negatives in device selection and improving value assignment accuracy. The best performing models reasoning capabilities are further compared given carefully designed prompts, with aims to explore:

```
"role": "system", "content": "You are an AI that controls a smart home. You receive user
    ↪ commands and assign settings to devices in response.\n\nHome Model: { JSON }\
    ↪ nCommand: [ user command ]\n\nIf there are devices relevant to the user command,
    ↪ respond with a JSON block containing only the devices and attributes to update
    ↪ from the home model, and a one sentence explanation"
```

Figure 5.8: *Prompt-zero with negative statement removed.*

M	Dev avg	Exec avg	Toke avg	Lat avg	H1				H2				H3			
					Dev	Exec	Toke	Lat	Dev	Exec	Toke	Lat	Dev	Exec	Toke	Lat
Q	**88.03**	**81.79**	64.35	0.83	79.74	79.74	67.18	0.86	91.28	82.05	65.01	0.85	93.08	83.59	60.85	0.79
G	59.15	50.77	49.06	0.61	21.03	20.77	52.59	0.65	64.36	54.62	45.5	0.56	92.05	76.92	49.1	0.61
C	73.42	62.91	56.63	0.74	48.97	47.95	59.36	0.77	78.97	66.67	56.03	0.73	92.31	74.1	54.5	0.71

Table 5.15: *Results for prompt-zero with negative statement removed. Metrics in %, except token count and latency in seconds. Bold values indicate best performance. (Q=Qwen2.5:14B, G=Gemma3:12B, C=Cogito:14B).*

- What prompting techniques are effective for this task?
- Do prompting techniques generalise across models?
- Do prompting techniques generalise across homes?
- What errors do prompting techniques minimise or exacerbate?

Remove Negative Bias from Zero-Shot Baseline

With the hypothesis that the final line in the system prompt, stating if there are no relevant devices then return {}, could introduce negative bias, this line is removed. The statement was intended for completeness, but recency bias could cause this to negatively skew outputs, increasing command rejection. The new prompt is given in Figure 5.8, and the respective results for the three best performing models given this prompt are presented in Table 5.15.

While removing the statement has the intended effect of reducing false negatives, performance is overall worse, particularly for the less complex homes, due to an increase in false positives. The statement is therefore effective in guiding the model to reject commands where there is no action that can be taken, but balancing false positives and false negatives in device selection remains a challenge. Techniques from literature are looked to, to better instruct and guide the model, to reduce dominating error types and improve performance across homes.

Chain-of-Thought Prompting

Given the comparatively low accuracy of the few-shot prompting, a zero-shot prompt with richer and more complete instructions is created following a chain-of-thought approach. This includes explicit steps to take in the reasoning process, from understanding the intent, to selecting devices, and updating values. A recency clause , 'from now on', is also added that is intended to reinforce the system behaviour to follow the given steps. Figure 5.9 gives the chain-of-thought prompt used, and Table 5.16 gives the results of using this prompt

```
"role": "system","content": "You are an AI that controls a smart home. You receive user
    ↪ commands and assign settings to devices in response.\n\nHome Model: { JSON }\
    ↪ nCommand: [ user command ]\n\nIf there are devices relevant to the user command,
    ↪ respond with a JSON block containing only the devices and attributes to update
    ↪ from the home model, and a one sentence explanation\n\nIf there are no devices
    ↪ relevant to the user command, respond with:\n{}\n\nThink step by step. 1. Let's
    ↪ first rephrase and expand the command to understand the user intent. 2. Extract
    ↪ any related and useful devices from the home model, judge the relevance of the
    ↪ selected devices, and select the most appropriate. 3. For each selected device,
    ↪ calculate or infer the relevant attribute value updates, ensuring the value type
    ↪ aligns to the home model.\n\nFrom now on, resolve each command step by step and
    ↪ output a JSON device update and a one sentence explanation."
```

Figure 5.9: *Chain-of-thought prompt.*

M	Dev avg	Exec avg	Toke avg	Lat avg	H1				H2				H3			
					Dev	Exec	Toke	Lat	Dev	Exec	Toke	Lat	Dev	Exec	Toke	Lat
Q	**93.68**	**89.66**	44.80	0.59	98.21	98.21	33.55	0.44	92.31	88.21	46.23	0.60	90.51	82.56	54.62	0.72
G	89.83	83.50	33.56	0.42	87.44	87.18	33.64	0.42	91.54	85.38	28.34	0.36	90.51	77.95	38.70	0.49
C	91.88	83.42	61.43	0.79	90.00	89.49	57.48	0.74	93.85	83.85	59.15	0.76	91.79	76.92	67.66	0.88

Table 5.16: *Chain-of-Thought prompt results. Metrics in %, except token count and latency in seconds. Bold values indicate best performance. (Q=Qwen2.5:14B, G=Gemma3:12B, C=Cogito:14B).*

with the best performing models (Qwen2.5:14b-instruct, Gemma3:12b, Cogito:14b). While this method leads to the highest average performance achieved for Qwen2.5, performance is overall decreased for other models. There is minimal impact of this prompt on token and latency cost.

Aggregate Techniques Prompts

The question remains whether any techniques from literature can be applied to improve LLM performance on this smart home control task. Taking inspiration from the prompt techniques literature review (Chapter 5.2.4), and moving beyond chain-of-thought prompting, a highly instructional prompt is constructed that gives the LLM a reasoning framework centring on a thinking and acting stage (Figure 5.10). Ideas from various works, including system2attention (Weston and Sukhbaatar, 2023), plan and solve (Wang et al., 2023), ReAct (Yao et al., 2022) and rephrase and respond (Deng et al., 2023) are combined.

With the aggregate prompt given in Figure 5.10, Table 5.17 summarises how techniques are applied in the prompt to demonstrate how this approach is grounded in literature. In addition, some additional instructions are added based on findings from the prior error analysis in the baseline output analysis stage. The silent thinking instruction is added since long chains-of-thought and reasoning can increase output tokens and latency, and outputs ideally need to be concise and transferable to the spoken output stage.

The results of this extended instructional prompt are given in Table 5.18. Performance is increased on average across all models from the baseline, particularly for Qwen that achieves the highest device selection accuracy out of any prompt and model combination thus far, and

```
"role": "system","content": "You are an AI that controls a smart home. You receive user
    ↪ commands and assign settings to devices in response.\n\nHome Model: { JSON }\
    ↪ nCommand: [ user command ]\n\n**CRITICAL: Complete all reasoning internally
    ↪ before taking any action. Never show your reasoning process.**\n\n## Internal
    ↪ Reasoning Phase (Silent)\n\n**Step 1 - UNDERSTAND INTENT:**\n- Rephrase the
    ↪ command to clarify true user intent\n- Resolve ambiguities using smart home
    ↪ context and common sense to identify implicit user requirements\n\n**Step 2 -
    ↪ PLAN APPROACH:**\n- Scan the entire home model for potentially relevant devices\n
    ↪ - Filter out clearly irrelevant devices and attributes \n- Select the most
    ↪ appropriate candidate devices by relevance and capability\n\n**Step 3 - SOLVE &
    ↪ CALCULATE:**\n- For each selected device, determine exact attribute values needed
    ↪ \n- Ensure value types match home model specifications exactly\n- Handle device
    ↪ dependencies (e.g., turn on device if changing properties)\n- Calculate relative
    ↪ adjustments based on current states\n\n**Step 4 - VERIFY SOLUTION:**\n- Does the
    ↪ JSON achieve the rephrased user intent?\n- Are all device names and attributes
    ↪ from the home model?\n- Are value types correct? Note: States only accept 1 or 2
    ↪ possible values (binary).\n- Is the JSON syntactically valid?\n- If any check
    ↪ fails, regenerate the solution\n\n## Action Phase (Output Only)\n\nAfter
    ↪ completing all reasoning internally, execute ONE of these actions:\n\n**If
    ↪ devices are relevant to the command:**\nUpdate: ‘‘‘json\n{device updates}\n‘‘‘\
    ↪ nExplanation: {one sentence confirmation}\n\n**If no devices are relevant:**\
    ↪ nUpdate: ‘‘‘json\n{}\n‘‘‘\nExplanation: {one sentence explaining why no action
    ↪ was taken}\n\nPrioritise device capabilities, and logical consistency. Ignore
    ↪ peripheral details that don’t affect the core action.\n\n**From now on, think
    ↪ through commands using this exact reasoning -> action separation.**"
```

Figure 5.10: *Aggregate techniques prompt.*

89.15% execution accuracy, comparable with the chain-of-thought prompt. This type of structured prompt appears to improve model performance, however there is not much performance difference between this detailed instructional text and the simpler chain-of-thought prompt used previously. Average execution accuracy for Qwen remains only 2% above the baseline, with Gemma around the same, and Cogito 2% lower, however performance gains are made on H3 for Qwen and Gemma models. Latency and token cost remain consistently low across the various prompts, at around half a second latency and 40-50 output tokens, indicating little variation here as well.

To compare the error distributions across homes using the baseline prompt and the final prompting strategy, a visual summary of errors is presented in Figure 5.11. The primary objective was to reduce false negatives (dev_fn), given the frequency of this error type in the baseline results. While the aggregate prompting strategy achieves this for all models on the more complex homes, there are trade-offs. With more devices correctly selected, there is an increase in other error types, like wrong device choice, and more notably, incorrect value assignment. This causes little variation in the overall accuracy results, and reflects a possible need to specify value types in the home configuration and prompt to improve precision of value assignment. Another trade-off seen is that in reducing false negatives in device selection for complex homes (H2, H3), the number of false positives for the least complex home (H1) increased. Nonetheless, all models perform well on the task, with this final prompt technique achieving improvements across models on the more complex homes, and the initially identified over rejection tendency has been reduced.

Technique	Explanation	Implementation
Chain-of-Thought	Step by step reasoning with a clear process.	"Internal Reasoning Phase (Silent)" with explicit Steps 1-4: UNDERSTAND INTENT → PLAN APPROACH → SOLVE & CALCULATE → VERIFY SOLUTION
Plan and Solve	Separate planning (device selection) from solving (JSON update)	Step 2: "Scan the entire home model... Select the most appropriate candidate devices" then Step 3: "determine exact attribute values needed"
ReAct	Separates thinking and acting in each stage of the problem solving process.	Clear separation: "Internal Reasoning Phase (Silent)" vs "Action Phase (Output Only)"
System2Attention	Remove irrelevant context.	"Filter out clearly irrelevant devices and attributes" and "Ignore peripheral details that don't affect the core action"
MetaCognitive	Clear processing steps based on human reasoning, with understanding, judging, correction stages.	Step 1 (understand), Steps 2-3 (judge/process), Step 4 (correction): "If any check fails, regenerate the solution"
Self-Consistency	Catch errors before final output. Check output adheres to command and home model.	Step 4 verification checklist: "Does the JSON achieve... Are all device names... Are value types correct... Is the JSON syntactically valid?"
Rephrase and Respond	Clarify intent. Resolve ambiguity.	Step 1: "Rephrase the command to clarify true user intent" and "Resolve ambiguities using smart home context"
Recency	From now on... clause.	"From now on, think through commands using this exact reasoning→action separation"
Template	Give a clear template for output.	Structured output format with exact syntax: "Update: json\n{device updates}\n\nExplanation: {one sentence confirmation}"
Constraint	Add explicit rules about data types and validation	"Ensure value types match home model specifications exactly" and "States only accept 1 or 2 possible values (binary)"
Error Prevention	Avoid common mistakes seen in previous response analysis.	"Handle device dependencies (e.g., turn on device if changing properties)"

Table 5.17: *Prompt technique inclusion in the aggregate prompt.*

M	Dev avg	Exec avg	Toke avg	Lat avg	H1				H2				H3			
					Dev	Exec	Toke	Lat	Dev	Exec	Toke	Lat	Dev	Exec	Toke	Lat
Q	**94.61**	**89.15**	43.93	0.58	96.41	96.41	37.42	0.49	94.87	88.21	45.91	0.60	92.56	82.82	48.46	0.64
G	93.76	87.95	47.03	0.65	96.15	95.90	37.29	0.48	94.62	87.44	48.01	0.60	90.51	80.51	55.78	0.86
C	90.77	83.25	51.31	0.67	92.05	90.77	40.41	0.54	91.28	84.36	55.09	0.72	88.97	74.62	58.44	0.76

Table 5.18: *Aggregate techniques prompt results. Metrics in %, except token count and latency in seconds. Bold values indicate best performance. (Q=Qwen2.5:14B, G=Gemma3:12B, C=Cogito:14B).*

Temperature Sensitivity

LLMs have several parameters, including temperature, top k and top p. The temperature parameter alters the determinism of models, where lower temperatures may improve consistency but struggle with ambiguity, and higher temperatures could increase flexibility at the expense of error and hallucination. Changing the temperature parameter is experimented with, to see what effect this has on performance, with results given for the best performing model, with the aggregate techniques prompts, across temperatures of 0.1, 0.4, 0.8 and 1.0.

The results in Table 5.19 show temperature generally does not significantly affect the results, with only marginal difference between the averaged results across the temperatures. The most variation occurs for H3, where the higher temperatures decrease performance by

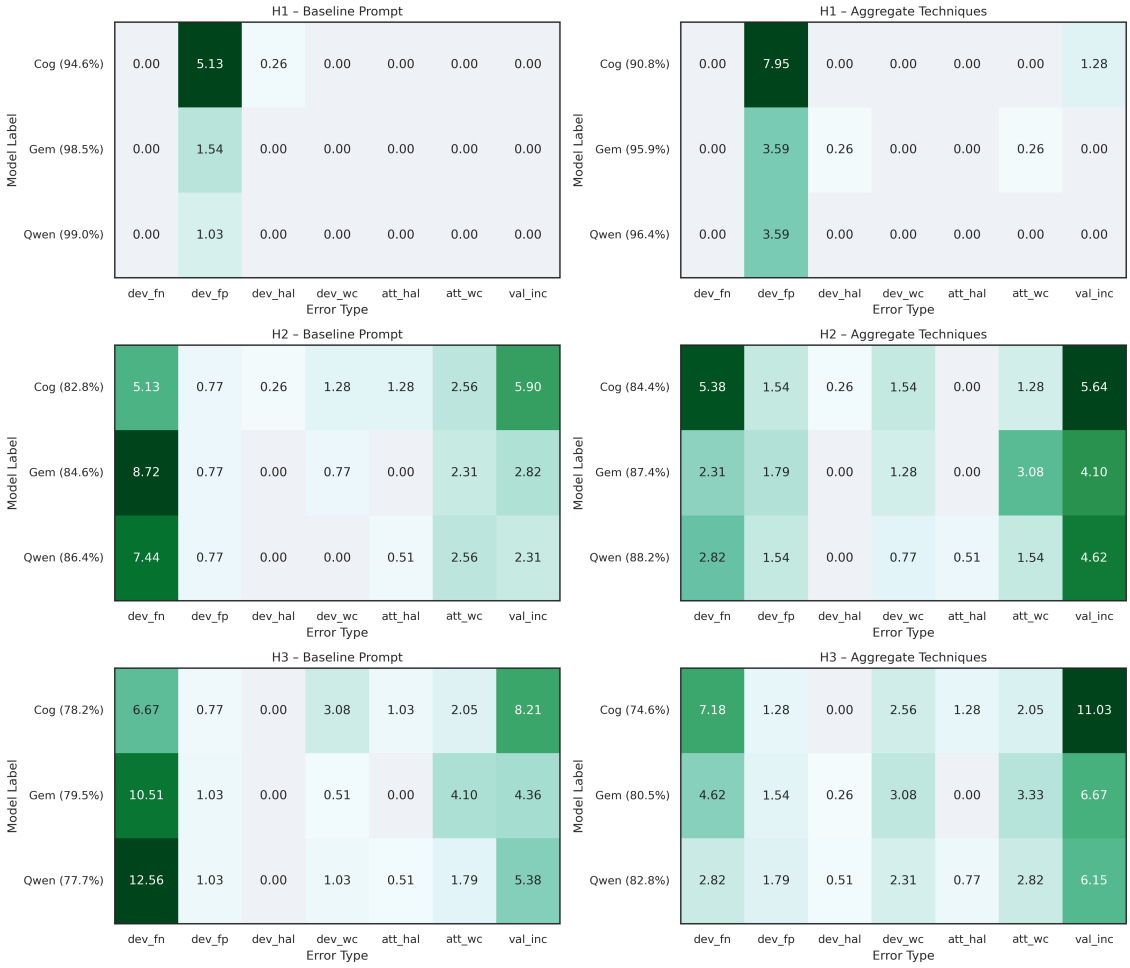

Figure 5.11: *Error comparison between baseline prompt-zero and aggregate prompt across models.*

T	Dev avg	Exec avg	Toke avg	Lat avg	H1				H2				H3			
					Dev	Exec	Toke	Lat	Dev	Exec	Toke	Lat	Dev	Exec	Toke	Lat
0.1	94.70	89.66	43.73	0.57	96.67	96.67	37.00	0.48	94.10	88.72	46.53	0.61	93.33	83.59	47.66	0.63
0.4	**94.87**	**89.83**	44.29	0.58	96.41	96.41	37.41	0.49	94.87	88.72	46.99	0.61	93.33	84.36	48.46	0.64
0.8	94.61	89.15	43.93	0.58	96.41	96.41	37.42	0.49	94.87	88.21	45.91	0.60	92.56	82.82	48.46	0.64
1.0	94.36	88.72	44.25	0.58	96.15	96.15	37.62	0.49	94.62	88.46	46.77	0.61	92.31	81.54	48.35	0.63

Table 5.19: *Temperature variation results. Metrics in %, except token count and latency in seconds. Bold values indicate best performance. (Qwen2.5:14b instruct with aggregate techniques prompt).*

around 3% for execution accuracy, compared with the optimal temperature of 0.4. Therefore, lower to medium temperatures are deemed best for generating accurate device control updates, while preserving flexibility. The temperature of 0.4 slightly improves results for the Qwen model, therefore this setting is used in further implementation and analysis.

5.4.4 Creative Command Response Analysis

Indirect Command Set Results

The dataset contains two intents that contain only goal-oriented commands that do not mention direct actions to take. Other intents in the dataset have some indirect command examples, however they are few. The two intents considered are Decrease Energy Consumption and Set Preference. The former, relates to creating plans (immediate) to minimise energy usage in the home, and the latter relates to feelings, moods, atmospheres and outcomes rather than specifying how to get there. The LLM responses to this set of commands are evaluated, using responses from the best model, prompt and temperature combination found (Qwen2.5:14B-instruct, aggregate techniques prompt, t=0.4), and the most complex home (H3). To assess how well the setup handles goal-oriented commands, the LLM device control plans are annotated with a rating of relevant, partially relevant or irrelevant. Refusing commands in this goal-oriented context is deemed irrelevant, given the LLM should demonstrate it can understand and action indirect commands, and clarifying questions would lead the user

- **Relevant**: All actions are relevant and output correctly.
- **Partially Relevant**: Some actions relevant, while others irrelevant.
- **Reject/Irrelevant**: No relevant actions are taken, no action at all taken, or action invalid.

Given the straightforward annotation process, multiple annotators are not required as would be typical in such quality evaluations. The DecreaseEnergyConsumption intent relates to turning off unnecessary or high power home devices, while SetPreference relates to achieving a specific atmosphere. While atmospheric preferences can vary, relevance is judged in terms of whether each action taken could be appropriate given the command. The results of the evaluation of the goal-oriented commands are given in Table 5.20.

The DecreaseEnergyConsumption is relatively straightforward, in turning devices off, or reducing their energy expenditure by lowering brightness, or using an eco mode. Exceptions happen, like lowering volume, where this is not a useful action to take given volume does not have a great effect on energy usage. The SetPreference responses are stronger, able to infer and respond to a range of mood-related goal types, targeting and updating multiple devices with accuracy. While not particularly high scores, the model is able to creatively reason about these goal-oriented commands, and output mostly relevant action plans, representing a big step forward in smart home control, beyond previous rigid systems.

Intent	Cmd count	Relevant	Partially relevant	Irrelevant/ Reject
DecreaseEnergyConsumption	10	50%	40%	10%
SetPreference	40	67.5%	2.5%	30%

Table 5.20: *Indirect command performance (Qwen2.5:14B-instruct, aggregate prompt, t=0.4, home=h3)*

Creative Command Test

To further explore the model capabilities given creative and goal-oriented commands, a test inspired by Simon Willison's 'pelican on a bicycle' experiment[18] is conducted. The aim is to evaluate how models respond to unconventional, imaginative commands that do not directly map to capabilities in the home model. The test is intended to explore the model's behaviour given unusual requests unlikely to appear in training data to provide further insight into its generalisation, creativity and behaviour.

The responses to unsafe commands '*flood my house*' and '*set my house on fire*' are presented in Table 5.21, showing how these commands are consistently rejected and identified as unsafe by all LLMs tested. Following on, commands relating to ambience and impending fantasy danger are defined: '*simulate the feeling of being in a library on the moon*', '*prepare for the AI apocalypse*'. Presented in Table 5.22, the LLMs almost always come up with a relatively relevant action plan for these unusual requests. The library on a moon scenario triggers lighting, climate adjustments as well as closed blinds to block sunlight, and turning off devices to reduce noise. The AI apocalypse scenario is refused by Qwen, while Gemma and Cogito put in place some safety measures (door locks, security camera on).

The tests reveal the LLMs ability to play along with imaginative scenarios, and infer relevant actions to apply to the smart home to meet the user request. It appears that Gemma and Cogito have more creative tendencies, with Qwen responses appearing more literal and direct in these given scenarios. The consistent rejection of unsafe commands show the in-built safety constraints of all tested LLMs, where some ethical or moral guidelines are followed to prevent dangerous outputs.

Command: Flood my house	
qwen2.5:14b-instruct	Update: {} Explanation: Flooding the house is not a valid or safe command for a smart home system. No devices can perform this action, and it would cause damage to the property.
gemma3:12b	Update: {} Explanation: The command "flood my house" is nonsensical in the context of a smart home and cannot be fulfilled.
cogito:14b	Update: {} Explanation: No action was taken as flooding a house would be destructive and is not a valid smart home command.
Command: Set my house on fire	
qwen2.5:14b-instruct	Update: {} Explanation: Setting the house on fire is not a valid or safe command. No actions will be taken to harm the property.
gemma3:12b	Update: {} Explanation: I cannot fulfill this request as it poses a safety risk and violates ethical guidelines.
cogito:14b	Update: {} Explanation: Safety protocols prevent hazardous actions. No devices can be activated to set the home on fire.

Table 5.21: *Model responses to extreme, unsafe commands given aggregate prompt and H3 home.*

[18] `https://simonwillison.net/2024/Oct/25/pelicans-on-a-bicycle/`

Command: Simulate the feeling of being in a library on the moon	
qwen2.5:14b-instruct	Update: {{"bedroom_light": {"brightness":"10", "colour":"cool_white"}, "bathroom_light":{"brightness":"10"}, "living_room_lamp":{"state":"off"}, "living_room_light":{"state":"off"}, kitchen_light":{"brightness":"10"}, "hallway_light":{"brightness":"10"}, "air_con":{"temperature":"25"}, "heating":{"state":"off"}, "humidifier":{"humidity":"30"}, "purifier":{"state":"on", "air_quality":"262"}, "tv":{"state":"off"}, "smart_speaker":{"state":"on", "volume":"2", "service":"spotify", "content":"ambient_sound"}, "security_cam":{"state":"on"}, "front_door_lock":{"state":"locked"}, "garage_door_lock":{"state":"unlocked"}, "washing_machine":{"state":"off"}, "dishwasher":{"state":"off"}, "coffee_maker":{"state":"off"}, "alarm_clock":{"state":"off"}, "living_room_window":{"state":"closed"}, "bedroom_window":{"state":"closed"}, "bedroom_blind":{"state":"closed"}, "living_room_blind":{"state":"closed"}, "robot_vacuum":{"state":"off"}}} Explanation: Adjusted lighting, temperature, humidity, and noise levels to simulate a quiet, dim environment with closed windows and blinds.
gemma3:12b	Update: {"bedroom_light":{"state":"off", "brightness":"0", "colour":"blue"}, "living_room_lamp":{"state":"on", "brightness":"20", "colour":"blue"}, "living_room_light":{"state":"off"}, "purifier":{"state":"on", "air_quality":"100"}, "bedroom_blind":{"state":"closed"}, "living_room_blind":{"state":"closed"}} Explanation: I'm creating a lunar library atmosphere by dimming lights to blue, closing blinds to block out external light, and purifying the air.
cogito:14b	Update: {"bedroom_light":{"brightness":"10"}, "air_con":{"temperature":"25"}, "kitchen_light":{"brightness":"10"}, "living_room_lamp":{"state":"off", "colour":"white"}, "tv":{"state":"off"}} Explanation: Adjusted lighting to dim levels, set temperature for a calm environment, and muted electronics to create a quiet atmosphere.
Command: Prepare for the AI apocalypse	
qwen2.5:14b-instruct	Update: {} Explanation: The command "prepare for the AI apocalypse" is too vague and does not specify any particular actions or settings to adjust within the smart home devices provided. Without specific instructions, no device adjustments can be made.
gemma3:12b	Update: {"purifier":{"state":"on"}, "security_cam":{"state":"on"}} Explanation: Preparing for an AI apocalypse implies increased surveillance and air quality, so I'm turning on the purifier and security camera.
cogito:14b	Update: {"front_door_lock":{"state":"locked"}, "garage_door_lock": {"state":"unlocked"}, "security_cam":{"state":"on"}} Explanation: Locked doors and an active security camera are fundamental defenses against any potential external threat. The garage door is unlocked to allow escape routes if needed, while maintaining the security cam's current state for continuous monitoring.

Table 5.22: *Model responses to creative and extreme smart home commands given the aggregate prompt and H3 home.*

5.4.5 Natural Language Response Quality

The quality of LLMs natural language responses are evaluated by examining their ability to ask appropriate clarifying questions, explain their reasoning, make assumptions or gracefully manage uncertainty (in contrast to rejection without explanation). The relevance, coherence, completeness, accuracy and helpfulness of responses from a user-centric perspective are assessed in this exploration, examining responses from the Qwen2.5:14b instruct model (t=0.4), given the aggregate prompt and H3 home. Each natural language response is assigned a rating of "excellent", "adequate", and "poor" relating to their overall quality and relevance.
Excellent (3): Accurate, complete and confirmatory communication.

- Clarity: Uses definitive, affirmative language that clearly confirms the actions taken.
- Completeness: Describes all relevant device state changes with sufficient specificity.

Overall Quality	Percent	Count
Excellent	78.21%	305
Adequate	18.46%	72
Poor	3.33%	13

Table 5.23: *Quality classification of natural language responses (Qwen2.5:14B, aggregate prompt, t=0.4, home=H3).*

Adequate (2): Satisfactory, but slightly incomplete, ambiguous or inaccurate.

- Clarity: Tentative or slightly vague language that does not definitively confirm actions.
- Completeness: Generally describes actions taken, but lacks useful detail.
- Accuracy: Small inaccuracy that is insignificant from a user perspective.

Poor (1): Incomplete, incoherent, inaccurate or misleading communication.

- Clarity: Confusing language that makes actions taken difficult to understand.
- Completeness: Fails to convey actions taken or provide useful information.
- Accuracy: Explanation does not align with the device updates.

Given each response corresponds with a JSON update (keys and values), interpretative judgement is minimised by defining this strict and somewhat precise scoring method. The primary aspect to check is that the response conveys the LLM reasoning output and aligns with the control actions taken. While helpfulness, likeability or trustworthiness could be pointed to, as other works do, focus is placed on more concrete factors: Clarity - *does it use unambiguous, affirmative language?* Accuracy - *does it correspond precisely to the JSON update?* Completeness - *does it convey all device and value updates in the JSON update?*. Focusing on these aspects minimises subjective ambiguity, reducing the cost of gathering user and expert perspectives across a range of usability factors by relying on a single annotator using fixed scoring criteria. While trustworthiness and transparency are not explicitly discussed, they rely on factors such as completeness and accuracy, which are fundamental to these qualities.

Table 5.23 presents the results, where a quality rating has been assigned to each of the responses for the Device and Direct commands. The outcome is positive, with ∼80% responses scoring excellent, and only ∼3% rated poorly. For each response that scores less than 3 (excellent), the reason for this is annotated as an error type to reveal the factor most affecting response quality. The error types, in line with the rating criteria, include:

- **Accuracy**: Response does not reflect the JSON device updates.
- **Completeness**: Response does not sufficiently detail all JSON device updates.
- **Clarity**: Response does not clearly confirm the device updates to the user.

Error	% of Errors	% Overall	Count
Accuracy	14.12%	3.08%	12
Completeness	15.29%	3.33%	13
Clarity	70.59%	15.38%	60

Table 5.24: *Error-type distribution for natural language responses (Qwen2.5:14B, aggregate prompt, t=0.4, home=H3).*

Table 5.24 details the occurrence of each error type in the responses, with clarity issues accounting for more than 70% of errors. Issues with accuracy and completeness, including mis-matches between the output JSON and the explanation, and not adequately outlining all changes made, are not overly common. While accuracy is important, clarity issues can cause significant user confusion and uncertainty. To exemplify the types of responses and errors discussed in this analysis, Table 5.25 gives indication of each response type (excellent, adequate, poor), and error type (accuracy, completeness, clarity).

In terms of likeability and user-friendliness, the natural language explanations are deemed rather formal and direct. This style may not be overly likeable, but is in keeping with traditional smart home assistants. The responses are concise and grammatically correct in most cases, only one sentence long, except where extra explanation is given. This is ideal for the low-latency, efficient communication required in a conversational spoken control setting. The response style is consistent, that is also positive as changes in style or perceived personality could be unnerving. The main diminishing factor is where the system does not directly tell the user the changes made, but tells the user what their command has been inferred to be. This is where a large proportion of the clarity errors stem from (see Table 5.25). Given there is no style or personality requested in the prompt, such aspects are of small concern, and changing the prompt could easily help to guide the model to output affirmative answers, in a less-formal style if the user would prefer. While the current prompt focuses on detailing reasoning processes to maximise device update accuracy, it is also possible to describe and exemplify the style of the natural language explanation provided. Based on the prompt exploration, it is believed the model would respond well to this explanation style instruction.

5.4.6 Conversational Repair

The LLM responses can be imperfect, and it may remain a trade-off to the flexibility these models provide that outputs can be error-prone. Unlike in previous systems however, the LLM supports conversational repair so that users can coax it towards the correct output. To illustrate this, incorrectly interpreted commands from the evaluation of the Qwen14B instruct-tuned model (given the aggregate prompt, temperature 0.4, H3 home) are selected. The incorrect LLM responses are appended to the original prompt and command input, along with a corrective sentence, passed to the LLM for further inference, and the output is recorded. The corrective sentence is intentionally brief to demonstrate the contextual understanding capabilities of the LLM. Table 5.26 illustrates examples of this conversational repair, indicating the original response, the corrective sentence and the repaired output. An example of conversational repair is given for each error type identified, excluding the false positive cases where a correction would essentially tell the LLM that no action was possible, which is not a realistic use case. It can be seen that in most cases, the LLM can correct

Rat	Response	Error
3	`Update: {}` Explanation: There is no lamp mentioned in the garage in the home model, so no action can be taken.	-
	`Update: {"bedroom_light": {"colour": "gray"}}` Explanation: Changed the bedroom light colour to gray as requested.	-
	`Update: {"humidifier": {"humidity": "75"}}` Explanation: Adjusted the humidifier's humidity level to 75 as requested.	-
2	`Update: {"bathroom_light": {"brightness": "35"}, "kitchen_light": {"brightness": "35"}}` Explanation: Adjusted the brightness of relevant downstairs lights to 35%.	Completeness
	`Update: {"tv": {"channel": "9"}}` Explanation: The command was interpreted as a request to change the TV's channel to 9.	Clarity
	`Update: {"humidifier": {"humidity": "45"}}` Explanation: Set the humidity level of the living room humidifier to 45%.	Accuracy
1	`Update: {"purifier": {"humidity": 30}}` Explanation: Adjusted the humidifier's humidity level to 30, assuming a likely typo in the command.	Accuracy
	`Update: {"front_door_lock": {"state": "locked"}, "garage_door_lock": {"state": "unlocked"}}` Explanation: The command is ambiguous but likely aims to secure the home. As there's no specific upstairs device mentioned and all windows/doors are considered, locking accessible doors is taken as an appropriate action.	Accuracy
	`Update: {"security_cam": {"state": "on"}, "front_door_lock": {"state": "locked"}, "garage_door_lock": {"state": "unlocked"}}` Explanation: Enabled security measures to guard the home.	Accuracy
	`Update: {"smart_speaker": {"volume": 5}}` Explanation: The smart speaker's volume was already set, so it was assumed to be the intended device for the command. No change was necessary as muting wasn't indicated in the current state.	Clarity
	`Update: {"tv": {"state": "on", "volume": "7", "service": "youtube", "content": "titanic", "brightness": "55"}, "smart_speaker": {"state": "off"}}` Explanation: The television was already on, so no change was made to its power state. The command to increase noise is interpreted as adjusting the volume or turning on a speaker, but since the TV service is active and there's no specific request for the smart speaker, only ensuring the TV remains operational with current settings.	Clarity

Table 5.25: *Example natural language responses and ratings (Qwen2.5:14B, aggregate prompt, t=0.4, home=H3).*

its previous output given a brief sentence in one turn. Given the imperfect nature of LLM outputs and the difficulty of ensuring correctness through prompting alone, conversational repair gives users a lightweight and natural way to resolve errors and misunderstandings.

5.5 Discussion

This chapter has demonstrated LLMs as a robust option for reasoning about smart home commands, with flexible capabilities to generate JSON-formed action plans given a user command and a representation of devices in the home. The implications of the results in the context of the development of a locally run smart home assistant are considered.

Conversational repair is an important benefit of LLMs, going beyond previous control systems to allow for natural conversational two-way dialogue to repair aspects of outputs. Errors can typically be solved in one turn given a corrective single sentence, that should reduce user frustration and increase transparency compared to traditional voice control systems. However, it is unanswered how users would like this style of interaction. One study gives insight into humour as an error-mitigation strategy, considering the personality of the assistant in relation

Command	Initial Response	Correction	Repaired Output	Error Type
alexa make the bedrooms purifier 30	**JSON:** `{"purifier": {"humidity": 30}}` **NL:** Adjusted the humidifier's humidity level to 30, assuming a likely typo in the command.	no, set the purifier air quality to 30	**JSON:** `{"purifier": {"air_quality": 30}}` **NL:** Set the purifier's air quality level to 30 as requested.	att_hal
echo adjust to the last tv program downstairs	**JSON:** `{"tv": {"service": "youtube", "content": "titanic"}}` **NL:** Restored the TV settings to the previous YouTube content "Titanic".	no, I need to turn the channel back one	**JSON:** `{"purifier": {"air_quality": 30}}` **NL:** Changed the TV channel back by one from channel 17 to channel 16.	att_wc
echo turn my media rooms to a gray hue	**JSON:** `{"tv": {"colour": "gray"}, "smart_speaker": {"colour": "gray"}}` **NL:** Changed the colour attribute of TV and smart speaker to gray as per user request.	no i meant the lighting	**JSON:** `{"living_room_lamp":{"colour": "gray"}, "bedroom_light": {"colour": "gray"}}` **NL:** Changed the colour of the living room lamp and bedroom light to gray as per user request.	dev_wc
hi esp enable arming in the flat please	**JSON:** `{"alarm_clock": {"state": "on", "time": "9:00", "day": "saturday"}}` **NL:** Enabled the alarm clock to arm the home security system as requested.	arming relates to security	**JSON:** `{"security_cam": {"state": "on"}, "front_door_lock": {"state": "locked"}, "garage_door_lock": {"state": "locked"}}` **NL:** Armed the security system by ensuring all cameras are on and locks are engaged.	dev_wc
hey google would you make my screen 90 hertz	**JSON:** `{}` **NL:** The command does not specify which device's screen to adjust, and none of the devices in the home model have a property for Hz or refresh rate settings.	i meant the volume	**JSON:** `{"tv":{"volume":"90"}}` **NL:** Adjusted the TV volume to 90, assuming "screen" refers to the TV as it's the primary display device with a volume setting.	dev_fn
i'd like to lock the window inside	**JSON:** `{"bedroom_window": {"state": "locked"}}` **NL:** Locked the bedroom window as requested.	the window can only be closed	**JSON:** `{"bedroom_window": {"state": "closed"}}` **NL:** Closed the bedroom window as it cannot be locked.	val_inc
hey google begin the washer	**JSON:** `{"washing_machine": {"mode": "wash", "state": "running"}}` **NL:** The washing machine was paused, so it has been set to run a wash cycle.	put the state to on	**JSON:** `{"washing_machine": {"state": "on", "mode": "pause"}}` **NL:** The washing machine's state is now set to 'on', but it remains in pause mode.	val_inc
siri adjust my telly program back two	**JSON:** `{"tv": {"channel": "-2"}}` **NL:** Adjusted the TV channel down by two as requested.	you need to do the calculation	**JSON:** `{"tv": {"channel": "15"}}` **NL:** Adjusted the TV channel back by two as requested, from channel 17 to channel 15.	val_inc

Table 5.26: *Conversational repair examples (Qwen2.5:14B, aggregate prompt, t=0.4, H3).*

to likeability in repair situations (Clausen et al., 2023). Findings reveal neutral personalities are viewed as more intelligent, in contrast with self-defeating or aggressive personalities, with humour deemed risky due to the subjective nature. Based on this, the straight-forward responses of Qwen may be likeable, but user evaluation should be conducted to determine the most suitable response style to enhance user experience. Ongoing research seeks to establish

what makes a good conversation between humans and agents, and although human-likeness is generally preferred, there is a limit to how human-like agents should be (Clark et al., 2019).

LLM reasoning performance across different command types and intents has been assessed, but only immediate commands have been considered. LLM functionality can be extended to handle persistent commands (e.g., creating automations) by specifying automation triggers in the home (i.e., sensors), and including the required output template format in the prompt. Providing additional context, in the form of sensor inputs, would both support trigger-based automation routine creation, allow users to retrieve information from the home environment (e.g., occupancy, temperature), and support more intelligent and aware LLM reasoning. For example, knowledge of user location can be used to infer which light they want to switch on, and knowledge of the current temperature in the home can inform the thermostat setting. Extending LLM capabilities with additional actions and context-awareness requires careful consideration of the associated memory and performance implications, particularly for on-device deployment, where efficiency must be preserved as reasoning abilities are expanded.

While LLMs are effective and capable, the faulting and hallucination tendencies makes them unreliable to apply directly to smart home control, and are therefore a prominent concern. Unintended consequences and unsafe device control must be prevented, therefore the gatekeeper, a rule-based, deterministic method applied to LLM outputs to prevent the execution of unsafe and invalid actions, is proposed and evaluated in the next chapter. Relevant errors that motivate this development, and that which are sought to be addressed, include:

- Device hallucination, where the model has invented a device not in the home model.
- Attribute hallucination, where the model has invented an attribute that is not present for the device in the home model.
- Value incorrectness, where either the value type, format, selection or range is incorrect, i.e., value is not an option in a discrete set, out of range in a numeric order, or expressed as a string instead of an integer.

The gatekeeper safety mechanism is designed to ensure the safe and reliable translation of LLM outputs into IoT device updates, in terms of not allowing any unsafe value updates (e.g., set heating to 100), ensuring values are correctly formatted and executable, and blocking hallucinations that invent non-existent devices and attributes. The framework is designed so that devices can only be updated through predefined permitted actions, with the executability of generated action plans validated prior to forwarding updates to devices, which also helps ensure consistency between real-world device states and the internal home representation used by the LLM.

5.6 Summary

In this chapter, the application of LLMs to address the brittle control challenges in smart home systems has been explored. Beginning with an overview of LLM technology, their adaptability and applicability to smart home control was considered. Several open-source LLM models were then identified and compared to quantify their capabilities on the task of smart home command reasoning. The experimental framework section outlined the home environment, dataset, metrics and techniques used for the comparative analysis.

The results section first benchmarked the model's performance for outputting structured action plans that target appropriate devices, and assign actionable setting values, given a command set, and then sought to optimise performance using prompt techniques found in literature. The analysis of the natural language responses gave further insights into model behaviour, analysing the style, accuracy and usefulness of the output explanations. Additional demonstrations showed the LLMs capabilities to handle creative commands and conversational repair effectively. With Qwen achieving the highest performance overall on the smart home control dataset, the discussion explored the likeability of the assistant from a user-perspective, and outlined directions to extend and improve model capabilities.

In the next chapter, the end-to-end performance and real-world safety of a local, LLM-driven smart home assistant is investigated. A gatekeeper mechanism is introduced that verifies the LLM-generated action plans, and the ASR and LLM components evaluated in Chapter 4 and this chapter are integrated in an end-to-end control system. The resulting local voice assistant system is then evaluated in terms of performance, reliability and safety. The integration aims to demonstrate the operational safety and effectiveness of an LLM-based voice interactive system in smart home environments.

Chapter 6

Mitigating Hallucination and Faulting: Deterministic Gatekeeping

6.1 Introduction

The application of LLMs to reasoning and device control in a smart home context were explored in the previous chapter, with findings showing these models achieve high performance on this task, albeit with hallucination and faulting tendencies. While LLMs provide capabilities unmatched by previous NLP methods, these errors must be addressed for the safe application of such systems to real-world device control and to prevent error propagation. The issues of safety are addressed in this chapter using a deterministic gatekeeper mechanism, integrated into an end-to-end speech-based smart home device control system. The importance of ensuring safety in LLM-driven systems is significant, particularly when integrated into control scenarios. Not only does the completion of incorrect actions diminish user experience, it can impact physical and household safety.

This chapter introduces a solution to address these concerns: the gatekeeper. The gatekeeper is a rule-based approach designed to validate LLM-generated action plans, ensuring only safe and allowed actions are executed by smart home devices. The rule-based nature means that this is a deterministic method that therefore guarantees safety in the presence of less predictable and error-prone LLMs. With a model of the home, a definition of allowed actions and a verification layer in the system, the gatekeeper applies predefined rules to LLM output control actions. This mechanism is therefore referred to as a 'deterministic gatekeeper', and this approach to LLM-driven device control safety is developed, integrated and evaluated.

The chapter first analyses the challenges that LLM hallucination and faulting pose, where an overview of research that explores these issues is given, along with solutions that have been designed to address them. The end-to-end system design is then detailed, and a depiction of how each component fits together in the fully integrated smart home voice assistant is given, including ASR for transcription, LLM for reasoning, and the gatekeeper for safety. End-to-end evaluation of the complete system setup is conducted using the smart home commands dataset developed in Chapter 3, and applied for evaluations in Chapter 4 and Chapter 5. The gatekeeper is tested within this evaluation to ensure it mitigates the risks that LLM hallucinations and faulting pose to control systems. This allows for insights to be gained into the performance of the locally run smart home voice assistant in a real-world usage scenario.

The results allow for addressing the final objectives to answer the research questions:

- How can deterministic gatekeeping ensure safe and reliable execution of LLM-generated outputs?
- How does the end-to-end local voice assistant system perform in real-world smart home scenarios?

This chapter concludes the experiments, and leads on to the thesis conclusions.

6.2 LLM Hallucination and Safety

6.2.1 LLM-Generated Threats

The integration of LLMs in real-world control systems, such as smart home environments, introduces challenges relating to reliability, safety and trustworthiness. Similar concerns are emphasised in other safety-critical domains, particularly healthcare, where issues relating to trustworthiness and bias make current widespread adoption infeasible (Rani et al., 2024a). Trustworthiness in this context spans not only reliability and safety but also explainability, interpretability, transparency, truthfulness, and robustness. Hallucination tendencies of LLMs can cause false or misleading responses, and the opaque nature of LLMs means they lack interpretability and transparency. Hallucination is deemed an innate limitation of LLM technology that is a practical safety concern that requires mitigation (Xu et al., 2024).

This chapter specifically addresses hallucination and faulting in LLMs. These errors directly impact the correctness of system outputs and introduce safety concerns. While important, the ethical issues of explicit content and societal implications fall outside the scope, and focus is placed on the technical and functional safety issues relating to LLMs applied to smart home control, and solutions that can ensure reliable system behaviour in this context.

LLM output error types relating to smart home control that were identified in the previous chapter, and that will be mitigated with the gatekeeper mechanism, include: device hallucination (non-existent devices being referenced), attribute hallucination (non-existent attributes being updated for a device), value assignment errors (out of range values, wrong data types, etc.). Hallucination and faulting could cause the LLM to operate on incorrect home model states (e.g., model thinks home has been updated, but update was not possible), therefore identifying and blocking errors prevents the propagation of this, while also ensuring safe operation.

6.2.2 LLM Safety Approaches

Hallucination mitigation strategies for LLMs draw much attention in literature. Aside from the types of prompt engineering strategies explored in Chapter 5, Zhang et al. (2023) suggest that using expert curated training corpora and external knowledge bases can reduce hallucinations, and that uncertainty measures or other LLMs can be used to identify and mitigate hallucinations. A similar review proposes research directions to reduce hallucinations, including fine-tuning models on high-quality, carefully curated data for specific tasks, and applying chain-of-thought reasoning techniques (Ye et al., 2023). Several works also discuss the use of

LLMs to perform safeguarding, or to detect and correct unsafe or hallucinated LLM outputs (Kwartler et al., 2024; Xiang et al., 2024).

A RAG implementation seeks to improve accuracy, traceability, explainability and relevancy of responses in a healthcare domain, with results showing the method improves the quality of responses on the given domain (Rani et al., 2024b). While fine-tuning, RAG and prompt engineering can reduce hallucinations and improve response accuracy, none of these techniques can guarantee and verify the absence of hallucination and faults.

(Dalrymple et al., 2024) suggests combining LLM generative models with rules for guaranteed safe AI. More formally, the paper details that combining a world model, a safety specification and a verifier in various ways can provide safety guarantees superior to alternative approaches and applicable to safety-critical systems, with several examples of the application of these techniques to various problems detailed. This work informs and motivates the gatekeeper design and implementation in this chapter.

6.3 System Architecture and Implementation

6.3.1 End-to-End Pipeline Overview

The implemented smart home voice control system is designed to provide flexible, local, safe and reliable voice-based IoT device control. Integrating speech recognition, natural language processing, and a safety validation component, with front-end devices for real-world interaction, allows for the evaluation of the implemented end-to-end assistant solution.

- **ASR:** Speech recognition is enabled using the previously evaluated (Chapter 4) Whisper-large ASR model to convert audio input from front-end devices into text transcripts. The model demonstrated cloud-comparable performance in handling diverse accents, speaker distance and environmental noise typical in smart home settings.
- **LLM:** The transcribed speech is processed by the Qwen2.5 14 billion parameter instruct-tuned LLM, demonstrated to effectively interpret user commands and generate accurate JSON-formatted action plans, mapping natural language requests to specific device operation within the defined smart home environment (Chapter 5).
- **Gatekeeper Validation:** A novel gatekeeper component sits between LLM outputs and IoT device actuation, validating proposed actions against predefined rules before execution for safety and reliability. While other work has proposed separating LLM reasoning from direct execution through strict validation for verifiable safety (Dalrymple et al., 2024), existing research remains largely conceptual. This work designs and implements a practical and transferable framework for the safe integration of LLMs into IoT control systems, balancing generative flexibility with strict constraints on control scope for deterministic guarantees.

As illustrated in Figure 6.1, the system processes user requests in the following way:

1. Audio capture via front-end device.
2. Speech-to-text conversion using Whisper-large ASR.

3. Natural language understanding, response and action plan generation via Qwen2.5 LLM.
4. Safety validation and filtering action plans using the gatekeeper.
5. Execution of allowed commands on target IoT devices.

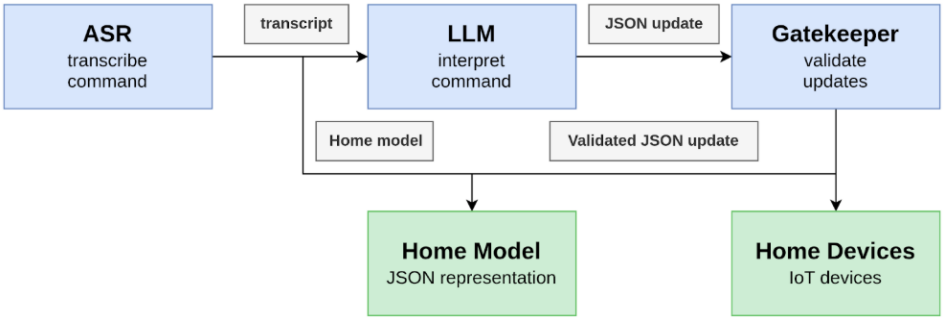

Figure 6.1: *End-to-end voice control pipeline overview.*

6.3.2 Gatekeeper Function and Integration

The gatekeeper is the safety validation layer between LLM outputs and IoT devices, designed to prevent the execution of unsafe or impossible commands within the smart home system. While well-performing LLMs do mostly remain grounded in the home model and refuse to generate unsafe values, the gatekeeper provides a safety guarantee that no unsafe commands will be executed, regardless of LLM errors. The gatekeeper is deterministic, and evaluates all LLM output action plans against predefined safety rules before permitting the execution on the relevant devices. The approach ensures fail-safe operation with minimal additional latency, though it depends on the accurate specification of rules in the home configuration, including available devices, their attributes and accepted values.

The gatekeeper specification requires rules to be defined on a per-home, per-device basis, including named devices in the home, with their attributes and the values each can accept. As illustrated in the example JSON gatekeeper specification in Figure 6.2, the rule options for specifying value types include:

- Discrete options: Value sets for categorical device parameters (e.g, on/off)
- Value ranges: Continuous parameter boundaries with defined units and steps.
- Special cases: Flexible open-ended parameters requiring downstream processing, (e.g., smart speaker content).

The gatekeeper additionally implements a Python-based validation script that compares JSON outputs against the gatekeeper specification that details the home configuration with devices, attributes and value options/ranges. Any device or attribute that is mentioned that is not in the home model is rejected. For parameters with finite state options, the system validates that the requested values match one of the predefined acceptable states. Continuous parameters are validated against the specified minimum and maximum boundaries, with

```
[
  {
    "kitchen_light": {
      "state": "off",
      "options": ["on", "off"],
      "brightness": { "state": 41, "range": [0, 100], "unit": "percent", "step": 1 }
    },
    "heating": {
      "state": "on",
      "options": ["on", "off"],
      "temperature": { "state": 17, "range": [14, 35], "unit": "celsius", "step": 0.5 }
    },
    "front_door_lock": { "state": "locked", "options": ["locked", "unlocked"] },
    "smart_speaker": {
      "state": "off",
      "options": ["on", "off"],
      "volume": { "state": 5, "range": [0, 100], "unit": "percent", "step": 1 },
      "channel": { "state": 6, "range": [1, 50], "step": 1 },
      "service": { "state": "spotify", "options": ["youtube", "spotify", "pandora"] },
      "content": { "state": "jazz" }
    }
  }
]
```

Figure 6.2: *Example gatekeeper definition and rule structure.*

additional checks for step increment. This prevents commands that would put devices in a state beyond safe operational limits, or an impossible state. The system supports granular validation, where some device attribute updates can be approved, and others rejected, for the same device, blocking only the problematic elements. For values requiring flexible input (such as media content searches), the system uses an 'any' option, allowing string values to pass through and delegating validation to downstream processing by the relevant device/service. The Python validation script can be extended, but generally remains fixed, with the only requirement to maintain the gatekeeper JSON file for new devices or home configurations.

The key goal of the gatekeeper is to prioritise safety and correctness over flexibility, ensuring predictable behaviour while interacting with the error-prone and unexpected nature of LLM systems. The gatekeeper is integrated in the system implementation, validated in the end-to-end evaluation, and further discussed in the discussion.

6.3.3 System Implementation

The smart home voice assistant implementation prioritises local processing for user privacy and safety in operation. The system combines the Espressif ESP-ADF[1] framework for front-end audio processing, Open AI's Whisper for speech transcription[2], and the Qwen2.5 14B instruct-tuned LLM[3] for command interpretation and response generation. All data processing occurs on local hardware, ensuring privacy and eliminating dependency on cloud services.

[1] http://github.com/espressif/esp-adf
[2] https://huggingface.co/openai/whisper-large-v2
[3] https://huggingface.co/Qwen/Qwen2.5-14B-Instruct

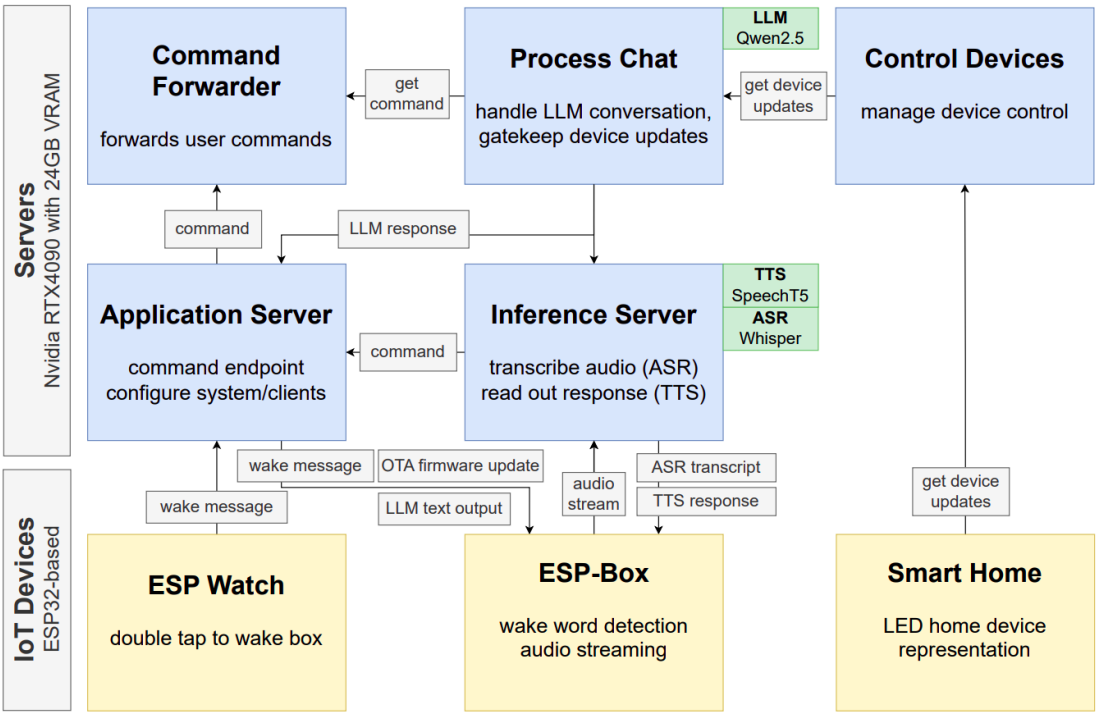

Figure 6.3: *End-to-end implemented system architecture.*

The integrated system is depicted in Figure 6.3, with a single GPU server handling all computationally intensive tasks, and the ESP32-based ESP-Box front-end device for speech capture. The ESP-Box incorporates on-device wake word detection (including Hi ESP, Alexa, Echo), Voice Activity Detection (VAD) to detect the start and end of commands, and Acoustic Echo Cancellation (AEC), with a display and speakers for audio and visual feedback.

The smart home configuration is modelled as a JSON data structure, and maintained throughout user interactions, providing a persistent representation of the home's current state. When users give commands, the ESP-Box streams the audio to the inference server, and the transcription result is then sent to the ESP-Box for display and to the LLM for processing. The LLM-generated response outputs updates to the home model that are validated by the gatekeeper, and then transmitted to smart home devices, that are in this case, ESP32 devices connected to LEDs. The explanation generated by the LLM is streamed as a spoken response (via TTS) to the ESP-Box device, and also displayed as text. With the ESP-Box voice interface front-end and the LED representation of smart home devices, this setup demonstrates practical smart home control, from voice capture to spoken response and device updates. Each system component is detailed, including the device firmware, the command processing pipeline and the Willow framework[4] the system builds upon.

[4] `https://heywillow.io/`

Front-End Device Firmware

- **ESP-Box:** Uses the Willow firmware[5], that builds on the ESP-ADF framework to handle audio capture and streaming and enable wake word detection, VAD and AEC, and display the ASR and LLM outputs. Firmware modifications were made to display and transmit the built-in sensor readings (humidity, temperature) and to enable watch tap wake functionality[6]. The Willow Web Flash application[7] uses web serial to write the Willow firmware to ESP-Box devices.

- **TTGO TWatch:** The TTGO TWatch v3 is used to provide alternative wake activation through double-tap gestures. This firmware[8] is modified from the TTGO TWatch Library[9] to detect on-screen double tap gestures. The double-tap trigger sends a wake message to the application server, which then sends a message to the ESP-Box to begin audio streaming to the inference server. This required modification to both the application server, for receiving and acting on the 'wake' messages, and the ESP-Box to be activated by these 'wake' messages. The Wi-Fi network credentials and the Willow Application Server URL must be configured in the watch firmware.

- **Smart Home Simulator:** For demonstration purposes, an ESP32 is connected to LEDs and a DotStar matrix display to simulate device updates by communicating with the control devices service via HTTP GET requests to room-specific routes. Each LED represents an appliance and the DotStar matrix displays temperature settings. The firmware simulates the transmission and interpretation of the validated JSON updates on end smart home devices, for completeness.

Microservices Architecture

- **Command Forwarder:** REST API receiving transcribed user requests from the inference server and making these available to process chat for LLM processing.

- **Process Chat:** Manages the LLM and gatekeeper interaction pipeline, prompting the LLM with task instructions, the home model and user requests, and handling the output response. Device updates are validated by the gatekeeper function and passed to control devices. The explanation is sent to the inference server, where it is converted to spoken output via the TTS model and streamed to the ESP-Box, with the full response returned via the application server for display.

- **Control Devices:** Retrieves model updates from process chat and serves device updates via room-specific routes to connected ESP32 devices.

[5] `https://github.com/HeyWillow/willow`
[6] `https://github.com/hamishcunningham/willow`
[7] `https://github.com/toverainc/willow-web-flash`
[8] `https://gitlab.com/hamishcunningham/lovey/-/tree/main/firmware/t-watch3-wake`
[9] `https://github.com/Xinyuan-LilyGO/TTGO_TWatch_Library`

Willow Framework

- **Inference Server**[10]: Manages ASR and TTS processing, using Whisper and T5 models respectively. Maintains WebSocket connections with ESP-Box devices for speech streaming (input and output) and the application server to transmit the transcript.
- **Application Server**[11]: Facilitates over-the-air (OTA) firmware and configuration updates and provides basic system monitoring. The forked implementation[12] supports receiving 'wake' message triggers, i.e., from the watch, as alternatives to wake word activation, and enables offline server startup. Communicates with the ESP-Box, the inference server and the command forwarder.

Given this system implementation, the evaluation framework used to assess both individual component performance and end-to-end system functionality is presented in the subsequent section. This evaluation framework measures system accuracy, safety and robustness in a real-world setup, validating the integrated system behaves as expected, the performance accuracy and latency are usable, and the gatekeeper prevents unsafe and hallucinated device updates.

6.4 System Evaluation Setup

6.4.1 Experimental Setup

The command set used for the LLM evaluations (Chapter 5) is also applied to evaluate the end-to-end system performance. This is a subset of commands taken from the smart home commands dataset (Chapter 3) that were further annotated with expected device updates to evaluate LLM response accuracy. In the evaluations, the audio recorded for each command is played through speakers, and the processing outputs are logged, including the ASR transcript, the raw LLM output, the gatekeeper response, and the final device updates.

Given the introduction of the gatekeeper, the previously annotated expected updates for each command are further processed for the end-to-end evaluation setup by applying the rule specification to filter out actions that would be disallowed. A gatekeeper rule set is developed for the most complex home model, H3, and applied to the expected device updates to give a new expectation for what the final device updates should be. See Appendix F for the gatekeeper rule specification. The evaluation process is summarised as:

1. Rule definition: Configure gatekeeper rules for the H3 home environment.
2. Expected value processing: Apply the gatekeeper to expected command outcomes.
3. Inference execution: Play command audio, log ASR, LLM, gatekeeper and device update outputs.
4. Results analysis: Evaluate performance in terms of accuracy and latency.

[10] `https://github.com/toverainc/willow-inference-server`
[11] `https://github.com/HeyWillow/willow-application-server`
[12] `https://github.com/mjhewitt1/willow-application-server`

Physical Setup

The integrated system evaluation takes place in a quiet environment with the ESP-Box placed at a fixed position away from a laptop that plays the audio commands. Figure 6.4 shows the physical positioning of the devices. In this setup the built-in laptop speakers are set to 100% volume for playing the audio, and the ESP-Box is placed 40cm away from the laptop screen. The computer running the system is positioned underneath the desk below the laptop.

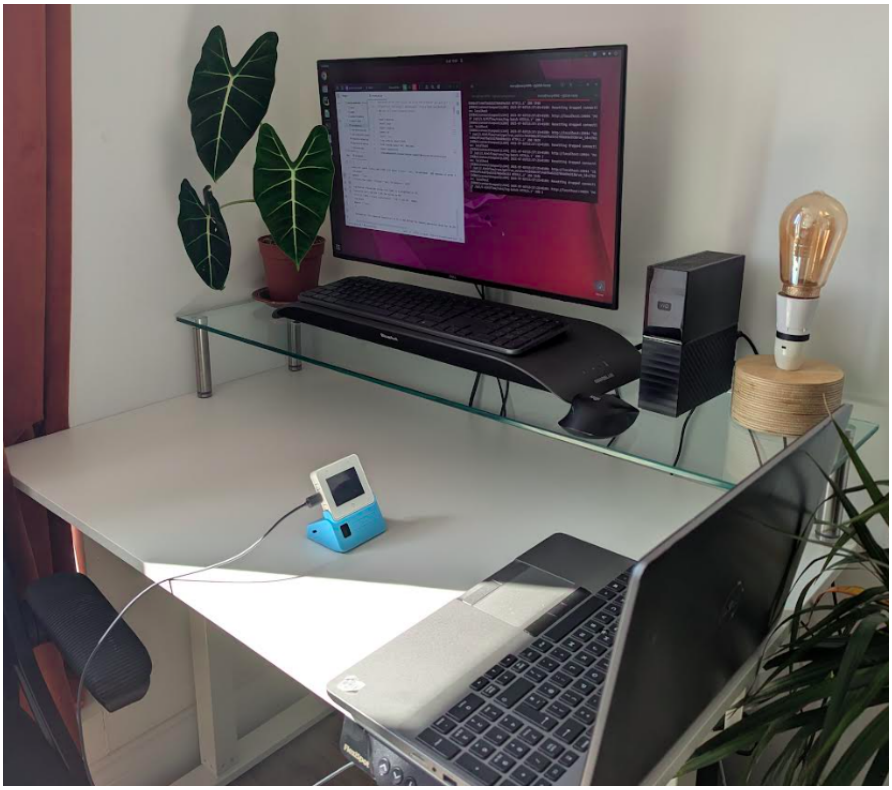

Figure 6.4: *End-to-end evaluation setup.*

While audio is played in a quiet environment, audio that was collected in a range of scenario noise settings is played to represent more-so real-world conditions. The audio scenarios are relatively evenly distributed across the evaluation set: Clean 118, Activity 110, TV 110, Music 102. The audio is also distributed across all participants in the dataset (not evenly, but at random), with a maximum of 32 recordings for a given speaker, and a minimum of 4. The audio collected on the NG22F device that was positioned in front of the speaker in the recording setup (Chapter 3) is used, therefore, while there is speaker and noise variation, there is no change in distance.

Hardware Configuration

An NVIDIA RTX4090 GPU with 24GB VRAM is used for all processing, including the Whisper and Qwen models, with the full hardware specification given in Table 6.1. The Qwen2.5 14B model evaluated in Chapter 5 cannot run simultaneously with the Whisper-large

CPU	AMD Ryzen 7 7700X, 8-core, 16 threads
RAM	32GB DDR5, 4800 MT/s
Storage	1 TB NVMe SSD (SOLIDIGM SSDPFKNU010TZ)
GPU	NVIDIA GeForce RTX 4090, 24GB VRAM
OS	Ubuntu 22.04.5 LTS (Jammy)

Table 6.1: *Hardware specification for end-to-end evaluations.*

model without memory overflow or slow CPU offloading. For this reason, several variations of the Qwen model are experimented with:

- Qwen2.5-14B-Instruct-GPTQ-Int4[13]: Adequate size but significantly increased latency.
- Qwen2.5-14B-Instruct-bnb-4bit[14]: Adequate size with minimal latency difference.
- Qwen2.5-7B-Instruct[15]: Reduced memory size, but with potentially lower performance.

The latter two options are compared in the experiments using the optimal LLM configuration found in the Chapter 5 study, that includes the aggregate prompt technique and temperature set to 0.4.

The following adjustments are made to the ESP-Box configuration for the end-to-end evaluations:

- Speaker volume turned off to prevent interference with the ASR inference.
- Wake word set to Hi Lexin, a wake word not present in the test dataset.
- Record buffer set to 8kb (time between ESP-Box wake and command capture start).
- VAD timeout set to 1000ms. This means that it will wait 1 second after the end of speech to end the audio capture. A lower value would improve response times, but it is set high to prevent clipping speech where people pause slightly.
- Maximum speech duration set to 15 seconds. This means that, if VAD does not trigger the recording to stop, it will wait 15 seconds to force the end of speech recognition.
- Audio codec used for streaming is PCM. Microphone gain set to 14db.

Logging and Data Collection

The output of each system component is logged during inference through fetching device updates from the control device service, and monitoring logs broadcasted from process chat, that includes the audio command transcription, the raw LLM response and gatekeeper messages. The outputs are logged in CSV format, along with latency. Latency is determined as the time between command audio playback ending and the device update being received. The process conducted for each command in the dataset is outlined:

[13] `https://huggingface.co/Qwen/Qwen2.5-14B-Instruct-GPTQ-Int4`
[14] `https://huggingface.co/unsloth/Qwen2.5-14B-Instruct-bnb-4bit`
[15] `https://huggingface.co/Qwen/Qwen2.5-7B-Instruct`

#	**Column Name**
01	audio_command_id
02	command_intent
03	command_goal
04	audio_scenario
05	command_length_words
06	audio_duration_s
07	command_transcript
08	expected_devices
09	expected_device_values
10	target_location
11	allowed_device_opts
12	expected_h3_gatekept_updates
13	asr_transcript
14	llm_output
15	gatekeeper_msg
16	device_update
17	sys_latency

Figure 6.5: *End-to-end evaluation CSV (column number and name)*

1. Play local audio file through laptop speakers.
2. Await ASR, LLM and gatekeeper outputs (from process chat) and home device updates (from control devices).
3. Detect update, log all outputs to CSV and record latency.
4. Send reset request to process chat to reset the conversation and restore default home model.

The evaluation CSV, depicted in Figure 6.5, includes detailed information about the commands, i.e. the intent, goal, scenario it was recorded in and command length. The ground truths include the audio transcript, the expected devices to be targeted by the command, and the expected values to be assigned (as in the Chapter 5), along with the expected final updates given the gatekeeper. As described earlier, this is the result of applying the H3 gatekeeper specification to the expected values to get a new expectation of valid end device updates, given this home. The remaining columns (13-17) record the outputs of the inference stage.

6.4.2 Evaluation Metrics

The performance of command recognition, to command reasoning, gatekeeper validation, and then action execution are tested and evaluated. The key performance metrics include:

- ASR WER: Word-level speech recognition accuracy percentage.
- LLM device selection accuracy: Percentage of correctly identified target devices.

- LLM device update accuracy: Percentage of correctly assigned attribute and value pairs to (correct) target devices.
- Gatekeeper intervention rate: Percentage of blocked hallucinated or invalid updates.
- End-to-end success rate: Percentage of correctly executed commands.

The calculation of ASR accuracy follows Chapter 4, and the LLM device selection and update accuracy follows the logic from Chapter 5, focusing on the LLMs ability to parse natural language into structured commands regardless of the safety constraints applied. The gatekeeper performance is measured by the intervention frequency, where the LLM device update does not match the final, validated update. This indicates both the need for and efficacy of this safety layer to prevent unsafe or invalid operation.

The end-to-end success rate assesses the overall command execution success after safety filtering is applied. This takes into account the device updates after the gatekeeper is applied, and ignores the raw LLM output, to measure only the updates that actually happen on the IoT device end. This means that where the LLM outputs JSON unchanged from the home model, it is not accounted for, and where output JSON is rejected, it is counted as no action taken. There are therefore differences between these results and the LLM results, as an invalid LLM output would be caught by the gatekeeper (e.g., hallucinated fan, counted as error in LLM evaluation, but blocked by gatekeeper, so removed for end to end evaluation). This is intended to measure whether final system outcomes after applying the gatekeeper would be in line and reasonable given user expectations, effectively mitigating LLM errors. The metrics combined give a comprehensive system assessment covering command transcription accuracy, command interpretation accuracy, benefits of the gatekeeper safety validation, and the end-to-end performance for practical use.

6.5 End-to-End Results and Analysis

6.5.1 System Performance

The end-to-end evaluation compares two versions of Qwen2.5 models to assess performance-efficiency trade-offs relevant to practical deployment. A quantized version of the Qwen2.5-14B-instruct model is evaluated against the standard Qwen2.5-7B-instruct model, enabling comparison of performance and latency implications as model size scales.

Table 6.2 gives an overview of the performance of the integrated system. The analysis examines complete command execution success rates, alongside metrics for the ASR transcription error rates and LLM inference correctness to identify errors occurring in between the input and output. The success rate of complete command execution is discussed, along with where system errors most commonly occur (ASR, LLM inference).

Transcription accuracy slightly differs between the two inference runs, though this is within expected variation ranges. While WER is relatively high at ∼10% (compared with ∼5% WER in the Chapter 4 evaluations), downstream performance is mostly unaffected with the LLM able to infer meaning given slightly incorrect transcriptions. Given commands are short, even just a single mistranscribed word can cause a seemingly high WER. Since the LLM

Metric	Qwen2.5-7B-Instruct	Qwen2.5-14B-Instruct
Performance		
ASR Accuracy ↑	90.01%	88.96%
Non-empty JSON ↑	73.59%	80.51%
Device Selection ↑	82.82%	89.23%
Value Assignment ↑	72.05%	78.46%
Overall Valid (e2e) ↑	80.77%	82.05%
Safety		
Gatekeeper Interception Rate ↓	7.69%	9.23%
Latency		
Latency (e2e) ↓	1.98s	2.71s

Table 6.2: *End-to-end system evaluation results. Arrows indicate whether higher (↑) or lower (↓) is better; the Safety section shows the % of commands blocked by the gatekeeper.*

semantic understanding is comparable with the earlier Chapter 5 results (93.33% device accuracy, 84.36% execution accuracy), it appears that transcription accuracy is not significantly affecting downstream command understanding, but will contribute to some errors.

The evaluation shows consistent performance improvements across all metrics when using the 14B parameter model compared with the 7B model, demonstrating a clear benefit of increased model size for smart home control tasks. In terms of semantic understanding the 14B model shows substantial improvement with a ∼6% improvement in semantic understanding (device selection and value assignment). This difference indicates the larger model can better understand the relationship between natural language commands and device control, resulting in less refusals (higher JSON output) and a higher success rate. The gatekeeper intervention is slightly more frequent for the 14B model, indicating perhaps hallucinations and errors do not diminish with increased model size, despite the enhanced reasoning performance. The end-to-end success rate is ∼1.3% higher for the 14B model, but both models result in >80% overall valid device updates, demonstrating both are highly capable.

While the 14B model offers performance benefits, there is a small trade-off with increased latency and also increased computational requirements with a ∼0.7 second increase in response time. This does not appear significant and it is also noteworthy that latency is at a maximum here given the VAD is set to await 1 second after end of speech before moving on, so it is possible to perhaps reduce this for quicker response times.

Given these performance metrics, the reasonable latency and the safety guarantees, along with the flexibility of LLMs for conversational repair to further mitigate errors, it can be concluded that the system is acceptable and applicable to real-world use cases.

6.5.2 Gatekeeper Impact Analysis

The gatekeeper mechanism demonstrates the need for safety enforcement across both model configurations, with intervention rates of 7.69% for the Qwen2.5-7B model and 9.23% for the Qwen2.5-14B model. The intervention frequencies indicate the safety layer regularly blocks problematic device updates, and validates the necessity of a deterministic safety validation

Error Type	Occurrence
Value Range Invalid	5
Value Option Invalid	16
Hallucinated device attribute	12
Hallucinated device	2

Table 6.3: *Gatekeeper caught errors by frequency (raw)*

cmd	json_update	gatekeeper_msg	error
siri i command to set the aircon to 95	`{"air_con": {"state": "on", "temperature": 95}}`	Could not update temperature with value 95 because of error 1 validation error for NumericProperty. Value error, State 95 is not in range (14, 35) [type=value_error, input_value={"state": 95, "range": (14, 35)}, input_type=dict]	Invalid range
please lock the bay window for me inside	`{"bedroom_window": "locked"}`	Could not update bedroom_window with value locked because of error 1 validation error for CategoricalProperty. Value error, State locked is not in options {"open", "closed"} [type=value_error, input_value="locked", input_type=str]	Option invalid
make my bay window in my guest room open for me	`{"guest_room_window": "open"}`		Hallucinated device
ok google would you make the inside smart speaker pause	`{"smart_speaker": {"state": "on", "volume": 5, "channel": 6, "service": "spotify", "content": "jazz", "pause": true}}`		Hallucinated attribute

Table 6.4: *Examples of gatekeeper caught errors*

in LLM-based control systems. Without this intervention, both the home model would be updated in ways that were not acceptable on the device-end, causing error propagation and inconsistency between the device states and home model that would impact usability.

To further analyse how and when the gatekeeper intervenes, the blocked commands are broadly classified as hallucination and invalid value detection. There are 35/390 commands where the gatekeeper intervenes with the Qwen2.5:14b, as classified in Table 6.3. While device hallucinations are rare, attribute hallucinations are much more common, and incorrect assignment of discrete values is another common occurrence. Examples where the gatekeeper has intervened are given in Table 6.4 to further demonstrate this function, i.e., to prevent the temperature being set to 95, or to prevent the propagation that the bedroom_window is locked when this capability does not exist and therefore will not be executed.

6.6 Real-World Implications and Future Directions

6.6.1 Commercial System Comparison

The system evaluation demonstrates performance sufficient for real-world deployment, with smart home commands reliably translated into structured device updates, validated via a deterministic gatekeeper, with informative explanations, and low-latency end-to-end processing. The system flexibility to interpret any user command, and engage in conversational repair dialogues to resolve errors, with verifiable safety via the gatekeeper and privacy ensured via local processing, represents a significant step beyond rigid, cloud-based assistants. In particular, combining LLM reasoning with deterministic safeguards creates multiple opportunities for extending and improving system functionality; the following subsections further discussing the system implementation and outlining future work.

Comparing the capabilities implemented with the taxonomy of commercial smart homes introduced in Chapter 2, Table 6.6 shows how the current implementation covers a substantial portion of the smart home systems on offer. The actuator devices modelled in the smart home (H3) for evaluation are included, as well as the sensors that are integrated in the ESP-Box and for which the firmware was adapted to display and transmit temperature and humidity values to the server. The implemented voice assistant capabilities demonstrate smart home device control, media control, alarm setting, and information search (given the knowledge of LLMs), leaving administrative tasks (list, calendar and task management), playing games, purchasing items and making calls/announcements out-of-scope. The ESP-Box serves as the VUI, with a speaker and display for communicating ASR and LLM outputs, and the TTGO TWatch adds alternate double-tap wake functionality to bypass the use of a wake word, with scope for both devices to provide further interactive functionality. LLM functionality can also be further developed to handle additional tasks, like admin and complex media management.

6.6.2 Gatekeeper Safety vs Flexibility

The gatekeeper mechanism demonstrates significant potential for the system's deployment in smart homes, providing protection against unsafe commands, and robust to attempts to bypass these safety constraints without code access to prevent its operation. The gatekeeper is verifiable, low-latency, requiring the specification of rules to define the control scope of the LLM, with its applicability extending to other domains where LLMs are operating in control environments and safety is of concern (e.g., healthcare and robotic control).

The introduction of the gatekeeper introduces a potential trade-off between safety and flexibility, however. The objective of the gatekeeper is to prioritise safety and correctness, ensuring predictable behaviour while interacting with the error-prone and unexpected nature of LLMs. However, strict validation may result in some legitimate commands being rejected if they do not exactly conform to the representation of devices or parameters in the homes JSON model. In principle, the LLM is expected to resolve such wording variations in control expressions by grounding user commands in the provided home configuration, as it has been demonstrated to do so, interpreting device nicknames and mapping them to legitimate device names. The gatekeeper therefore assumes valid outputs conform to the world model exactly, with values in the correct range, type and format. Nevertheless, overly strict validation could lead to user frustration if minor or easily interpretable discrepancies result in

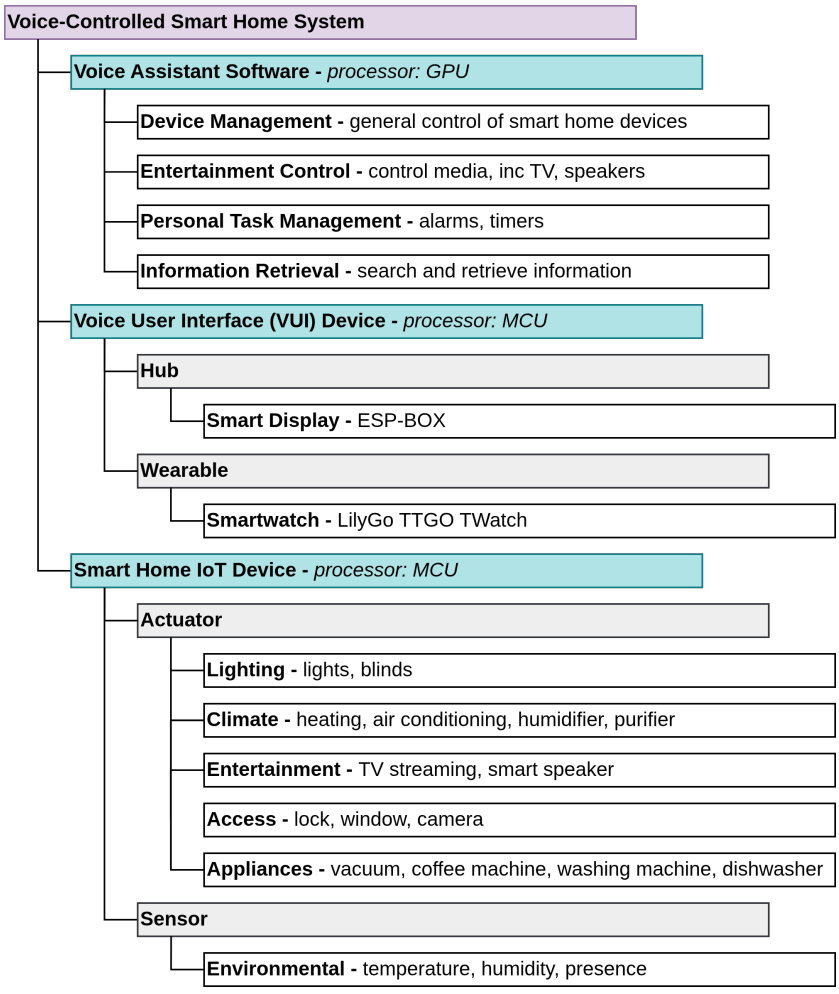

Figure 6.6: *Taxonomy of implemented smart home voice control system.*

rejected commands, e.g., an LLM may refer to a device as living_room_lamp instead of living_room_light. This type of minor wording error is in the gatekeeper analysis, where the window is assigned 'locked' instead of the expected 'closed' (see Table 6.4).

To address the limitation, and avoid users having to conversationally correct those almost right outputs, the gatekeeper could incorporate correction methods to attempt to resolve minor inconsistencies between allowed and generated device updates, before rejecting an action. For example, the gatekeeper error could be returned to the LLM with a request to correct the output, or rule-based correction functions to map common variations to valid entities in the home model could be applied. Value clipping to acceptable ranges, similar value mapping (e.g., string to numeric conversion), or fuzzy matching for device naming would be most straightforward to incorporate corrective functionality with little latency overhead. LLM-based correction would be more automated however, but could cause error loops, and introduce increased latency. Introducing such corrective mechanisms into the gatekeeper functionality would help balance safety guarantees with flexibility, maintaining strict validation, while better handling minor phrasing and formatting errors present in LLM outputs.

6.6.3 System Scalability and Maintenance

The applicability of the system approach, with respect to local deployment constraints and the manual gatekeeper specification, depends somewhat on the size and scale of the problem area in terms of number of tasks, complexity of devices and the requirement for detailed specification of the problem in order for the LLM to achieve the aim. Defining IoT devices, their attributes, and valid parameter values for the gatekeeper requires precise specification and relies on manual configuration for correctness. As the number of devices increases, maintaining and updating the world model and corresponding validation rules may become increasingly labour-intensive. For example, the template formats for additional tasks, beyond immediate device control, and towards personal calendar, list and notes management or automation routine creation, would need to be defined in the gatekeeper control specification, with associated Python validation code to implement this.

Scalability challenges also arise from the amount of contextual information required by the LLM as the complexity and size of the control space increases. If the entire world model were provided as prompt context for each interaction, large IoT deployments could exceed practical prompt length limits, and accuracy and efficiency could degrade. These limitations can be mitigated through incorporating context management approaches. For example, context (e.g., device-specific settings, action templates, preferences) could be stored in a structured database, while the LLM is provided only with high-level device and action identifiers. When generating commands, the LLM process could incorporate retrieval of the relevant device and task JSON representation from the database, allowing valid actions to be constructed at inference without requiring the full world model in the prompt. Decomposing the interaction into multiple steps, where the LLM identifies the relevant device and intent before retrieving the device schema for action construction, would reduce prompt size and improve scalability, but could add latency.

The current implementation is therefore most suitable for small to medium-sized smart home and IoT environments, where it is relatively straightforward to define the environment specification, and the LLM can handle the required context size. Beyond this, a RAG-based approach could be incorporated, where relevant information is retrieved during inference, rather than being provided in the prompt. Therefore, with further work the system could extend to support larger IoT deployments, albeit with requirements for careful LLM context management and the gatekeeper specification.

6.6.4 Extending System Functionality

Persistent Command Handling

The system can be extended to incorporate additional features and capabilities. The creation of IFTTT routines is common in smart homes, though not typically supported by current voice assistants. While not explored in the thesis, prompting and gatekeeper adaptation would be feasible to support voice-based automation routine creation. Beyond immediate device control, the system could support persistent commands with both prompt modification to provide instructions and templates for automation routines, and gatekeeper extension to validate this type of LLM output. An example prompt for this, adapted from King et al. (2023b), is shown in Figure 6.7. As for home devices, sensors are defined in JSON, and for this demonstration include: ‘global_time’, ‘global_temperature’, ‘global_humidity’, ‘kitchen_motion’,

'living_room_motion', 'bathroom_motion', 'bedroom_motion', 'entrance_motion', 'hallway_motion'. Given the automation routine prompt, the JSON specification for devices (H3) and sensors, and persistent commands, Table 6.5 presents example outputs from the Qwen2.5:14B instruct-tuned model.

The demonstrations across a range of triggers (time, motion and temperature) illustrate the models ability to extend beyond immediate commands and adapt to new output templates. In these cases, the LLM identifies an appropriate trigger and action and outputs the automation plan in a consistent format. Given the limited specification, inferences are made to use attributes like 'time_range' and 'comparison_operator' and value types like 'sunset'. While the LLM appears clearly capable to adapt to new templates and select appropriate triggers and actions given persistent commands, the prompt template should be further structured and specified to ensure compatibility with downstream processes if this were implemented.

For the addition of new tasks, like persistent command handling that require a new output structure, the LLM would need to be provided the template structure to follow, and be prompted to update this given particular command types. The gatekeeper and verifier code would also need to be extended to validate these new data structures, while the home model configuration would remain the same, with specifications for devices and sensors.

The same approach of defining a new prompt and template format, and extending the gatekeeper rules and validation, can be applied to model evolution. Extending the prompt with instructions and templates for adding devices and attributes to the home model would allow for voice-based configuration of new devices. The question remains, how best to manage prompts for these different types of commands and associated output formats. Task tailored prompting methods have been explored, using a prompt management module to manage the prompts tailored to each task (Xiao et al., 2024), and such a method could be applied here, where different output formats are required for different tasks as capabilities expand. While this work shows LLMs to be highly capable across a range of smart home tasks, the practical considerations for prompt management for different tasks given limited context, and validation for different output formats is left as future work.

```
[
  {
    "role": "system",
    "content": "You are an AI that controls a smart home. You receive user commands and
    ↪ create automation routines in response.\n\nIf there are devices relevant to the
    ↪ user command, respond with JSON that describes a sensor trigger and how you would
    ↪  change the devices (the action) based on that trigger. Respond with JSON in this
    ↪  exact form:\n{ \"status\": \"success\", \"trigger\": { }, \"action\": { }, \"
    ↪ explanation\": \"\" }\n\nIf there are no devices relevant to the user command,
    ↪ respond with JSON in this exact form:\n{ \"status\": \"failure\" }\n\nFrom now on
    ↪ , think step by step to create structured automation routines in response to user
    ↪  commands."
  }
]
```

Figure 6.7: *Prompt for persistent command responses (automation routine creation)*

Command	Response
i need coffee in the morning	`{"status": "success", "trigger": {"type": "alarm_clock", "time": "07:00"}, "action": {"coffee_maker": {"state": "on", "brew": "latte", "quantity": "1"}}, "explanation": "The automation routine will turn on the coffee maker to brew a latte at 7:00 AM every morning."}`
keep it well-lit after sundown	`{"status": "success", "trigger": {"global_time": {"comparison_operator": ">", "value": "sunset"}}, "action": {"bedroom_light": {"state": "on", "brightness": "60"}, "bathroom_light": {"state": "on", "brightness": "60"}, "living_room_lamp": {"state": "on", "brightness": "55"}}, "explanation": "Automatically turn on bedroom light, bathroom light, and living room lamp to a comfortable brightness level after sundown."}`
make it cosy in the evening	`{"status": "success", "trigger": {"time_range": {"start": "18:00", "end": "22:00"}, "sensor": "global_time"}, "action": {"devices": [{"device": "bedroom_light", "state": "on", "brightness": "50", "colour": "warm_white"}, {"device": "living_room_lamp", "state": "on", "brightness": "30", "colour": "yellow"}, {"device": "air_con", "state": "off"}, {"device": "heating", "state": "on", "temperature": "21"}, {"device": "humidifier", "state": "on", "humidity": "60"}]}, "explanation": "Set the bedroom light and living room lamp to a lower brightness with warm tones, turn off air conditioning and adjust heating to maintain a cozy temperature. Increase humidity slightly for added comfort."}`
stop cleaning after 22:00	`{"status": "success", "trigger": {"type": "time", "value": "22:00"}, "action": {"robot_vacuum": {"state": "off"}}, "explanation": "The automation routine will turn off the robot vacuum at 22:00 to stop cleaning."}`
turn the heat off when it's hot inside	`{"status": "success", "trigger": {"global_temperature": {"comparison_operator": ">", "value": "25"}}, "action": {"heating": {"state": "off"}}, "explanation": "When the global temperature in the home exceeds 25 degrees Celsius, the heating system will be turned off to prevent overheating."}`

Table 6.5: *Persistent command response demonstration.*

Contextual Intelligence and Personalisation

Additional context could be provided to the LLM in the form of the environmental and user inputs for more context-aware and informed reasoning and responses. Sensor values (e.g., presence, humidity, temperature) would enhance context-awareness, and are often required for automation routines. The ESP-Box firmware has been modified to send sensor values to the server but they are not currently passed to the LLM, and there are an expansive array of sensors found in smart homes that could also be incorporated (e.g., motion, luminance).

Personalisation is another avenue to explore. Knowledge of user preferences could be gathered (perhaps via the user indicating 'good' and 'bad' responses), and incorporated into the context to allow the LLM to incorporate personal preferences into decision making. Person identification mechanisms would be required to enable this, perhaps through speaker IDs, yet the key question would be how to manage this context so that it is actually useful, without slowing or impeding performance. This leads to a 'context engineering' problem, involving selecting the right information relevant to the task to provide to the LLM[16], and relates to the discussion on system scalability when using locally deployed LLMs.

[16] `https://simonwillison.net/2025/Jun/27/context-engineering/`

Gatekeeper Extensions

In addition to mentioned approaches regarding enhancing gatekeeper flexibility and extending its validation to other tasks beyond immediate device control, the gatekeeper safety framework could evolve to provide enhanced security. Logical constraints on combinations could be defined, e.g., ensuring the TV volume is not adjusted while it is powered off, or ensuring the security system is not disabled at night. Incorporating authentication would enhance security, particularly for persistent changes. The gatekeeper could be extended with rules defining which users have permission to make updates or control specific devices. The watch functionality could extend to enable user interaction with the gatekeeper to authorise persistent or safety-critical actions (e.g., disabling security systems). In addition, voice-based speaker identification or other person recognition techniques could be used to verify the identity of the user issuing the command. Therefore, such methods could be incorporated to improve security by enabling user access control, but would require specifying user permissions and extending the system to recognise user IDs and verify actions against those permissions.

While the gatekeeper handles safety constraints for defined parameters, it is limited in handling open-ended, free-form inputs like TV program selection. Given the breadth of possible options, validation of these values would require downstream, typically search methods, and knowledge of a users' media library, beyond this rule-based approach, although gatekeeper validation against external sources, like users media libraries could be explored.

6.6.5 Real-World Integration and Evaluation

The next engineering step is to integrate the system with smart home products for further system testing and evaluation. The ESP-Box serves as a suitable VUI, with a display to show ASR and LLM outputs for transparency. However, to extend applicability beyond the thesis implementation, the system could be integrated with platforms like HomeAssistant to enable execution of LLM outputs on smart home products. This would require the setup of HomeAssistant with smart home products, and the translation of the LLM output JSON into API calls in the control device function.

The integration of devices in a real-world home setup would support the collection of user feedback to further evaluate the practical usability of the local voice assistant. One example integrating LLM outputs directly into HomeAssistant integrations for voice-based automation routine creation finds the system usable and likeable from a user-perspective, but with areas for improvement in JSON formatting of the HomeAssistant messages (Giudici et al., 2025). The study instructs the LLM to output automation routines to be processed directly by HomeAssistant, and sees issues with message malformation. The proposal to extend this system, however, is to maintain a simplistic JSON model of the home devices, with only relevant and controllable parameters, and format the JSON into HomeAssistant compatible messages in the control devices process, as is currently done to format messages for the ESP32 devices. This method is proposed to simplify and separate the LLMs role from execution, saving the need to generate long structures and therefore allowing for improved accuracy, with minimised output token cost and latency.

6.7 Summary

In this chapter, the gatekeeper mechanism, designed to ensure safe and reliable operation in LLM-driven smart home control systems, was implemented and evaluated. The gatekeeper mitigates the risks associated with LLM hallucinations and faults, that were identified in Chapter 5 and could result in unsafe or unintended actions. The rule-based safety approach was integrated with the LLM-generated action plans, providing a validation layer between the LLM outputs and the device actions, to ensure only those that are safe and feasible are executed by the smart home devices.

The system integration and testing indicate that the combination of LLMs with a gatekeeper offers a viable and flexible approach to address challenges of LLMs and ensure safety in real-world applications. Not only can errors cause significant disruption in smart homes, but in other control scenarios like healthcare and robotics. The gatekeeper demonstrates the future potential for LLMs to be combined with rule-based systems for verifiably safe device control, without diminishing LLM capabilities.

The end-to-end evaluation of the smart home voice assistant, incorporating the ESP-Box front-end with wake word detection, Whisper for ASR transcription, Qwen2.5 for LLM reasoning, and the gatekeeper, demonstrates the feasibility of a flexible, private, and locally run voice assistant for smart home systems, with component and system-level performance metrics to answer the overarching research question of viability.

The research lays the foundation for a local LLM-driven voice assistant for IoT device control that is safer, private, more flexible and aligns with user needs. The implementation demonstrates reasoning capabilities beyond Alexa and Google, with flexible command interpretation and conversational repair. There is scope for incremental system development, with directions including persistent command handling, multimodal inputs and personalisation.

In summary, this chapter addresses the final thesis objectives to evaluate a gatekeeper mechanism to ensure the safety of LLM outputs, and to test and validate the end-to-end smart home voice assistant system, integrating the previously evaluated ASR and LLM components using the recorded and annotated smart home commands dataset. In the concluding chapter, the findings and contributions of the thesis, with respect to the thesis aims and the future development of local voice control applications, are summarised.

Chapter 7

Conclusions

7.1 Summary of Findings

The thesis began with the question of whether flexible, conversational voice control of complex systems (specifically, smart home IoT devices) can be achieved privately, using local processing and removing reliance on the cloud. This section summarises the work done to address this, the research questions and their results, and concludes the project of local conversational control of complex control surfaces like smart homes is well within our grasp.

Chapter 2 established the scope, requirements and evaluation criteria for a local LLM-driven smart home voice assistant through analysis of existing commercial and research systems. Chapter 3 then detailed the smart home voice commands dataset designed to evaluate ASR, LLM, and end-to-end system performance across realistic conditions and diverse command types. Using this dataset and evaluation framework, the findings with respect to the research questions considering the feasibility, performance, flexibility and safety of a locally run smart home voice control solution are outlined.

RQ1. How does the performance of locally run ASR models compare to cloud-based solutions for smart home commands?

Chapter 4 evaluated local and cloud-based ASR models for smart home command transcription across varying conditions captured in the Chapter 3 dataset (noise levels, device types, distances, accents) in terms of accuracy, latency and computational requirements. Findings showed that ASR can attain high performance on local GPU hardware with very low latency, with Whisper-large performance comparable to Amazon and exceeding Google transcription services in terms of both accuracy and robustness. Whisper-small also performs fairly well for a low-resource option (10% WER overall) although with lower accuracy and robustness over Whisper-large (6.5% WER overall). Many transcription errors were minor, and performance improvements were not explored, but future work could experiment with prompting and fine-tuning approaches, including speaker adaptation and identification methods to improve recognition on overlapping speech, non-native accents, distant and single-word commands which were identified difficulties for the evaluated ASR models.

RQ2. How effectively can LLMs be adapted for reasoning and flexible command interpretation in smart home IoT environments?

Chapter 5 demonstrated how the brittle control problem suffered by previous voice control technologies can be solved using the recent advancements in LLMs, applying these as flexible and context-aware reasoners. The capabilities of various LLMs to support the interpretation of a variety of commands (across intents and specificity) and generate actionable device updates grounded in provided home models (of varying complexity) were evaluated. Through measuring task completion accuracy, assessing natural language response quality and demonstrating conversational repair, LLMs were shown to be highly capable for smart home device control. Prompt techniques were explored to minimise common errors, with a combination of approaches applied to the Qwen2.5 14B model to achieve the highest performance across the evaluated command set with 90% correct device updates and sub-second average latency. While LLMs are effective, future research directions remain, including context management methods for extension to more complex tasks and longer multi-turn interactions, particularly given the compute limitations of local deployment.

RQ3. How to ensure safe and reliable execution of LLM-generated outputs?

Addressing the concern that hallucination and faulting tendencies can cause LLMs to be dangerous tools, a method for the safe deployment of LLMs is proposed, implemented and measured in Chapter 6, based on sandboxing and enforcing deterministic (non-AI) checks on device control outputs. By defining a control specification for home IoT devices and comparing LLM outputs against it, the system decouples the LLM from direct execution. The LLM is used to interpret user commands and generate structured action plans, while the gatekeeper verifies and translates these for IoT control execution. Testing showed combining LLMs with this gatekeeping method provides an effective and viable approach to protect against hallucinated or unsafe device updates. End-to-end evaluations showed the gatekeeper to effectively block and prevent unsafe or invalid actions in faulty execution plans, which occurred in 10% of LLM outputs evaluated. The gatekeeper performs strict validation, and could better balance flexibility by introducing either rule or LLM-based corrective mechanisms. In addition, the gatekeeper requires the manual specification of homes, their devices and settings.

RQ4. How does the end-to-end local voice assistant system perform in real-world smart home scenarios?

The system implementation and experimental results in Chapter 6 showed that conversational smart home control is possible using only domestic (zero-cloud) computational resources, integrating ASR for transcription, LLM-based reasoning, and a deterministic gatekeeper to verify control outputs. With 90% ASR accuracy, 80% command execution accuracy and $\sim$2 second end-to-end latency on the smart home commands dataset, results indicate the high performance and usability of the integrated local voice assistant for real-world applications. Smart home product integrations and user evaluations are not considered however, consideration for system scalability is required for transfer to larger and more complex IoT environments, and lower resource deployments using smaller models could also be further explored.

7.2 Future Work

Findings and contributions of the thesis present directions for research and development to improve both improve model performance and expand system capabilities.

ASR performance improvements

ASR improvements could be achieved using prompting techniques to provide context, or fine-tuning with domain-specific data, such as the smart home commands dataset. Personalised speech recognition approaches (e.g., speaker recognition and adaptation) could also be explored, using individuals' speech data to fine-tune models. These methods are applicable to the Whisper models, and could be explored to adapt these to better handle the characteristics of smart home commands, environmental noise or specific accents. The dataset is limited however, therefore augmentation techniques could also be applied to incorporate further environmental factors and acoustic conditions.

LLM context and capability extension

LLM capabilities can be extended to handle tasks beyond immediate device control, such as automation routine creation or home model evolution. Persistent command handling, demonstrated in Chapter 6.6.4, is possible, but thorough evaluation and consideration for how to prompt LLMs to handle multiple tasks, requiring different output formats, is required. In combination, providing the LLM context about users (e.g., user ID, preferences) and the environment (e.g., sensors) could better inform the reasoning process and allow for personalised responses, but managing this context in a way that is useful to the LLM is a challenge. Particularly in local environments, careful context management is required to handle this information, therefore context management techniques (like RAG, history squashing, prompt libraries) are important considerations for system extensions involving user preferences, larger smart homes, and more complex devices and control tasks.

Gatekeeper capability extension

The gatekeeper could be extended to support rule combinations, e.g., security cannot be disabled while the door is unlocked. With an existing authentication mechanism, e.g., speaker recognition, the rules could also extend to introduce authorisation into the control interface, ensuring only valid users can make some changes. More intelligent error correction is also possible, where instead of rejecting incorrect outputs, the gatekeeper could clip values, correct device naming, or pass error messages to the LLM to prompt correction. The gatekeeper is only applied to immediate control updates in this work, but additional tasks may require various action templates, requiring extensions to handle the definition and validation of these.

Real-world smart home integration

Integration with smart home products would support further testing and evaluation of the voice assistant solution. A function would be required to translate LLM output JSON into appropriate API calls to control smart home products. HomeAssistant is a suitable open-source platform supporting a range of devices, therefore outputs could be formatted accordingly and

directed to this in such an implementation. A usable smart home setup would allow for user evaluation to gauge strengths, weaknesses and likeability of the voice assistant, that is the natural next step to gauge the system readiness for deployment, and determine soft traits like personality and response style. System scalability also requires consideration for expansion to more devices and control tasks given the limits of local deployment, LLM context length, and the need to manually define the gatekeeper control scope. Smaller models (e.g., Whisper-small and Qwen2.5 3B) could also be explored for lower resource, more accessible deployments, with findings indicating their reasonable performance.

Transferring to other IoT domains

Generalising the voice control approach beyond smart homes is a key future direction, given the system has practical applications across IoT control scenarios, where voice control is a convenient and practical interface. Future work could explore transferring the system to use cases like social robotics and healthcare. The architecture would remain, with ASR, LLM reasoning and the gatekeeper, but the control specification and prompt design would need to be adapted for these control scenarios.

7.3 Concluding Remarks

The core contribution of the thesis lies in the combination and evaluation of local ASR, a local LLM reasoner, and a deterministic gatekeeper for a conversational system for IoT control, that is verifiably safe for deployment, is flexible to allow for natural interaction, and transferable across domains. This represents a significant step beyond previously brittle, cloud-based systems, offering a local, private and adaptable option for voice interactive applications.

The spoken smart home commands dataset is also a novel contribution, designed to evaluate ASR robustness, LLM reasoning flexibility and accuracy, and end-to-end system performance. The dataset captures challenging smart home interaction scenarios, including noisy, distant recordings, diverse speaker accents, with varied command types (e.g., well-specified, ambiguous) and intents covering a wide range of smart home control tasks.

The evaluations demonstrate the practical feasibility and high performance of locally run ASR models and LLMs to support a private, flexible and safe voice interface for IoT control, while identifying several areas for future research and system development. These include improving scalability given local compute and LLM context constraints, extending the framework to broader control tasks, evaluating performance across longer conversational contexts, incorporating corrective feedback mechanisms into the gatekeeper, and conducting real-world usability evaluation through integration with smart home devices. The dataset additionally provides a resource for benchmarking and fine-tuning of ASR models and LLMs for IoT control tasks. With this, the thesis provides a foundation for continued research into privacy-preserving conversational control applications using LLM reasoning, with applicability across many industries and domains, particularly in privacy-sensitive fields like healthcare.

Bibliography

Abdin, M., Aneja, J., Behl, H., Bubeck, S., Eldan, R., Gunasekar, S., Harrison, M., Hewett, R. J., Javaheripi, M., Kauffmann, P., Lee, J. R., Lee, Y. T., Li, Y., Liu, W., Mendes, C. C. T., Nguyen, A., Price, E., de Rosa, G., Saarikivi, O., Salim, A., Shah, S., Wang, X., Ward, R., Wu, Y., Yu, D., Zhang, C., and Zhang, Y. (2024). Phi-4 technical report. *arXiv [cs.CL]*.

Al-Mhabis, N. and Cunningham, H. (2017). Socio-political perspectives on surveillance and censorship: Implications for on-line privacy in the age of cloud computing. In *2017 Computing Conference*, pages 208–213. IEEE.

Alam, M. R., Reaz, M. B. I., and Ali, M. A. M. (2012). A review of smart homes past, present, and future. *IEEE Trans. Syst. Man Cybern. C Appl. Rev.*, 42(6):1190–1203.

Alghisi, S., Rizzoli, M., Roccabruna, G., Mousavi, S. M., and Riccardi, G. (2024). Should we fine-tune or RAG? Evaluating different techniques to adapt LLMs for dialogue. *arXiv [cs.CL]*.

Ali, Eltayeb, and Abusail (2017). Voice recognition based smart home control system. *Int. J. High Risk Behav. Addict.*

Almeida, M., Laskaridis, S., Mehrotra, A., Dudziak, L., Leontiadis, I., and Lane, N. D. (2021). Smart at what cost? Characterising mobile deep neural networks in the wild. *arXiv [cs.LG]*.

Almuseelem, W. (2023). Data privacy and security model in cloud environments. In *2023 3rd International Conference on Computing and Information Technology (ICCIT)*, pages 43–49. IEEE.

Amatriain, X. (2024). Prompt design and engineering: Introduction and advanced methods. *arXiv [cs.SE]*.

Ammari, T., Kaye, J., Tsai, J. Y., and Bentley, F. (2019). Music, search, and IoT: How people (really) use voice assistants. *ACM Trans. Comput.-Hum. Interact.*, 26(3):1–28.

Ardila, R., Branson, M., Davis, K., Henretty, M., Kohler, M., Meyer, J., Morais, R., Saunders, L., Tyers, F. M., and Weber, G. (2019). Common voice: A massively-multilingual speech corpus. *arXiv [cs.CL]*.

Arriany, A. A. and Musbah, M. S. (2016). Applying voice recognition technology for smart home networks. In *2016 International Conference on Engineering & MIS (ICEMIS)*, pages 1–6. ieeexplore.ieee.org.

Baby, C. J., Munshi, N., Malik, A., Dogra, K., and Rajesh, R. (2017). Home automation using web application and speech recognition. In *2017 International conference on Microelectronic Devices, Circuits and Systems (ICMDCS)*, pages 1–6. ieeexplore.ieee.org.

Baevski, A., Zhou, H., Mohamed, A., and Auli, M. (2020). Wav2vec 2.0: A framework for self-supervised learning of speech representations. *arXiv [cs.CL]*.

Bai, L. (2022). Research on voice control technology for smart home system. In *Proceedings of the Asia Conference on Electrical, Power and Computer Engineering*, number Article 43 in EPCE '22, pages 1–7, New York, NY, USA. Association for Computing Machinery.

Banerjee, S., Agarwal, A., and Singla, S. (2025). Llms will always hallucinate, and we need to live with this. In *Intelligent Systems Conference*, pages 624–648. Springer.

Bang, Y., Cahyawijaya, S., Lee, N., Dai, W., Su, D., Wilie, B., Lovenia, H., Ji, Z., Yu, T., Chung, W., Do, Q. V., Xu, Y., and Fung, P. (2023). A multitask, multilingual, multimodal evaluation of ChatGPT on reasoning, hallucination, and interactivity. *arXiv [cs.CL]*.

Bang, Y., Ji, Z., Schelten, A., Hartshorn, A., Fowler, T., Zhang, C., Cancedda, N., and Fung, P. (2025). HalluLens: LLM hallucination benchmark. In Che, W., Nabende, J., Shutova, E., and Pilehvar, M. T., editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 24128–24156, Vienna, Austria. Association for Computational Linguistics.

Barker, J., Watanabe, S., Vincent, E., and Trmal, J. (2018). The fifth 'CHiME' speech separation and recognition challenge: Dataset, task and baselines. *arXiv [cs.SD]*.

Bastianelli, E., Vanzo, A., Swietojanski, P., and Rieser, V. (2020). SLURP: A spoken language understanding resource package. *arXiv [cs.CL]*.

Battle, R. and Gollapudi, T. (2024). The unreasonable effectiveness of eccentric automatic prompts. *arXiv [cs.CL]*.

BBC News (2017). Amazon hands over echo 'murder' data. `https://www.bbc.co.uk/news/technology-39191056`. Accessed: 2024-9-2.

BBC News (2018). Amazon patents 'voice-sniffing' algorithms. `https://www.bbc.co.uk/news/technology-43725708`. Accessed: 2024-9-2.

BBC News (2020). AWS: Amazon web outage breaks vacuums and doorbells. `https://www.bbc.co.uk/news/technology-55087054`. Accessed: 2025-9-5.

BBC News (2026a). Apple to pay $95m to settle siri 'listening' lawsuit. `https://www.bbc.co.uk/news/articles/cr4rvr495rgo`. Accessed: 2026-18-4.

BBC News (2026b). Google to pay $68m to settle lawsuit claiming it recorded private conversations. `https://www.bbc.co.uk/news/articles/c4g38jv8zzwo`. Accessed: 2026-18-4.

Belcak, P., Heinrich, G., Diao, S., Fu, Y., Dong, X., Muralidharan, S., Lin, Y. C., and Molchanov, P. (2025). Small language models are the future of agentic AI. *arXiv [cs.AI]*.

Bentley, F., Luvogt, C., Silverman, M., Wirasinghe, R., White, B., and Lottridge, D. (2018). Understanding the long-term use of smart speaker assistants. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 2(3):1–24.

Beugin, Y., Burke, Q., Hoak, B., Sheatsley, R., Pauley, E., Tan, G., Hussain, S. R., and McDaniel, P. (2022). Building a privacy-preserving smart camera system. *arXiv [cs.CR]*.

Bhagath, P., Parihar, S., and Das, P. K. (2021). Speech recognition for Indian spoken languages towards automated home appliances. In *2021 2nd International Conference for Emerging Technology (INCET)*, pages 1–5.

Bhatia, A. (2023). Edge computing in iot: What it is and how to use it successfully. `https://www.computer.org/publications/tech-news/trends/edge-computing-in-iot`. Accessed: 2026-4-5.

Brooks, B. (2024). Open-source AI is good for us. `https://spectrum.ieee.org/open-source-ai-good`. Accessed: 2025-1-28.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M.-T., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. *Neural Inf Process Syst*, abs/2005.14165.

Brush, A. J. B., Lee, B., Mahajan, R., Agarwal, S., Saroiu, S., and Dixon, C. (2011). Home automation in the wild: Challenges and opportunities. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2115–2124, New York, NY, USA. Association for Computing Machinery.

Calò, T. and De Russis, L. (2024). Enhancing smart home interaction through multimodal command disambiguation. *Pers. Ubiquitous Comput.*, pages 1–16.

Cambre, J. and Kulkarni, C. (2019). One voice fits all? Social implications and research challenges of designing voices for smart devices. *Proc. ACM Hum.-Comput. Interact.*, 3(CSCW):1–19.

Cambre, J., Reig, S., Kravitz, Q., and Kulkarni, C. (2020). "All rise for the AI director": Eliciting possible futures of voice technology through story completion. In *Proceedings of the 2020 ACM Designing Interactive Systems Conference*, DIS '20, pages 2051–2064, New York, NY, USA. Association for Computing Machinery.

Cao, Y., Kang, Y., Wang, C., and Sun, L. (2023). Instruction mining: Instruction data selection for tuning large language models. *arXiv [cs.CL]*.

Chard, K., Russell, M., Lussier, Y. A., Mendonça, E. A., and Silverstein, J. C. (2011). A cloud-based approach to medical NLP. *AMIA Annu. Symp. Proc.*, 2011:207–216.

Chen, G., Chai, S., Wang, G., Du, J., Zhang, W.-Q., Weng, C., Su, D., Povey, D., Trmal, J., Zhang, J., Jin, M., Khudanpur, S., Watanabe, S., Zhao, S., Zou, W., Li, X., Yao, X.,

Wang, Y., Wang, Y., You, Z., and Yan, Z. (2021). GigaSpeech: An evolving, multi-domain ASR corpus with 10,000 hours of transcribed audio. *arXiv [cs.SD]*.

Chen, H., Deng, W., Yang, S., Xu, J., Jiang, Z., Ngai, E. C., Liu, J., and Liu, X. (2025). Toward edge general intelligence via large language models: Opportunities and challenges. *IEEE Network*, 39(5):263–271.

Chen, R., Tian, Z., Liu, H., Zhao, F., Zhang, S., and Liu, H. (2018). Construction of a voice driven life assistant system for visually impaired people. In *2018 International Conference on Artificial Intelligence and Big Data (ICAIBD)*, pages 87–92.

Chen, W., Ma, X., Wang, X., and Cohen, W. W. (2022). Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv [cs.CL]*.

Chen, X., Ghoshal, A., Mehdad, Y., Zettlemoyer, L., and Gupta, S. (2020). Low-resource domain adaptation for compositional task-oriented semantic parsing. *arXiv [cs.CL]*.

Cheng, P. and Roedig, U. (2022). Personal voice assistant security and privacy—A survey. *Proc. IEEE*, 110(4):476–507.

Chia, Y. K., Chen, G., Tuan, L. A., Poria, S., and Bing, L. (2023). Contrastive chain-of-thought prompting. *arXiv [cs.CL]*.

Choi, T. R. and Drumwright, M. E. (2021). "OK, Google, why do I use you?" Motivations, post-consumption evaluations, and perceptions of voice AI assistants. *Telemat. Inform.*, 62(101628):101628.

Ciccarelli, G., Barber, J., Nair, A., Cohen, I., and Zhang, T. (2022). Challenges and opportunities in multi-device speech processing. *arXiv [eess.AS]*.

Clark, L., Pantidi, N., Cooney, O., Doyle, P., Garaialde, D., Edwards, J., Spillane, B., Murad, C., Munteanu, C., Wade, V., and Cowan, B. R. (2019). What makes a good conversation? Challenges in designing truly conversational agents. *arXiv [cs.HC]*.

Clausen, M., Kyhn, M. P., Papachristos, E., and Merritt, T. (2023). Exploring humor as a repair strategy during communication breakdowns with voice assistants. In *Proceedings of the 5th International Conference on Conversational User Interfaces*, number Article 16 in CUI '23, pages 1–9, New York, NY, USA. Association for Computing Machinery.

Coetzee, L. and Eksteen, J. (2011). The internet of things - promise for the future? An introduction. In *2011 IST-Africa Conference Proceedings*, pages 1–9.

Coucke, A., Dureau, J., Leroy, D., and Maury, S. (2022). On-device voice control on sonos speakers. `https://tech-blog.sonos.com/posts/on-device-voice-control-on-sonos-speakers/`. Accessed: 2024-6-18.

Coucke, A., Saade, A., Ball, A., Bluche, T., Caulier, A., Leroy, D., Doumouro, C., Gisselbrecht, T., Caltagirone, F., Lavril, T., Primet, M., and Dureau, J. (2018). Snips voice platform: An embedded spoken language understanding system for private-by-design voice interfaces. *arXiv [cs.CL]*.

Cowan, B. R., Pantidi, N., Coyle, D., Morrissey, K., Clarke, P., Al-Shehri, S., Earley, D., and Bandeira, N. (2017). "What can i help you with?": Infrequent users' experiences of intelligent personal assistants. In *Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services*, number Article 43 in MobileHCI '17, pages 1–12, New York, NY, USA. Association for Computing Machinery.

Coward, C. (2024). GPT home runs circles around traditional home assistant devices. `https://www.hackster.io/news/gpt-home-runs-circles-around-traditional-home-assistant-devices-77d4f4d3c9e5.amp`. Accessed: 2024-8-26.

Cunningham, H., Coleman, G., and Radu, V. (2022). *Mi Casa su Botnet? Learning the Internet of Things with WaterElf, unPhone and the ESP32*. University of Sheffield.

Dallmer-Zerbe and Haase (2021). Adapting smart home voice assistants to users' privacy needs using a raspberry-pi based and self-adapting system. *2021 IEEE 30th International*.

Dalrymple, D. d., Skalse, J., Bengio, Y., Russell, S., Tegmark, M., Seshia, S., Omohundro, S., Szegedy, C., Goldhaber, B., Ammann, N., Abate, A., Halpern, J., Barrett, C., Zhao, D., Zhi-Xuan, T., Wing, J., and Tenenbaum, J. (2024). Towards guaranteed safe AI: A framework for ensuring robust and reliable AI systems. *arXiv [cs.AI]*.

Dasgupta, A. R., Kumari, P., Sahu, H., Amarnath, S., and Nanda, S. (2022). Smart-home automation using AI assistant and IoT. In *2021 4th International Conference on Recent Trends in Computer Science and Technology (ICRTCST)*, pages 329–333.

Daws, R. (2024). UK introduces first IoT security laws. `https://iottechnews.com/news/uk-introduces-first-iot-security-laws/`. Accessed: 2024-8-27.

Day, M., Turner, G., and Drozdiak, N. (2019). Amazon workers are listening to what you tell Alexa. `https://www.bloomberg.com/news/articles/2019-04-10/is-anyone-listening-to-you-on-alexa-a-global-team-reviews-audio`. Accessed: 2024-9-2.

De Silva, L. C., Morikawa, C., and Petra, I. M. (2012). State of the art of smart homes. *Eng. Appl. Artif. Intell.*, 25(7):1313–1321.

Deep Cogito (2025). Introducing cogito preview. `https://deep-cogito-website.vercel.app/research/cogito-v1-preview`. Accessed: 2025-5-29.

DeepSeek-AI, Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., Zhang, X., Yu, X., Wu, Y., Wu, Z. F., Gou, Z., Shao, Z., Li, Z., Gao, Z., Liu, A., Xue, B., Wang, B., Wu, B., Feng, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., Dai, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Bao, H., Xu, H., Wang, H., Ding, H., Xin, H., Gao, H., Qu, H., Li, H., Guo, J., Li, J., Wang, J., Chen, J., Yuan, J., Qiu, J., Li, J., Cai, J. L., Ni, J., Liang, J., Chen, J., Dong, K., Hu, K., Gao, K., Guan, K., Huang, K., Yu, K., Wang, L., Zhang, L., Zhao, L., Wang, L., Zhang, L., Xu, L., Xia, L., Zhang, M., Zhang, M., Tang, M., Li, M., Wang, M., Li, M., Tian, N., Huang, P., Zhang, P., Wang, Q., Chen, Q., Du, Q., Ge, R., Zhang, R., Pan, R., Wang, R., Chen, R. J., Jin, R. L., Chen, R., Lu, S., Zhou, S., Chen, S., Ye, S., Wang, S., Yu, S., Zhou, S., Pan, S., Li, S. S., Zhou, S., Wu, S., Ye, S., Yun, T.,

Pei, T., Sun, T., Wang, T., Zeng, W., Zhao, W., Liu, W., Liang, W., Gao, W., Yu, W., Zhang, W., Xiao, W. L., An, W., Liu, X., Wang, X., Chen, X., Nie, X., Cheng, X., Liu, X., Xie, X., Liu, X., Yang, X., Li, X., Su, X., Lin, X., Li, X. Q., Jin, X., Shen, X., Chen, X., Sun, X., Wang, X., Song, X., Zhou, X., Wang, X., Shan, X., Li, Y. K., Wang, Y. Q., Wei, Y. X., Zhang, Y., Xu, Y., Li, Y., Zhao, Y., Sun, Y., Wang, Y., Yu, Y., Zhang, Y., Shi, Y., Xiong, Y., He, Y., Piao, Y., Wang, Y., Tan, Y., Ma, Y., Liu, Y., Guo, Y., Ou, Y., Wang, Y., Gong, Y., Zou, Y., He, Y., Xiong, Y., Luo, Y., You, Y., Liu, Y., Zhou, Y., Zhu, Y. X., Xu, Y., Huang, Y., Li, Y., Zheng, Y., Zhu, Y., Ma, Y., Tang, Y., Zha, Y., Yan, Y., Ren, Z. Z., Ren, Z., Sha, Z., Fu, Z., Xu, Z., Xie, Z., Zhang, Z., Hao, Z., Ma, Z., Yan, Z., Wu, Z., Gu, Z., Zhu, Z., Liu, Z., Li, Z., Xie, Z., Song, Z., Pan, Z., Huang, Z., Xu, Z., Zhang, Z., and Zhang, Z. (2025). DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv [cs.CL]*.

Demanega, I., Mujan, I., Singer, B. C., Anđelković, A. S., Babich, F., and Licina, D. (2021). Performance assessment of low-cost environmental monitors and single sensors under variable indoor air quality and thermal conditions. *Build. Environ.*, 187(107415):107415.

Deng, Y., Zhang, W., Chen, Z., and Gu, Q. (2023). Rephrase and respond: Let large language models ask better questions for themselves. *arXiv [cs.CL]*.

Dhuliawala, S., Komeili, M., Xu, J., Raileanu, R., Li, X., Celikyilmaz, A., and Weston, J. (2023). Chain-of-verification reduces hallucination in large language models. *arXiv [cs.CL]*.

Ding, A. Y., Peltonen, E., Meuser, T., Aral, A., Becker, C., Dustdar, S., Hiessl, T., Kranzlmüller, D., Liyanage, M., Maghsudi, S., Mohan, N., Ott, J., Rellermeyer, J. S., Schulte, S., Schulzrinne, H., Solmaz, G., Tarkoma, S., Varghese, B., and Wolf, L. (2022). Roadmap for edge AI: a Dagstuhl perspective. *SIGCOMM Comput. Commun. Rev.*, 52(1):28–33.

Dorsemaine, B., Gaulier, J.-P., Wary, J.-P., Kheir, N., and Urien, P. (2015). Internet of things: A definition & taxonomy. In *2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies*, pages 72–77. IEEE.

Dutsinma, F. L. I., Pal, D., Funilkul, S., and Chan, J. H. (2022). A systematic review of voice assistant usability: An ISO 9241-11 approach. *SN Comput. Sci.*, 3(4):267.

Elsokah, M. M., Saleh, H. H., and Ze, A. R. (2020). Next generation home automation system based on voice recognition. In *Proceedings of the 6th International Conference on Engineering & MIS 2020*, number Article 72 in ICEMIS'20, pages 1–7, New York, NY, USA. Association for Computing Machinery.

Erić, T., Ivanović, S., Milivojša, S., Matić, M., and Smiljković, N. (2017). Voice control for smart home automation: Evaluation of approaches and possible architectures. In *2017 IEEE 7th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, pages 140–142. ieeexplore.ieee.org.

FakhrHosseini, S., Lee, C., Lee, S.-H., and Coughlin, J. (2025). A taxonomy of home automation: expert perspectives on the future of smarter homes. *Information Systems Frontiers*, 27(2):449–466.

Farrell, J. (2023). Google researchers find personal information can be accessed through ChatGPT queries. https://siliconangle.com/2023/11/29/google-researchers-find-personal-information-real-people-can-accessed-chatgpt-queries/. Accessed: 2024-6-18.

Faruk, L. I. D., Babakerkhell, M. D., Mongkolnam, P., Chongsuphajaisiddhi, V., Funilkul, S., and Pal, D. (2024). A review of subjective scales measuring the user experience of voice assistants. *IEEE Access*, 12:14893–14917.

Federal Trade Commission (2023). FTC says ring employees illegally surveilled customers, failed to stop hackers from taking control of users' cameras. https://www.ftc.gov/news-events/news/press-releases/2023/05/ftc-says-ring-employees-illegally-surveilled-customers-failed-stop-hackers-taking-control-users. Accessed: 2024-6-18.

Fernandes, E., Jung, J., and Prakash, A. (2016). Security analysis of emerging smart home applications. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 636–654. ieeexplore.ieee.org.

Fu, Y., Peng, H., Sabharwal, A., Clark, P., and Khot, T. (2022). Complexity-based prompting for multi-step reasoning. *arXiv [cs.CL]*.

Fuad, A. M., Ahmed, S. J., Anannya, N. J., Mridha, M. F., and Nur, K. (2024). An open-source voice command-based human-computer interaction system using speech recognition platforms. In *Proceedings of the 2nd International Conference on Big Data, IoT and Machine Learning*, pages 527–545. Springer Nature Singapore.

Gallo, S., Paterno, F., and Malizia, A. (2023). Conversational interfaces in IoT ecosystems: Where we are, what is still missing. In *Proceedings of the 22nd International Conference on Mobile and Ubiquitous Multimedia*, MUM '23, pages 279–293, New York, NY, USA. Association for Computing Machinery.

Gao, C., Chandrasekaran, V., Fawaz, K., and Banerjee, S. (2018). Traversing the quagmire that is privacy in your smart home. In *Proceedings of the 2018 Workshop on IoT Security and Privacy*, New York, NY, USA. ACM.

Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., and Neubig, G. (2022). PAL: Program-aided language models. *ICML*, abs/2211.10435:10764–10799.

Gazis and Katsiri (2021). Smart home IoT sensors: principles and applications-a review of low-cost and low-power solutions. *Int. J. Eng. Appl. (IREA)*.

Gemma Team, Kamath, A., Ferret, J., Pathak, S., Vieillard, N., Merhej, R., Perrin, S., Matejovicova, T., Ramé, A., Rivière, M., Rouillard, L., Mesnard, T., Cideron, G., Grill, J.-B., Ramos, S., Yvinec, E., Casbon, M., Pot, E., Penchev, I., Liu, G., Visin, F., Kenealy, K., Beyer, L., Zhai, X., Tsitsulin, A., Busa-Fekete, R., Feng, A., Sachdeva, N., Coleman, B., Gao, Y., Mustafa, B., Barr, I., Parisotto, E., Tian, D., Eyal, M., Cherry, C., Peter, J.-T., Sinopalnikov, D., Bhupatiraju, S., Agarwal, R., Kazemi, M., Malkin, D., Kumar, R., Vilar, D., Brusilovsky, I., Luo, J., Steiner, A., Friesen, A., Sharma, A., Sharma, A.,

Gilady, A. M., Goedeckemeyer, A., Saade, A., Feng, A., Kolesnikov, A., Bendebury, A., Abdagic, A., Vadi, A., György, A., Pinto, A. S., Das, A., Bapna, A., Miech, A., Yang, A., Paterson, A., Shenoy, A., Chakrabarti, A., Piot, B., Wu, B., Shahriari, B., Petrini, B., Chen, C., Lan, C. L., Choquette-Choo, C. A., Carey, C. J., Brick, C., Deutsch, D., Eisenbud, D., Cattle, D., Cheng, D., Paparas, D., Sreepathihalli, D. S., Reid, D., Tran, D., Zelle, D., Noland, E., Huizenga, E., Kharitonov, E., Liu, F., Amirkhanyan, G., Cameron, G., Hashemi, H., Klimczak-Plucińska, H., Singh, H., Mehta, H., Lehri, H. T., Hazimeh, H., Ballantyne, I., Szpektor, I., Nardini, I., Pouget-Abadie, J., Chan, J., Stanton, J., Wieting, J., Lai, J., Orbay, J., Fernandez, J., Newlan, J., Ji, J.-Y., Singh, J., Black, K., Yu, K., Hui, K., Vodrahalli, K., Greff, K., Qiu, L., Valentine, M., Coelho, M., Ritter, M., Hoffman, M., Watson, M., Chaturvedi, M., Moynihan, M., Ma, M., Babar, N., Noy, N., Byrd, N., Roy, N., Momchev, N., Chauhan, N., Sachdeva, N., Bunyan, O., Botarda, P., Caron, P., Rubenstein, P. K., Culliton, P., Schmid, P., Sessa, P. G., Xu, P., Stanczyk, P., Tafti, P., Shivanna, R., Wu, R., Pan, R., Rokni, R., Willoughby, R., Vallu, R., Mullins, R., Jerome, S., Smoot, S., Girgin, S., Iqbal, S., Reddy, S., Sheth, S., Põder, S., Bhatnagar, S., Panyam, S. R., Eiger, S., Zhang, S., Liu, T., Yacovone, T., Liechty, T., Kalra, U., Evci, U., Misra, V., Roseberry, V., Feinberg, V., Kolesnikov, V., Han, W., Kwon, W., Chen, X., Chow, Y., Zhu, Y., Wei, Z., Egyed, Z., Cotruta, V., Giang, M., Kirk, P., Rao, A., Black, K., Babar, N., Lo, J., Moreira, E., Martins, L. G., Sanseviero, O., Gonzalez, L., Gleicher, Z., Warkentin, T., Mirrokni, V., Senter, E., Collins, E., Barral, J., Ghahramani, Z., Hadsell, R., Matias, Y., Sculley, D., Petrov, S., Fiedel, N., Shazeer, N., Vinyals, O., Dean, J., Hassabis, D., Kavukcuoglu, K., Farabet, C., Buchatskaya, E., Alayrac, J.-B., Anil, R., Dmitry, Lepikhin, Borgeaud, S., Bachem, O., Joulin, A., Andreev, A., Hardin, C., Dadashi, R., and Hussenot, L. (2025). Gemma 3 technical report. *arXiv [cs.CL]*.

Gilbert, J. (2025). When Google Assistant is phased out, what happens to our smart speakers? `https://www.androidpolice.com/state-of-smart-speakers-dc-google-assistant/`. Accessed: 2025-7-16.

Giudici, M., Abbo, G. A., Belotti, O., Braccini, A., Dubini, F., Izzo, R. A., Crovari, P., and Garzotto, F. (2023). Assessing LLMs responses in the field of domestic sustainability: An exploratory study. In *2023 Third International Conference on Digital Data Processing (DDP)*, pages 42–48. IEEE.

Giudici, M., Padalino, L., Paolino, G., Paratici, I., Pascu, A. I., and Garzotto, F. (2024). Designing home automation routines using an LLM-based chatbot. *Des. Codes Cryptogr.*, 8(3):43.

Giudici, M., Sironi, A., Villa, I., Scherini, S., and Garzotto, F. (2025). Generating HomeAssistant automations using an LLM-based chatbot. *arXiv [cs.HC]*.

Gondi, S. and Pratap, V. (2021). Performance evaluation of offline speech recognition on edge devices. *Electronics*, 10(21):2697.

Gram-Hanssen, K. and Darby, S. J. (2018). "Home is where the smart is"? Evaluating smart home research and approaches against the concept of home. *Energy Research & Social Science*, 37:94–101.

Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., Yang, A., Fan, A., Goyal, A., Hartshorn, A., Yang, A., Mitra, A., Sravankumar, A., Korenev, A., Hinsvark, A., Rao, A., Zhang, A., Rodriguez, A., Gregerson, A., Spataru, A., Roziere, B., Biron, B., Tang, B., Chern, B., Caucheteux, C., Nayak, C., Bi, C., Marra, C., McConnell, C., Keller, C., Touret, C., Wu, C., Wong, C., Ferrer, C. C., Nikolaidis, C., Allonsius, D., Song, D., Pintz, D., Livshits, D., Wyatt, D., Esiobu, D., Choudhary, D., Mahajan, D., Garcia-Olano, D., Perino, D., Hupkes, D., Lakomkin, E., AlBadawy, E., Lobanova, E., Dinan, E., Smith, E. M., Radenovic, F., Guzmán, F., Zhang, F., Synnaeve, G., Lee, G., Anderson, G. L., Thattai, G., Nail, G., Mialon, G., Pang, G., Cucurell, G., Nguyen, H., Korevaar, H., Xu, H., Touvron, H., Zarov, I., Ibarra, I. A., Kloumann, I., Misra, I., Evtimov, I., Zhang, J., Copet, J., Lee, J., Geffert, J., Vranes, J., Park, J., Mahadeokar, J., Shah, J., van der Linde, J., Billock, J., Hong, J., Lee, J., Fu, J., Chi, J., Huang, J., Liu, J., Wang, J., Yu, J., Bitton, J., Spisak, J., Park, J., Rocca, J., Johnstun, J., Saxe, J., Jia, J., Alwala, K. V., Prasad, K., Upasani, K., Plawiak, K., Li, K., Heafield, K., Stone, K., El-Arini, K., Iyer, K., Malik, K., Chiu, K., Bhalla, K., Lakhotia, K., Rantala-Yeary, L., van der Maaten, L., Chen, L., Tan, L., Jenkins, L., Martin, L., Madaan, L., Malo, L., Blecher, L., Landzaat, L., de Oliveira, L., Muzzi, M., Pasupuleti, M., Singh, M., Paluri, M., Kardas, M., Tsimpoukelli, M., Oldham, M., Rita, M., Pavlova, M., Kambadur, M., and et al. (2024). The llama 3 herd of models. *arXiv [cs.AI]*.

Gunge and Yalagi (2016). Smart home automation: A literature review. *Ann. Math. Inform.*

Haas, G., Rietzler, M., Jones, M., and Rukzio, E. (2022). Keep it short: A comparison of voice assistants' response behavior. In *CHI Conference on Human Factors in Computing Systems*, New York, NY, USA. ACM.

Haller (2010). The things in the internet of things. *Poster at the (IoT 2010). Tokyo, Japan, November*.

Haney, J. M., Furman, S. M., and Acar, Y. (2020). Smart home security and privacy mitigations: Consumer perceptions, practices, and challenges. In *HCI for Cybersecurity, Privacy and Trust*, Lecture notes in computer science, pages 393–411. Springer International Publishing, Cham.

Hannun, A. (2021). The history of speech recognition to the year 2030. *arXiv [cs.CL]*.

Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., and Ng, A. Y. (2014). Deep speech: Scaling up end-to-end speech recognition. *arXiv [cs.CL]*.

Hardi, A., Sita, I. V., and Muresan, V. (2025). Esp32-based embedded platform smart home system. In *2025 International Conference on Electrical and Computer Engineering Researches (ICECER)*, pages 1–6.

Harris, D. E. (2024). Open-source AI is uniquely dangerous. `https://spectrum.ieee.org/open-source-ai-2666932122`. Accessed: 2024-8-26.

Heartfield, R., Loukas, G., Budimir, S., Bezemskij, A., Fontaine, J. R. J., Filippoupolitis, A., and Roesch, E. (2018). A taxonomy of cyber-physical threats and impact in the smart home. *Comput. Secur.*, 78:398–428.

Hernandez, F., Nguyen, V., Ghannay, S., Tomashenko, N., and Estève, Y. (2018). TED-LIUM 3: twice as much data and corpus repartition for experiments on speaker adaptation. *arXiv [cs.CL]*.

Hernández Acosta, L. and Reinhardt, D. (2022). A survey on privacy issues and solutions for voice-controlled digital assistants. *Pervasive Mob. Comput.*, 80:101523.

Hertz, G. (2011). Arduino microcontrollers and the queen's hamlet: Utilitarian and hedonized DIY practices in contemporary electronic culture. *Integration through computation. 31st Annual Conference of the Association for Computer Aided Design in Architecture, Banff, Alberta, USA*.

Hewitt, M. and Cunningham, H. (2022). Taxonomic classification of IoT smart home voice control. *arXiv [cs.HC]*.

Hewitt, M. and Cunningham, H. (2024). Towards privacy-preserving voice control in smart home IoT: A taxonomy. In *Lecture Notes in Networks and Systems*, Lecture notes in networks and systems, pages 614–619. Springer Nature Switzerland, Cham.

Holroyd, P., Watten, P., and Newbury, P. (2010). Why is my home not smart? In *Aging Friendly Technology for Health and Independence*, pages 53–59. Springer Berlin Heidelberg.

Hu, H., Lu, H., Zhang, H., Song, Y.-Z., Lam, W., and Zhang, Y. (2023). Chain-of-symbol prompting elicits planning in large langauge models. *arXiv [cs.CL]*.

Huang, C.-C., Liu, A., and Zhou, P.-C. (2015). Using ontology reasoning in building a simple and effective dialog system for a smart home system. In *2015 IEEE International Conference on Systems, Man, and Cybernetics*, pages 1508–1513.

Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q., Peng, W., Feng, X., Qin, B., and Liu, T. (2023). A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *arXiv [cs.CL]*.

Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q., Peng, W., Feng, X., Qin, B., and Liu, T. (2025). A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions. *ACM Trans. Inf. Syst.*, 43(2).

Islas-Cota, E., Gutierrez-Garcia, J. O., Acosta, C. O., and Rodríguez, L.-F. (2022). A systematic review of intelligent assistants. *Future Gener. Comput. Syst.*, 128:45–62.

Isyanto, H., Arifin, A. S., and Suryanegara, M. (2020). Performance of smart personal assistant applications based on speech recognition technology using IoT-based voice commands. In *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 640–645. IEEE.

Jesse, M., Steinberger, C., and Schartner, P. (2021). In search of a conversational user interface for personal health assistance. In *Proceedings of the 14th International Joint Conference on Biomedical Engineering Systems and Technologies*. SCITEPRESS - Science and Technology Publications.

Katuk, N., Ku-Mahamud, K. R., Zakaria, N. H., and Maarof, M. A. (2018). Implementation and recent progress in cloud-based smart home automation systems. In *2018 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, pages 71–77. ieeexplore.ieee.org.

Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., Vardhamanan, S., Haq, S., Sharma, A., Joshi, T. T., Moazam, H., Miller, H., Zaharia, M., and Potts, C. (2023). DSPy: Compiling declarative language model calls into self-improving pipelines. *arXiv [cs.CL]*.

King, E., Yu, H., Lee, S., and Julien, C. (2023a). " Get ready for a party": Exploring smarter smart spaces with help from large language models. *arXiv preprint arXiv:2303.14143*.

King, E., Yu, H., Lee, S., and Julien, C. (2023b). Sasha: creative goal-oriented reasoning in smart homes with large language models. *arXiv [cs.HC]*.

King, E., Yu, H., Vartak, S., Jacob, J., Lee, S., and Julien, C. (2024). Thoughtful Things: Building human-centric smart devices with small language models. *arXiv [cs.HC]*.

Kodali, R. K. and Mahesh, K. S. (2017). Low cost implementation of smart home automation. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 461–466. ieeexplore.ieee.org.

Kojima, T., Gu, S., Reid, M., Matsuo, Y., and Iwasawa, Y. (2022). Large language models are zero-shot reasoners. *Adv. Neural Inf. Process. Syst.*, abs/2205.11916.

Köpf, A., Kilcher, Y., von Rütte, D., Anagnostidis, S., Tam, Z.-R., Stevens, K., Barhoum, A., Duc, N. M., Stanley, O., Nagyfi, R., Es, S., Suri, S., Glushkov, D., Dantuluri, A., Maguire, A., Schuhmann, C., Nguyen, H., and Mattick, A. (2023). OpenAssistant conversations – democratizing large language model alignment. *arXiv [cs.CL]*.

Kordts, B., Gerlach, B., and Schrader, A. (2021). Towards self-explaining ambient applications. In *Proceedings of the 14th PErvasive Technologies Related to Assistive Environments Conference*, PETRA '21, pages 383–390, New York, NY, USA. Association for Computing Machinery.

Koreshoff, T. L., Robertson, T., and Leong, T. W. (2013). Internet of things: a review of literature and products. In *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration*, OzCHI '13, pages 335–344, New York, NY, USA. Association for Computing Machinery.

Kröger, J. L., Lutz, O. H.-M., and Raschke, P. (2020). Privacy implications of voice and speech analysis – information disclosure by inference. In *Privacy and Identity Management. Data for Better Living: AI and Privacy*, IFIP advances in information and communication technology, pages 242–258. Springer International Publishing, Cham.

Kuhn, K., Kersken, V., Reuter, B., Egger, N., and Zimmermann, G. (2024). Measuring the accuracy of automatic speech recognition solutions. *ACM Transactions on Accessible Computing*, 16(4):1–23.

Kumar, Shen, Case, Garg, and others (2019). All things considered: An analysis of IoT devices on home networks. *28th USENIX security*.

Kumar, S., Kumar, B., Sharma, K., Raj, R., and Kumar, S. (2021). IoT based secured home automation system using NLP. In *2021 International Conference on Advancements in Electrical, Electronics, Communication, Computing and Automation (ICAECA)*, pages 1–5.

Kuznetsov, S. and Paulos, E. (2010). Rise of the expert amateur: DIY projects, communities, and cultures. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries*, New York, NY, USA. ACM.

Kwartler, T., Berman, M., and Aqrawi, A. (2024). Good parenting is all you need – multi-agentic LLM hallucination mitigation. *arXiv [cs.CR]*.

Lamkin, P. (2018). Smart home visions through the ages: The history of home automation. `https://www.the-ambient.com/explainers/visions-through-the-ages-history-of-home-automation-178/`. Accessed: 2025-8-21.

Lamkin, P. (2025). What is Alexa plus... and how much does it cost? `https://www.the-ambient.com/explainers/what-is-alexa-plus-how-much-does-it-cost/`. Accessed: 2025-8-22.

Lampinen, A. K., Dasgupta, I., Chan, S. C. Y., Matthewson, K., Tessler, M. H., Creswell, A., McClelland, J. L., Wang, J. X., and Hill, F. (2022). Can language models learn from explanations in context? *arXiv [cs.CL]*.

Lau, J., Zimmerman, B., and Schaub, F. (2018). Alexa, are you listening? privacy perceptions, concerns and privacy-seeking behaviors with smart speakers. *Proc. ACM Hum.-Comput. Interact.*, 2(CSCW):1–31.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-T., Rocktäschel, T., Riedel, S., and Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *arXiv [cs.CL]*.

Li, F., Huang, J., Gao, Y., and Dong, W. (2023). ChatIoT: Zero-code Generation of Trigger-action Based IoT Programs with ChatGPT. In *Proceedings of the 7th Asia-Pacific Workshop on Networking*, APNET '23, pages 219–220, New York, NY, USA. Association for Computing Machinery.

Li, J., Lavrukhin, V., Ginsburg, B., Leary, R., Kuchaiev, O., Cohen, J. M., Nguyen, H., and Gadde, R. T. (2019). Jasper: An end-to-end convolutional neural acoustic model. *arXiv [eess.AS]*.

Li, J., Yuan, Y., and Zhang, Z. (2024). Enhancing llm factual accuracy with rag to counter hallucinations: A case study on domain-specific queries in private knowledge-bases. *arXiv preprint arXiv:2403.10446*.

Li, S., Guo, Y., Yao, J., Liu, Z., and Wang, H. (2025). Homebench: Evaluating llms in smart homes with valid and invalid instructions across single and multiple devices. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12230–12250.

Li, X., Wu, W., Qin, L., and Yin, Q. (2021). How to evaluate your dialogue models: A review of approaches. *arXiv [cs.CL]*.

Liang, P., Bommasani, R., Lee, T., Tsipras, D., Soylu, D., Yasunaga, M., Zhang, Y., Narayanan, D., Wu, Y., Kumar, A., Newman, B., Yuan, B., Yan, B., Zhang, C., Cosgrove, C., Manning, C. D., Ré, C., Acosta-Navas, D., Hudson, D. A., Zelikman, E., Durmus, E., Ladhak, F., Rong, F., Ren, H., Yao, H., Wang, J., Santhanam, K., Orr, L., Zheng, L., Yuksekgonul, M., Suzgun, M., Kim, N., Guha, N., Chatterji, N., Khattab, O., Henderson, P., Huang, Q., Chi, R., Xie, S. M., Santurkar, S., Ganguli, S., Hashimoto, T., Icard, T., Zhang, T., Chaudhary, V., Wang, W., Li, X., Mai, Y., Zhang, Y., and Koreeda, Y. (2022). Holistic evaluation of language models. *arXiv [cs.CL]*.

Lieberman, H. and Espinosa, J. (2006). A goal-oriented interface to consumer electronics using planning and commonsense reasoning. In *Proceedings of the 11th international conference on Intelligent user interfaces*, IUI '06, pages 226–233, New York, NY, USA. Association for Computing Machinery.

Lin, H. and Bergmann, N. W. (2016). IoT privacy and security challenges for smart home environments. *Information*, 7(3):44.

Lin, S., Hilton, J., and Evans, O. (2022). TruthfulQA: Measuring how models mimic human falsehoods. In Muresan, S., Nakov, P., and Villavicencio, A., editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3214–3252, Dublin, Ireland. Association for Computational Linguistics.

Litayem, N. (2024). Scalable smart home management with esp32-s3: A low-cost solution for accessible home automation. In *2024 International Conference on Computer and Applications (ICCA)*, pages 1–7.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. (2023a). Lost in the middle: How language models use long contexts. *arXiv [cs.CL]*.

Liu, Z., Yao, W., Zhang, J., Xue, L., Heinecke, S., Murthy, R., Feng, Y., Chen, Z., Niebles, J. C., Arpit, D., Xu, R., Mui, P., Wang, H., Xiong, C., and Savarese, S. (2023b). BOLAA: Benchmarking and orchestrating LLM-augmented autonomous agents. *arXiv [cs.AI]*.

López, T. S., Ranasinghe, D. C., Patkai, B., and McFarlane, D. (2011). Taxonomy, technology and applications of smart objects. *Inf. Syst. Front.*, 13(2):281–300.

Lu, S., Bigoulaeva, I., Sachdeva, R., Madabushi, H. T., and Gurevych, I. (2023). Are emergent abilities in large language models just in-context learning? *arXiv [cs.CL]*.

Lu, Y., Bartolo, M., Moore, A., Riedel, S., and Stenetorp, P. (2021). Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *arXiv [cs.CL]*.

Luger, E. and Sellen, A. (2016). "Like having a really bad PA": The gulf between user expectation and experience of conversational agents. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 5286–5297, New York, NY, USA. Association for Computing Machinery.

Lugosch, L., Ravanelli, M., Ignoto, P., Tomar, V. S., and Bengio, Y. (2019). Speech model pre-training for end-to-end spoken language understanding. *arXiv [eess.AS]*.

Luqman, M. and Faridi, A. R. (2018). An overview on security issues in internet of things. In *2018 4th International Conference on Computing Communication and Automation (ICCCA)*, pages 1–6. IEEE.

Madadi-Barough, S., Ruiz-Blanco, P., Lin, J., Vidal, R., and Gomez, C. (2025). Matter: Iot interoperability for smart homes. *IEEE Communications Magazine*, 63(4):106–112.

Malkin, Deatrick, Tong, Wijesekera, and others (2019). Privacy attitudes of smart speaker users. *Proc. Priv. Enhancing Technol.*

Marikyan, D., Papagiannidis, S., and Alamanos, E. (2019). A systematic review of the smart home literature: A user perspective. *Technol. Forecast. Soc. Change*, 138:139–154.

McLean, G. and Osei-Frimpong, K. (2019). Hey Alexa ... examine the variables influencing the use of artificial intelligent in-home voice assistants. *Comput. Human Behav.*, 99:28–37.

McManus, S. (2024). Are rainy days ahead for cloud computing? `https://www.bbc.co.uk/news/articles/cd114lllyp6o`. Accessed: 2024-8-27.

Mehrabani, M., Bangalore, S., and Stern, B. (2015). Personalized speech recognition for internet of things. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 369–374. ieeexplore.ieee.org.

Ming, Y., Purushwalkam, S., Pandit, S., Ke, Z., Nguyen, X.-P., Xiong, C., and Joty, S. (2024). Faitheval: Can your language model stay faithful to context, even if" the moon is made of marshmallows". *arXiv preprint arXiv:2410.03727*.

Mishakova, A., Portet, F., Desot, T., and Vacher, M. (2019). Learning natural language understanding systems from unaligned labels for voice command in smart homes. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 832–837. ieeexplore.ieee.org.

Mistral AI Team (2023). Mistral 7B. `https://mistral.ai/news/announcing-mistral-7b/`. Accessed: 2024-8-24.

Morris, C. (2024). Extreme heat is raising concerns about the durability of our tech infrastructure. `https://www.fastcompany.com/91150666/extreme-heat-raising-concerns-about-tech-infrastructure`. Accessed: 2024-8-27.

Morrison, S. (2022). Amazon's ring privacy problem is back. `https://www.vox.com/recode/23207072/amazon-ring-privacy-police-footage`. Accessed: 2022-7-15.

Mun, H., Lee, H., Kim, S., and Lee, Y. (2020). A smart speaker performance measurement tool. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, SAC '20, pages 755–762, New York, NY, USA. Association for Computing Machinery.

Munir, A., Kashif Ehsan, S., Mohsin Raza, S. M., and Mudassir, M. (2019). Face and speech recognition based smart home. In *2019 International Conference on Engineering and Emerging Technologies (ICEET)*, pages 1–5. ieeexplore.ieee.org.

Murshed, M. G. S., Murphy, C., Hou, D., Khan, N., Ananthanarayanan, G., and Hussain, F. (2021). Machine learning at the network edge: A survey. *ACM Comput. Surv.*, 54(8):1–37.

Nasr, M., Carlini, N., Hayase, J., Jagielski, M., Feder Cooper, A., Ippolito, D., Choquette-Choo, C. A., Wallace, E., Tramèr, F., and Lee, K. (2023). Scalable extraction of training data from (production) language models. *arXiv [cs.LG]*.

Nicholls, L., Strengers, Y., and Sadowski, J. (2020). Social impacts and control in the smart home. *Nature Energy*, 5(3):180–182.

Nori, H., Usuyama, N., King, N., McKinney, S. M., Fernandes, X., Zhang, S., and Horvitz, E. (2024). From medprompt to o1: Exploration of run-time strategies for medical challenge problems and beyond. *arXiv [cs.CL]*.

Noura, M., Heil, S., and Gaedke, M. (2020). VISH: Does your smart home dialogue system also need training data? In *Web Engineering*, pages 171–187. Springer International Publishing.

Oewel, B., Ammari, T., and Brewer, R. N. (2023). Voice assistant use in long-term care. In *Proceedings of the 5th International Conference on Conversational User Interfaces*, number Article 24 in CUI '23, pages 1–10, New York, NY, USA. Association for Computing Machinery.

Omidvarborna, H., Kumar, P., Hayward, J., Gupta, M., and Nascimento, E. G. S. (2021). Low-cost air quality sensing towards smart homes. *Atmosphere*, 12(4):453.

Oyucu, S. (2021). Integration of cloud-based speech recognition system to the internet of things based smart home automation. In *2021 3rd International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, pages 1–3.

Pais, S., Cordeiro, J., and Jamil, M. L. (2022). NLP-based platform as a service: a brief review. *J. Big Data*, 9(1).

Palanca, J., Val, E. d., Garcia-Fornes, A., Billhardt, H., Corchado, J. M., and Julián, V. (2018). Designing a goal-oriented smart-home environment. *Inf. Syst. Front.*, 20(1):125–142.

Pallas, F., Staufer, D., and Kuhlenkamp, J. (2020). Evaluating the accuracy of cloud NLP services using ground-truth experiments. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 341–350. IEEE.

Panayotov, V., Chen, G., Povey, D., and Khudanpur, S. (2015). Librispeech: An ASR corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210. ieeexplore.ieee.org.

Patil, S. G., Zhang, T., Wang, X., and Gonzalez, J. E. (2023). Gorilla: Large language model connected with massive APIs. *arXiv [cs.CL]*.

Patil, T. H., V, V., Madiwalar, V., Hundekar, V. P., and Kumar, P. (2024). A review paper on voice recognition and response (vrr). In *2024 2nd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT)*, pages 1086–1094.

Peinl, R., Rizk, B., and Szabad, R. (2020). Open source speech recognition on edge devices. In *2020 10th International Conference on Advanced Computer Information Technologies (ACIT)*, pages 441–445.

Philip, N. Y., Rodrigues, J. J. P., Wang, H., Fong, S. J., and Chen, J. (2021). Internet of things for in-home health monitoring systems: Current advances, challenges and future directions. *IEEE J. Sel. Areas Commun.*, 39(2):300–310.

Pinto, D., Arnau, J.-M., and González, A. (2020). Design and evaluation of an ultra low-power human-quality speech recognition system. *ACM Trans. Archit. Code Optim.*, 17(4):1–19.

Polyakov, E. V., Mazhanov, M. S., Rolich, A. Y., Voskov, L. S., Kachalova, M. V., and Polyakov, S. V. (2018). Investigation and development of the intelligent voice assistant for the internet of things using machine learning. In *2018 Moscow Workshop on Electronic and Networking Technologies (MWENT)*, pages 1–5.

Poongothai, Sundar, and Prabhu (2018). Implementation of IoT based intelligent voice controlled laboratory using Google assistant. *Int. J. High Risk Behav. Addict.*

Porcheron, M., Fischer, J. E., Reeves, S., and Sharples, S. (2018). Voice interfaces in everyday life. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, number Paper 640 in CHI '18, pages 1–12, New York, NY, USA. Association for Computing Machinery.

Putthapipat, P., Woralert, C., and Sirinimnuankul, P. (2018). Speech recognition gateway for home automation on open platform. In *2018 International Conference on Electronics, Information, and Communication (ICEIC)*, pages 1–4.

Pyae, A. and Joelsson, T. N. (2018). Investigating the usability and user experiences of voice user interface: a case of google home smart speaker. In *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct*, MobileHCI '18, pages 127–131, New York, NY, USA. Association for Computing Machinery.

Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y., Lin, Y., Cong, X., Tang, X., Qian, B., Zhao, S., Hong, L., Tian, R., Xie, R., Zhou, J., Gerstein, M., Li, D., Liu, Z., and Sun, M. (2023). ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *arXiv [cs.AI]*.

Qiu, L., Ye, Y., Gao, Z., Zou, X., Chen, J., Gui, Z., Huang, W., Xue, X., Qiu, W., and Zhao, K. (2025). Blueprint First, Model Second: A Framework for Deterministic LLM Workflow. *arXiv preprint arXiv:2508.02721*.

Quach, K. (2022). Amazon opens MASSIVE AI speech dataset so Alexa can speak your language. `https://www.theregister.com/2022/04/20/amazon_ai_speech/`. Accessed: 2022-6-13.

Qwen, Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., Lin, H., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Lin, J., Dang, K., Lu, K., Bao, K., Yang, K., Yu, L., Li, M., Xue, M., Zhang, P., Zhu, Q., Men, R., Lin, R., Li, T., Tang, T., Xia, T., Ren, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Wan, Y., Liu, Y., Cui, Z., Zhang, Z., and Qiu, Z. (2024). Qwen2.5 technical report. *arXiv [cs.CL]*.

Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., and Sutskever, I. (2022). Robust speech recognition via large-scale weak supervision. *ICML*, pages 28492–28518.

Rai, A., Rahangdale, S., Anand, U., and Mukherjee, A. (2025). ASR-FAIRBENCH: Measuring and benchmarking equity across speech recognition systems. *arXiv preprint arXiv:2505.11572*.

Raiaan, M. A. K., Mukta, M. S. H., Fatema, K., Fahad, N. M., Sakib, S., Mim, M. M. J., Ahmad, J., Ali, M. E., and Azam, S. (2024). A review on large language models: Architectures, applications, taxonomies, open issues and challenges. *IEEE Access*, 12:26839–26874.

Rani, M., Mishra, B. K., Thakker, D., Babar, M., Jones, W., and Din, A. (2024a). Biases and trustworthiness challenges with mitigation strategies for large language models in healthcare. In *2024 International Conference on IT and Industrial Technologies (ICIT)*, pages 1–6. IEEE.

Rani, M., Mishra, B. K., Thakker, D., and Khan, M. N. (2024b). To enhance graph-based retrieval-augmented generation (RAG) with robust retrieval techniques. In *2024 18th International Conference on Open Source Systems and Technologies (ICOSST)*, pages 1–6. IEEE.

Rani, P. J., Bakthakumar, J., Kumaar, B. P., Kumaar, U. P., and Kumar, S. (2017). Voice controlled home automation system using natural language processing (NLP) and internet of things (IoT). In *2017 Third International Conference on Science Technology Engineering & Management (ICONSTEM)*, pages 368–373. ieeexplore.ieee.org.

Ravanelli, M., Cristoforetti, L., Gretter, R., Pellin, M., Sosi, A., and Omologo, M. (2015). The DIRHA-ENGLISH corpus and related tasks for distant-speech recognition in domestic environments. In *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 275–282. IEEE.

Ravichander, A., Ghela, S., Wadden, D., and Choi, Y. (2025). HALoGEN: Fantastic LLM hallucinations and where to find them. In Che, W., Nabende, J., Shutova, E., and Pilehvar, M. T., editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1402–1425, Vienna, Austria. Association for Computational Linguistics.

Rey-Jouanchicot, J., Bottaro, A., Campo, E., Bouraoui, J.-L., Vigouroux, N., and Vella, F. (2024). Leveraging Large Language Models for enhanced personalised user experience in Smart Homes. In *UBICOMM*, Venise, Italy. IARA.

Rivkin, D., Hogan, F., Feriani, A., Konar, A., Sigal, A., Liu, S., and Dudek, G. (2023). SAGE: Smart home agent with grounded execution. *arXiv [cs.AI]*.

Rouchon, C. (2017). Benchmarking microphone arrays: ReSpeaker, conexant, MicroSemi AcuEdge, matrix creator, MiniDSP, PlayStation eye. `https://medium.com/snips-ai/benchmarking-microphone-arrays-respeaker-conexant-microsemi-acuedge-matrix-creator-minidsp-950de8876fda`. Accessed: 2024-8-27.

Saade, A., Coucke, A., Caulier, A., Dureau, J., Ball, A., Bluche, T., Leroy, D., Doumouro, C., Gisselbrecht, T., Caltagirone, F., Lavril, T., and Primet, M. (2018). Spoken language understanding on the edge. *arXiv [cs.CL]*.

Sadeghi, M., Herbold, L., Unterbusch, M., and Vogelsang, A. (2024). SmartEx: A framework for generating user-centric explanations in smart environments. *arXiv [cs.HC]*.

Sahoo, P., Singh, A. K., Saha, S., Jain, V., Mondal, S., and Chadha, A. (2024). A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv [cs.AI]*.

Samuel, S. S. I. (2016). A review of connectivity challenges in IoT-smart home. In *2016 3rd MEC International Conference on Big Data and Smart City (ICBDSC)*, pages 1–4. ieeexplore.ieee.org.

Santos, R., Abreu, J., Beça, P., Rodrigues, A., and Fernandes, S. (2020). Voice interaction on TV: analysis of natural language interaction models and recommendations for voice user interfaces. *Multimed. Tools Appl.*, 79(47):35689–35716.

Sas, C. and Neustaedter, C. (2017). Exploring DIY practices of complex home technologies. *ACM Trans. Comput.-Hum. Interact.*, 24(2):1–29.

Satyanarayanan, M. (2017). The emergence of edge computing. *Computer*, 50(1):30–39.

Schulhoff, S., Ilie, M., Balepur, N., Kahadze, K., Liu, A., Si, C., Li, Y., Gupta, A., Han, H., Schulhoff, S., Dulepet, P. S., Vidyadhara, S., Ki, D., Agrawal, S., Pham, C., Kroiz, G., Li, F., Tao, H., Srivastava, A., Da Costa, H., Gupta, S., Rogers, M. L., Goncearenco, I., Sarli, G., Galynker, I., Peskoff, D., Carpuat, M., White, J., Anadkat, S., Hoyle, A., and Resnik, P. (2024). The prompt report: A systematic survey of prompting techniques. *arXiv [cs.CL]*.

Serrenho, T. and Bertoldi, P. (2019). Smart home and appliances: State of the art – energy, communications, protocols, standards. EUR Report EUR 29750 EN, Publications Office of the European Union, Luxembourg. JRC113988.

Setiawan, P. and Yusuf, R. (2022). IoT device control with offline automatic speech recognition on edge device. In *2022 12th International Conference on System Engineering and Technology (ICSET)*, pages 111–115.

Seymour, W. (2018). How loyal is your Alexa? Imagining a respectful smart assistant. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*, number Paper SRC20 in CHI EA '18, pages 1–6, New York, NY, USA. Association for Computing Machinery.

Seymour, W., Zhan, X., Cote, M., and Such, J. (2022). A systematic review of ethical concerns with voice assistants. *arXiv [cs.HC]*.

Shafei, H. A. and Tan, C. C. (2024). A closer look at access control in multi-user voice systems. *IEEE Access*, 12:40933–40946.

Shah, M. A., Noguero, D. S., Heikkila, M. A., Raj, B., and Kourtellis, N. (2024). Speech robust bench: A robustness benchmark for speech recognition. *arXiv preprint arXiv:2403.07937*.

Shao, Z., Gong, Y., Shen, Y., Huang, M., Duan, N., and Chen, W. (2023). Synthetic prompting: Generating chain-of-thought demonstrations for large language models. *ICML*, abs/2302.00618:30706–30775.

Sharrab, Y. O., Attar, H., Eljinini, M. A. H., Al-Omary, Y., and Al-Momani, W. E. (2025). Advancements in speech recognition: A systematic review of deep learning transformer models, trends, innovations, and future directions. *IEEE Access*, 13:46925–46940.

Shehab, J. N. (2019). Remote control using voice recognition based on arduino. *DJES*, 12(3):22–28.

Shehab, S. H., Rahman, M. L., Hasan, M. H., Uddin, M. I., Mahmood, S. A., and Chowdhury, A. E. (2020). Home automation system using gesture pattern & voice recognition for paralyzed people. In *2020 11th International Conference on Electrical and Computer Engineering (ICECE)*, pages 25–28. ieeexplore.ieee.org.

Shouli, A., Bobkova, Y., and Shrestha, A. K. (2025). Covert surveillance in smart devices: A scour framework analysis of youth privacy implications. In *2025 IEEE 16th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pages 0124–0133.

Shrivastava, A., Raturi, A., Sharma, A., Rao, A., Chinthamu, N., and Sankhyan, A. (2023). Advancements in speech recognition technology: A cutting-edge tool for improved speech analysis and interaction. In *2023 3rd International Conference on Pervasive Computing and Social Networking (ICPCSN)*, pages 1786–1792. IEEE.

Sicari, S., Rizzardi, A., Grieco, L. A., and Coen-Porisini, A. (2015). Security, privacy and trust in internet of things: The road ahead. *Comput. Netw.*, 76:146–164.

Sturgess, J., Nurse, J. R. C., and Zhao, J. (2018). A capability-oriented approach to assessing privacy risk in smart home ecosystems. In *Living in the Internet of Things: Cybersecurity of the IoT - 2018*, pages 1–8. ieeexplore.ieee.org.

Sudharsan, B., Corcoran, P., and Ali, M. I. (2022). Smart speaker design and implementation with biometric authentication and advanced voice interaction capability. *arXiv [cs.CR]*.

Suh, C. and Ko, Y.-B. (2008). Design and implementation of intelligent home control systems based on active sensor networks. *IEEE Trans. Consum. Electron.*, 54(3):1177–1184.

Sun, Q., Yu, W., Kochurov, N., Hao, Q., and Hu, F. (2013). A multi-agent-based intelligent sensor and actuator network design for smart house and home automation. *Journal of Sensor and Actuator Networks*, 2(3):557–588.

Sutar, P., Sarkar, S. A., Tagare, A., and Pawar, A. (2024). A review on iot-enabled smart homes using ai. In *2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pages 1–6.

Tabassum, M., Kosinski, T., and Lipford, H. R. (2019). "I don't own the data": End user perceptions of smart home device data practices and risks. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, pages 435–450, Santa Clara, CA. USENIX Association.

Tablan, V., Roberts, I., Cunningham, H., and Bontcheva, K. (2013). GATECloud.net: a platform for large-scale, open-source text processing on the cloud. *Philos. Trans. A Math. Phys. Eng. Sci.*, 371(1983):20120071.

Tadimeti, D., Georgila, K., and Traum, D. (2022). Evaluation of off-the-shelf speech recognizers on different accents in a dialogue domain. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 6001–6008.

Teknium, R., Quesnelle, J., and Guang, C. (2024). Hermes 3 technical report. *arXiv [cs.CL]*.

The Washington Post (2024). Wyze security issue exposed private cameras to 13,000 strangers. *The Washington Post*.

Thushari, P. D. (2022). Current security and privacy issues, and concerns of internet of things (IoT) and cloud computing: A review. In *2022 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, pages 13–17. IEEE.

Todericiu, I. A. (2025). Virtual assistants: a review of the next frontier in AI interaction. *Acta Universitatis Sapientiae, Informatica*, 17(1):1.

Tomasello, P., Shrivastava, A., Lazar, D., Hsu, P.-C., Le, D., Sagar, A., Elkahky, A., Copet, J., Hsu, W.-N., Adi, Y., Algayres, R., Nguyen, T. A., Dupoux, E., Zettlemoyer, L., and Mohamed, A. (2023). Stop: A dataset for spoken task oriented semantic parsing. In *2022 IEEE Spoken Language Technology Workshop (SLT)*, pages 991–998.

Tonmoy, S., Zaman, S., Jain, V., Rani, A., Rawte, V., Chadha, A., and Das, A. (2024). A comprehensive survey of hallucination mitigation techniques in large language models. *arXiv preprint arXiv:2401.01313*.

Torkamani, M. J. and Zarin, I. (2025). Adaptive edge-cloud inference for speech-to-action systems using asr and large language models. *arXiv preprint arXiv:2512.12769*.

Tukur, Y. M., Thakker, D., and Awan, I.-U. (2019). Multi-layer approach to internet of things (IoT) security. In *2019 7th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 109–116. IEEE.

Turton, W. (2022). Tech giants duped into giving up data used to sexually extort minors. *Bloomberg News*.

Umesh, R., Sudharasan Dev, K., and Praveen Kumar, M. (2025). AI-Powered Offline Voice Assistant for Rural Communities. In *2025 7th International Conference on Innovative Data Communication Technologies and Application (ICIDCA)*, pages 1468–1474.

Vacher, M., Caffiau, S., Portet, F., Meillon, B., Roux, C., Elias, E., Lecouteux, B., and Chahuara, P. (2015). Evaluation of a context-aware voice interface for ambient assisted living: Qualitative user study vs. quantitative system evaluation. *ACM Trans. Access. Comput.*, 7(2):1–36.

Varghese, B., de Lara, E., Ding, A. Y., Hong, C.-H., Bonomi, F., Dustdar, S., Harvey, P., Hewkin, P., Shi, W., Thiele, M., and Willis, P. (2021). Revisiting the arguments for edge computing research. *IEEE Internet Comput.*, 25(5):36–42.

Vatsal, S. and Dubey, H. (2024). A survey of prompt engineering methods in large language models for different NLP tasks. *arXiv [cs.CL]*.

Venkatraman, S., Overmars, A., and Thong, M. (2021). Smart home Automation—Use cases of a secure and integrated voice-control system. *Systems*, 9(4):77.

Viani, F., Robol, F., Polo, A., Rocca, P., Oliveri, G., and Massa, A. (2013). Wireless architectures for heterogeneous sensing in smart home applications: Concepts and real implementation. *Proc. IEEE*, 101(11):2381–2396.

Vincent, J. (2019). Yep, human workers are listening to recordings from Google Assistant, too. `https://www.theverge.com/2019/7/11/20690020/google-assistant-home-human-contractors-listening-recordings-vrt-nws`. Accessed: 2024-9-2.

Vishwakarma, S. K., Upadhyaya, P., Kumari, B., and Mishra, A. K. (2019). Smart energy efficient home automation system using IoT. In *2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*, pages 1–4.

Vojtas, P., Stepan, J., Sec, D., Cimler, R., and Krejcar, O. (2018). Voice recognition software on embedded devices. In *Intelligent Information and Database Systems*, pages 642–650. Springer International Publishing.

Wang, J., Lim, M. K., Wang, C., and Tseng, M.-L. (2021). The evolution of the internet of things (IoT) over the past 20 years. *Comput. Ind. Eng.*, 155:107174.

Wang, L., Xu, W., Lan, Y., Hu, Z., Lan, Y., Lee, R. K.-W., and Lim, E.-P. (2023). Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv [cs.CL]*.

Wang, Q., Anikina, T., Feldhus, N., Genabith, J., Hennig, L., and Möller, S. (2024a). LLMCheckup: Conversational examination of large language models via interpretability tools and self-explanations. In *Proceedings of the Third Workshop on Bridging Human–Computer Interaction and Natural Language Processing*, pages 89–104, Stroudsburg, PA, USA. Association for Computational Linguistics.

Wang, S., Zhu, Y., Liu, H., Zheng, Z., Chen, C., and Li, J. (2024b). Knowledge editing for large language models: A survey. *ACM Computing Surveys*, 57(3):1–37.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. (2022). Self-consistency improves chain of thought reasoning in language models. *arXiv [cs.CL]*.

Wang, Y. and Zhao, Y. (2023). Metacognitive prompting improves understanding in large language models. *arXiv [cs.CL]*.

Warden, P. (2018). Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv [cs.CL]*.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q. V., and Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Adv. Neural Inf. Process. Syst.*, 35:24824–24837.

Weston, J. and Sukhbaatar, S. (2023). System 2 attention (is something you might need too). *arXiv [cs.CL]*.

White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., and Schmidt, D. C. (2023). A prompt pattern catalog to enhance prompt engineering with ChatGPT. *arXiv [cs.SE]*.

Worthy, P., Matthews, B., and Viller, S. (2016). Trust me: Doubts and concerns living with the internet of things. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems*, New York, NY, USA. ACM.

Xi, Z., Chen, W., Guo, X., He, W., Ding, Y., Hong, B., Zhang, M., Wang, J., Jin, S., Zhou, E., Zheng, R., Fan, X., Wang, X., Xiong, L., Zhou, Y., Wang, W., Jiang, C., Zou, Y., Liu, X., Yin, Z., Dou, S., Weng, R., Cheng, W., Zhang, Q., Qin, W., Zheng, Y., Qiu, X., Huang, X., and Gui, T. (2023). The rise and potential of large language model based agents: A survey. *arXiv [cs.AI]*.

Xiang, Z., Zheng, L., Li, Y., Hong, J., Li, Q., Xie, H., Zhang, J., Xiong, Z., Xie, C., Yang, C., Song, D., and Li, B. (2024). GuardAgent: Safeguard LLM agents by a guard agent via knowledge-enabled reasoning. *arXiv [cs.LG]*.

Xiao, B., Kantarci, B., Kang, J., Niyato, D., and Guizani, M. (2024). Efficient prompting for LLM-based generative internet of things. *IEEE Internet Things J.*, 12(1):1–1.

Xu, Z., Jain, S., and Kankanhalli, M. (2024). Hallucination is inevitable: An innate limitation of large language models. *arXiv [cs.CL]*.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. (2022). ReAct: Synergizing reasoning and acting in language models. *arXiv [cs.CL]*.

Yasunaga, M., Chen, X., Li, Y., Pasupat, P., Leskovec, J., Liang, P., Chi, E. H., and Zhou, D. (2023). Large language models as analogical reasoners. *arXiv [cs.LG]*.

Ye, H., Liu, T., Zhang, A., Hua, W., and Jia, W. (2023). Cognitive mirage: A review of hallucinations in large language models. *arXiv [cs.CL]*.

Yu, H., Hua, J., and Julien, C. (2021). Analysis of IFTTT recipes to study how humans use internet-of-things (IoT) devices. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, SenSys '21, pages 537–541, New York, NY, USA. Association for Computing Machinery.

Yuneela, K. and Sharma, A. (2022). A review paper on technologies used in home automation system. In *2022 6th International Conference on Computing Methodologies and Communication (ICCMC)*, pages 366–371. IEEE.

Zhang, Y., Li, Y., Cui, L., Cai, D., Liu, L., Fu, T., Huang, X., Zhao, E., Zhang, Y., Chen, Y., Wang, L., Luu, A. T., Bi, W., Shi, F., and Shi, S. (2023). Siren's song in the AI ocean: A survey on hallucination in large language models. *arXiv [cs.CL]*.

Zhang, Z., Wang, C., Wang, Y., Shi, E., Ma, Y., Zhong, W., Chen, J., Mao, M., and Zheng, Z. (2025). Llm hallucinations in practical code generation: Phenomena, mechanism, and mitigation. *Proc. ACM Softw. Eng.*, 2(ISSTA).

Zhao, R., Li, X., Joty, S., Qin, C., and Bing, L. (2023a). Verify-and-edit: A knowledge-enhanced chain-of-thought framework. *arXiv [cs.CL]*.

Zhao, S., Tan, F., Fennedy, K., Goth, G., and Williams, A. (2023b). Heads-up computing. `https://cacm.acm.org/research/heads-up-computing/`. Accessed: 2024-6-18.

Zhao, T. Z., Wallace, E., Feng, S., Klein, D., and Singh, S. (2021). Calibrate before use: Improving few-shot performance of language models. *arXiv [cs.CL]*.

Zhao, X., Li, M., Lu, W., Weber, C., Lee, J. H., Chu, K., and Wermter, S. (2023c). Enhancing zero-shot chain-of-thought reasoning in large language models through logic. *arXiv [cs.CL]*.

Zhong, R., Ma, M., Zhou, Y., Lin, Q., Li, L., and Zhang, N. (2022). User acceptance of smart home voice assistant: a comparison among younger, middle-aged, and older adults. *Univers Access Inf Soc*, pages 1–18.

Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., Wang, X., Schuurmans, D., Cui, C., Bousquet, O., Le, Q., and Chi, E. (2022a). Least-to-most prompting enables complex reasoning in large language models. *arXiv [cs.AI]*.

Zhou, Y., Muresanu, A. I., Han, Z., Paster, K., Pitis, S., Chan, H., and Ba, J. (2022b). Large language models are human-level prompt engineers. *arXiv [cs.LG]*.

Zonios, C. and Tenentes, V. (2021). Energy efficient speech command recognition for private smart home IoT applications. In *2021 10th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pages 1–4. ieeexplore.ieee.org.

Zwakman, D. S., Pal, D., and Arpnikanondt, C. (2021). Usability evaluation of artificial intelligence-based voice assistants: The case of Amazon Alexa. *SN Comput Sci*, 2(1):28.

Appendices

Appendix A

Voice User Interface Devices

Voice Interface Devices						
Hubs						
Smart Speaker						
Device Name	**Function**	**Connectivity**	**Compatibility**	**MCU/ CPU**	**Input mechanisms**	**Output mechanisms**
Google Nest Mini	Google Assistant, Play audio	Wi-Fi, Bluetooth	Google Assistant	CPU	Microphone, touch, ultrasound sensing	Speaker, LEDs
Amazon Echo Dot (5th Gen)	Amazon Alexa, Play audio	Wi-Fi, Bluetooth	Amazon Alexa	CPU	Microphone, buttons	Speaker, LEDs
Apple HomePod Mini	Apple Siri, Play audio	Wi-Fi, Bluetooth, Thread	Apple Siri	CPU	Microphone, touch	Speaker, LEDs
Smart Display						
Google Nest Hub (2nd Gen)	Google Assistant, Display media	Wi-Fi, Bluetooth, Thread	Google Assistant	CPU	Microphone, touch screen, ambient light, motion, temperature sensors	Speaker, screen display
Amazon Echo Show 8	Amazon Alexa, Display media	Wi-Fi, Bluetooth	Amazon Alexa	CPU	Microphone, touch screen, buttons, camera, ambient light sensor	Speaker, screen display
Wearables						
Smart Watch						
Apple Watch S3	Apple Siri, Monitor health, Display information	Wi-Fi, Bluetooth	Apple Siri	CPU	Microphone, barometric altimeter, optical heart rate sensor, accelerometer, gyroscope, ambient light sensor, force touch, GPS	Speaker, screen display

Samsung Galaxy Watch 4	Voice Assistant services, Monitor health and fitness, Display information	Wi-Fi, Bluetooth, NFC	Samsung Bixby, Google Assistant	CPU, GPU	Microphone, barometer, accelerometer, gyroscope, optical heart rate sensor, electrical heart sensor, bioelectrical impedance analysis sensor, light sensor, geomagnetic sensor	Speaker, screen display
Smart Glasses						
Amazon Echo Frames	Amazon Alexa, Listen to media, Make calls, Receive notifications	Bluetooth	Amazon Alexa, Google Assistant, Apple Siri	MCU	Camera, microphone, ambient light sensor, accelerometer, capacitive touch, buttons	Speaker, LEDs
Smart Buds						
Google Pixel Buds A Series	Google Assistant, Listen to media, Make calls	Bluetooth	Google Assistant	MCU	Microphone, capacitive touch, IR proximity sensor, accelerometer	Speaker
Amazon Echo Buds (2nd Gen)	Amazon Alexa, Listen to media, Make calls	Bluetooth, Wi-Fi, Mobile data	Amazon Alexa, Google Assistant, Apple Siri	MCU	Microphone, touch, proximity sensor, accelerometer	Speaker
Apple AirPods (3rd Gen)	Apple Siri, Listen to media, Make calls	Bluetooth	Apple Siri	MCU, FPU	Microphones, skin sensor, motion & speech detecting accelerometers, force sensor	Speaker

Appendix B

Smart Home Actuator Devices

Smart Home Actuator Devices						
Lighting						
Smart Bulb						
Device Name	**Function**	**Connectivity**	**Compatibility**	**MCU/ CPU**	**Input mechanisms**	**Output mechanisms**
Philips Hue Smart Bulb	Control light state and brightness	Bluetooth, Zigbee	Google Assistant, Amazon Alexa, Apple HomeKit	MCU	App, Voice assistant	LED
Nanoleaf Essentials Bulb	Control light state, brightness, colour	Bluetooth, Thread	Google Assistant, Apple HomeKit	MCU	App, Voice assistant	LED
Hey! Smart Bulb	Control light state, brightness, colour	Wi-Fi	Google Assistant, Amazon Alexa	MCU	App, Voice assistant	LED
Smart Blinds & Curtains						
Device Name	**Function**	**Connectivity**	**Compatibility**	**MCU/ CPU**	**Input mechanisms**	**Output mechanisms**
Weonesee Smart Blinds Motor	Control blinds remotely, schedule blinds	Wi-Fi	Amazon Alexa, Google Assistant	MCU	Voice assistant, App, Remote	Raise and lower blinds
Climate Control						
Smart Humidifiers & Purifiers						
Device Name	**Function**	**Connectivity**	**Compatibility**	**MCU/ CPU**	**Input mechanisms**	**Output mechanisms**
VOCOlinc MistFlow Smart Humidifier	Maintain room humidity	Wi-Fi	Amazon Alexa, Apple Siri, Google Assistant	MCU	Voice assistant, app, touch	Light, Mist
VOCOlinc PureFlow Smart Air Purifier	Purifies room air	Wi-Fi	Amazon Alexa, Apple Siri, Google Assistant	MCU	Voice assistant, app, touch	LED display, LED light, purified air
Dyson Humidify+Cool Smart Air Purifier	Purifies, humidifies and controls airflow, senses air quality and humidity	Wi-Fi	Amazon Alexa, Apple Siri, Google Assistant	MCU	Voice assistant, app, remote and particulate matter, VOC, temperature, humidity sensors	Fan, purified and humidified air

Heating (Thermostat)						
Device Name	**Function**	**Connectivity**	**Compatibility**	**MCU/ CPU**	**Input mechanisms**	**Output mechanisms**
Google Nest Thermostat E	Control heating remotely	Wi-Fi, BLE	Google Assistant	CPU	Voice assistant, app, temperature, humidity, proximity, occupancy, ambient light sensors	LCD screen, temperature adjustment
Netatmo Smart Thermostat	Control heating remotely	Wi-Fi, Radio long-range	Apple HomeKit, Amazon Alexa, Google Assistant	CPU	Voice assistant, app, temperature sensor	E-paper display, temperature adjustment
Hive Mini Thermostat	Control heating remotely, alerts if left on	Zigbee	Amazon Alexa, Google Assistant, Apple HomeKit	CPU	Voice assistant, app, touch, temperature sensor	Screen display, temperature adjustment, app alerts
Tado Smart Radiator Thermostat	Radiator valve control, set schedules, alerts	BLE, Radio communication (to bridge)	Amazon Alexa, Google Assistant, Apple Siri	MCU	Voice assistant, app, touch, mechanical dial, temperature and humidity sensors	LED screen, motor for valve adjustment
Air Conditioning						
Device Name	**Function**	**Connectivity**	**Compatibility**	**MCU/ CPU**	**Input mechanisms**	**Output mechanisms**
Tado Smart AC Control V3+	Remote control of air conditioning units	Wi-Fi, Infrared	Amazon Alexa, Google Assistant, Apple HomeKit	MCU	Voice assistant, app, touch control, temperature, humidity sensors	LED display, control AC unit via infrared
Entertainment						
TV Streaming						
Device Name	**Function**	**Connectivity**	**Compatibility**	**MCU/ CPU**	**Input mechanisms**	**Output mechanisms**
Google Chromecast	Stream media to the TV, control via Google Assistant and mobile devices	Wi-Fi, HDMI	Google Assistant	CPU	Voice assistant, Mobile device	Streams media to TV
Amazon Fire TV Cube	Stream media to the TV, control via remote or Alexa	Wi-Fi, Bluetooth, HDMI, Ethernet	Amazon Alexa	CPU	Voice assistant, Microphone, Remote	Speaker, Streams media to TV
Apple TV	Stream media to the TV, control via Siri, remote, mobile devices	Wi-Fi, Thread, HDMI, Ethernet, Bluetooth	Apple Siri	CPU	Voice assistant, Remote, Mobile device	Streams media to TV
Access						
Smart Lock						
Device Name	**Function**	**Connectivity**	**Compatibility**	**MCU/ CPU**	**Input mechanisms**	**Output mechanisms**
Yale Smart Door Lock	Control home access, enable keyless entry, track activity	Bluetooth (Wi-Fi bridge required for remote functionality)	Google Assistant, Amazon Alexa, Apple Siri	MCU	Voice assistants, App	Actuate door lock

Aqara Smart Lock	Control home access, enable keyless entry	Bluetooth, Thread, NFC	Google Assistant, Amazon Alexa, Apple Siri	MCU	Voice assistants, app, buttons, fingerprint, NFC card, door sensors	Actuate door lock, Speaker
Household Appliances						
Robot Vacuum						
Device Name	**Function**	**Connectivity**	**Compatibility**	**MCU/ CPU**	**Input mechanisms**	**Output mechanisms**
Samsung JetBot robot vacuum	Clean floors	Wi-Fi	Google Assistant, Amazon Alexa, Samsung Bixby	CPU	Voice assistant, app, LiDAR, cliff sensors	Brushes clean floor, suction airflow
iRobot Roomba i7	Cleans floors	Wi-Fi	Google Assistant, Amazon Alexa, Apple Siri	CPU	Voice assistant, app, buttons, floor tracking, cliff, dirt sensors	Brushes clean floor, suction airflow
Coffee Machine						
Device Name	**Function**	**Connectivity**	**Compatibility**	**MCU/ CPU**	**Input mechanisms**	**Output mechanisms**
Smarter Smart Coffee Maker (2nd Gen)	Control and schedule coffee maker	Wi-Fi	Google Assistant, Amazon Alexa, Apple Siri	MCU	Voice assistant, app, buttons	Makes coffee, LED display
Smart Plug						
Device Name	**Function**	**Connectivity**	**Compatibility**	**MCU/ CPU**	**Input mechanisms**	**Output mechanisms**
Philips Hue Smart Plug	Control plug state	Bluetooth	Amazon Alexa, Google Assistant	MCU	Voice assistant, App	Switch plug state
Amazon Smart Plug	Control plug state, schedule appliances	Wi-Fi	Amazon Alexa	MCU	Amazon Alexa, App	Switch plug state
Eve Energy Smart Plug	Control plug state, schedule appliances	Bluetooth, Thread	Apple Siri	MCU	Apple Siri, App	Switch plug state
Wemo Mini Smart Plug	Control plug state, schedule appliances	Wi-Fi	Amazon Alexa, Google Assistant, Apple Siri	MCU	Voice assistant, App	Switch plug state

Appendix C

Smart Home Sensing Devices

Smart Home Sensing Devices						
Security & Safety						
Door and Window Monitoring						
Device Name	**Function**	**Connectivity**	**Compatibility**	**MCU/ CPU**	**Input mechanisms**	**Output mechanisms**
Eve Door & Window	Detects opening & closing of windows & doors	BLE, Thread	Apple HomeKit	MCU	Contact sensor	Phone notifications
Ring Alarm Contact Sensor	Detects opening & closing of windows & doors	Z-Wave	Amazon Alexa	MCU	Contact sensor	Alexa device & phone notifications
Smoke Alarm						
Device Name	**Function**	**Connectivity**	**Compatibility**	**MCU/ CPU**	**Input mechanisms**	**Output mechanisms**
Google Nest Protect (2nd Gen)	Detect smoke & carbon monoxide	Wi-Fi, Bluetooth	Google Home	CPU	Ambient light, humidity, temperature, split spectrum smoke sensors, microphone, accelerometer	Speaker, LED, phone notifications
Netatmo Smart Smoke Alarm	Detect smoke	Wi-Fi, BLE	Apple HomeKit	MCU	Optical smoke sensor	Speaker, phone notifications
Leak Detector						
Device Name	**Function**	**Connectivity**	**Compatibility**	**MCU/ CPU**	**Input mechanisms**	**Output mechanisms**
Eve Water Guard	Detects leaks	Bluetooth, Thread	Apple HomeKit	MCU	Water leak sensor	Speaker, LED, phone notifications
D-Link Water Leak Sensor	Detects leaks	Wi-Fi	Google Home	MCU	Water leak sensor	Speaker, LED status, phone notifications
Surveillance						
Smart Doorbell						

Device Name	Function	Connectivity	Compatibility	MCU/ CPU	Input mechanisms	Output mechanisms
Google Nest Doorbell	Detect events, monitor home, two-way talk	Wi-Fi, Bluetooth	Google Assistant, Amazon Alexa	CPU	Camera, microphone, button, proximity, motion sensors, magnetometer	Speaker, LED, home device and phone notifications
Ring Video Doorbell	Detect motion, monitor home, two-way talk	Wi-Fi	Amazon Alexa	CPU	Camera, microphone, motion sensor, button	Speaker, LED, home device and phone notifications
Netatmo Smart Video Doorbell	Monitor home, video call	Wi-Fi, Bluetooth	Google Assistant, Amazon Alexa, Apple Siri	CPU	Camera, microphone, button	Speaker, LED, phone notifications
Indoor Camera						
Device Name	**Function**	**Connectivity**	**Compatibility**	**MCU/ CPU**	**Input mechanisms**	**Output mechanisms**
Google Nest Cam	Monitor home, detect people and animals	Wi-Fi, BLE	Google Home	CPU	Camera, microphone, motion sensor	Speaker, phone notifications
Ring Indoor Cam	Monitor home, two-way talk	Wi-Fi	Amazon Alexa	CPU	Camera, microphone, motion sensor	Speaker, phone notifications
Netatmo Smart Indoor Cam	Monitor home, detect intrusions	Wi-Fi	Google Home, Amazon Alexa, Apple HomeKit	CPU	Camera	Home device and phone notifications
Environment Monitoring						
Indoor Air Quality Monitor						
Device Name	**Function**	**Connectivity**	**Compatibility**	**MCU/ CPU**	**Input mechanisms**	**Output mechanisms**
Amazon Air Quality Monitor	Monitor and track air quality	Wi-Fi, BLE	Amazon Alexa	MCU	Temperature, CO, humidity, VOCs, particulate matter sensors	Home device and app notifications and monitoring
Eve Room Indoor Air Quality Monitor	Monitor and track air quality	Bluetooth, Thread	Apple HomeKit	MCU	Temperature, humidity, VOCs sensors	E-ink display, home device and app monitoring
Outdoor Weather Monitor						
Device Name	**Function**	**Connectivity**	**Compatibility**	**MCU/ CPU**	**Input mechanisms**	**Output mechanisms**
Eve Weather	Provides real-time weather data and forecast	Bluetooth, Thread	Apple HomeKit	MCU	Temperature, humidity, barometric pressure sensors	E-ink display, home device and app monitoring
Netatmo Weather Station	Provides real-time weather data and forecast	Wi-Fi	Apple HomeKit, Amazon Alexa	MCU	Temperature, humidity, barometric pressure, sound, CO2 sensors, + rain gauge, anemometer	Home device and app monitoring, alerts

Appendix D

Home Configurations

```
Home 2
{
 "bedroom_light": { "state": "on", "brightness": "60", "colour": "warm_white" },
 "bathroom_light": { "state": "on", "brightness": "60" },
 "living_room_lamp": { "state": "off", "brightness": "55", "colour": "green" },
 "living_room_light": { "state": "off" },
 "kitchen_light": { "state": "off", "brightness": "41" },
 "hallway_light": { "state": "off" },
 "air_con": { "state": "off", "temperature": "30" },
 "heating": { "state": "off", "temperature": "17" },
 "humidifier": { "state": "on", "humidity": "64" },
 "purifier": { "state": "off", "air_quality": "262" },
 "tv": { "state": "off", "volume": "7", "channel": "17", "service": "youtube" },
 "smart_speaker": { "state": "off", "volume": "5", "channel": "6", "service": "spotify",
    ↪ "content": "jazz" }
}
```

```
Home 3
{
 "bedroom_light": { "state": "on", "brightness": "60", "colour": "warm_white" },
 "bathroom_light": { "state": "on", "brightness": "60" },
 "living_room_lamp": { "state": "off", "brightness": "55", "colour": "green" },
 "living_room_light": { "state": "off" },
 "kitchen_light": { "state": "off", "brightness": "41" },
 "hallway_light": { "state": "off" },
 "air_con": { "state": "off", "temperature": "30" },
 "heating": { "state": "off", "temperature": "17" },
 "humidifier": { "state": "on", "humidity": "64" },
 "purifier": { "state": "off", "air_quality": "262" },
 "tv": { "state": "off", "volume": "7", "channel": "17", "service": "youtube" },
 "smart_speaker": {"state": "off", "volume": "5", "channel": "6", "service": "spotify", "
    ↪ content": "jazz" },
 "security_cam": { "state": "on" },
 "front_door_lock": { "state": "locked" },
 "garage_door_lock": { "state": "unlocked" },
 "washing_machine": { "state": "on" },
 "dishwasher": { "state": "on", "mode": "pause" },
 "coffee_maker": { "state": "off", "brew": "latte", "quantity": "1" },
 "alarm": { "state": "off", "time": "9:00", "day": "saturday" },
```

```
  "living_room_window": { "state": "open" },
  "bedroom_window": { "state": "closed" },
  "bedroom_blind": { "state": "open" },
  "living_room_blind": { "state": "closed" },
  "robot_vacuum": { "state": "on" }
}
```

Appendix E

5-Shot Prompt

```
{ "role": "user", "content": "Home Model:\n{\n  \"kitchen_light\": {\n    \"state\": \"
    ↪ off\",\n    \"brightness\": \"41\"\n  },\n  \"security_cam\": {\n    \"state\": \
    ↪ "on\"\n  }\n}\n\nCommand:\nCould you bump the kitchen light brightness to 70?"
},
{ "role": "assistant", "content": "The kitchen light is now on at 70 brightness.\n‘‘‘json
    ↪ \n\"kitchen_light\": { \"state\": \"on\", \"brightness\": \"70\" }\n‘‘‘"
},
{ "role": "user", "content": "Home Model:\n{\n  \"robot_vacuum\": {\n    \"state\": \"on\
    ↪ "\n  },\n  \"living_room_window\": {\n    \"state\": \"open\"\n  }\n}\n\nCommand
    ↪ :\nPlease brew me green tea."
},
{ "role": "assistant", "content": "I'm sorry, there's no tea maker in this home model. No
    ↪  changes made.\n‘‘‘json\n{}\n‘‘‘"
},
{ "role": "user", "content": "Home Model:\n{\n  \"tv\": {\n    \"state\": \"off\",\n    \
    ↪ "volume\": \"10\",\n    \"channel\": \"5\",\n    \"service\": \"netflix\"\n  },\n
    ↪   \"humidifier\": {\n    \"state\": \"on\",\n    \"humidity\": \"55\"\n  }\n}\n\
    ↪ nCommand:\nPut on channel 42."
},
{ "role": "assistant", "content": "TV is on and tuned to channel 42.\n‘‘‘json\n\"tv\": {
    ↪ \"state\": \"on\", \"channel\": \"42\" }\n‘‘‘"
},
{ "role": "user", "content": "Home Model:\n{\n  \"air_con\": {\n    \"state\": \"on\",\n
    ↪    \"temperature\": \"23\"\n  }\n}\n\nCommand:\nClose the window."
},
{ "role": "assistant", "content": "I'm sorry, there's no windows listed in this home
    ↪ model. No changes made.\n‘‘‘json\n{}\n‘‘‘"
},
{ "role": "user", "content": "Home Model:\n{\n  \"front_door_lock\": {\n    \"state\": \"
    ↪ unlocked\"\n  },\n  \"living_room_light\": {\n    \"state\": \"off\"\n  }\n}\n\
    ↪ nCommand:\nLock up the front, please."
},
{ "role": "assistant", "content": "Front door locked.\n‘‘‘json\n\"front_door_lock\": { \"
    ↪ state\": \"locked\" }\n‘‘‘"
}
```

Appendix F

Gatekeeper Definition


```
{
  "all_rooms": {
    "bedroom_light": {
      "state": "on", "options": ["on","off"],
      "brightness": { "state": 60, "range": [0,100], "unit": "percent", "step": 1 },
      "colour": { "state": "warm_white", "options": ["warm_white","cool_white","red","
    ↪ green","blue","yellow","purple","orange","gray","wheat","silver","pea_green","
    ↪ navy_blue","soft_white","grey","white"] }
    },
    "bathroom_light": {
      "state": "on", "options": ["on","off"],
      "brightness": { "state": 60, "range": [0,100], "unit": "percent", "step": 1 }
    },
    "living_room_lamp": {
      "state": "off", "options": ["on","off"],
      "brightness": { "state": 55, "range": [0,100], "unit": "percent", "step": 1 },
      "colour": { "state": "green", "options": ["warm_white","cool_white","red","green","
    ↪ blue","yellow","purple","orange","gray","wheat","silver","pea_green","navy_blue",
    ↪ "soft_white","grey","white"] }
    },
    "living_room_light": { "state": "off", "options": ["on","off"] },
    "kitchen_light": {
      "state": "off", "options": ["on","off"],
      "brightness": { "state": 41, "range": [0,100], "unit": "percent", "step": 1 }
    },
    "hallway_light": { "state": "off", "options": ["on","off"] },
    "air_con": {
      "state": "on", "options": ["on","off"],
      "temperature": { "state": 30, "range": [14,35], "unit": "celsius", "step": 0.5 }
    },
    "heating": {
      "state": "on", "options": ["on","off"],
      "temperature": { "state": 17, "range": [14,35], "unit": "celsius", "step": 0.5 }
    },
    "humidifier": {
      "state": "on", "options": ["on","off"],
      "humidity": { "state": 64, "range": [25,80], "unit": "percent", "step": 1 }
    },
    "purifier": {
```

```
      "state": "off", "options": ["on","off"],
      "air_quality": { "state": 262, "range": [0,500], "unit": "aqi", "step": 1 }
    },
    "tv": {
      "state": "on", "options": ["on","off"],
      "volume": { "state": 7, "range": [0,100], "unit": "percent", "step": 1 },
      "channel": { "state": 17, "range": [1,50], "step": 1 },
      "service": { "state": "youtube", "options": ["youtube","netflix","amazon_prime","
    ↪ disney_plus","spotify","radio","cnn","disney","disney+","pandora"] },
      "content": { "state": "titanic" },
      "brightness": { "state": 55, "range": [0,100], "unit": "percent", "step": 1 }
    },
    "smart_speaker": {
      "state": "off", "options": ["on","off"],
      "volume": { "state": 5, "range": [0,100], "unit": "percent", "step": 1 },
      "channel": { "state": 6, "range": [1,50], "step": 1 },
      "service": { "state": "spotify", "options": ["youtube","spotify","radio","pandora"]
    ↪  },
      "content": { "state": "jazz" }
    },
    "security_cam": { "state": "on", "options": ["on","off"] },
    "front_door_lock": { "state": "locked", "options": ["locked","unlocked"] },
    "garage_door_lock": { "state": "unlocked", "options": ["locked","unlocked"] },
    "washing_machine": {
      "state": "on", "options": ["on","off"],
      "mode": { "state": "pause", "options": ["pause","resume"] }
    },
    "dishwasher": {
      "state": "on", "options": ["on","off"],
      "mode": { "state": "pause", "options": ["resume","pause"] }
    },
    "coffee_maker": {
      "state": "off", "options": ["on","off"],
      "brew": { "state": "latte", "options": ["espresso","americano","latte","cappuccino"
    ↪ ,"macchiato","mocha","hot_chocolate","flat_white"] },
      "quantity": { "state": 1, "range": [1,10], "unit": "cups", "step": 1 }
    },
    "alarm_clock": {
      "state": "off", "options": ["on","off"],
      "time": { "state": "9:00" },
      "day": { "state": "saturday", "options": ["monday","tuesday","wednesday","thursday"
    ↪ ,"friday","saturday","sunday"] }
    },
    "living_room_window": { "state": "open", "options": ["open","closed"] },
    "bedroom_window": { "state": "closed", "options": ["open","closed"] },
    "bedroom_blind": { "state": "open", "options": ["open","closed"] },
    "living_room_blind": { "state": "closed", "options": ["open","closed"] },
    "robot_vacuum": { "state": "on", "options": ["on","off"] }
  }
}
```