

Advancing the Development and Utilization of Data Infrastructure for Smart Homes

Sheik Murad Hassan Anik

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Application

Na Meng, Co-chair

Xinghua Gao, Co-chair

Eli Tilevich

Bo Ji

Haibo Zeng

Soowon Chang

August 16, 2024

Blacksburg, Virginia

Keywords: Data Infrastructure, Machine Learning Application, Building Engineering, IoT,
Smart Home System, Automation Configuration.

Copyright 2024, Sheik Murad Hassan Anik

Advancing the Development and Utilization of Data Infrastructure for Smart Homes

Sheik Murad Hassan Anik

(ABSTRACT)

The smart home era is inevitably arising towards our everyday life. However, the scarcity of publicly available data remains a major hurdle in the domain, limiting people's capability of performing data analysis and their effectiveness in creating smart home automations. To mitigate this hurdle and its influence, our research explored three research directions to (1) create a better infrastructure that effectively collects and visualizes indoor-environment sensing data, (2) create a machine learning-based approach to demonstrate a novel way of analyzing indoor-environment data to facilitate human-centered building design, and (3) conduct an empirical study to explore the challenges and opportunities in existing smart home development.

Specifically, we conducted three research projects. First, we created an open-source IoT-based cost-effective, distributed, scalable, and portable indoor environmental data collection system, Building Data Lite (BDL). We deployed this research prototype in 12 households, which deployment so far has collected more than 2 million records that are available to public in general. Second, building occupant persona is a very important component in human-centered smart home design, so we investigated an approach of applying state-of-the-art machine-learning models to data collected by an existing infrastructure, to enable the automatic creation of building occupant persona while minimizing human effort. Third, Home

Assistant (HA) is an open-source off-the-shelf smart home platform that users frequently use to transform their residences into smart homes. However, many users seem to be stuck with the configuration scripts of home automations. We conducted an empirical study by (1) crawling posts on HA forum, (2) manually analyzing those posts to understand users' common technical concerns as well as frequently recommended resolutions, and (3) applying existing tools to assess the tool usefulness in alleviating users' pain. All our research projects will shed light on future directions in smart home design and development.

Advancing the Development and Utilization of Data Infrastructure for Smart Homes

Sheik Murad Hassan Anik

(GENERAL AUDIENCE ABSTRACT)

My research aims to address the gaps in the smart home systems domain in terms of data availability, utilization, and, development issues. In this dissertation, I developed an IoT-based wireless sensor network to mitigate the lack of publicly available actual building data. I used machine learning tools for developing building occupant persona with real-world data which is a necessary element in human-centered smart home design. I conducted an empirical study to understand the automation configuration issues in smart home systems and presented a root-cause taxonomy of the issues investigated. The combined findings of this research can help the smart home development community and open new doors in research directions.

Dedication

To my loving family and dearest wife, Anupama.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my co-advisors, Dr. Na Meng and Dr. Xinghua Gao, for their unwavering support and mentorship throughout my Ph.D. journey. I will always cherish the memory of our first meeting in Dr. Meng's office, where we discussed my initial research project. Dr. Na guided me through the intricate world of software engineering research, while Dr. Gao provided invaluable insights into Building Engineering research. Their combined expertise and guidance were instrumental in the successful publications of our research in esteemed journals. I consider myself extremely fortunate to have had such exceptional co-advisors.

I am profoundly thankful to my advisory committee members, Dr. Eli Tilevich, Dr. Bo Ji, Dr. Haibo Zeng, and Dr. Soowon Chang, for their insightful guidance and constructive feedback, which have greatly enhanced my work. Their mentorship has not only shaped my research but will also leave a lasting impact on my future career. I am also sincerely grateful to my current and former lab mates for their invaluable knowledge-sharing, collaborative spirit, and thoughtful feedback on my projects, papers, and presentations.

My sincere thanks go to the Computer Science department for supporting me as a Graduate Teaching Assistant throughout numerous semesters at Virginia Tech. In addition, I would like to thank Sharon Kinder-Potter, the graduate coordinator for the computer science department, who was my first point of contact for every administrative need.

I would like to express my heartfelt thanks to my friends back home. I cherish every moment I had the opportunity to spend with them. Their encouragement and friendship inspired me to keep going in the most challenging times. My gratitude goes out to all my friends at Virginia Tech. When I first arrived in the United States, I knew no one, but it wasn't

long before I found myself surrounded by people who made Blacksburg feel like home. These friends became my second family, and I am incredibly fortunate to have all of them.

I am grateful to my parents, my brother Arnob, my sister-in-law Sara, my niece Zara, and my cousin Shathi for their unconditional love and unwavering support. I am also deeply thankful to my mother-in-law, whose complete support and love have been a constant source of inspiration for me.

Last but certainly not least, I am forever indebted to my beloved wife, Anupama, for her friendship, encouragement, care, and love. I appreciate the sacrifices she had to make, the struggles she had to overcome, and the challenges she had to tackle throughout these years. Without her steadfast support, reaching this stage in life would have been impossible.

Contents

List of Figures	xii
List of Tables	xiv
1 Introduction	1
1.1 Dissertation Goals	2
1.2 Dissertation Organization	3
2 Related Works	4
2.1 Building Data Sensing Systems	4
2.2 Persona Development and Machine Learning	7
2.2.1 Building Occupant Persona Development	7
2.2.2 Machine Learning tools for modeling Occupant Behavior	9
2.3 Smart Home Development and Issues	11
2.3.1 Empirical Studies on IoT Systems	11
2.3.2 Automatic Bug Detection in IoT Systems	13
3 A Cost-Effective, Scalable, and Portable IoT Data Infrastructure for Indoor Environment Sensing	14
3.1 Motivation	14

3.2	The System Design of BDL	18
3.2.1	System Architecture	19
3.2.2	Central Server	21
3.2.3	Sensing Node	22
3.2.4	Database Configuration and Data Transmission	24
3.2.5	Network Configuration	26
3.2.6	Data Integration	26
3.2.7	Graphical User Interface	28
3.3	Prototype Deployment	30
3.4	A Case Study	34
3.5	Discussion	38
3.5.1	Challenges and Solutions	38
3.5.2	Potential Use Cases	40
3.5.3	Limitations and Future Research	40
3.6	Summary	41
4	Machine learning approach in the development of building occupant per- sonas	43
4.1	Motivation	43
4.2	Methodology	46

4.2.1	Challenges of using machine learning tools in building occupant persona development process	47
4.2.2	Methodology Overview	48
4.2.3	Dataset description	49
4.2.4	Pre-processing	51
4.2.5	Feature selection	55
4.2.6	Target variables	58
4.2.7	Machine learning models	61
4.3	Evaluation	66
4.3.1	Experiment settings	67
4.3.2	Evaluation metrics	67
4.3.3	Model Performances	69
4.3.4	Answer to the research questions	72
4.4	Discussion	76
4.4.1	Sample Occupant Persona Prediction	77
4.4.2	Limitations and future research	79
4.5	Summary	80
5	An empirical study of automation configuration bugs in Smart Homes	82
5.1	Motivation	82
5.2	Background	86

5.3	Methodology	88
5.3.1	Data Collection	89
5.3.2	Data Analysis	92
5.4	Experiment Results	93
5.4.1	RQ1: Challenges in Handling Automation Configuration	93
5.4.2	RQ2: Frequent Resolution Strategies	103
5.4.3	RQ3: Current Debugging Tool Support	108
5.5	Threats to Validity	112
5.6	Lessons Learned	113
5.7	Summary	115
6	Conclusion	116
6.1	Summary	117
6.2	Future Works	118
6.3	Impact	121
	Bibliography	122

List of Figures

3.1	The BDL system architecture.	18
3.2	An example of BDL deployment.	19
3.3	Entity relationship diagram of the central database.	20
3.4	Data flow in the BDL system.	21
3.5	Sensing node prototype version 1.	22
3.6	Local database design.	24
3.7	Sample data format	27
3.8	BDL system’s Graphical User Interface	29
3.9	Sensing node prototype version 2 (left) and version 3 (right).	31
3.10	Structure of the prototype’s sensing node.	32
3.11	Sensors used in sensing node prototype.	32
3.12	Image a deployed sensing node.	35
3.13	A photo of a house in the case study.	36
3.14	Visualization of collected data from all sensors of a deployed sensing node.	37
3.15	Visualization of collected data of a single sensor.	38
4.1	Our research methodology involving data curation, ML model training, and ML-based prediction for persona development	49

4.2	Feature distribution in categories.	52
4.3	Histogram of occupant age data before (left) and after (right) pre-processing step.	53
4.4	Histogram of TEMPGONE before (left) and after (right) pre-processing step.	54
5.1	An exemplar YAML file for home automation [32]	83
5.2	The consumer pattern in Home Assistant	86
5.3	The taxonomy of 190 threads, where * means some issues belong to more than one category	91

List of Tables

4.1	Target attribute details.	57
4.2	Accuracy (A), F1-score (F), Precision (P), and, Recall (R) obtained across the target variables	69
4.3	Average accuracy of the machine learning models over 10-fold validation process	70
4.4	Average precision (P), recall (R), and, f1-scores (F) of classification models .	72
4.5	Sample occupant characteristics prediction	77
5.1	The representative resolution strategies (i.e., each strategy occurs at least four times in our dataset)	103
5.2	The bug detection capabilities of current tools	110
5.3	The confusing tool-generated reports for Listing 6	111

List of Abbreviations

ADB AdaBoost

BDL Building Data Lite

DIY Do it yourself

DT Decision Tree Classifier

EIA Energy Information Administration

HBI Human-Building Interaction

IoT Internet of Things

KNN K Nearest Neighbor

LDA Linear Discriminant Analysis

RECS Residential Energy Consumption Survey

RFC Random Forest Classifier

SVM Support Vector Machine

USD United States Dollar

Chapter 1

Introduction

The terms ‘Internet of Things (IoT)’ and ‘Smart Home’ have been brought up more frequently than ever in recent times. The Internet of Things (IoT) describes the network of physical objects embedded with sensors, software, and, other technologies for the purpose of connecting and exchanging data with other devices and systems over the Internet. A smart home is a home that incorporates advanced automation systems to provide the inhabitants with sophisticated monitoring and control over the building’s functions [106]. In recent years, the buzz about this new type of domestic technology have been gaining a lot of popularity. Devices like smart mirrors, robot vacuum cleaners, and wireless kitchen appliances combined with a smart home system are bound to transform the indoor living experience. In 2022, IoT smart home device sales grossed \$115 billion US dollars, and it is forecasted to reach a value of more than \$195 billion US dollars by 2026 [110].

Smart home systems have become inevitable in recent times, and as this field is still in its emerging phase, it still requires a lot of research to bring perfection to the system and provide the most convenience to its users. The first hurdle researchers face in the domain of smart home research is the lack of publicly available data in the heterogeneous IoT platforms. The vast number of facility management systems, home automation systems, and the ever-increasing number of Internet of Things (IoT) devices are in constant need of environmental monitoring. Indoor environment data can be utilized to improve indoor facilities and better occupants’ working and living experience, however, such data are scarce because many

existing facility monitoring technologies are expensive and proprietary for certain building systems, such as building automation systems, energy management systems, and maintenance systems.

In the domain of smart home research, the next step after the data collection is analyzing the data and identifying useful insights from it. The data obtained on different home automation systems can tell us different stories and provide important insights on the environment as well as its occupants. Such information can be utilized for the improvement of the habitat's living conditions. A user persona is a communication tool for designers to generate a mental model that describes the archetype of users. Developing building occupant personas is proven to be an effective method for human-centered smart building design, which considers occupant comfort, behavior, and energy consumption. Optimization of building energy consumption also requires a deep understanding of occupants' preferences and behaviors.

Each smart home system consists of multiple smart devices, sometimes scaling up to hundreds or thousands in number. These devices are controlled by the automation configurations defined by developers. Writing the automation configuration for smart devices in smart home systems is a challenging task. An issue in automation configuration can result in variable consequences, for example, improper behavior of the device or errors in the whole system. A study on automation configuration issues in the smart home domain is indeed necessary due to the increasing enthusiasm of do-it-yourself users trying to develop their own smart home systems.

1.1 Dissertation Goals

With the increasing number of smart home users in recent days, its different areas are in need of investigation to ensure proper growth. To mitigate issues related to the development

and usage of smart homes, this dissertation focuses on three aspects, (i) developing a cost-effective, distributed, scalable, and portable indoor environmental data collection system, (ii) using machine learning tools for semi-automated building occupant persona generation, and, (iii) conducting an empirical study on automation configuration bugs smart home systems.

1.2 Dissertation Organization

The following is the outline of the organization of this dissertation:

- Chapter 2 discusses the prior research works related to IoT data infrastructure development, building occupant characterization with persona development, and, development issues of smart home systems.
- Chapter 3 provides an overview of the work done on IoT data infrastructure - Building Data Lite (BDL), a cost-effective, distributed, scalable, and portable indoor environmental data collection system.
- Chapter 4 narrates the work done on the usage of machine learning tools on residential energy consumption data in a semi-automated approach to generate building occupant personas.
- Chapter 5 describes the completed work on the empirical study of automation configuration issues in smart home systems.
- Chapter 6 concludes the dissertation with the view of future research direction.

Chapter 2

Related Works

This chapter elaborates prior works relevant to development and usage of data collection infrastructure for smart home development. Section 2.1 discusses the data collection systems proposed in previous researchers. Section 2.2 describes the related works conducted in building occupant characterization and persona development. Section 2.3 explores the smart home development issue investigations conducted previously.

2.1 Building Data Sensing Systems

The first step to make a building “smart” is to establish the data infrastructure. Data collection has always been a challenge in the smart building research domain [82]. Wireless sensor network technologies have been implemented as solutions to this challenge [77, 128]. However, there is still a lack of cost-effective means to generate, collect, and process the ubiquitous indoor environment data in existing buildings. In this section, related works in the domain of indoor environmental sensing system are discussed. The innovation of the proposed BDL system lies in comprehensively achieving cost-effectiveness, portability, scalability, and generalizability. Therefore, the related works are discussed regarding these properties.

Ferdoush *et al.* [61] presented a wireless sensor network system using open-source hardware platforms, Arduino, Raspberry Pi, and XBee module. The system is low-cost and scalable

both in terms of the type of sensors and the number of sensor nodes. It is suited for a wide variety of applications related to environmental monitoring. Each sensing node centers around an Arduino board, which incorporates multiple sensors. The designed prototype collected temperature and humidity data. The Arduino board incorporates an XBee module to communicate with the base station. The base station is based on a Raspberry Pi which is connected to the Internet via a router. The XBee module on the base station works as a coordinator device which has a limitation of supporting a maximum of 10 sensing nodes. The system is portable in the sense of placing the sensing nodes with the approximation of the base station.

To lower energy consumption, Alva *et al.* [40] used video-based human occupancy sensing to optimize the lighting strategy. This study involves the development of a human occupancy sensing system in MATLAB and the hardware for lighting controls. Video cameras in this system can be placed in a portable fashion but the system remains purpose-specific, and the cameras significantly increase the system's cost.

Kumar *et al.* [77] proposed a low-cost Smart Living System, which uses an Android-based user interface for control of home appliances. This model integrates temperature and humidity sensor, DHT11, sound sensor, LM393, and gas sensor, MQ135, with Raspberry Pi 3B microcontroller board. The system uses an external GPRS module to connect to the internet. The work presents an experiment on monitoring the air quality, sound, temperature, and humidity detection. The system uploads the collected data to a cloud server, which decides whether the pollution level has crossed a certain threshold.

Zakaria *et al.* [128] developed an air quality sensing system by utilizing MQ-135 gas sensor, coupled with the temperature and humidity sensor DHT-22. The system can detect gaseous components such as NH_3 , NO_2 , Alcohol, and Benzene, along with sensing temperature and humidity. Coleman *et al.* [55] developed a low-cost indoor air quality sensing system

capable of measuring volatile organic compounds (VOCs) and other gaseous concentrations while monitoring temperature and humidity. They identified the Indoor Air Quality (IAQ) matrix as composed of CO_2 , VOC , CO , PM , $HCHO$, and NO_2 . Sahal *et al.* [104] proposed a Raspberry Pi v3 based model to capture and monitor gas and sound properties from the environment. It uses an LM393 sound sensor, an MQ135 gas sensor, a DHT11 temperature and humidity sensor, and a GPRS module. The objective of their research is to monitor the air quality and measure noise intensity to mitigate sound pollution. Marques *et al.* [88] provided an IoT-based real-time indoor air quality monitoring system named iAir. It features an ESP8266 with ATmega168PA MCU as a communication and processing unit. The system utilizes MICS6814 sensor to detect gases such as Carbon Monoxide, Nitrogen Dioxide, Ethanol, Methane, and, Propane. It uses WiFi connectivity and smartphone application to provide data access and real-time notification.

The systems proposed in [55, 88, 104, 128] are focused on the specific task of air quality monitoring with a set of fixed sensors, lacking the feature of scalability. The system in [40] is also task-specific but significantly more expensive than the other works mentioned here. The systems proposed in [61, 77] offer the property of cost-effectiveness, portability, and scalability to some extent.

In contrast to the above-mentioned works, the BDL system proposed in chapter 3 aims in attaining cost effectiveness, code-free scalability, and, portability, while keeping the usability of the system more generic and task-independent. The use of general-purpose input and output ports of Raspberry Pi enables the system to be compatible with a wide range of sensors and makes the system scalable. The use of these sensors with the Raspberry Pi keeps the overall cost of the sensing nodes inexpensive. The wireless communication of the sensing nodes makes the system attain the goal of portability. BDL's architecture is similar to the one presented by Aheleroff *et al.* [37], which describes the system of mirroring

digital representatives of physical assets with two-way dynamic mapping. BDL offers a code-free sensing system. The general concept of a code-free sensing network system can help research work across researchers of different backgrounds because data regarding building environments can be useful in multiple academic areas, in which not all researchers have a background in programming.

2.2 Persona Development and Machine Learning

Related previous studies in occupant characterization and persona development can be grouped into the following two categories: building occupant persona development and machine learning tools for modeling occupant behavior.

2.2.1 Building Occupant Persona Development

Human-Building Interaction (HBI) usually reflects some characteristics of the building occupants, and the building occupant persona is an effective and concise way to represent these characteristics for human-centered building design and operation. Agee *et al.* [35] conducted a study on a human-centered approach to smart housing. The study employed a multi-phase, mixed-methods research design and collected data from 309 residential housing units in Virginia, U.S. The authors collected longitudinal energy use data via the deployed electricity sensors and the occupants' attitudes, preferences, and self-perceived energy behaviors via surveys and semi-structured interviews. Affinity diagramming was then used to identify the topics and themes of occupants' HBI activities. The output of the affinity diagramming analysis and energy analysis led to the development of data-driven Personas that communicate smart housing user needs. Malik *et al.* [87] researched the occupant behavior

of 1223 low-income households in India through a transverse field study. Three occupant archetypes, indifferent consumer, considerate saver, and conscious conventional, were established based on the occupants' behavioral and psychographic characteristics gathered from field surveys. Occupants' age and education were determined as significant factors influencing occupant archetype clustering and their actions within the indoor built environment. The segmentation results further inform that there exists considerable diversity in occupant behavior and attitudes within similar socio-economic groups, which consequently leads to differential energy demand and occupant comfort preferences. The findings can assist the building simulation community in accurately estimating energy demand through realistic occupant profiles and providing pragmatic occupant-centric building designs for the future low-income housing stock.

Studying occupant behavior related to their comfort needs is critical for better understanding home energy consumption. Ortiz *et al.* [96] studied the motivations behind comfort behaviors and the energy consumption discrepancies among occupants with different behavioral patterns. The authors grouped the occupants into five categories based on their psychological and behavioral models — locus of control, emotions towards their home environment, and the importance they give to comfort affordances. The findings show that each of the archetypes has a distinct valence of opinions when asked about topics regarding energy use, energy awareness, general comfort, and affordances, but what they express verbally is not always congruent to the general results of their self-reported answers. Buttitta *et al.* [51] proposed a method to develop occupancy-integrated archetypes that allow the annual final heating energy required (by building stock) characterized by different occupancy profiles, which play a significant role in influencing heat demand in residential buildings. The authors showed that, for UK residential archetypes, the discrepancy between the heat demand calculated using the proposed occupancy-integrated archetypes and the BRE Domestic En-

ergy Model (BREDEM) calculation procedures can be up to 30%, pointing out that the use of BREDEM occupancy profiles is not necessarily appropriate, particularly when disaggregated and differentiated energy profiles are involved.

2.2.2 Machine Learning tools for modeling Occupant Behavior

Occupant behavior remains one of the main sources of uncertainty in building energy modeling [57]. Amasyali *et al.* [41] proposed a data-driven method of modeling occupant behavior for simultaneously reducing energy consumption and improving comfort. The proposed method consists of two components: 1) a set of machine learning-based occupant-behavior-sensitive models for predicting energy consumption and thermal and visual comfort, and 2) a genetic algorithm-based optimization model for optimizing occupant behavior. To test and evaluate the proposed method, an office building was instrumented and data about energy consumption, outdoor weather conditions, occupant behavior, and occupant comfort were collected for about three months. Based on the collected data, the authors developed a set of machine learning models for predicting energy consumption of cooling and lighting and occupant comfort of thermal and visual. Carlucci *et al.* [52] presented a similar study with a machine-learning approach for predicting building energy consumption in an occupant-behavior-sensitive manner. In their approach, a model learns from a large set of energy-use cases that were modeled and simulated in EnergyPlus [56]. The machine-learning prediction model was trained using a large dataset including 3-month hourly data for 5760 use cases representing different combinations of building characteristics, outdoor weather conditions, and occupant behaviors. The results of the study demonstrated the high impact of occupant behavior on building energy consumption and identified opportunities for behavioral energy-saving measures.

To create comfortable indoor environments for building occupants, Deng *et al.* [58] developed artificial neural network (ANN) models for predicting indoor thermal comfort by using thermal sensations and occupants' behavior. The models were trained by data about air temperature, relative humidity, clothing insulation, metabolic rate, thermal sensations, and occupants' behavior collected in ten offices and ten houses/apartments. The models were able to predict similar acceptable air temperature ranges in offices. The occupants' behavior in multi-occupant offices was more complex, which led to a different prediction of thermal comfort. This investigation demonstrates alternative approaches to the prediction of thermal comfort.

Peng *et al.* [100] conducted an analysis of occupants' behavior in an office building with the goal of increasing the efficiency of HVAC systems. They proposed a demand-driven control strategy that automatically responds to occupants' energy-related behavior to reduce energy consumption and maintain room temperature for occupants with similar performances. In this control strategy, two types of machine learning methods, unsupervised and supervised learning, were applied to learn occupants' behavior in two learning processes. The occupancy-related information learned by the algorithms was used by a set of specified rules to infer real-time room set points for controlling the office's space cooling system. This learning-based approach intends to reduce the need for human intervention in the cooling system's control. The proposed strategy was applied to control the cooling system of the office building under real-world conditions.

Automating the development of a human-centered building occupant persona requires vast data on both energy consumption and occupant behavior. The gap in either one not only hinders the comprehensiveness of the building occupant persona development process but also leads to inaccuracy in results. Although occupant behavior is difficult to model due to the stochastic nature and variability of humans, it is necessary to explore the generic

pattern of their behaviors and integrate the information with the building energy model [71]. Modeling occupant behavior remains one of the key steps in the process of building occupant persona development. A valid occupant behavior model needs to have the potential to simulate realistic building users' reactions to different built environments, and the lack of real occupant behavioral data accounts for a gap in research. The human-centered building occupant persona development process proposed in [35] requires manual human work of labeling, filtering, and clustering data from large datasets. These steps are both labor and time extensive, which impede the entire process. To address these research gaps, our work described in chapter 5 utilizes real-world occupant data with machine learning tools for occupant characterization to accelerate the process of building occupant persona development.

2.3 Smart Home Development and Issues

The related work on Smart Home Development includes empirical studies on Internet-of-Thing (IoT) systems, and bug detection in those systems.

2.3.1 Empirical Studies on IoT Systems

Studies were recently performed to characterize issues or problems in IoT systems [39, 48, 62, 66, 86, 122, 131]. For instance, Fernandes *et al.* [62], Alrawi *et al.* [39], and Zhou *et al.* [131] analyzed the security properties of IoT platforms and systems. In contrast, our research focuses on coding issues in IoT automation configuration, covering three categories: (1) implementing new features, (2) debugging, and (3) optimization. He *et al.* [66] did an online survey with 72 users of smart home systems, to learn their negative user experiences.

The participants reported fears of breaking the system by writing code, and struggles of diagnosing or recovering from system failures. Similarly, Makhshari and Mesbah [86] did interviews and surveys with IoT developers; they also found testing and debugging as the major challenges. However, neither study examines any widely used smart home platform to characterize the bug patterns or fixing strategies, let alone to provide concrete actionable advice to tool builders. Our study is motivated by and complements both studies.

Brackenbury *et al.* [48] focused on the trigger-action programming (TAP) model. They systematized the temporal paradigms through which TAP systems could express rules, and classified TAP programming bugs into three categories: control logic, timing, and inaccurate user expectation. As with Brackenbury *et al.*, we also identified bugs related to these three general categories (e.g., wrong loop structure and wrong data specification for matching). However, our taxonomy is finer-grained and more comprehensive, as we derived bug patterns from real-world bugs instead of speculation on the TAP model. Our observations reflect the real-world bug distribution among patterns; we also characterized (1) bugs violating syntactic rules (e.g., wrong formats) and more diverse semantic rules (e.g., data access), (2) developers' bug fixes, (3) recurring implementation requests, and (4) frequent optimization needs.

Wang *et al.* [122] inspected 330 device integration bugs mined from HAC to characterize any root cause, fix, trigger condition, and impact of those bugs. Our study is closely related to the work by Wang *et al.*, as we also examined data mined from HAC. However, our study is irrelevant to device integration; instead, it focuses on the coding issues in automation configuration.

2.3.2 Automatic Bug Detection in IoT Systems

Tools were created to automatically detect bugs in IoT systems [53, 65, 69, 79, 80, 120, 125]. Specifically, VulHunter [125] detects new vulnerabilities by analyzing the known vulnerability patch packs in Industry IoT. SOTERIA [53] verifies whether IoT apps adhere to the identified safety, security, and functional properties via model checking. Liang *et al.* [80] and Fu *et al.* [65] separately created tools to reveal bugs in IoT operating systems (OSes).

Some tools [69, 79, 120] detect conflicting interactions between smart home IoT applications, because these conflicts can result in undesired actions like locking a door during a fire. For instance, Trimananda *et al.* [120] studied 198 official and 69 third-party apps on Samsung SmartThings, and found 3 major categories of app conflicts. Based on their observations, the researchers created a conflict detector that uses model checking to detect up to 96% of the conflicts. Li *et al.* [79] categorized conflicts in a totally different way. They also invented a graph structure named *IA graph* to represent the controls in each IoT app and event schedules. With that representation, they created an efficient algorithm to leverage first-order logic and SMT solvers to detect conflicts.

Our study complements all tools mentioned above; it focuses on a different type of IoT bugs—bugs in automation configuration or implementation. Such implementation bugs are less relevant to security, IoT kernels, or inter-app conflicts. However, they are still important as these bugs can prevent end-users from realizing the desired automation rules or achieving high-quality automations.

Chapter 3

A Cost-Effective, Scalable, and Portable IoT Data Infrastructure for Indoor Environment Sensing

3.1 Motivation

Buildings are one of the most important aspects of human life, and, people spend 87% of their time inside buildings [94]. Buildings have both direct and indirect impacts on human health, comfort, and productivity [46, 94, 116, 117]. Maintaining optimal conditions for a better experience is critical for built environments. New techniques and methods are becoming available for this purpose with the developments of the Internet of Things (IoT) technologies [70, 90]. For example, an air quality monitoring system can benefit occupants in many ways from detecting harmful gases to monitoring oxygen levels. There are sensing systems, such as the ones presented in Zakaria *et al.* [128] and Marques *et al.* [88], that provide indoor air quality monitoring. However, these sensing systems are dedicated to air quality monitoring only without the capability of extension to other features such as temperature or humidity monitoring. Another important application of the indoor environment sensing system is to support the optimization of building energy consumption. Buildings consume about 40% of total consumed energy, and half of that goes for heating, ventilation, and air conditioning

(HVAC) systems [60]. A significant portion of this consumed energy is wasted [89], and, using a proper monitoring and control system can reduce this waste significantly [60]. Even though many building systems already incorporate proprietary networks of sophisticated sensors and devices, normally there is very limited inter-system connectivity or exposure to the larger networks of IoT devices, which hinders the establishment of a comprehensive IoT data infrastructure for building automation and management.

There are four major challenges in establishing a comprehensive IoT data infrastructure for buildings. First, the high cost of the sensing devices is a major reason for the reluctance of building owners and managers to install a data system in the first place. For example, the average cost per square foot to outfit a facility with a Building Automation System (BAS) was \$2.30, in 2017, which means it will cost about \$230,000 on BAS alone for a typical 100,000 square foot building [102]. Moreover, building systems such as BAS will not last for the entire life cycle of a building, and there will be upgrade fees for about every ten years or less. Second, the installation of building systems can be a challenge for existing buildings. The best time to install a building system, with numerous wires and devices, is during construction. After a building starts to operate, installing additional systems or changing the existing ones is usually disruptive to occupants.

The third challenge is that most of the data generated by commercial building systems are self-proprietary because most of the existing building systems lack interoperability. Typically, there is a lack of means to use the data collected in one system for another system's functions. Different systems need to generate the indoor environment data they need, and the data are not in a format that can be used for other purposes, which significantly limits the potential of smart building innovations. Scalability is the fourth challenge. For many building systems, after the initial installation during construction, it is usually difficult to modify the system by adding or removing features, or increasing the coverage to other building spaces. The

system’s functions and scale, including what kind of indoor environment data to generate, are usually settled by the time of initial installation. Scaling up the system may cost more than installing a brand-new system that can cover the required functions and spaces. In addition, current building systems are lacking portability. Usually, after a system is installed in a building, it normally cannot be ported to a different location later on. It is very challenging to take out the system from the installed location and place it in another location without extensive engineering and reconstruction.

In this work, I present a cost-effective, scalable, portable, and distributed indoor environment sensing system, Building Data Lite (BDL), to address the building data availability challenges, and thus, to provide a means for establishing the IoT data infrastructure of a smart building. To collect indoor environment data, the system has three types of components: sensing nodes, central nodes, and inter-node connections. The sensing nodes are an array of digital and analog sensors to collect data from the surroundings. Each sensing node has its local database. A central node is a web-based central server that integrates data from all local databases and visualizes data on a website. The inter-node connections link sensing nodes with the central node. The system is scalable such that each sensing node’s function (what data to generate) is customizable, and it is easy to add sensing nodes to or remove nodes from an established system. The system also includes a web-based Graphical User Interface (GUI) for users to extract, view, and analyze the generated data. Industry 4.0 [36] introduced a system that enables appliances such as refrigerators or air-conditioners to capture relevant data and pass it to cloud storage. The BDL system, on the contrary, uses dedicated sensing nodes to capture targeted data through connected sensors. This research proposes an innovative means of establishing cost-effective, scalable, and portable IoT infrastructure for indoor environment data generation, integration, processing, and presentation. The research team developed a prototype of the BDL system to demonstrate how it

overcomes the limitations of existing building systems regarding indoor environment sensing in the following manners:

a) Cost: The BDL system is built with a mini-computer Raspberry Pi and compatible sensor modules. The cost of each sensing node is limited to \$40 to \$60, depending on the sensing requirements. Typically, each sensing node can cover one building zone, and therefore, for a building with 100 zones, the entire system only costs about \$5,000. The BDL system is fairly affordable for indoor environment sensing requirements by smart building research and innovations.

b) Installation: The distributed sensing nodes (Raspberry Pi computers) of the BDL system are connected with a centralized server via wireless communication. Each node can capture multiple indoor environment values of a particular building space. The installation of the sensing nodes requires only a power supply and Internet/WiFi connection.

c) Comprehensiveness: The BDL system uses multiple sensors to collect different environment data simultaneously, and establish a comprehensive database that can be used for developing multiple smart building applications. The developed prototype can capture data related to temperature, humidity, light, sound, motion, vibration, flame, and three types of gas.

d) Scalability: The BDL system is customizable, and the users have the option to add or remove devices (Raspberry Pis), sensor modules, and functions.

e) Portability: The nodes of BDL system are connected to the central server via local WiFi or the Internet, which makes them portable to any location with a power supply and under the coverage of a wireless network. The sensing nodes can be removed from one location and then deployed in another location without any engineering on the building.

f) Open-source: The source code of BDL has been published by the authors via GitHub [42].

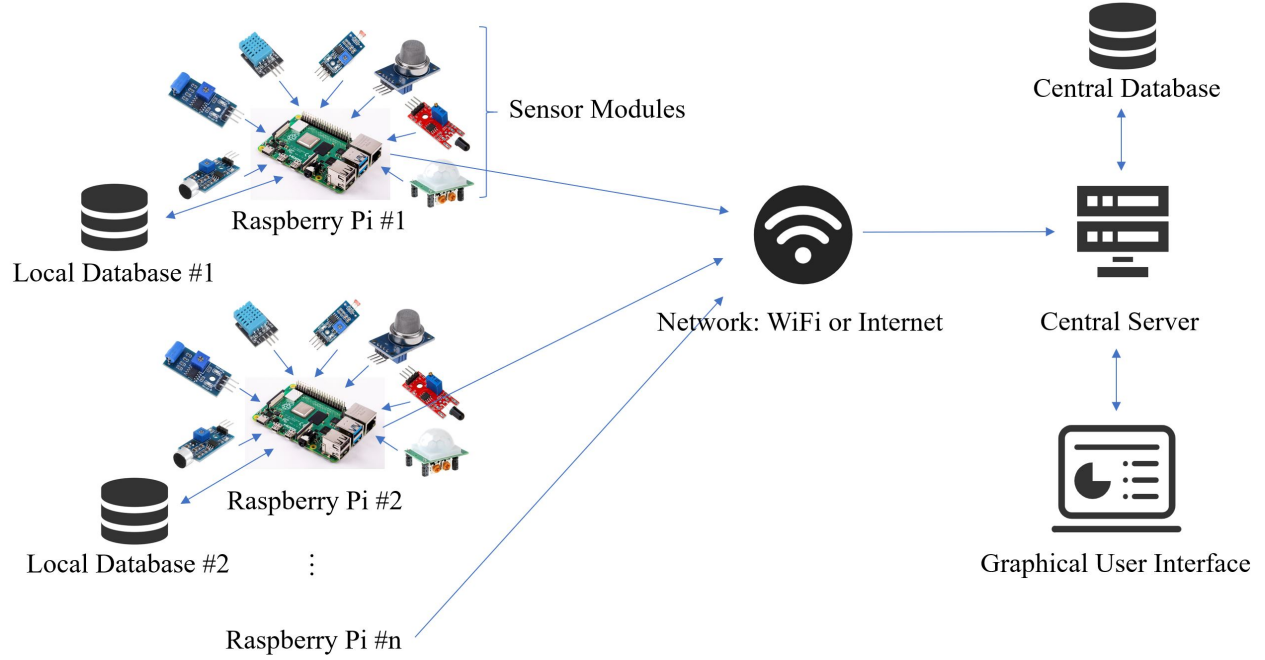


Figure 3.1: The BDL system architecture.

The BDL also provides an open-source, integrated database system that makes it easier for other systems to utilize the generated data.

This chapter is structured as follows: Section 2.1 reviews relevant studies conducted in the indoor environment sensing field. Section 3.2 presents the BDL system’s overall design. Section 3.3 demonstrates the prototype development. Section 3.4 presents the case study in which the BDL system is deployed in 12 households. Section 3.5 discusses the challenges encountered and corresponding solutions, potential use cases of the BDL system, and the current limitations and future research directions. Section 3.6 concludes the work.

3.2 The System Design of BDL

Unlike previous studies that developed a sensing system for a particular use case, this research focuses on the development of a more generic-purpose, cost-efficient, scalable, portable,

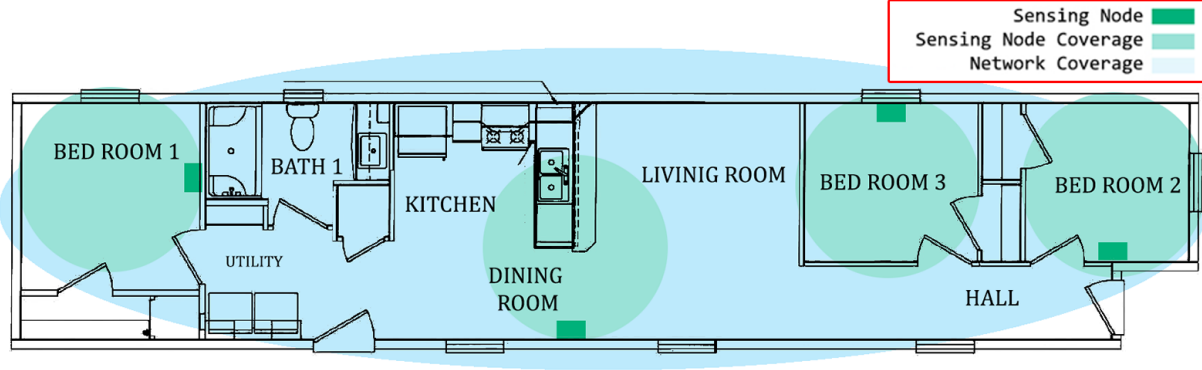


Figure 3.2: An example of BDL deployment.

and distributed indoor environment data sensing system that can be used to establish the data infrastructure for smart building innovations. This section demonstrates the system design of BDL by describing the system architecture, central server, sensing node, database configuration and data transmission, network configuration, data integration, and graphic user interface.

3.2.1 System Architecture

The BDL system, as shown in Figure 3.1, consists of a central server and multiple individual sensing nodes, which are built around Raspberry Pis. The sensors in each sensing node capture real-time data from the environment and transfer the data to the connected Raspberry Pi, which has its own local database. After a certain interval, the Raspberry Pi communicates with the central server, and uploads the newly generated data to the central database. The GUI communicates with the central database to visualize data, and provides functions such as downloading (selected) data, downloading the error log, and modifying the system by adding or removing sensing modules or sensing nodes.

Figure 3.2 illustrates an example of the BDL system deployment. The green portions rep-

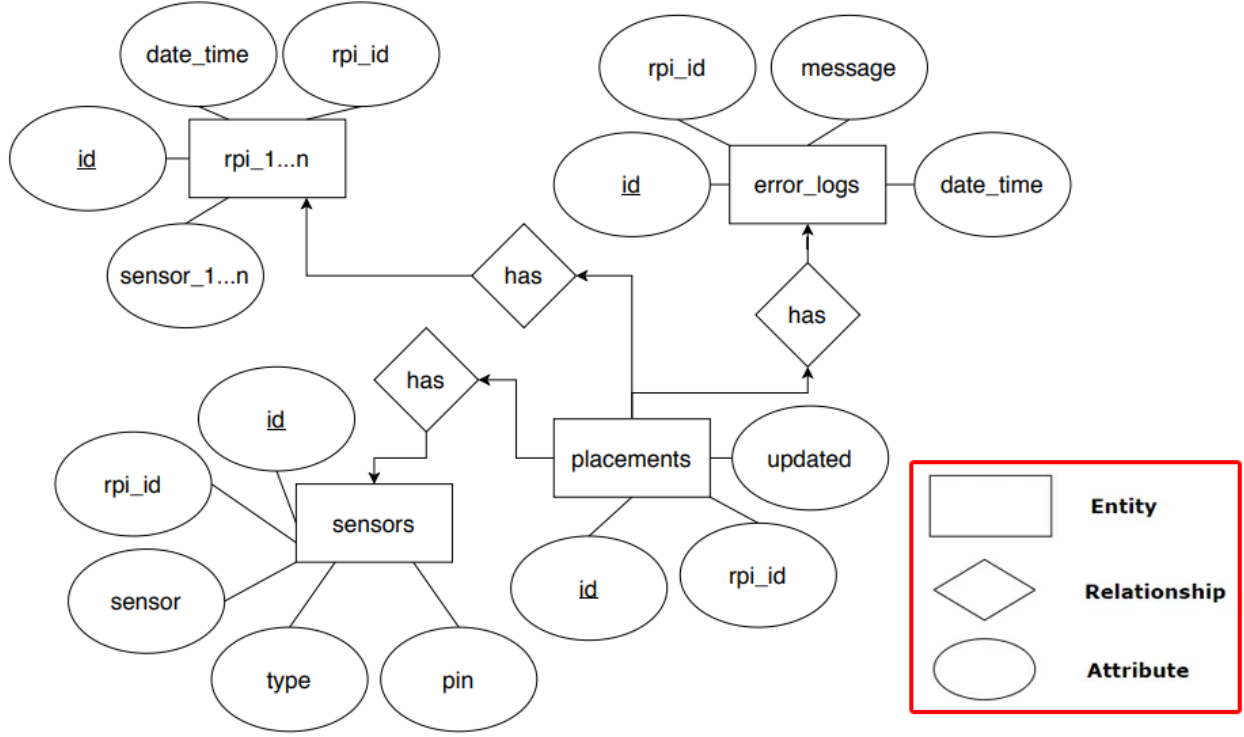


Figure 3.3: Entity relationship diagram of the central database.

resent the locations where individual sensing nodes collect environmental data. The blue area denotes the area covered by the wireless network. The central server is also connected to this network either locally or through a live server. The sensing nodes are modular, and each node can have different sensors. They are portable and can be deployed anywhere with power and network connections.

All software used in the system is open-sourced, and both the database systems used, MariaDB and MySQL, are free. The system software including both the server and the sensing node has been made publicly available and can be found in the GitHub repository [42]. The server is currently available for open access to help researchers and industry practitioners with the need of indoor environment sensing.

3.2.2 Central Server

The central server of BDL is designed as a web-based system. The prototype uses PHP v7 as the server language. The front-end was written in HTML, CSS, and JavaScript. The system is designed to use Asynchronous JavaScript and XML (AJAX) to create a fast and dynamic GUI. JQuery and Chart.js libraries are used for data visualization. The GUI fetches data from the central database. The server is live and being hosted in a third-party hosting site with the domain: www.building-data-lite.com. Figure 3.3 shows the entity relationship diagram of the central database, and it is a general overview of the database schema. Here the rectangular shapes denote the entities, the circular shapes denote the attributes of the entities and the rhombus shapes denote the relation between each entity.

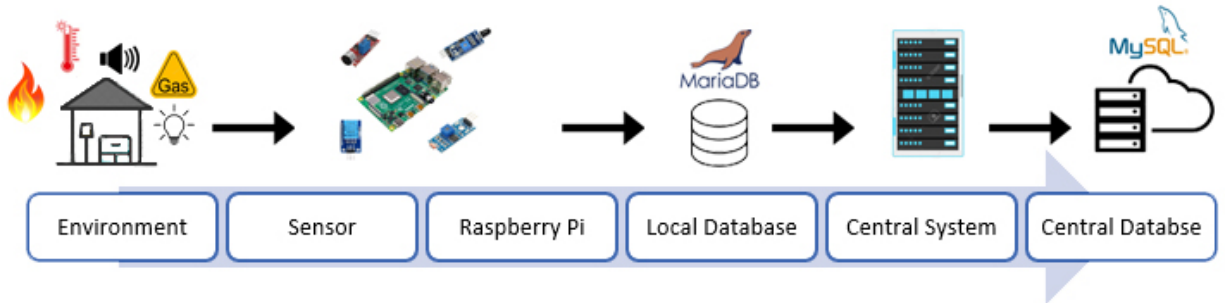


Figure 3.4: Data flow in the BDL system.

Each sensing node has its local database system to store offline data. This is essential because there can be disruptions of communication between the sensing node and the central server. The sensing nodes use MariaDB for the local database. To keep data consistency, a temporary data file is being created every hour consisting of new records that have not been sent to the central server. The central server provides the information (for example, the timestamp of the last synchronization) to make sure the data in this file are not redundant. In each synchronization, the file is transferred to the central server, and then, replaced by a new one to reduce space wastage. Approximately 60 records are present in each hourly generated

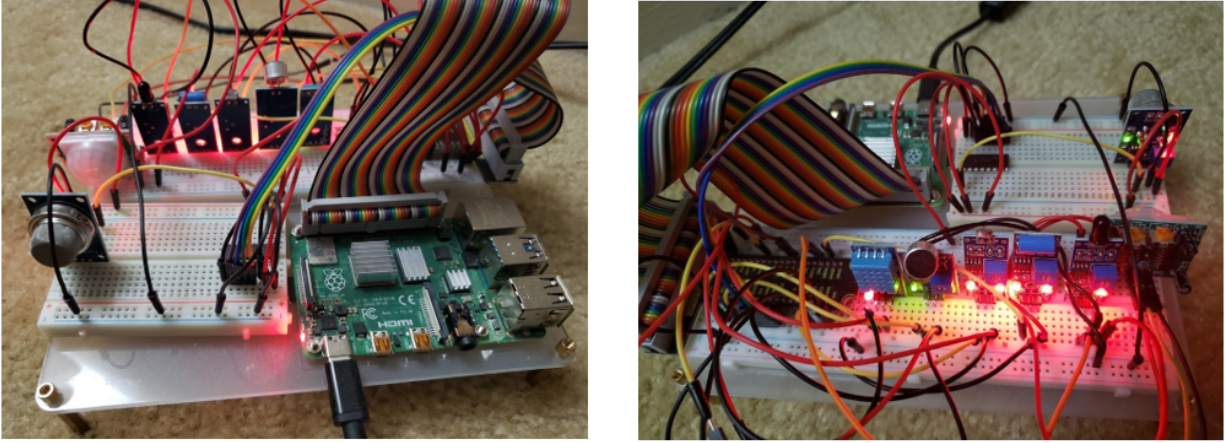


Figure 3.5: Sensing node prototype version 1.

file, but this number will be doubled for each occurrence of communication disruption. The central server performs a synchronization check of the received data before storing it in the central database. Figure 3.4 illustrates the data flow in the BDL system.

3.2.3 Sensing Node

In each sensing node, the Raspberry Pi connects with multiple sensor modules to collect data from the surroundings. Figure 3.5 shows the prototype of a sensing node. The sensors are connected via the GPIO ports of the Raspberry Pi, with a GPIO extension breadboard for easier deployment. The Raspberry Pi can take digital inputs only, which is a problem in incorporating analog sensors, such as the light sensor and the gas sensors. To resolve this issue, the authors use an analog-to-digital converter, MCP3008 [85], to capture readings from the analog sensors. The MCP3008 has 8 channels, which means it can connect up to 8 analog sensors with the Raspberry Pi. The rest of the GPIO pins onboard will be able to connect one digital sensor each. The sensors have been categorized in three types depending on their implementation with the Raspberry Pi. Type 1 is the direct input sensors, for example, the light-dependent resistance sensor (LDR) [76]. These sensors connect to the GPIO pins

and Raspberry Pi can be directly read data from these pins. Type 2 refers to the sensors that provide a feedback signal to the connected pin when an event occurs, for example, the sound sensor [104, 113] or the motion sensor [111]. The third type of sensor does not share a common implementation pattern and needs to have a specific implementation code. The repository will be continuously updated with new sensors of this type for users to just select and use. Advanced users can add their custom code to the system for this type of sensor. The code repository [42] includes a section to add custom codes for this sensor type.

The range of each sensor is different and some sensors include a potentiometer to tune the amplitude of readings or range of the sensor. As the system is designed to be of generic purpose, the range of a sensing node will depend on the sensors used in the particular node. For example, the default radius of a PIR motion sensor is 6 meters with a 120-degree angle while a temperature sensor should capture the temperature of the air surrounding the sensor. The sensing node captures environment data through the connected sensor modules and stores them in its local database, which is designed based on the connected sensing modules. It captures one value from each sensor at every given time frame, for example, 60 seconds. The Raspberry Pi must have an operating system installed and include the required packages to run Python scripts of the sensing program. The packages required to run the program are listed in a note with more instructions on the code repository [42].

There are multiple options for the core of the sensing node design, such as ESP8266, ESP32, Nodemcu, Arduino, Pycom, and our selected board of Raspberry Pi. While the other boards are slightly cheaper than Raspberry Pi (model 4B), they are mostly microcontrollers and/or with WiFi connectivity modules. These boards provide easier connectivity with low-level sensors but Raspberry Pi boards provide the functionalities of a complete computer system with operating system, memory, database, built-in WiFi module, HDMI, and USB extension ports. It is easier to program, collect, and store data in a Raspberry Pi than the

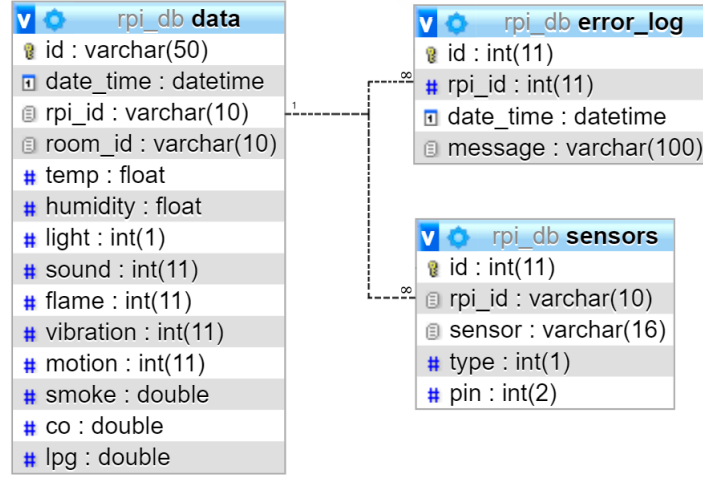


Figure 3.6: Local database design.

other boards, especially for a non-technical individual.

3.2.4 Database Configuration and Data Transmission

There are two separate database designs in the BDL system. One is used in the sensing nodes (shown in Figure 3.6) and another in the central server (shown in Figure 3.3). The database shares some common entities such as “error_logs” and “sensors”. The “data_storage” table of the local database represents a specific Raspberry Pi table in the central database with the name “rpi_n” where n represents the Raspberry Pi identification number.

The specific implementation of the system will vary according to what sensor modules are used by each customized sensor node. The “placements” table records Raspberry Pi nodes present in the system. It has a column “updated” which refers to changes made in the Raspberry Pi. It will be 1 if a sensor has been added to the system or removed from the system and 0 otherwise. The entity “rpi_1...n” represents sensing nodes based on Raspberry Pi boards. A new table marked with an increasing identification number is created every time a new node is inserted into the system. Every node can have a different set of sensors

connected to it. The dynamic set of sensors makes changes to sensor columns “sensor1_1...n” in the corresponding Raspberry Pi table. The sensor information of each sensing node is also recorded in the “sensors” table which contains implementation-specific information, such as sensor type and connecting pin number. Each record in the “rpi” table is denoted by a unique id generated from the combination of date_time and rpi_id. The date_time and rpi_id (Raspberry Pi ID) are string data and are present in separate columns to enable faster queries. The rest of the columns represent different information records related to the sensors, such as light presence, temperature, sound, etc. The central database contains additional tables to keep track of the sensing node placements and sensor modules. The “error_logs” table contains exception message logs. The local database system used in each sensing module is MariaDB, which is usual for Raspberry Pi systems. The central database is created using MySQL. MariaDB and MySQL are similar in many aspects, and it is convenient to maintain consistency between these.

Data transmission in the system is done through post methods in Hypertext Transfer Protocol Secure (HTTPS) which is secure. Relying on the security of HTTPS, external encryption and decryption mechanisms have not yet been introduced in the current BDL system but there are plans to introduce those in the future. The frequency of requests in the system will depend on the number of sensing nodes in the system and not the number of sensors in each node. This is because a sensing node will send a chunk of data collected from all its connected sensors in a preset interval which can be an hour, a day, or a month. This practically eliminates the limit of server request bottleneck because it is very unlikely that multiple sensing nodes will hit the server at the exact same time. Even if such a scenario occurs, the HTTPS server can handle at least 200 requests a minute.

3.2.5 Network Configuration

The BDL system is designed to run in both local and global connectivity. For the global setup, the server needs to be running in a live domain. Many live domain hosting providers are capable of running a PHP server. It is one of the most common server languages and is easy to implement. After the server project folder has been uploaded to the server's public HTML folder, the server can start running at the domain address. The domain address needs to be set in all sensing nodes, by the change of a single line of code. Each sensing connects to a wireless network. If a server is deployed in local mode (on one of the Raspberry Pis rather than on an independent computer), then all sensing nodes and the server need to be connected to the same network. If working in the global mode, then each node and the server need to be connected to the Internet.

The DBL system prototype v1 adopts the Raspberry Pi model 4B, and the prototypes v2 and v3 adopt the Raspberry Pi Zero W because they all have built-in WiFi modules for wireless network connections. Data are transferred from the sensing nodes to the central server via the network, which can be WiFi or wireless Internet (4G, 5G, etc.). If the network connection is broken for some reason, the sensing nodes will keep working in offline mode, in which they will keep generating data and storing them in their local databases. The data will be synced whenever the connection is resumed.

3.2.6 Data Integration

In each sensing node, the sensors output data to connect Raspberry Pi either directly or through an analog-to-digital converter (ADC), and the Raspberry Pi receives the data through the GPIO ports. The sensing node is programmed to run in an infinite loop to collect data continuously. In the prototypes, there is a 60-second interval between the read-

ings, and it can be changed according to demand. The data record interval does not need to be manually adjusted and the user can view the data in different intervals on the GUI regardless of the input interval. The viewing interval however can be either equal or greater than the recorded interval, for example, if data is recorded every minute then the records cannot be viewed per 30 seconds but they can be viewed as per minute or as per 10 minutes. The Raspberry Pi collects all sensor readings each minute and merges them together to make a record. The record is then stored in the local database with a timestamp and other necessary information, such as location and the identifiers of sensor modules. If any of the sensors fail to provide data, a system exception is thrown. The sensing node records the exception details with the timestamp in a separate table in the local database. The data is stored in numeric format. Each sensor is different in regards to its readings, for example, the temperature sensor (DHT11) provides readings in degrees centigrade while the humidity reading is in percentage. Figure 3.7 illustrates some data format samples from the sensors connected in the designed prototype. Here, each column from ‘temperature’ to ‘lpg’ refers to a sensor name and the records show the different types of values each sensor generates.

date_time	temperature	humidity	light	sound	flame	vibration	motion	smoke	co	lpg
3/4/2021 15:28	23.7	12	0	6	0	0	0	1.674034	1.536071	0.47363
3/4/2021 15:27	23.7	11	0	6	0	0	0	1.717727	1.536071	0.485193
3/4/2021 15:26	23.8	13	0	7	0	0	0	1.652684	1.485697	0.47363
3/4/2021 15:25	23.7	12	0	7	0	0	0	1.350867	1.125592	0.390769
3/4/2021 15:24	23.7	13	0	6	0	0	0	1.318867	1.058022	0.374653
3/4/2021 15:22	23.7	13	0	12	0	0	0	1.165735	0.930124	0.343073
3/4/2021 15:21	23.7	14	0	6	0	0	0	1.097811	0.860598	0.319926
3/4/2021 15:20	23.7	14	0	7	0	0	0	1.100925	0.889631	0.319926
3/4/2021 15:19	23.6	14	0	9	0	1	0	1.272542	1.041864	0.370496
3/4/2021 15:18	23.5	13	0	10	0	0	0	1.228476	0.995414	0.370496
3/4/2021 15:17	23.4	13	0	13	0	0	2	1.100925	0.863759	0.323356
3/4/2021 15:16	23.4	14	0	10	0	0	0	1.085458	0.848087	0.316553
3/4/2021 15:15	23.4	13	0	7	0	0	0	1.085458	0.848087	0.316553
3/4/2021 15:14	23.5	13	0	6	0	0	0	1.03785	0.800263	0.305919
3/4/2021 15:13	23.7	13	0	9	0	0	0	1.049515	0.788796	0.302735

Figure 3.7: Sample data format

In the prototype system, the sensing node is programmed to communicate with the central

server once every hour (which can be changed). Each sensing node sends its identification string to the central server, and the server returns the ID of the latest stored data from the corresponding sensing node. After receiving that ID, the sensing node queries the local database for all the records inserted after that given ID. The local database then returns the newly inserted records to the sensing node. These are the records that are present in the sensing node but not yet inserted in the central database. The sensing node then creates a CSV file with these records and sends the generated file to the server. The central server receives the data and performs another synchronization check before inserting the records into the central database.

3.2.7 Graphical User Interface

The BDL system's GUI has five functions, which are 1) data visualization, 2) data download, 3) error log download, 4) add or remove sensing nodes (Raspberry Pis), and 5) modify sensor modules on each sensing node. Figure 3.8(a) illustrates the home page, which updates dynamically on the value change of the control fields. To visualize the data, the user can specify the sensing node(s), the time range, time interval, and sensor module(s). The data is generated by the sensors connected to the sensing nodes. The data is represented in a graph chart with the date-time on the x-axis and sensor reading on the y-axis. The user can also download the numeric data in a CSV file through the data download page shown in Figure 3.8(c). The user has the option to select the sensing node and sensors from the list of connected sensors. The system's error log collects error information from all the sensing nodes and it can be downloaded for each sensing node and specified time span, similar to the data download page as shown in Figure 3.8(c).

The BDL system architecture is designed in a way that it is capable of including or removing

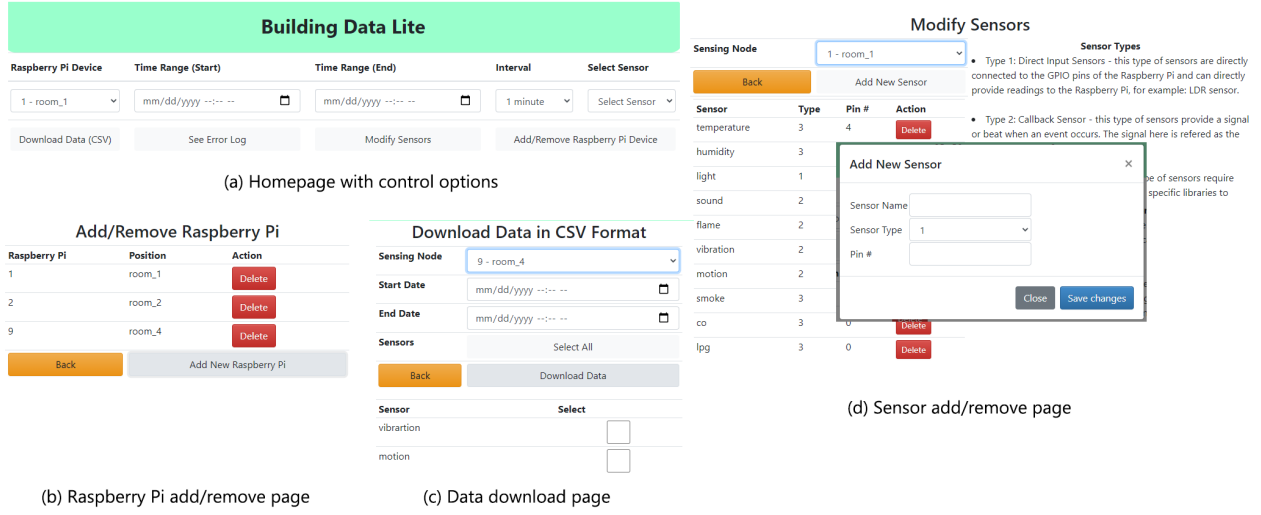


Figure 3.8: BDL system's Graphical User Interface

sensing nodes and sensors of each node as per the need. The user has to connect and disconnect the hardware manually, and the BDL system will adjust the code based on the user inputs from GUI. The GUI has the option to add/remove sensing nodes and sensors, which can be done by clicking the button. The user does not need to write any code to add or remove a sensing node or a sensor. After the operations on hardware, which are setting up the Raspberry Pi, and connecting or disconnecting sensor modules to it, the user can use the GUI to finish the system configuration, as shown in Figure 3.8(b) and Figure 3.8(d).

The sensors are categorized into three types as described in section 3.2.3. The GUI offers a dropdown list for the user to select the sensor type when adding a new sensor. The user needs to specify the sensor name, type, and, connected pin in the GUI as shown in figure 3.8(d). As mentioned in section 3.2.3, the sensors of type-3 require specific implementation code, to minimize the effort on the user, the GUI of BDL offers a list of supported sensors which the user can directly choose from. This list will be kept up to date to support the latest sensors. The advanced users also have the option to write and add their custom code in the system for this type of sensor. The code repository[42] includes a section to add

custom codes for this sensor type.

3.3 Prototype Deployment

In the BDL system, each sensing node is centered around its own Raspberry Pi, which is connected with a set of analogue and digital sensor modules. Any sensor that is capable of communicating through the General-Purpose Input Output (GPIO) ports of the Raspberry Pi system can be connected in this system. The first two prototypes of the BDL system were developed using the following sensors: DHT11 (temperature and humidity) sensor, light sensor, sound sensor, vibration sensor, motion sensor, MQ2 (smoke, natural gas and carbon mono-oxide) sensor. The third prototype version includes a sensor array named Enviro Plus [101], which includes light, proximity, gas, and sound sensor. Each sensing node is connected to the central server via either a WiFi network or a wireless Internet connection. The prototype 1 and 2 use the central server in local connectivity mode, in which the server and the sensing nodes need to be connected in the same wireless network to communicate. The prototype 3 uses the global mode, in which the server is implemented in a live server in the cloud, eliminating the need for being in the same wireless network with the sensing nodes. In this mode, each sensing node needs to be connected to the Internet via either wired or wireless connection. The Raspberry Pi model 4B used in the first prototype features both types of connections while the other two prototypes use Raspberry Pi Zero which uses the wireless communication.

The sensing nodes of the BDL system are customizable and can be implemented in different ways. To demonstrate the different implementations, three separate variations of the sensing node were created and tested. While the server-side remained the same, the physical hardware setup of the sensing nodes varied. Figure 3.5 shows the bread-board setup which



Figure 3.9: Sensing node prototype version 2 (left) and version 3 (right).

is completely scalable with plug-in sensors (prototype version 1). Figure 3.9 (left) shows the Printed Circuit Board (PCB) version of the bread-board variant using a Raspberry Pi Zero, which is prototype version 2, and the third version of the prototype, which uses a Raspberry Pi Zero connected to an Enviro Plus sensor module, is shown in Figure 3.9 (right).

Figure 3.10 shows the structure of one of the prototype sensing nodes (v1 & v2). It contains seven sensors and provides 10 types of readings. In the figure, the solid lines represent the digital data and the dashed lines represent the analog data. Prototype v3 has a similar design with a different set of sensors.

The sensors used in the prototype v1 & v2 are listed as follows:

- Temperature and Humidity sensor (DHT11) : This sensor provides readings for temperature and humidity[44, 103, 121]. The system uses degree centigrade as the unit of temperature and percentage as the unit of humidity. It uses a capacitive humidity

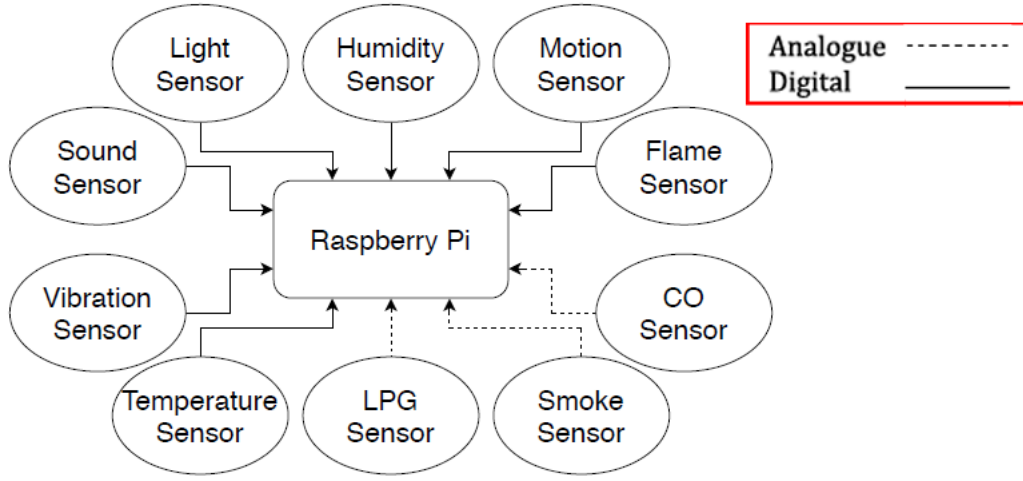


Figure 3.10: Structure of the prototype's sensing node.

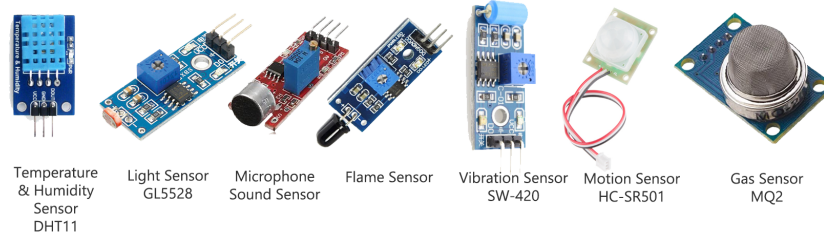


Figure 3.11: Sensors used in sensing node prototype.

sensor and a thermistor to measure the surrounding air, and can generate new data every 2 seconds.

- Light sensor (GL5528): The light sensor is one of the digital sensors. It uses photo-detector GL5528 to detect light intensity of the surrounding environment. The resistance of the sensor varies depending on the amount of light it is exposed to which changes the output voltage. It provides 0 when it detects light and 1 if it does not detect any light. The sensor has a tuning potentiometer (POT) for adjusting the reading threshold [76].
- Microphone Sound sensor: The sound sensor is also a digital sensor. When the module

in the intensity of the sound environment does not reach a threshold (set through the included variable resistance), the sensor will output a high signal. Otherwise, it will output a low signal. The sensor detects a change in sound amplitude and provides a reading [104, 113]. This reading can be called a beat. The designed sensing node counts the number of beats in a time interval for the data record.

- Flame sensor: This sensor detects the presence of flame nearby. It is sensitive to flame wavelengths between 760nm to 1100nm in infrared. It is a binary sensor and provides 1 for the presence of flame and 0 for otherwise. There is a tuning potentiometer to adjust the reading threshold of the sensor [114].
- Vibration sensor (SW420): The vibration sensor can detect nearby vibration beyond a threshold defined by a potentiometer and provide a binary reading [47]. The beat counting mechanism has been used to present the readings of this sensor (how many beats per minute).
- Passive Infrared Motion sensor (HC-SR501): Passive Infrared motion sensor, also known as PIR sensor, records the intensity of motion detected nearby [111]. It comes with two tuning potentiometers to adjust sensitivity and reading delay. The output goes high when objects enter the sensing range, and automatically returns to low when object leaves.
- Gas Sensor (MQ2): This is an analog gas sensor that requires an additional analog to digital converter circuit to incorporate this sensor with the Raspberry Pi. It can provide continuous readings of smoke, carbon monoxide (CO), and liquefied petroleum gas (LPG) [67].

Prototype v3 uses a sensor array, Enviro Plus [101], which packs light, proximity, temperature, humidity, gas, and sound sensors. The prototype's source code is shared through

an open-source project published by the authors [42]. The wiring settings of each sensor are written in comments of the source codes. A GPIO extension for the Raspberry Pi has been used in the prototype development for ease of implementation in a breadboard setup (v1). The digital sensors are directly connected to the GPIO pins of the Raspberry Pi. The analog sensors are connected to the MCP3008 analog-to-digital converter, which forwards a digital output to the Raspberry Pi for the respective analog sensor connected. Figure 3.11 shows the different sensors used in the sensing node prototype. From the left, DHT11, light sensor, sound sensor, flame sensor, vibration sensor, motion sensor, and, MQ2 gas sensor. The version 2 uses the same setup in a printed circuit board setup, which makes the setup more compact.

To demonstrate the flexibility and stability of the sensing nodes, a total of 30 nodes including all three variants have been developed and deployed which are continuously collecting data for over 4 months.

3.4 A Case Study

To evaluate and demonstrate the capabilities of the BDL system, the research team has been conducting a case study. A total of 48 sensing nodes are being deployed and collecting data continuously in 12 households (four sensing nodes each, distributed in different rooms) of an affordable community in Richmond, VA. The data collected by the sensing nodes of the BDL prototype 3 system are being transferred to a live cloud server.

The typical sensing node deployment is illustrated in Figure 3.2, in which one of the house's floor plans and the locations of the devices are shown. Figure 3.12 shows a set of images about the deployed devices, and Figure 3.13 shows an image of a house in the case study.



Figure 3.12: Image a deployed sensing node.

The research team used BDL prototype version 3 for the case study because it is more compact and, with the 3D-printed cases, it is more acceptable for the house occupants participating in this study. BDL v3 uses Raspberry Pi Zero with a built-in WiFi module, connecting with the Enviro or Enviro Plus sensor bundles. The Enviro Plus bundle includes proximity, humidity, pressure, light, sound, and gas sensors. It is the advanced version of the Enviro bundle, which includes all these sensors except for the gas sensor. The research team has to use some of the Enviro bundle because the global chip shortage is impacting the availability of the Enviro Plus bundle. The server-side code is slightly modified so that



Figure 3.13: A photo of a house in the case study.

both types of sensor bundles can be installed in the system with just a single click.

The deployed system's central server operates in global mode on an Apache server [93] of the Xampp module [59]. The server is established in a laptop with Microsoft Windows 10 as the operational system. The central database is built with MySQL [107]. Local WiFi networks are used to connect each house's sensing nodes to the cloud-based central server. The sensing nodes' local databases are built with MariaDB [73]. If the server becomes unavailable in situations such as lost Internet connection, the data transfer will be canceled. Instead of making a second communication attempt immediately, the sensing node will keep collecting more data and then repeat the data transfer process after another hour. In this process, the data may pile up in the sensing node, and the generated file size will keep increasing every hour but 60 records per hour will not make the file size larger than the transfer capability of the current network systems.

The web-based GUI displays graphs of the data collected from the 12 houses and stored in the central server. With the proper selection of the control options (shown in Figure 3.8a),

the graph representation of the collected data can be presented on the website (building-data-lite.com). The system can show data graphs for all sensors of a sensing node on a single page, as Figure 3.14 shows, or the data graph of a particular sensor with additional detailed information such as minimum, maximum, and average, as Figure 3.15 shows.



Figure 3.14: Visualization of collected data from all sensors of a deployed sensing node.

Figure 3.14 illustrates a truncated view of 6 sensor readings of a sensing node. The graph presents the collected data of this sensing node from 1st August 2021 to 31 August 2021. On the top row, the data from the proximity sensor and the humidity sensor are shown, and followed by the data of the pressure sensor and the light sensor on the second row, and then, the data of oxidised gases and reduced gases on the third row. Figure 3.15 shows the temperature data captured by the sensor node Raspberry Pi #10 (rpi_10). The minimum, maximum, and average values are 27.7, 26.3, and, 26.98 degrees centigrade respectively. In all graphs, the x-axis represents the date-time of data and the y-axis represents the reading of a particular sensor.

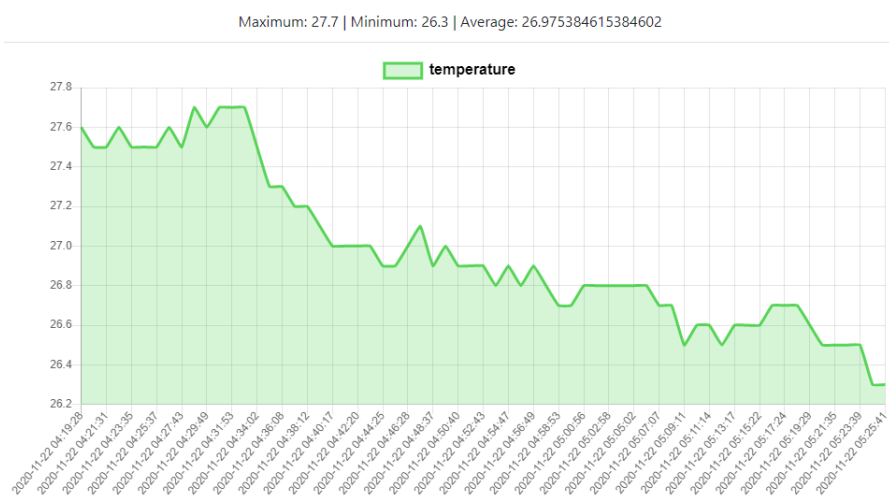


Figure 3.15: Visualization of collected data of a single sensor.

The Enviro and Enviro Plus sensor boards also include an LCD display. The BDL v3 sensing nodes utilize this display to provide the house occupants with a reading of the most recent data.

The deployed devices in this case study are planned to collect data for at least one year. The data will be collected with full respect to the privacy of the tenants. These indoor environment data will be used for research studies related to low-income households, the performance of manufactured houses, and other related topics.

3.5 Discussion

3.5.1 Challenges and Solutions

This section discusses the technical challenges encountered by the authors and provides corresponding solutions.

Binary sensors: Some of the sensors provide only yes-no data. For example, the sound

sensor gives a reading when it detects a sound, but it cannot provide any measurement of the amplitude of the noise. A beat counting system was introduced to count the frequency of sounds during a time interval to achieve an understandable reading from the sensor.

Sudden crashes: There were some sudden run-time crashes during the data collection phase. The problems were caused by faults in sensor readings, bad connection, overloaded access, and unavailability of server. Python’s run-time exception-handling feature was used to tackle these issues. Two types of exceptions were caught during this phase. The first one occurred due to a problem in sensor reading or sensor connection and the second one occurred due to server unavailability, longer wait time, or bad network connectivity. These exceptions were caught and logged in a local database and later the log is transferred to the central database.

Data synchronization: Synchronization is important for the data continuity and integrity between each sensing module and the central server. The authors solved this issue using a two-fold solution combined together. First, a routine update mechanism between the sensing node and the central server was designed. Instead of transferring data records, we transferred files consisting of multiple records. Second, a record-checking system was implemented in the sensing node to keep data consistent. The databases in both sensing nodes and the central server were restructured to reduce the number of entries (the database designs for the sensing node and the central server are shown in Figure 3.3 and Figure 3.6, respectively).

File size limit: The sensing nodes transmit the newly added records to the central database through a data file. If the transmission is not complete for some reason, such as the server is unavailable, the sensing node will wait another hour for the next transmission attempt. The file size increases as the number of unsent records piles up. The problem occurs when the file size exceeds 2 Megabytes, which is the default limit of file transfer in the local PHP server. The authors addressed this issue by manually overwriting the maximum transferable file size limit in the “php.ini” file.

3.5.2 Potential Use Cases

The proposed BDL system is an innovative method to create the data infrastructure for existing buildings. It is also one step towards many potential smart building applications. Several example use cases are listed as follows:

Space utilization analysis: The BDL system can be used to collect historical data on building space usage. These data can be used for space utilization analysis to deduce whether the space is being used properly or not; whether it is underused or overused; what is the indoor environment when the space is used, etc.

Building control system: The BDL system can provide data to, or be merged with a building control system to enable automated building control. With sufficient data and analysis (such as machine learning on occupant behavior), the system can automatically control the indoor environment, such as light, temperature, humidity, ventilation, etc.

Emergency management system: The BDL system can be modified to become an emergency warning system. It can warn the building owner if it detects any abnormal characteristics in the indoor environment. The user can set a threshold for a particular reading to be marked as abnormal, or machine learning models can be developed to detect anomalies.

Activity monitoring: The BDL system can be used as a monitoring system for detecting unusual activities. It can provide data to, or be merged with a building security system that can warn the building owner that there is an unusual activity, such as burglary, in a particular building space.

3.5.3 Limitations and Future Research

Although the proposed BDL system's first prototype has been finished and experimented with, the system is still under development, and there are some limitations, which will be

resolved in the future. Some of the sensors require fine-tuning and calibration to provide a more accurate reading. The prototype has been calibrated, but for more accurate results, the system needs to be calibrated in different environmental conditions. The authors have plans to create more sensing nodes and deploy them in different locations. It will help us in tuning and calibrating the sensors for improved accuracy.

3.6 Summary

This work in this chapter presents the research study that aims to establish a distributed wireless sensing network for collecting indoor environment data in existing buildings. This research contributes to the body of knowledge by proposing an innovative way to establish a cost-effective, scalable, and portable IoT data infrastructure for indoor environment sensing. A wireless distributed sensing network named Building Data Lite (BDL) is proposed with the design of the system and three developed prototype sensing nodes. A case study with one of the prototypes has been conducted with 12 households in an affordable community, equipped with a total of 48 sensing nodes. The data that can be generated by the prototypes involve temperature, humidity, light, sound, flame, vibration, motion, smoke, carbon monoxide (CO), and liquefied petroleum gas (LPG).

This system is envisioned to be implemented in various domains, such as the Internet of Things, and occupant behavior-related studies. The BDL system provides a code-free system that means researchers without any coding background can implement the system and the sensing nodes by following the guidelines. The system includes a graphical user interface that enables the user to access, visualize, and download the data, and modify the sensing nodes and sensor modules as needed. The experiment has shown that the proposed system is functional, portable, and scalable. Each sensing node costs only \$70, and the software

used is completely free.

Currently, the system utilizes the general-purpose input-output ports of the Raspberry Pi for communication with the sensors. There are sensors or devices such as a microphone or a camera that connect to the Raspberry Pi through the Universal Serial Bus (USB) ports which are not yet supported by this system. Some of the sensors require customized libraries to provide readings which calls for some code scripts. It can be further optimized and automated. The plan is to integrate these features in the future to further improve the system's functionality and usability. With the data infrastructure developed and deployed, the task comes is the utilization of the data collected from such data infrastructures. The next chapter discusses the work conducted with residential energy consumption data and state-of-the-art machine learning tools.

Chapter 4

Machine learning approach in the development of building occupant personas

4.1 Motivation

People spend most of the time indoors [74], and the impact of a building on its occupants is significant [45]. Smart Buildings are buildings that integrate intelligence, enterprise, control, materials, and construction as an entire building system, with adaptability, not reactivity, at its core, to meet the drivers for building progression: energy and efficiency, longevity, comfort, and, satisfaction [50]. In recent years, human-centeredness has become an increasingly important factor for smart building design and operation. Optimizing building performance requires a deep understanding of occupants' behavior and preferences. Developing building occupant personas is proven to be effective in creating occupant profiles for human-centered smart buildings [35]. An occupant persona is a tool of communication for designers to develop a conceptual model that describes the archetype of occupants [49]. The development of a building occupant persona can assist the building designers, developers, and, engineers in accurately estimating resource demand through realistic and representative occupant profiles, enabling pragmatic occupant-centric building design and operation. The increasing avail-

ability of data from a wide range of sources will allow smart buildings to become adaptable, and prepare themselves for context and change over all timescales [50].

One of the key objectives of smart buildings is to maximize occupant comfort while minimizing energy consumption [92]. Buildings and urban spaces increasingly incorporate artificial intelligence and new forms of interactivity, raising a wide span of research questions about the future of human experiences with, and within, built environments. This emerging area is defined as Human-Building Interaction (HBI) [38]. HBI affects human well-being and the surrounding environment. Although energy and technology are placed into buildings by people for their own comfort and recreation, often the decision-making process excludes the human factor from the design itself. Smart buildings provide an opportunity to consider the physical, physiological, and, psychological needs of occupants in the building design. Human-centered design (HCD) places humans at the center of the building design, which can accelerate smart housing design for people. To maximize human well-being and the optimize performance of smart buildings, an iterative, human-centered approach to building design must be employed.

Agee *et al.* [35] introduced a human-centered approach to smart housing, leading to the development of data-driven smart housing personas that communicate smart housing user needs. In their work, the authors utilized descriptive statistics and behavioral analysis to describe the physical, physiological, and psychological needs of the occupants. Personas help designers anchor their work in a fictional user's needs in the design of products and systems [115]. Their proposed approach can reduce the risk of designers designing for themselves.

Developing building occupant personas benefits both the occupants and building designers in many ways. A user-centric building design can help understand the needs, behaviors, and preferences of the potential occupants, building designers can create spaces that are tailored to the users' needs, leading to a more user-centered design approach. By designing buildings

with the occupant persona in mind, the building can provide a more pleasant and satisfying experience for the occupants, leading to higher levels of user satisfaction and productivity.

Designing buildings that meet the needs of the occupant persona can lead to improved building performance, such as energy efficiency and indoor air quality, as the building is designed to meet the specific needs of its occupants. It can also reduce the need for retrofits and renovations, resulting in cost savings over the building’s lifecycle. Understanding the needs and preferences of potential occupants can help building owners and developers to better market and lease the building, leading to higher occupancy rates and increased revenue. However, the task faces multiple challenges beginning with the limited availability of building occupant data. The lack of data related to occupants and their living environment is one of the major pullbacks in smart housing persona development research. Even with the limited data that is available, the process of developing a building occupant persona is manual and time-consuming. Currently, the occupant persona development process requires researchers to manually analyze data, conduct interviews, cluster the occupant profiles based on multiple criteria, and finally construct a persona [35]. The entire process takes much time because it involves manual labor in all the steps. Automating a single step of this procedure can accelerate the entire persona development task. Moreover, although there is a scarcity of residential building data, there has been some work to address this issue recently. For example, Anik *et al.* [43] presented a cost-effective and portable framework for indoor data collection, and Song, *et al.* [109] discussed data collection and analysis methods for analyzing human comfort in the indoor environment. With more building data available, and the development of machine learning technologies, new opportunities are emerging. Machine learning is an automated process that extracts patterns from data [72]. In the field of predictive data analytics, machine learning is a method used to devise complex prediction algorithms and models [91]. Machine learning models that can deliver fast and accurate results given

appropriate data are provided [97].

In this work, we apply machine learning to the 2015 Residential Energy Consumption Survey Data for building occupant characteristics classification and prediction which can lead to automating some steps of building occupant persona development. The procedure includes data processing, feature engineering, selection of target variables, training, and evaluating the models. This work aims to answer the following research questions:

- RQ1: How effectively can machine learning tools predict individual building occupant characteristics?
- RQ2: How do machine learning algorithms compare with each other when predicting building occupant characteristics?

The rest of the chapter is structured as follows: Section 4.2 presents the methodology, description of the data, and machine learning models used. Section 4.3 demonstrates the machine learning evaluation details. Section 4.4 discusses the development of building occupant persona using the findings of this work, the limitations, and, the future research directions. Finally, section 4.5 concludes the research.

4.2 Methodology

Developing occupant characteristics persona is a difficult and time-consuming task [35]. Using the latest machine learning tools can help alleviate the process. However, there are some challenges to using machine learning tools in the development of building occupant persona. This section describes the challenges and the methodology followed by this work to overcome these challenges in using state-of-the-art machine learning models in occupant characteristics classification and prediction.

4.2.1 Challenges of using machine learning tools in building occupant persona development process

This work aimed towards using state-of-the-art machine learning models in the development of building occupant personas. Along the path, the hurdles in the task were identified. The following are the key challenges of using machine learning models in building occupant persona development:

- (i) Acquiring the necessary data: Obtaining a dataset that accurately represents the target population can be challenging and time-consuming. Collecting data on a variety of factors such as demographics, lifestyle, and behavior can be costly and may require specialized knowledge.
- (ii) Identifying relevant variables: Determining which variables to include in the model can be difficult due to the large number of potential variables that may be relevant. This requires a deep understanding of the target population and the factors that influence their behavior.
- (iii) Choosing the appropriate model: Selecting which ML models to use: There are many different machine learning models that can be used for occupant persona development, each with its own strengths and weaknesses. Selecting the most appropriate model for a given problem requires a thorough understanding of the data and the problem at hand. Additionally, different models may require different levels of computational resources and expertise to implement, which can affect the feasibility of using a particular model.
- (iv) Fine-tuning hyperparameters of models for individual target variables: Finally, even after selecting an appropriate machine learning model, there is a significant amount of tuning that must be done to ensure that the model is effective at predicting the target variables. This involves adjusting the hyper-parameters of the model to optimize performance, which can be time-consuming and require significant computational resources. Moreover, hyperparameter

tuning may need to be done separately for each target variable, which further increases the complexity of the problem.

(v) Interpreting the results: Interpreting the results of a machine learning model can be challenging, especially when dealing with complex models. Understanding the factors that contribute to the model’s predictions can be difficult, which may limit the usefulness of the model for developing occupant personas.

(vi) Data privacy concerns: Collecting and storing data on individuals raises privacy concerns. Care must be taken to ensure that data is collected and used in compliance with relevant privacy laws and regulations.

4.2.2 Methodology Overview

Figure 4.1 illustrates the framework of this research, which is inspired by a study conducted by Zhongguo *et al.* [130]. The process begins with processing the 2015 Residential Energy Consumption Dataset [34]. The metadata is extracted, and the data is cleaned in the pre-processing step. Then, 16 target variables are chosen to represent occupant characteristics, such as age, education, income, thermal preference, etc. The rest of the attributes remain as descriptive variables and are filtered through the feature selection step, in which the irrelevant and redundant variables are dropped. Then, six machine learning models for classification prediction tasks are selected and tested. These models are trained through the 10-fold cross-validation and the evaluated results are presented here. The following sections describe the different components of the methodology of this work.

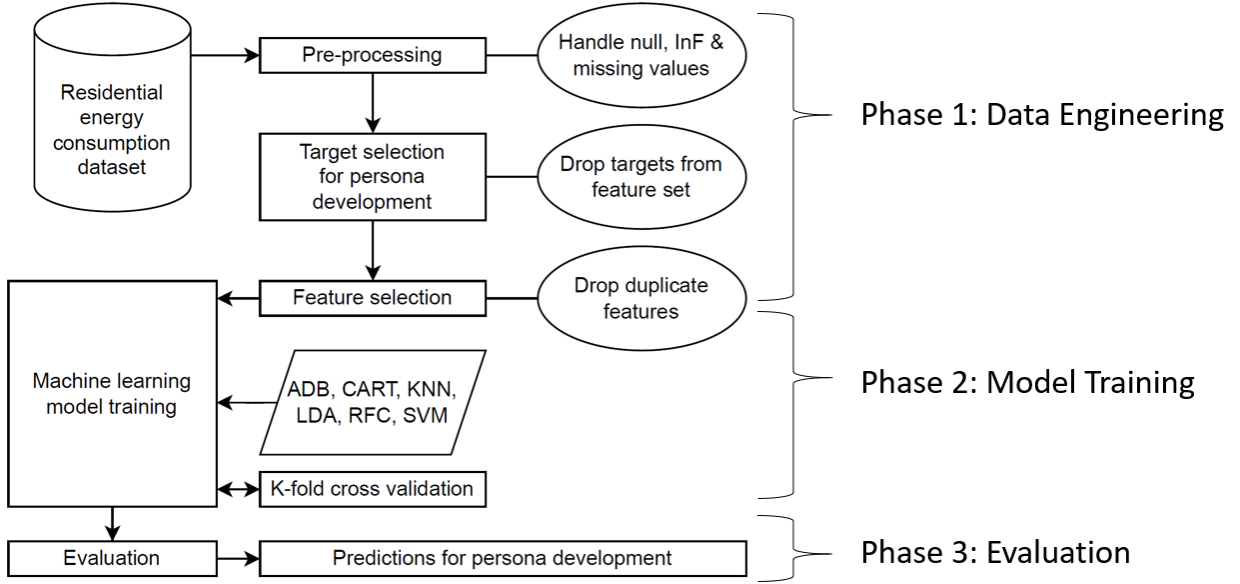


Figure 4.1: Our research methodology involving data curation, ML model training, and ML-based prediction for persona development

4.2.3 Dataset description

The U.S. Energy Information Administration (EIA) conducts the Residential Energy Consumption Survey (RECS) periodically to gather detailed information on energy usage in American households [34]. RECS is a comprehensive project that includes a household survey, data collection from energy suppliers, and estimation of end-use consumption and expenditures. The survey gathers data on energy-related features and usage patterns from a nationally representative sample of housing units. The energy supplier survey collects data on the consumption of electricity, natural gas, propane/LPG, fuel oil, and kerosene in the sampled housing units during the reference year. RECS focuses on homes used as primary residences and excludes secondary homes, vacant units, military barracks, and common areas in apartment buildings. Consequently, RECS estimates are not meant to represent total sector-level figures but are particularly useful for comparing different characteristics of homes within the residential sector.

The total number of responding households is 5686 in the 2015 RECS. Each record includes a total of 759 attributes. These attributes have been categorized in 12 sections by EIA:

- A: Structural Characteristics – house type, construction time, renovation time, floor plan, building materials, etc.
- B: Kitchen Appliances – kitchen stove, microwave, stove fuels, oven, hood, ventilation, grill, refrigerator, freezer, dishwasher, etc.
- C: Home Appliances and Electronics – clothes washer, dryer, washing cycle, television, gaming console, phone, computer, etc.
- D: Space Heating – heating appliances, heater usage during winter, heating fuel, thermostat controls, hot water system, fireplace, etc.
- E: Air Conditioning – air conditioners, cooling systems, air conditioning heat pump, air filter, programmable thermostat, thermostat control during summer days, etc.
- F: Water Heating – water heaters types, fuels, usage, age, tank, etc.
- G: Miscellaneous – light bulbs, swimming pool, energy audit, energy assistance, etc.
- H: Fuels Used – back-up generator, onsite power system details, fuel usage in home, energy bills, natural gas usage in home, etc.
- I: Housing Unit Measurement – size, shape and area of the different floors of the housing units.
- J: Fuel Bills – fuel suppliers, bill types, different fuel bills like electricity, gas, wood, smart meter, etc.

- K: Housing Unit Characteristics – occupants’ age, gender, education, employment status, number of people living in the household, income, etc.
- L: Energy Insecurity and Assistance – challenges paying bills, struggles due to unsafe or unhealthy temperature, utility discontinuation, energy assistance, requirement of medication, etc.

Figure 4.2 illustrates the number of features in each of the 12 categories in pairs. For every pair, the left bar (colored in blue) refers to the original number of features in that category, and, the right bar (colored in orange) refers to the number of features selected after the feature selection process. 16 of these attributes are chosen to be the target variables because they relate to individual occupant characteristics. The rest of the attributes remain as input variables. The next section discusses the feature selection process in detail. In the training process, the k-fold cross-validation is utilized with $k = 10$ dividing the 5686 records into 10 random folds where 9 folds (5118 data points) were used for training and the remaining 1 fold of data (568 data points) was used for testing. The process is repeated 10 times to ensure the validity of the evaluation results.

4.2.4 Pre-processing

Data pre-processing is the process of transforming raw data into a machine-understandable format. It is a critical step in any machine-learning pipeline as it can have a significant impact on the performance of the model. In many cases, the raw data used for machine learning may be incomplete, noisy, or contain errors. Sometimes, such noisy data make the model unusable. Pre-processing can help to clean and correct these issues, which can improve the accuracy of the model. This work utilizes the data from the RECS 2015 dataset where most of the columns are ready to be directly fed to the machine learning models.

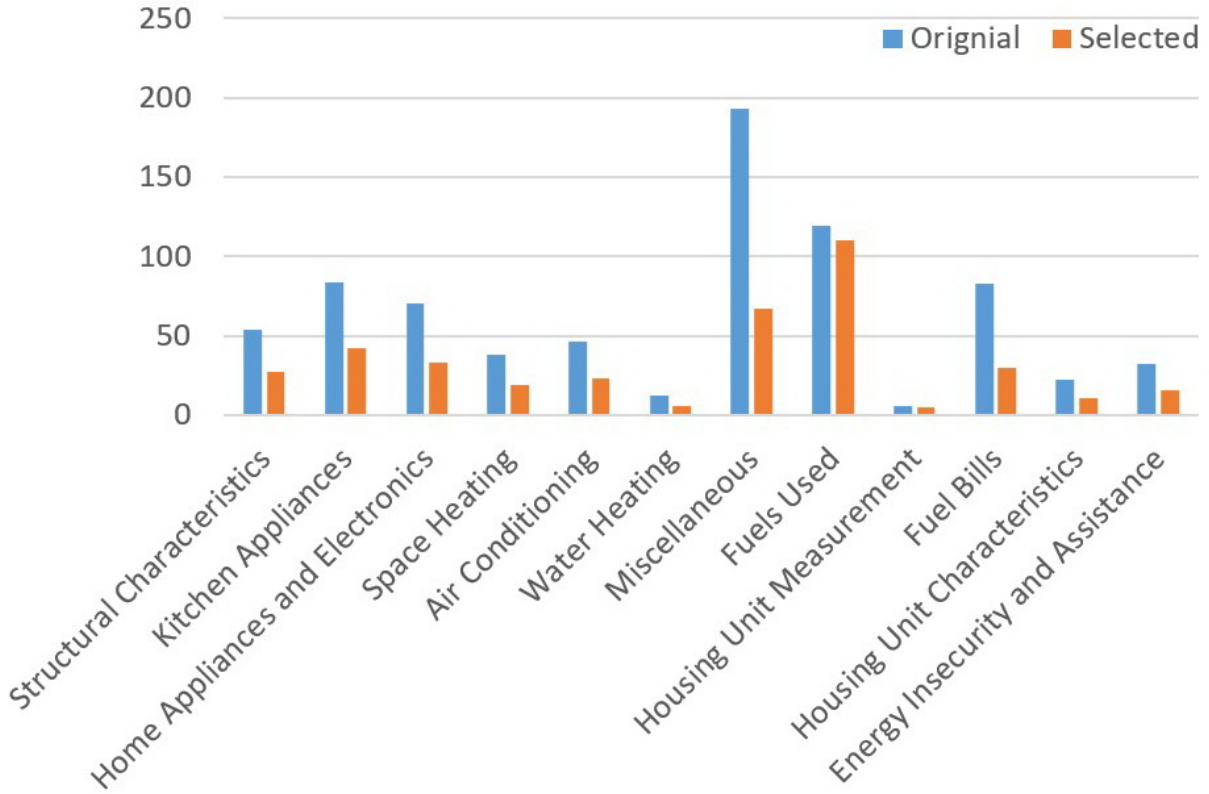


Figure 4.2: Feature distribution in categories.

However, there are some columns that contain infinite, missing, blank, and, null values. For machine learning models to understand all records, these values needed to be transformed. The following are the different pre-processing steps done in this work:

4.2.4.1 Missing, null, InF values

Some of the data records contained missing or blank values. Some records had null values. These missing, blank, and, null values have been replaced with -1 to establish the value is missing. Some rows included infinite values written as 'inf' instead of a number. These records are particularly difficult to process by the machine learning models because the model expects all values of a particular column to be of the same data type. Having both

numbers and text or string on the same column provides a mixed signal to the machine learning pipeline. The infinite values have been replaced with a large enough number, i.e. the largest integer in the programming language Python.

4.2.4.2 Age classification

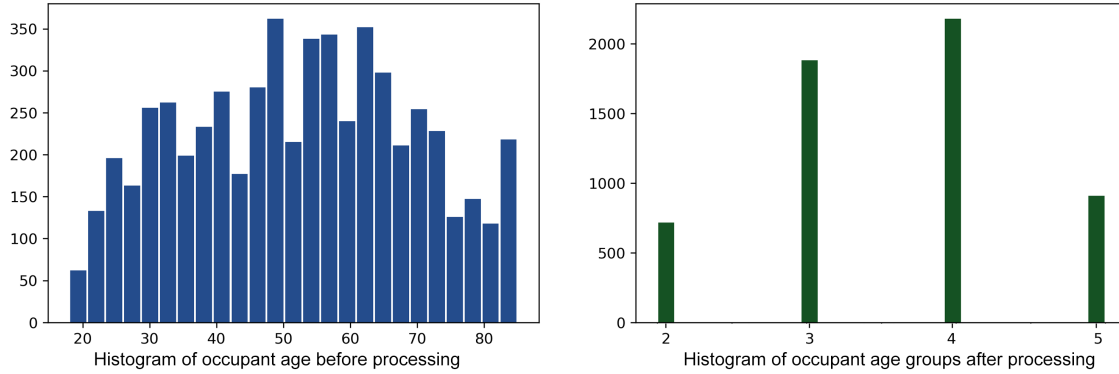


Figure 4.3: Histogram of occupant age data before (left) and after (right) pre-processing step.

The target variable, HHAGE ((SL. 9 on Table 4.1)) which refers to the age of the responding occupant, is continuous from 18 to 110. Previous studies [81, 126] used age groups instead of continuous age numbers in years which is more feasible in machine learning classification tasks. This study follows the age range used in [81]. The records are categorized in the following age groups (in years): Children (0 to 12), Young Adult (13 to 30), Middle Adult (31 to 50), Senior Adult (51 to 70), and Senior (71 to 110). Figure 4.3 illustrates the histogram of the occupant age data before (left) and after (right) the pre-processing step. On the left, the data is distributed across 18 to 110 years. On the right, there are only 4 bars representing each age group. As the occupant age data starts from 18 years, there are no records in the Children age group (0 to 12 years).

4.2.4.3 Temperature classification

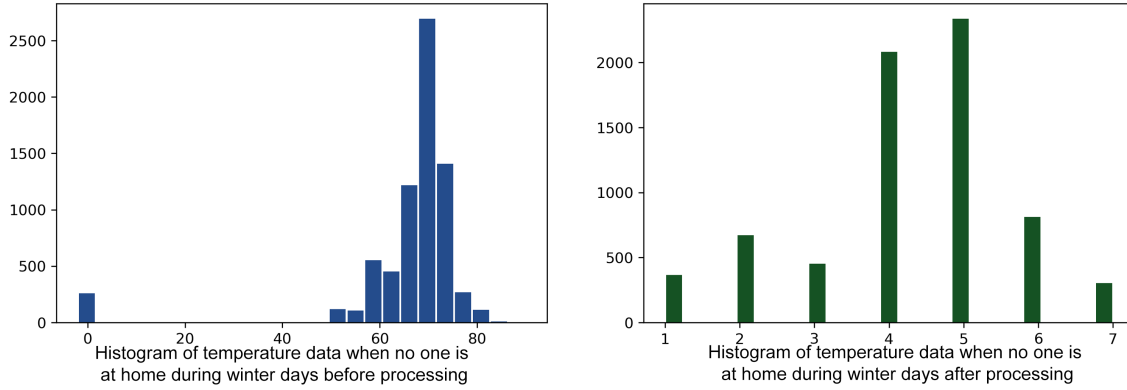


Figure 4.4: Histogram of TEMPGONE before (left) and after (right) pre-processing step.

The temperature attributes (SL. 3 to 8 on Table 4.1) are in a continuous form representing temperature records in degree Fahrenheit (°F). Similar to the age group, it is preferable to use regression models for continuous numbers. In the case of a classification task, it is better to have groups or classes of numbers. In the RECS 2015 dataset, the temperature data in all 6 target columns range from 40 to 96 °F. As an example of the data distribution of the temperature records, the left side of Figure 4.4 illustrates the histogram of the attribute TEMPGONE which represent the temperature data when no one is at home during the winter days. The other temperature attributes demonstrate similar characteristics. Here, it can be observed that most of the data fall between 50 to 80 °F. However, there is a peak on 0 °F. This portion represents missing or null data. The continuous temperature data is then distributed in 7 groups. Details of these groups can be found on SL. 3 of Table 4.1. All temperature attributes have been categorized following the same distribution. This categorization is more feasible for machine learning classification tasks. In this way, machine learning models are used to predict a temperature group for a particular input instead of identifying the exact temperature value. On the right side of Figure 4.4 the histogram after the processing of TEMPGONE attribute is shown. Here, it can be seen that most of the

data lie in groups 4 and 5 which represent 65 to 68°F and 69 to 72°F respectively.

4.2.5 Feature selection

Feature selection is an important step in the performance of effective machine learning models. It involves selecting a subset of the most relevant features (or variables) from a larger set of available features that are likely to have the most predictive power for the target variable. This process can lead to better model performance, faster training times, avoiding over-fitting, and, more interpretable models.

The RECS 2015 dataset includes a total of 759 columns each representing an attribute. The authors went through a manual inspection of each of the features to identify the features that are important in the occupant characteristics prediction and the features that can be removed to reduce the total number of features to feed the machine learning models. A total of 370 attributes were dropped through the manual feature selection process leaving 389 attributes for the target and training features.

The RECS 2015 dataset included imputation flags for most of the attributes which refer to whether a record was imputed or not. A record can be imputed if it is not directly obtained from the occupants, but it is observed from other sources such as the structure, surrounding, or measuring device. Either way, in this work, the imputation flag is not a necessary component. Hence, the imputation flags are dropped. There were a total of 217 imputation flag attributes.

The energy consumption attributes recorded in the dataset contained both energy units and corresponding costs in US dollars. The dollar amounts are not in the scope of this study and thus are dropped from the dataset. There were a total of 53 dollar amount attributes. The data also included replicated weights which are used for variance estimation. These

attributes were also not in the scope of this study and thus a total of 97 replicated weight attributes were dropped from the dataset.

Each record in the dataset was given an ID represented in the first column named DOEID. The ID is unique to each record and unnecessary in the classification task. It was dropped in the selection process. Finally, there were 2 attributes representing the number of phones and cell phones. These 2 columns have a direct correlation with the number of occupants living in the household which is one of the characteristics this work aims to estimate. Thus these 2 attributes were also dropped from the dataset.

Each of the 759 attributes was manually examined in this study and a total of 370 features were dropped in the manual feature selection process which includes imputation flags, utility bills in US dollar amount (actual unit is kept), and replicated weights (used for variance estimation), ID, and, phone counts. These are safe to drop because they do not represent any data that might affect the output of the machine-learning models for developing building occupant personas. The orange bars in Figure 4.2 (right bar of each category pair) represent the number of features selected for the training models. The remaining 389 attributes include features from all the categories of the initial dataset. After cautious examination of these 389 features, 16 target features were selected. More details on the target features are provided in the next section. Leaving the target features aside, there remains a total of 373 attributes which were kept as the input variables of the machine-learning models. This study utilized all of the remaining 373 features as inputs of the machine-learning models.

Table 4.1: Target attribute details.

SL.	Attribute Name	Description	Data Range	# of classes	Class Details	
1	EQUIPMUSE	Main heating equipment control description	1 to 5, & 9	6	1: Set one temperature and leave it there most of the time 2: Manually adjust the temperature at night or when no one is at home 3: Program the thermostat to automatically adjust the temperature during the day and night at certain times 4: Turn equipment on or off as needed 5: Our household does not have control over the equipment 9: Other	
2	USEWWAC	Main cooling equipment control description	1 to 5, & 9	6	Same as 1 (EQUIPMUSE)	
3	TEMPHOME	Typical temperature when someone is at home during the winter days	40 to 96 °F	7	1: 40 to 50 °F 2: 51 to 60 °F 3: 61 to 64 °F 4: 65 to 68 °F	5: 69 to 72 °F 6: 73 to 76 °F 7: 77 to 96 °F
4	TEMPGONE	Typical temperature when no one is at home during the winter days	40 to 96 °F	7	Same as 3	
5	TEMPNITE	Typical temperature when someone is at home during the winter nights	40 to 96 °F	7	Same as 3 (TEMPHOME)	
6	TEMPHOMEAC	Typical temperature when someone is at home during the summer days	40 to 96 °F	7	Same as 3 (TEMPHOME)	
7	TEMPGONEAC	Typical temperature when no one is at home during the summer days	40 to 96 °F	7	Same as 3 (TEMPHOME)	
8	TEMPNITEAC	Typical temperature when someone is at home during the summer nights	40 to 96 °F	7	Same as 3 (TEMPHOME)	
9	HHAGE	Age of the respondent	18 to 110 years	5	1: Children (0 to 12) 2: Young Adult (13 to 30) 3: Middle Adult (31 to 50)	4: Senior Adult(51 to 70) 5: Senior (71 to 110)
10	EMPLOYHH	Employment Status of the respondent	1 to 4	4	1: Employed full-time 2: Employed part-time	3: Retired 4: Not employed
11	EDUCATION	Highest degree or level of school completed by the respondent	1 to 5	5	1: Less than high school diploma or GED 2: High school diploma or GED 3: Some college or Associate's degree 4: Bachelor's degree (e.g.: BA, BS) 5: Master's or higher degree (e.g.: MS, MBA, MD, PhD)	
12	NHSLDMEM	Number of people living in the home	1 to 20	20	Each number of people is converted into a class itself	
13	NUMADULT	Number of adults living in the home	1 to 20	20	Each number of people is converted into a class itself	
14	NUMCHILD	Number of children living in the home	1 to 20	20	Each number of people is converted into a class itself	
15	ATHOME	Number of weekdays someone is at home	0 to 5	6	0: None 1: 1 day 2: 2 days	3: 3 days 4: 4 days 5: 5 days
16	MONEYPY	Combined household income (USD)	1 to 16	16	1: Less than 5,000 2: 5,000 - 7,499 3: 7,500 - 9,999 4: 10,000 - 12,499 5: 12,500 - 14,999 6: 15,000 - 19,999 7: 20,000 - 24,999 8: 25,000 - 29,999	9: 30,000 - 34,999 10: 35,000 - 39,999 11: 40,000 - 49,999 12: 50,000 - 59,999 13: 60,000 - 74,999 14: 75,000 - 99,999 15: 100,000 - 149,999 16: 150,000 or more

4.2.6 Target variables

The authors examined the remaining 389 attributes after the feature selection step to select the target variables. A total of 16 attributes were found that actively relate to occupant characteristics. These attributes can provide key information in developing building occupant persona. Table 4.1 provides a detailed overview of these 16 target attributes. The first column (SL.) is a regular serial number of these variables, the second column (Attribute Name) denotes the respective target variable, the third column (Description) provides a description of the attribute, the fourth column (Data Range) represents the range span of the data of the respective attribute, the fifth column (# of classes) denotes the total number of classes present in that attribute, and, the last column (Class Details) provides detail information of the classes present in the particular target attribute. The following are the 16 target variables selected for this study:

- EQUIPMUSE: Main heating equipment household behavior, including values such as setting one temperature and leaving it there most of the time; manually adjusting the temperature at night or when no one is at home; programming the thermostat to automatically adjust the temperature during the day and night at certain times; turn equipment on or off as needed, etc. The datapoints are distributed in 6 classes. The class details can be found on the first row of table 4.1.
- USEWWAC: Most-used individual air conditioning unit household behavior including values, such as setting one temperature and leaving it there most of the time; manually adjusting the temperature at night or when no one is at home; programming the thermostat to automatically adjust the temperature during the day and night at certain times; turn equipment on or off as needed, etc. Similar to EQUIPMUSE, the data points are categorized in 6 classes.

- TEMPHOME: Winter temperature when someone is at home during the day. The values range from 50 to 90 degrees Fahrenheit. The temperature attributes have been processed in the pre-processing step. The processed data falls into 7 classes. The class details can be found on row 3 of table [4.1](#).
- TEMPGONE: Winter temperature when no one is at home during the day with the value range of 50 to 90 degrees Fahrenheit. It was also processed in the pre-processing step to distribute the data into 7 classes. Class distribution is the same as TEMPHOME.
- TEMPNITE: Winter temperature at night with the value range of 50 to 90 degrees Fahrenheit. A similar processing step has been followed here with 7 classes.
- TEMPHOMEAC: Summer temperature when someone is at home during the day with the value range of 50 to 90 degrees Fahrenheit. The same temperature processing was conducted in this attribute and the data was distributed into 7 classes.
- TEMPGONEAC: Summer temperature when no one is at home during the day with the value range of 50 to 90 degrees Fahrenheit. Similar to the other temperature attributes, this one was also processed and the data is classified into 7 classes.
- TEMPNITEAC: Summer temperature at night with the value range of 50 to 90 degrees Fahrenheit. A similar processing step has been followed here with 7 classes.
- HHAGE: Respondent age, values ranging from 18 to 110. This attribute was processed in the pre-processing step and the data has been classified into 5 classes, from children to seniors. The class details are provided in table [4.1](#).
- EMPLOYHH: Respondent employment status. Values covering employed full-time, part-time and unemployed or retired. Data is classified into 4 classes, starting from 1

to 4.

- **EDUCATION:** Highest education completed by the respondent. Replies cover less than high school diploma or GED, high school diploma or GED, some college or associate's degree, bachelor's degree (for example: BA, BS), master's, professional, or doctorate degree (for example: MA, MS, MBA, MD, JD, PhD). Data is distributed in 5 classes ranging from 1 to 5. Detailed class distribution can be found on row 11 of table 4.1.
- **NHSLDMEM:** Number of household members, values ranging from 1 to 20. Each occupant number was automatically considered as a class by the machine learning models thus making the total number of classes to 20 for this attribute.
- **NUMADULT:** Number of household members age 18 or older, values ranging from 1 to 20. Again, each occupant number was automatically considered as a class by the machine learning models thus making the total number of classes 20 for this attribute.
- **NUMCHILD:** Number of household members age 17 or younger, values ranging from 1 to 20. Similar to the other attribute featuring occupant number, this attribute also contains 20 classes each representing a number between 1 to 20.
- **ATHOME:** Number of weekdays someone is at home. The values range from 0 to 5, creating a total of 6 classes.
- **MONEYPPY:** Annual gross household income for the previous year. The data is distributed in 16 classes ranging from less than 5,000 USD to more than 150,000 USD. Detailed class distribution can be found on row 16 of table 4.1.

These target attributes were chosen to describe different occupant characteristics. Together, these attributes can provide information on the age, lifestyle, behavior, family size, income, education, and, thermal comfortability of the occupants. The goal of this study is to see if we

can predict these occupant characteristics from the selected 373 input features that contain respective information on energy usage, housing type, kitchen appliances, electronics usage, heating, and, cooling elements. If the models can successfully predict the target occupant characteristics from these selected input features, it can benefit in identifying the needs of particular residents from the data of their current household.

4.2.7 Machine learning models

Classification is a supervised learning approach in which a target variable is categorical or discrete. The task of choosing a specific classification model is a critical step [75], and each model has its own strengths and weaknesses in a given scenario. There is no cut-and-dried flowchart that can be used to determine which model should be used or will outperform the rest. A simple example can be, back propagation neural networks achieve higher accuracy than the decision tree method on Iris and Appendicitis data but a lower accuracy on Breast cancer and Thyroid data [123, 130]. Pedregosa *et al.* [99] suggests a guide on choosing the proper machine learning algorithm for a given task. In this work, six machine learning models have been used parallelly on the library recommended settings [99]. The following are the classification models used in this work:

4.2.7.1 Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA) is a supervised learning algorithm used for classification tasks [99]. It is a technique used to find a linear combination of features that best separates two or more classes of objects or events. It can be used to classify data into different categories based on their features. It is a linear model for classification and is most commonly used for feature extraction in pattern classification problems. It is used in finding the projection

hyper-plane that minimizes the inter-class variance and maximizes the distance between the projected means of the classes [124].

In LDA, it is assumed that the classes have the same covariance matrix and that the data is normally distributed. The goal is to find a projection of the data onto a lower-dimensional space, that is, a subspace such that the different classes are well separated and the intra-class variance is minimized. This projection is obtained by finding the eigenvectors of the between-class scatter matrix and the within-class scatter matrix. The between-class scatter matrix measures the variation between different classes, while the within-class scatter matrix measures the variation within each class. The eigenvectors of the between-class scatter matrix and the within-class scatter matrix are used to construct a transformation matrix that maps the data to a new subspace. In the new subspace, each sample is represented by a new set of features, which are called discriminant functions. These functions can be used to classify new data points by assigning them to the class with the highest score. This study used the LDA model with its default parameters [99]. The solver parameter was set to “svd” indicating Singular value decomposition (default). This is recommended for data with a large number of features which is the case of this study.

4.2.7.2 K Nearest Neighbors Classifier (KNN)

K Nearest Neighbors Classifier (KNN) is also a supervised machine learning algorithm used for classification tasks [99]. It is a non-parametric method, which means it does not make any assumptions about the underlying distribution of the data. Instead, it uses the nearest neighbors of a new data point to predict its class. The k-NN algorithm is among the simplest and yet most efficient classification rules and is widely used in practice [78].

The KNN algorithm works by finding the K training examples that are closest to a new

test example in terms of some distance metric, for example, Euclidean distance, and, cosine distance. The value of K is a hyperparameter that differs in different tasks. Once the K nearest neighbors are identified, the algorithm assigns the new test example to the class that is most common among its K nearest neighbors. One of the strengths of KNN is that it can handle non-linear decision boundaries and can be applied to both binary and multi-class classification problems which is the case of this study. In this study, the parameter `n_neighbors` (also referred to as K) was set to 5. This is also the default recommended value of the parameter.

4.2.7.3 Decision Tree Classifier (DT)

Decision Tree Classifier (DT) is another supervised learning algorithm used for classification and regression tasks. It is a tree-based model that works by recursively partitioning the feature space into regions, each of which is associated with a class label or a continuous target value [112]. The DT algorithm builds a binary tree in which each internal node corresponds to a decision rule based on the value of a single feature. The decision rule splits the data into two or more subsets based on a threshold value or a set of categorical values. Each leaf node of the tree corresponds to a predicted class or target value. To build a decision tree, the DT algorithm uses a greedy approach that maximizes the information gain or the Gini impurity [127] at each split. Information gain measures the reduction in entropy (or some other measure of uncertainty) of the target variable, while Gini impurity measures the probability of misclassification if a random example is assigned to a particular class. In this work, the Gini impurity parameter was used as criteria for the split with the splitter strategy set to “best”.

The DT algorithm can be used for both binary and multi-class classification problems, as well as for regression tasks. As the tasks related to this study are multi-class classification, DT

remains a good fit. It is a simple and interpretable model that can handle both categorical and continuous features, and, can capture non-linear relationships between the features and the target variable.

4.2.7.4 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a popular supervised learning algorithm used for classification, regression, and outlier detection tasks [129]. SVM aims to find the hyperplane that best separates the examples of different classes in a high-dimensional feature space. The basic idea of SVM is to map the input data points into a high-dimensional feature space using a kernel function, and then find the hyperplane that maximizes the margin between the two closest examples of different classes. The examples that are closest to the hyperplane are called support vectors, and they determine the position and orientation of the hyperplane.

SVM can handle both linearly separable and non-linearly separable data by using different kernel functions, such as linear, polynomial, and radial basis function (RBF) kernels. The default choice of kernels, RBF is used in this study. One of the strengths of SVM is its ability to handle high-dimensional data and small sample sizes, which makes it suitable for a wide range of applications. It can also handle imbalanced datasets by adjusting the class weights or using cost-sensitive learning. Another advantage of SVM is its ability to provide a sparse solution, meaning that only a subset of the examples (i.e., the support vectors) are used to define the hyperplane, which can improve its scalability and interpretability.

4.2.7.5 AdaBoost Classifier (ADB)

AdaBoost Classifier (ADB) is a machine learning algorithm that belongs to the family of ensemble methods. It is a meta-algorithm that combines multiple weak classifiers to form

a strong classifier. The basic idea behind AdaBoost is to iteratively train a series of weak classifiers on weighted versions of the training data, and then combine their predictions to obtain the final classification [64, 105]. At each iteration, AdaBoost adjusts the weights of the training examples to give more importance to the misclassified examples. The weak classifier is then trained on the weighted data and added to the ensemble with a weight that depends on its classification accuracy. The weights of the examples are then updated again based on the classification errors of the current ensemble, and the process is repeated for a predefined number of iterations or until a desired accuracy is achieved. The base estimator parameter was set to Decision Tree Classifier with max depth = 1 which is the default. The number of estimators parameter was set to 50 by default.

One of the strengths of AdaBoost is its ability to improve the performance of weak classifiers and handle complex decision boundaries. It can also handle imbalanced datasets and noisy data, by adjusting the weights of the examples and focusing on the difficult examples. Another advantage of AdaBoost is its simplicity and interpretability, as it combines simple and easily understandable models to form a more complex and accurate model.

4.2.7.6 Random Forest Classifier (RFC)

Random Forest Classifier (RFC) is a machine learning algorithm that belongs to the family of ensemble methods. It is a meta-algorithm that combines multiple decision trees to form a powerful and robust classifier. The basic idea behind Random Forest is to build a collection of decision trees on randomly selected subsets of the training data, and then combine their predictions to obtain the final classification [98]. At each iteration, Random Forest randomly selects a subset of the features and a subset of the training features, and, trains a decision tree on the selected data. The trees are grown to maximum depth and may be pruned later to improve their generalization performance. The predictions of the individual trees are then

combined using a majority vote or weighted vote, depending on the specific implementation. In this study, the number of estimators in RFC which denotes the number of trees in the forest was set to 100. The random state parameter was set to 0 which controls both the randomness of the bootstrapping of the samples used when building trees and the sampling of the features to consider when looking for the best split at each node.

One of the strengths of Random Forest is its ability to handle complex decision boundaries and nonlinear relationships between the features and the target variable. It can also handle missing data, outliers, and irrelevant features, by randomly selecting subsets of the features. Another advantage of Random Forest is its ability to provide estimates of the importance of the features, which can help in feature selection and understanding the underlying data. It can also be easily parallelized and scaled to large datasets and distributed computing environments.

4.3 Evaluation

The evaluation of the machine learning models plays a crucial role in assessing the performance of the classification models. Employing appropriate evaluation settings and metrics, analyzing performance scores, and, identifying key findings can draw meaningful conclusions about the model's effectiveness and its relevance to the research objectives. The evaluation section forms an essential component of the study, contributing to the overall understanding and interpretation of the classification model's performance. This section provides insights into the model's performance by employing evaluation settings and metrics. It also discusses the obtained performance scores, key findings, and, the answer to the research question posed in this study.

4.3.1 Experiment settings

Machine learning classification models are evaluated using various settings and metrics to assess their performance and effectiveness. In this study, 6 machine learning models were evaluated in a 10-fold cross-validation setting. The total number of records present in the dataset is 5686. During the process, 5118 records were used in the training process and 586 records were used for the evaluation at each of the 10-folds. At every iteration, the data were randomly shuffled.

4.3.2 Evaluation metrics

The models were evaluated with the most common classification evaluation metrics used by scholars [68]. The evaluation metrics used in this study are accuracy (A), precision (P), recall (R), and, f1-score (F). Classification accuracy is a performance metric used to measure the accuracy of a machine learning model in predicting categorical outcomes [118]. It is the proportion of correct predictions made by the model out of all the predictions made. Classification accuracy of the developed models on unseen data is one of the principal metrics for evaluating classification models. The following formula has been used to determine classification accuracy:

$$\text{Accuracy (A)} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \quad (4.1)$$

In machine learning classification tasks, precision is another important performance metric [118] that measures the ability of a classification model to identify only the relevant instances of a particular class in a particular dataset. More specifically, precision is the proportion of true positives (i.e., instances that are correctly classified as belonging to a certain class) out

of all the instances that the model classified as belonging to that class (i.e., true positives plus false positives). In mathematical terms, it can be expressed as:

$$\text{Precision (P)} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \quad (4.2)$$

Precision is an important metric in many real-world applications, especially to avoid falsely labeling instances as belonging to a certain class. It is generally used in conjunction with recall to get a better understanding of the model's performance. Recall is another important performance metric in machine learning classification tasks [118]. It measures the ability of a classification model to identify all relevant instances of a particular class in the dataset. More specifically, recall is the proportion of true positives (i.e., instances that belong to a certain class and are correctly classified as such) out of all the instances that actually belong to that class (i.e., true positives plus false negatives). In mathematical terms, it can be expressed as:

$$\text{Recall (R)} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \quad (4.3)$$

F1-score is another performance metric that combines both precision and recall into a single metric. It provides a way to balance the two metrics and evaluate the overall performance of a classification model. The F1-score is calculated as the harmonic mean of precision and recall, giving equal weight to both measures. The harmonic mean balances the impact of precision and recall, ensuring that the F1-score remains high only if both precision and recall are high. The F1-score is calculated as:

$$\text{F1-score (F)} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (4.4)$$

4.3.3 Model Performances

Table 4.2: Accuracy (A), F1-score (F), Precision (P), and, Recall (R) obtained across the target variables

Metrics	A		F		P		R	
Target	Min	Max	Min	Max	Min	Max	Min	Max
ATHOME	0.41	0.60	0.38	0.51	0.31	0.57	0.40	0.60
EDUCATION	0.25	0.45	0.17	0.41	0.11	0.48	0.25	0.45
EMPLOYHH	0.46	0.75	0.31	0.72	0.23	0.76	0.46	0.75
EQUIPMUSE	0.33	0.69	0.21	0.68	0.14	0.69	0.33	0.69
HHAGE	0.38	0.65	0.21	0.64	0.15	0.68	0.38	0.65
MONEYPY	0.19	0.37	0.08	0.35	0.05	0.35	0.19	0.37
NHSLDMEM	0.37	1.00	0.20	1.00	0.13	1.00	0.37	1.00
NUMADULT	0.54	1.00	0.38	1.00	0.30	1.00	0.54	1.00
NUMCHILD	0.65	1.00	0.54	1.00	0.46	1.00	0.65	1.00
TEMPGONE	0.30	0.61	0.20	0.59	0.13	0.59	0.30	0.61
TEMPGONEAC	0.29	0.67	0.15	0.68	0.10	0.69	0.29	0.68
TEMPHOME	0.33	0.65	0.24	0.63	0.17	0.65	0.33	0.65
TEMPHOMEAC	0.29	0.69	0.13	0.68	0.09	0.70	0.29	0.69
TEMPNITE	0.31	0.62	0.18	0.61	0.12	0.64	0.31	0.62
TEMPNITEAC	0.28	0.66	0.15	0.65	0.10	0.67	0.28	0.66
USEWWAC	0.71	0.89	0.64	0.87	0.57	0.87	0.71	0.89

Table 4.2 shows the combined minimum and maximum Accuracy (A), F1-score (F), Precision (P), and, Recall (R) obtained by the machine learning models over 10-fold cross-validation process for the 16 target variables. The target features are placed in the left-most column followed by the minimum and maximum results obtained in all 4 evaluation metrics by the classification models. These results span over all 10-folds and 6 machine learning models, thus a larger gap between the minimum and maximum values can be observed. The weighted average has been used in the calculation of precision, recall, and, F1-score as this study deals with multi-class classification tasks. Precision measures the accuracy of positive predictions, while recall measures the completeness of positive predictions. High precision and high recall are desirable, but there may be a trade-off between the two metrics in some cases. Precision evaluates the correctness of positive predictions, whereas recall assesses the inclusiveness

Table 4.3: Average accuracy of the machine learning models over 10-fold validation process

Models	ADB	DT	KNN	LDA	RFC	SVM	Mean
ATHOME	0.58	0.43	0.43	0.54	0.57	0.55	0.52
EDUCATION	0.38	0.3	0.28	0.40	0.40	0.33	0.35
EMPLOYHH	0.71	0.61	0.48	0.70	0.71	0.48	0.62
EQUIPMUSE	0.55	0.61	0.37	0.60	0.66	0.38	0.53
HHAGE	0.59	0.52	0.40	0.59	0.61	0.38	0.52
MONEYPY	0.34	0.26	0.21	0.31	0.35	0.22	0.28
NHSLDMEM	0.37	0.99	0.53	0.96	0.76	0.37	0.66
NUMADULT	0.77	0.99	0.64	0.96	0.77	0.54	0.78
NUMCHILD	0.69	0.99	0.67	0.97	0.75	0.68	0.79
TEMPGONE	0.56	0.51	0.32	0.51	0.55	0.37	0.47
TEMPGONEAC	0.47	0.63	0.31	0.58	0.64	0.31	0.49
TEMPHOME	0.55	0.60	0.37	0.55	0.63	0.41	0.52
TEMPHOMEAC	0.56	0.66	0.32	0.62	0.67	0.30	0.52
TEMPNITE	0.58	0.54	0.33	0.50	0.58	0.35	0.48
TEMPNITEAC	0.57	0.61	0.30	0.53	0.63	0.31	0.49
USEWWAC	0.86	0.85	0.74	0.83	0.88	0.76	0.82

of positive predictions. Ideally, both high precision and high recall are preferred; however, there can be instances where there is a trade-off between these two metrics. The F1-score is a measure of a model’s accuracy that considers both precision and recall. The maximum accuracy, precision, recall, and, F1-score of 100% was observed in the case of the 3 target variables indicating the number of household members (NHSLDMEM), number of adults (NUMADULT), and, number of children (NUMCHILD) in the house indicating the machine learning models performed best in classifying these characteristics. The lowest accuracy, precision, recall, and, F1-score were obtained for the gross household income (MONEYPY) which yields the machine learning models didn’t perform well in classifying this particular feature. It also indicates that the gross income of a household doesn’t rely on the selected input features of this study. Low precision of 10% was observed in classifying average room temperature during summer nights (TEMPNITEAC) and summer days when no one is at home (TEMPGONEAC). The highest precision for these features are 67% and 69% respec-

tively. This indicates that some models performed poorly in classifying these two target features however some models performed above average in classifying these characteristics as there are 7 classes of data in each of these features.

Table 4.3 reveals the average accuracy obtained by each classification model during the 10-fold cross-validation process over the 16 target variables. The targets are positioned in the left-most column, followed by the average accuracy obtained by the six classification models. The right-most column shows the mean accuracy obtained for the respective target feature. The best average score for each target is highlighted in bold numbers. Across all 16 target variables, DT, LDA, and, RFC classifiers achieved the highest average accuracy (63%). KNN and SVM achieved the lowest average accuracy (42%). DT and LDA achieved over 95% average accuracy for 3 target variables which are the number of household members (NHSLDMEM), number of adults (NUMADULT), and, number of children (NUMCHILD). Most of the models also obtained above 75% accuracy for classifying main cooling equipment control behavior (USEWWAC). The average accuracy among all the models for USEWWAC is 82% meaning the models performed moderately well in classifying this feature. The models achieved more than 50% accuracy for all target variables except 2 which are the level of education (EDUCATION) and the total household income (MONEYPY) where the average accuracy obtained by the models are 35% and 28% respectively. The data in MONEYPY is distributed in 16 classes which is almost double compared to the other targets. This might answer the lower accuracy, however, the data in EDUCATION is distributed in only 5 classes and this does not support the previous claim. A more concrete reason can be the level of education and gross household income may not depend on the energy usage and housing detail input variables. Here, education refers to the education of individual occupants (RECS survey respondents) and MONEYPY is the gross household income. It is plausible that these two characteristics are independent of the chosen input variables in this

study.

Table 4.4: Average precision (P), recall (R), and, f1-scores (F) of classification models

Models	ADB			DT			KNN			LDA			RFC			SVM		
Target	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
ATHOME	0.45	0.58	0.49	0.45	0.44	0.44	0.36	0.43	0.39	0.48	0.54	0.49	0.42	0.57	0.45	0.31	0.55	0.39
EDUCATION	0.39	0.38	0.38	0.31	0.31	0.31	0.27	0.28	0.27	0.40	0.40	0.35	0.41	0.40	0.37	0.11	0.33	0.17
EMPLOYHH	0.68	0.71	0.68	0.62	0.62	0.61	0.44	0.48	0.46	0.63	0.70	0.66	0.65	0.71	0.67	0.23	0.48	0.31
EQUIPMUSE	0.47	0.55	0.48	0.62	0.61	0.61	0.34	0.37	0.34	0.60	0.60	0.58	0.66	0.66	0.65	0.14	0.38	0.21
HHAGE	0.60	0.59	0.59	0.52	0.52	0.52	0.39	0.40	0.39	0.64	0.59	0.55	0.62	0.61	0.59	0.15	0.38	0.21
MONEYPY	0.31	0.34	0.32	0.26	0.26	0.26	0.19	0.21	0.20	0.32	0.31	0.29	0.31	0.35	0.31	0.05	0.22	0.08
NHSLDMEM	0.14	0.37	0.20	0.99	0.99	0.99	0.50	0.53	0.51	0.96	0.96	0.95	0.71	0.76	0.72	0.14	0.37	0.20
NUMADULT	0.69	0.77	0.71	0.99	0.99	0.99	0.59	0.64	0.60	0.95	0.96	0.96	0.72	0.77	0.71	0.30	0.54	0.38
NUMCHILD	0.68	0.69	0.66	0.99	0.99	0.99	0.59	0.67	0.62	0.96	0.97	0.96	0.68	0.75	0.69	0.46	0.68	0.55
TEMPGONE	0.55	0.56	0.55	0.52	0.52	0.52	0.29	0.32	0.30	0.51	0.51	0.49	0.54	0.55	0.50	0.13	0.37	0.20
TEMPGONEAC	0.40	0.47	0.38	0.63	0.63	0.63	0.31	0.31	0.31	0.56	0.58	0.53	0.62	0.64	0.61	0.10	0.31	0.15
TEMPHOME	0.53	0.55	0.52	0.60	0.59	0.59	0.33	0.37	0.34	0.56	0.55	0.52	0.62	0.63	0.60	0.17	0.41	0.24
TEMPHOMEAC	0.54	0.56	0.54	0.66	0.66	0.67	0.30	0.32	0.30	0.65	0.62	0.59	0.66	0.67	0.64	0.09	0.30	0.14
TEMPNITE	0.57	0.58	0.57	0.55	0.54	0.55	0.30	0.33	0.31	0.50	0.50	0.45	0.58	0.58	0.52	0.12	0.35	0.18
TEMPNITEAC	0.59	0.57	0.57	0.61	0.61	0.61	0.28	0.30	0.28	0.53	0.53	0.47	0.63	0.63	0.61	0.10	0.31	0.15
USEWWAC	0.80	0.86	0.82	0.85	0.85	0.85	0.62	0.74	0.66	0.86	0.83	0.83	0.85	0.88	0.86	0.57	0.76	0.65

4.3.4 Answer to the research questions

Analysis of the results obtained by the machine learning models provides multiple insights that can help answer the research questions poised in section 4.1. These findings can elaborate on the effectiveness of using ML models in the task of predicting building occupant characteristics.

RQ1: How effectively can machine learning tools predict individual building occupant characteristics?

Understanding the effectiveness of using machine learning tools in predicting individual building occupant characteristics is crucial as it explains the usability and feasibility of the task. The evaluation results can benefit in understanding the effectiveness of using the machine learning tools of this study. This work incorporates 6 state-of-the-art machine learning models for 16 selected occupant characteristics classification and prediction tasks.

From the performance measures presented in table 4.2, it can be deduced that the machine learning models performed efficiently in the task. The models performed significantly well for 4 occupant characteristics and moderately well for 10 characteristics. The models obtained below-average accuracy in classifying 2 characteristics (education and income) which in itself is a finding denoting these characteristics may not depend on the energy usage and selected housing features.

From table 4.2 it is clear that the performance of the machine learning models varied over the 16 target variables. The ML models obtained different accuracy, precision, recall, and, f1-scores for the different occupant characteristics. The 16 target attributes of this work can be separated into 3 categories, (i) Temperature preference and equipment control, (ii) Household characteristics, and, (iii) Individual characteristics. The ML models achieved different effectiveness in these 3 categories. The following is a description of the machine learning model performances over these categories:

(i) Temperature preference and equipment control: Targets 1 through 8 of table 4.1 fall in this category. It includes temperature preferences during winter and summer along with the usage of heating and cooling equipments. These attributes represent the entire household but are specific to temperature control. The temperature preference attributes achieved an average accuracy of 50%. The data in these variables are distributed in 7 categories which makes 50% a moderately good number for accuracy. The accuracy scores among these variables also do not vary more than 3 which makes sense with the same input features. Surprisingly, the classification of the usage of the main cooling equipment during the summer (USEWWAC) achieved significantly better average accuracy (82%) than the classification of the usage of the main heating equipment (EQUIPMUSE) during the winter (52%). The random forest classifier (RFC) achieved the highest accuracy in both of these target labels (88% for USEWWAC and 66% for EQUIPMUSE). This result indicates that the input

features may provide more correlating information towards cooling equipment usage than heating equipment usage.

(ii) Household characteristics: Targets 12 to 16 of table 4.1 fall in this category. It includes the number of household members, number of adults, number of children, number of days, someone present at home, and, the combined household income. These attributes represent the entire household. The target attributes related to the number of people performed surprisingly well. The decision tree classifier achieved 99% accuracy in all three. It can be deduced that these variables have a high correlation with the input features present in the dataset. The input features like energy consumption, and, housing structure like number of bedrooms, total area, etc. can tell a lot about the number of habitats present. However, the machine learning models did not perform well in classifying the number of weekdays some is present at home (ATHOME) and the combined household income (MONEYPY) achieving an average accuracy of 51% and 28% respectively. The random forest classifier achieved the highest accuracy in these two variables (57% and 35% respectively). It can be the case that these variables are independent of the input features present in the dataset.

(iii) Individual characteristics: Targets 9 to 11 of table 4.1 fall in this category. It includes the age, employment status, and, level of education of the respondent. These attributes represent the individual who responded to the survey questionnaire. The machine learning models achieved an average accuracy of 51% for the respondent's age, 61% for the employment status of the respondent, and, 35% for the level of education of the respondent. These results indicate that the correlating input features were not enough for the machine learning model to classify these target attributes. These variables are solely about independent respondents and not the entire household however most of the input variables in this work represent the entire housing unit. This makes it difficult for machine learning models to classify individual occupant characteristics from household data.

RQ2: How do machine learning algorithms compare with each other when predicting building occupant characteristics?

The comparison of the machine learning algorithms for any classification or prediction task is very important as it facilitates performance evaluation, model selection, optimization, interpretability, and, understanding of model behavior. It helps to make informed decisions, improve performance, and gain insights into the classification task and the models' characteristics. For comparing machine learning classification models, factors like accuracy, precision, recall, and, F1-score are typically considered to assess the performance and determine which model is better suited for the task. The models used in this study are the most commonly used machine learning classification models used by scholars [99, 118]. Even though the same input and output features were provided to each model, the evaluation results significantly varied. Analyzing table 4.3 and table 4.4 can provide insights on the performance comparison among the machine learning tools used in this study.

Table 4.3 shows the average Accuracy obtained by each model. The reason behind the success of RFC can be the resourcefulness for making accurate predictions needed in strategic decision-making. Overall, the random forest classifier (RFC), the decision tree classifier (DT), and, the linear discriminant analysis classifier (LDA) are tied in the first position with an average accuracy of 63%. However, RFC outperformed the rest by achieving the best accuracy for 9 out of 16 target variables. The Adaboost classifier (ADB) can be placed in the second position as it achieved the best accuracy for 4 out of 16 target variables. The decision tree classifier achieved the highest accuracy of 99% for 3 of the target variables. Both K nearest neighbor classifier (KNN) and support vector machine classifier (SVM) scored an average of 42% overall accuracy.

Table 4.4 shows the average Precision (P), Recall (R), and, F1-score (F) obtained by the six machine learning models throughout the 10-fold validation process across the 16 target

variables. The target variables are placed in the left-most column, followed by average P, R, and, F obtained by each individual classification model. The results of each model are separated by vertical lines. Precision measures the accuracy of the positive predictions. Higher precision values indicate a lower false positive rate. For most target variables, RFC and DT have relatively high precision values, indicating their ability to accurately classify positive instances. Recall measures the ability of the model to identify positive instances correctly. Higher recall values indicate a lower false negative rate. The RFC and DT tend to have higher recall values, closely followed by LDA for most target variables, indicating their ability to capture a higher percentage of positive instances. F1-score is the harmonic mean of precision and recall, providing a balanced measure of the model’s performance. For most target variables, the RFC algorithm achieved higher F1-scores, indicating a good balance between precision and recall. Different algorithms performed better or worse depending on the target variables. For example, the DT and LDA tend to perform well for the population attributes while ADB and RFC performed relatively better in classifying education and gross household income target variables. This suggests that the performance and predictability of the classification model vary depending on the specific target variable being predicted. In summary, the table highlights the importance of selecting the appropriate machine learning model for a given classification task, as well as the potential for variation in performance across different targets.

4.4 Discussion

The classification and prediction of occupant characteristics using machine learning tools can be useful in many ways. It can provide a way of partially automating the building occupant persona generation procedure which can facilitate both the occupants and the building de-

signers in many ways. The following is an example case study of occupant persona generated from the feature predictions from this work followed by a discussion on the shortcomings of this study along with the possible future research directions.

4.4.1 Sample Occupant Persona Prediction

Table 4.5: Sample occupant characteristics prediction

Feature	Prediction	Description
EQUIPMUSE	1	Set a heating temperature and leave it there most of the time during the winter
USEWWAC	1	Set a cooling temperature and leave it there most of the time during the summer
TEMPHOME	4	Typical temperature during winter days when people are at home is 65 - 68 °F
TEMPGONE	2	Typical temperature during winter days when no one at home is 51 - 60 °F
TEMPNITE	4	Typical temperature during winter nights when people are at home is 65 - 68 °F
TEMPHOMEAC	5	Typical temperature during summer days when people are at home is 69 - 72 °F
TEMPGONEAC	7	Typical temperature during summer days when no one at home is 77 - 96 °F
TEMPNITEAC	4	Typical temperature during summer nights when people are at home is 65 - 68 °F
HHAGE	3	Middle age adult (31 -50)
EMPLOYHH	1	Employed full time
EDUCATION	3	Some college or associate degree
NHSLDMEM	2	2 People live in the house
NUMADULT	2	2 Adults live in the house
NUMCHILD	0	There are no children living in the house
ATHOME	0	Noone is present at home during the weekdays
MONEYPY	13	Gross household income is about \$60,000 to \$74,999

Table 4.5 shows a sample of prediction data from the machine learning models involving all 16 occupant characteristics. A random row from the RECS-2015 was provided as input to the ML models which predicted the 16 features. Table 4.5 reveals predictions from models with the best average accuracy (shown in table 4.3) for each target feature. The prediction column of table 4.5 shows the actual prediction value from the models (numerical). The description column provides the textual description of each prediction from the class details of table 4.1. As an example of the applicability of the results, OpenAI’s ChatGPT [95] was utilized with the information present in table 4.5 for generating a descriptive occupant persona that yielded the following output:

“The individual is a middle-aged adult, between the ages of 31 and 50. The residents

are employed full-time and have attained some college education or an associate degree. Their household consists of two adults, with no children residing in the house. The gross household income falls within the range of approximately \$60,000 to \$74,999. This income level can help inform decisions related to energy consumption, budgeting, and overall financial considerations.

During the winter, the heating temperature is set and left at a consistent level most of the time. When people are at home during winter days, the typical temperature is maintained between 65 and 68 °F, providing a comfortable and cozy environment. However, when no one is at home during winter days, the temperature is adjusted to a lower range of 51 to 60 °F to conserve energy and reduce heating costs. During winter nights when people are at home, the temperature is again set between 65 and 68 °F to ensure a pleasant sleep environment.

In the summer, a cooling temperature is set and maintained throughout most of the time. When people are at home during summer days, the typical temperature is kept between 69 and 72 °F, providing a cool and comfortable atmosphere. However, when no one is present at home during summer days, the temperature is allowed to rise to a higher range of 77 to 96 °F, conserving energy and minimizing cooling expenses. During summer nights when people are at home, the temperature is adjusted back to the comfortable range of 65 to 68 °F for a restful sleep.

Considering their employment, the household members are likely absent from home during weekdays. This schedule suggests that the heating or cooling systems may be adjusted accordingly to conserve energy and optimize comfort during those periods.”

The above is the ChatGPT-generated building occupant persona that harnessed 16 occupant characteristics predicted by the best-performing machine learning models of this study. This is a representation of a composite person who is likely to occupy a particular building or

space. It can help designers and developers to better understand and empathize with their target audience, which can lead to better design decisions and a more user-centric approach to building design. It can act as a reference point throughout the design process and building designers can ensure that the building is tailored to the needs and preferences of its intended occupants. This demonstrates the possible usage and feasibility of using machine learning tools in the development of building occupant persona. It also indicates the direction of automating the occupant persona development process.

4.4.2 Limitations and future research

The purpose of this research article was to investigate the application of state-of-the-art machine learning models in classifying building occupant characteristics. The ultimate objective was to automate the process of building occupant persona development, aiming to reduce manual effort and time consumption. The study successfully demonstrated the possibility of automating several steps in the generation of smart housing personas, thereby potentially streamlining the overall process. However, it is important to note that complete automation was not achieved in this study, leaving room for potential future research in this area. Further investigation is suggested focusing on fully automating the task, which would be a promising direction for future studies. This work utilized six different machine learning models in their analysis. Despite the extensive experimentation, this study couldn't distinguish any individual model that could effectively classify all 16 occupant characteristics with better confidence than the rest. Consequently, the adoption of a model selection strategy, where the best-performing models are chosen for each specific characteristic is suggested. This approach allows for more accurate predictions overall.

Future research works may explore deep-learning models for the classification and prediction

of building occupant characteristics. Deep-learning techniques have recently shown superior performance in dedicated tasks compared to traditional machine-learning models. Therefore, incorporating deep-learning models into future research endeavors may yield improved accuracy and results. Additionally, the study proposes investigating the ensembling technique of machine learning models for occupant characteristic prediction. Ensemble learning combines the predictions of multiple models to enhance overall accuracy and robustness. Applying this approach to building occupant persona development could potentially yield more reliable results. The research conducted in this study has practical implications for automating the persona development process in the context of building performance simulation, design behavior interventions, and smart building management solutions. By leveraging the findings of this research, future studies can work towards fully automating the persona development process. This holistic automation can lead to enhanced functionality and accuracy in various applications related to building performance and management.

4.5 Summary

This research investigates the feasibility of using machine learning to automate some steps of building persona development. Six machine learning models were trained using the 2015 Residential Energy Consumption Survey data provided by U.S. Energy Information Administration for the task of occupant characterization. The results indicate that it is possible to use machine learning tools to partially automate the process of building occupant persona development. This research contributes to the body of knowledge by proposing a machine learning based approach for facilitating the development of building occupant personas. It proves that given the necessary data, it is possible to automatically predict the occupant characteristics with a certain confidence, and thus, help us to better understand occupants

and improve living conditions to meet occupant demands. After the development of data collection infrastructure and building occupant characterization, the next chapter describes the proposed work to be conducted on the development issues in smart home systems.

Chapter 5

An empirical study of automation configuration bugs in Smart Homes

5.1 Motivation

A **smart home** is a residence that uses internet-connected devices to remotely monitor and manage appliances/systems. According to Fortune Business Insights, the global Smart Home Market size is projected to reach USD 338.28 billion by 2030, at a Compound Annual Growth Rate (CAGR) of 20.1% during the forecast period 2023–2030 [24]. As explained by researchers, the increasing number of internet users, surging disposable income of consumers within emerging economies, the growing significance of home monitoring in remote areas, and the increasing demand for low-carbon emission and energy-saving-oriented solutions are anticipated to drive the market competencies [16].

A **smart home platform** is a software framework that controls and manages multiple devices from multiple manufacturers, usually through a smartphone or tablet app. Various smart home platforms are available: some are commercial systems and close-source (e.g., Samsung SmartThings [108]); some are free and open-source (e.g., openHAB [84]). *Among the alternatives, Home Assistant (HA) has become one of the most widely used platforms [20, 21, 22] mainly because it is free, open-source, and designed specially for local control as well as privacy.*

```

automation:
- alias: "Turn on office lights"
  trigger:
    - platform: state
      entity_id: sensor.office_motion_sensor
      to: "on"
  condition: "{{ state_attr('sun.sun', 'elevation') < 4 }}"
  action:
    - service: scene.turn_on
      target:
        entity_id: scene.office_lights

```

Figure 5.1: An exemplar YAML file for home automation [32]

Up till March 2024, HA has got over 332 thousand active installations; HA developers estimated the actual number of HA users (e.g., residents of HA-based smart homes) to be 3 times of this number (i.e., about 1 million) [26]. The widespread usage of HA motivated us to do research on HA-based smart homes, because (1) the data from these systems can represent many smart homes, and (2) our research findings will help shape the future of smart homes.

To create a smart home with HA, users need to (1) integrate components (e.g., devices) into the system and (2) create YAML files to automatically control those components. YAML [31] is a human-friendly data serialization language for all programming languages. With YAML, HA users can define an automation rule by specifying a **trigger**, an **action**, and (optionally) a **condition**; such a rule expresses that HA performs an action when a trigger event occurs and optionally the specified condition is met. For instance, the rule in Fig. 5.1 means to turn on office lights, when someone moves (i.e., the sensor detects motion) and the outside is dark (i.e., the solar elevation angle is less than four degrees).

With some initial inspection of the online forum Home Assistant Community [27], we found lots of questions concerning automation configuration. On the forum, over 10,000 ques-

tions were tagged with “automation” and over 1,000 questions were tagged with “configuration” [29]. This may be because many HA users have no programming background and automatic control is challenging. As a result, misconfigured smart homes may misbehave [14], waste energy [8], frustrate users [6], or jeopardize home safety.

Inspired by our initial observations, we conducted a novel empirical study on automation configuration issues in HA-based smart homes, to understand the technical challenges and identify research opportunities. Specifically, we crawled the discussion threads of “Configuration” during 03/2022–08/2022 using tag “automation”, and retrieved 438 potentially relevant threads. Then we manually analyzed all threads to identify the root cause and resolution strategy of each issue under discussion, and filtered out 248 threads because they lack necessary information. We explored the following research questions (RQs) and observed interesting phenomena:

- **RQ1:** *What challenges do users face when configuring automation?* Among the 190 threads, 129, 52, and 9 discussions separately focus on debugging, implementation, and optimization. It means that users get stuck with debugging more often than other issues. We also observed significant concept commonality/similarity among threads (e.g., data specification for matching), which implies users’ strong need for help or tool support in implementing, debugging, or optimizing some features.
- **RQ2:** *How do users address challenges in automation configuration?* Users frequently applied eight strategies to address common challenges. Two of the strategies correct formats (i.e., quotes and indentation); two conditionally call various services; one replaces the trigger type; one correctly accesses or calculates data; one correctly specifies data for matching; one handles a group/list of same-typed entities. All strategies imply desired tool support.

- **RQ3:** *How effectively do existing tools detect or fix buggy YAML files?* We searched online for all existing tools that can validate YAML files. By applying the 6 publicly available tools to 190 buggy files, we observed all tools to report only 16–20 files/snippets as invalid ones, and 1–5 of these reports are false positives. All tools achieved high precision, very low recall, and very low F score when detecting bugs; their error-reporting mechanism is poor. No tool fixes bugs.

In this paper, we made the following research contributions:

- We conducted a novel and comprehensive empirical study to characterize real-world issues of automation configuration in HA-based smart homes. No prior work did that.
- We revealed the common root causes and recurring resolution strategies for HA-related automation issues; many of our findings are revealed for the first time.
- We novelly applied 6 publicly available tools to 129 buggy YAML files/snippets, and surprisingly found all tools ineffective in detecting or fixing bugs.
- We open-sourced our dataset and tool-application results, to facilitate future research in related areas.

HA is one of the most widely used smart home platforms, and commonly shares the automation configuration mechanism (if-this-then-that rules) with many other platforms (e.g., IFTTT [28] and Samsung SmartThings [108]). Thus, our study will shed light on future research in smart homes, help with end-user IoT programming, and enlighten potential ways of improving smart home quality. Our dataset is available at <https://figshare.com/s/7aa8ea9f4af98c371114>

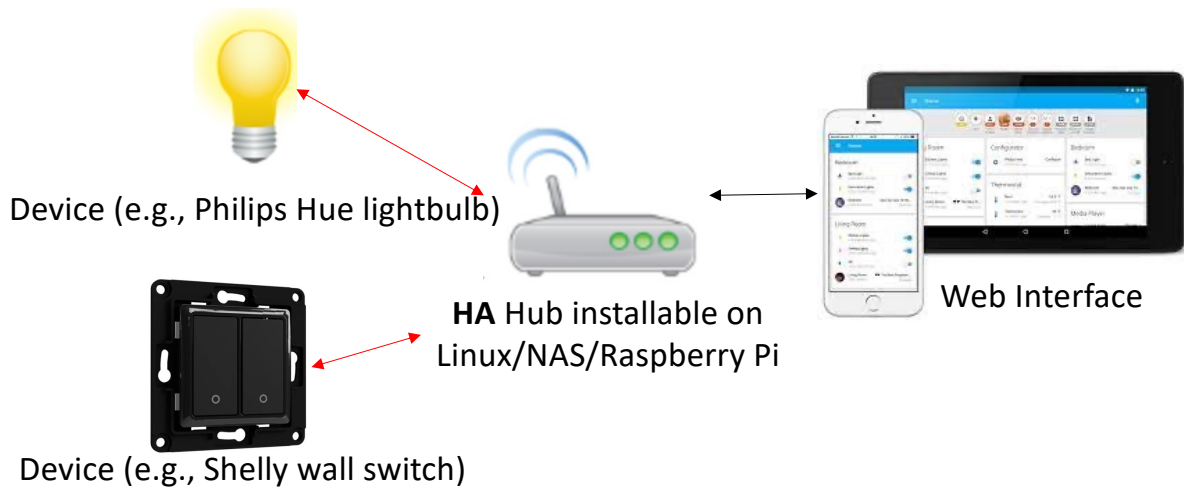


Figure 5.2: The consumer pattern in Home Assistant

5.2 Background

This section introduces the domain knowledge of Home Assistant.

HA can be installed on various devices, from full Linux systems to some network-attached storage (NAS) environment or even a Raspberry Pi. As shown in Fig. 5.2, users can access HA through a dashboard or web-based user interface by using companion apps for smartphones, or by using web browsers for tablets and PCs. Once the HA software is installed, it acts as a hub—a central control system for home automation. Without any mandatory dependence on vendor-specific cloud services, devices, or mobile apps, the HA hub can have local control of the IoT devices, software, applications, and services that are supported by modular integration components. **Home Assistant Community (HAC)** [27] is an on-line forum dedicated for HA developers/users to discuss and resolve issues in smart home systems. HAC contains discussion threads. Each thread has one question post, and zero or more answer posts; at most one answer in each thread is the accepted answer.

Integrations are pieces of software that allow HA to connect to other software and platforms. For example, a product by Philips called Hue smart light (see Fig. 5.2) can be included

into a smart home via the Philips Hue integration, which integration allows HA to talk to the hardware controller Hue Bridge, so that any HA-compatible physical devices connected to Hue Bridge appear in HA as logical devices (i.e, virtual objects) and can be controlled by the hub.

Entities are the basic building blocks to hold data in HA. An entity represents a sensor, actor, or function in HA, which can monitor physical properties or control other entities. Each entity has a **state** to hold information of interest (e.g., whether a light is on or off); an entity's state only holds one value at a time. Entities can store **attributes** related to their state, such as the brightness of a turned-on light. **Sensors** return information about an object, such as the level of water in a tank. **Devices** are logical groups for entities. A device may represent a physical device with one or more sensors; the sensors are entities associated with the device.

Automation is a set of repeatable actions that can be run automatically. **Automation Configuration** is about defining YAML files to specify automations, after HA is installed and all components are integrated. In a YAML file, an automation rule has three key segments:

(i) **Trigger** describes what starts an automation [30]. An automation can be triggered by an event (i.e., signal emitted when something happens), certain entity state (e.g., when a light is on), or a given time. Multiple triggers can be specified simultaneously for one automation. When any of the automation's triggers becomes true (i.e, trigger fires), HA will validate the conditions if any, and call the action. For the example in Fig. 5.1, the trigger is an event: when the state of a motion sensor is changed to “on”.

(ii) **Condition** is optional; it describes the **predicates** (i.e., tests) that must be met before actions get run [32]. After a trigger occurs, all conditions are checked. If any of them returns

false, the action is not run. The condition in Fig. 5.1 tests whether the solar elevation angle is less than four degrees.

(iii) **Action** describes what is executed when a rule fires [23]. It can interact with anything via services or events. **Service** carries out a task, such as turning on the light in the living room. A service can have a target and data. For instance, entity `scene.office_lights` in Fig. 5.1 is a **scene** that prescribes a series of actions, with each action setting an entity's state. The defined action interacts with this scene by calling service `scene.turn_on`, to turn on lights and set their states as prescribed by the scene.

Scripts are also repeatable actions, similar to automations. The difference is that scripts do not have triggers: scripts cannot automatically run unless they are used in an automation.

Templates are used for formatting outgoing messages to present to users or processing incoming data from entities. They are expressed with Jinja2 [2]—a general-purpose templating language. For instance, `"{{state_attr('sun.sun', 'elevation') < 4}}"` in Fig. 5.1 is a template, which gets the solar elevation degree from entity `sun.sun` and compares that value with 4 to define a predicate for condition checking. Templates enable smart homes to flexibly respond to varying entity states. To define templates, people must surround single-line templates with double quotes (") or single quotes ('). When defining multi-line templates, people must use `{% %}` to enclose the program structures (e.g., loop) in templates.

5.3 Methodology

To understand the challenges and opportunities in home automation, we explored the following three research questions:

- **RQ1:** *What challenges do users face when configuring automations?* What kind

of automation-related questions do users ask? Are there questions frequently asked? What are the root causes of the frequently asked questions?

- **RQ2:** *How do experts/users address challenges related to automation configuration?*

Namely, when people ask similar or identical questions, is there any answer repetitively suggested? Do answers present resolution strategies?

- **RQ3:** *How effectively do existing tools detect or fix buggy YAML files?* When users have difficulty debugging erroneous YAML files, can existing tools reveal or fix those errors?

This section first introduces our procedure of data collection, and then explains our method of exploring RQs.

5.3.1 Data Collection

The program data related to HA-based smart home systems can be majorly found on two websites: GitHub [25] and HAC [27]. We decided to crawl HAC for two reasons. First, HAC organizes discussion threads based on categories and tags. Such an organization enables us to quickly locate automation configuration-related program data. Second, many questions on HAC are resolved, with some resolutions well explained and finally accepted by askers. Such a high availability of technical solutions and rational explanations enables us to characterize questions and answers with high rigor.

Specifically, we manually went through the discussion threads under the category “Configuration” of HAC, to locate candidates tagged with “automation” from 03/01/2022 to 08/31/2022. That six-month period was chosen because we collected data in September 2022. As HAC ranked threads based on timestamps of the latest activity on each thread

(e.g., question posting or answer updating), we successfully retrieved a dataset of 438 candidate discussion threads purely based on the HAC ranking and timestamp range. Next, we manually inspected each thread to record (1) ID of the question post, (2) the URL, (3) the original YAML file provided by the asker, and (4) the suggested YAML file mentioned, referred to, or implied by the accepted answer. In this process, we filtered out a thread if

- (i) the question asker does not present a YAML file,
- (ii) no answer was explicitly marked as accepted by the asker,
- (iii) the accepted answer does not suggest or lead to any concrete version of YAML file, or
- (iv) the discussion is too confusing for us to understand.

We defined the four filters mentioned above, to ensure the high quality of our data analysis results. With more details, based on our experience so far, when people asked questions at HAC, they sometimes failed to provide YAML files to clearly show the programming context or automation scenarios. Consequently, analyzing such questions can be very challenging and time-consuming, as it is hard to tell what is the major issue and what types of resolutions askers inquiry for. To avoid such confusion, we introduced filter (i) to guarantee that each covered question post provides sufficient details on the programming context, including but not limited to askers' automation needs, the attempts they made, the technical issues they faced, and their specific requests for help. When processing each thread, we manually inspected all posts involved in the discussion. As long as the asker provided their YAML files in any of the posts, we kept those threads.

Filter (ii) ensures that there is a correct answer for us to refer to, when characterizing people's concern and resolution for each covered thread. Filter (iii) makes sure that the suggested resolution is clear and concrete. Namely, we do not have to speculate on the correct version

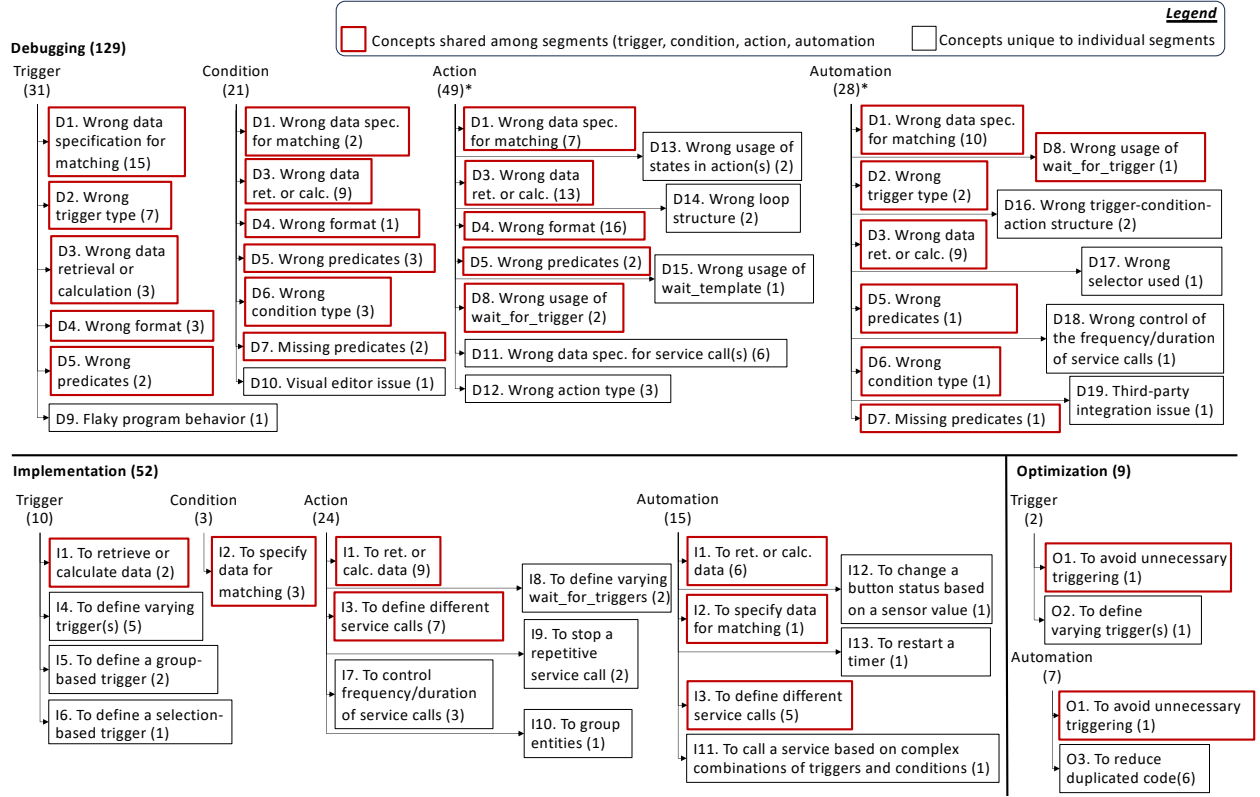


Figure 5.3: The taxonomy of 190 threads, where * means some issues belong to more than one category

of YAML files, and thus will not commit mistakes in speculation. In reality, however, an asker might accept an answer that provides no concrete YAML files, while the concrete version was mentioned in a different post of the same thread. To tolerate such inaccuracy in askers' labeling of accepted answers, we read all posts in each thread. As long as a concrete YAML file is suggested by an accepted answer or any answer related to that one, we counted in the thread. In this way, we made our dataset as representative as possible. Filter (iv) ensures that we have high confidence in our interpretation of threads. After applying all filters mentioned above, we kept 190 threads for further analysis.

5.3.2 Data Analysis

For RQ1, we took an open-coding approach to classify threads, because we had no prior knowledge of people’s concerns about automation configuration. Specifically, two authors manually inspected all threads to identify keywords/phrases, that characterize each thread in terms of the (1) issue type, (2) automation component under discussion, and (3) involved technical concepts. Then they separately defined an initial taxonomy and categories by clustering recurring or similar keywords/phrases. Next, they held a meeting to compare and discuss their initial results to refine the taxonomy and improve thread classification. Using the classification labels they agreed upon, author A rechecked all threads to label categories and to reveal overlooked or wrong categories. Afterwards, author B examined A’s labels for all threads; whenever he disagreed upon any labels, he had discussions with author A until reaching a consensus. This procedure sometimes involved multiple iterations of thread labeling, and lasted until both authors agreed upon all labels.

For RQ2, we also took an open-coding approach to identify frequently adopted resolution strategies. With more details, inside each category, author A summarized the resolution suggestion for each issue under discussion and then clustered issues if their resolutions were identical or similar in certain aspects. In this procedure, A also defined resolution strategies based on the common summaries. Next, author B manually inspected all strategies as well as related threads, and initiated discussion with A for any disagreement. The discussion lasted until all disagreements were addressed.

For RQ3, we first defined a ground-truth dataset of buggy YAML files/snippets, according to the classification results in RQ1. In this dataset, we also included the fixes suggested by accepted answers. Next, we applied all state-of-the-art YAML validation tools to the dataset, to study how effectively existing tools detect and fix bugs.

5.4 Experiment Results

This section presents and explains our results for RQ1–RQ3.

5.4.1 RQ1: Challenges in Handling Automation Configuration

Fig. 5.3 shows our thread classification based on issue types, issue locations, and involved technical concepts.

5.4.1.1 Classification Based on Issue Types

We observed three types of technical issues: (1) implementation, (2) debugging, and (3) optimization. **Implementation** means that people described their program context and automation requirements; they asked for implementation to fit the context and satisfy those requirements. **Debugging** means that users provide erroneous automations, requesting to diagnose the root cause and fix bugs. Compared with implementation, debugging questions often include askers’ unsuccessful trials (i.e., buggy YAML files) to implement certain requirements. **Optimization** means that people present the correct YAML files satisfying certain requirements, express their optimization goals, and ask for refined YAML files.

Our dataset includes 129 debugging issues, 52 implementation issues, and 9 optimization issues. It means that most askers had difficulty debugging YAML files, while fewer askers were bothered by implementation and optimization issues. This may be because automation configurations have relatively simple control logic. It is not quite difficult for people to get started and have initial implementation done. However, it is harder for people to successfully debug initial implementation, in order to deliver high-quality automation configurations. Consequently, even fewer people bother to optimize correct configurations for refinement.

Finding 1: *68% (129/190) of the examined issues are about debugging, implying the significant challenge of addressing buggy YAML files.*

5.4.1.2 Classification Based on Segments Involved

Under each issue category, we classified threads based on where issues occurred. Namely, if an issue resided in purely the trigger, condition, or action segment of an automation rule, we labeled it accordingly. Otherwise, if an issue resided in multiple segments or we could not clearly tell which specific segment an issue was about, then we labeled the issue with “Automation”. As shown in Fig. 5.3, for both debugging and implementation issues, “Action” is the largest among four categories. This implies that actions are harder to develop than triggers and conditions. One reason is that HA defines the script syntax [15], which allows people to define complex control structures (e.g., if-then and loop), or special behaviors in actions (e.g., `wait_for_trigger` to wait for a trigger before calling a service). The syntax poses challenges for people to understand and properly use all syntactic components. Additionally, there are more threads on triggers than conditions. It implies that triggers are harder to develop, probably because there are more alternative ways of defining triggers, and more delicate constraints on trigger definition. Finally, automation issues widely exist in all three categories. It means that users have difficulty in developing multiple segments for consistency, or in creating code that can exist in any of the segments.

Finding 2: *There are 73, 50, 43, and 24 issues separately about action, automation, trigger, and condition, implying that automation and action segments are more challenging to create or improve.*

5.4.1.3 Classification Based on Technical Concepts

We further classified issues based on the technical concepts involved. In this procedure, whenever we observed similar issues in different segments (i.e., trigger, condition, or action), we used the same terms to capture the similarity, and to identify segment-specific concepts as well as segment-agnostic ones. In Fig. 5.3, the 129 debugging issues cover 19 unique concepts; 8 concepts are shared among segments (D1–D8).

D1. Wrong data specification for matching means that data (e.g., state or attribute) was wrongly provided for predicate-value matching in trigger, condition, action, or the whole automation, making the automation run abnormally. For instance, lines 2–6 in Listing 1 show a buggy snippet from a debugging thread [12]. This snippet defines a state trigger, to start an automation when an entity `input_boolean.ecobee_fan_on_off` changes from the 'off' state to 'on'. However, as both states are misspelled, HA is case-sensitive and does not recognize those states. Consequently, the automation is never started because the prescribed state transition never happens.

```
1 # Bug: Wrong states are specified for the to: and from: options
2 trigger:
3   - platform: state
4     entity_id: input_boolean.ecobee_fan_on_off
5     to: 'On'      # Fix: 'On' should be 'on'
6     from: 'Off'  # Fix: 'Off' should be 'off'
7 # Implementation: To properly specify an unwanted target state with the not_to: option
8 trigger:
9   - platform: state
10  entity_id:
11    - sensor.pc1_printjob
12    - sensor.pc2_printjob
13    - sensor.pc3_printjob
14    - sensor.pc4_printjob
15  not_to: 'unknown'
```

Listing 1: Two YAML snippets to show the common challenge of “data specification for matching” [11, 12]

D2. Wrong trigger type: There are multiple types of triggers (e.g., state trigger and device

```

1 # Bug: A template is involved in a device trigger definition
2 trigger:
3   - type: no_motion
4     platform: device
5     device_id: e0954ea41d7a6da69baeff2e9558ed13
6     entity_id: binary_sensor.v4b01_motion
7     domain: binary_sensor
8     id: '1'
9     for:
10      hours: 0
11      minutes: '{{states('input_number.timeout_offices') | int(0) }}'
12      seconds: 0
13 # Fix: Replace the device trigger with a state trigger to support template usage
14 trigger:
15   - platform: state
16     entity_id: binary_sensor.v4b01_motion
17     id: '1'
18     to: 'off'
19     for:
20      minutes: "{{ states('input_number.timeout_offices') | int(0) }}"

```

Listing 2: A bug and fix related to D2 [17]

trigger) usable to specify the triggering logic of an automation. However, when users wrongly choose a trigger type, the specified logic does not work because the chosen type does not support that logic. For instance, the buggy version in Listing 2 defines a device trigger, to fire when sensor `binary_sensor.v4b01_motion` detects no motion for the number of minutes users specified via `input_number.timeout_offices`, a numeric input box in GUI. However, device triggers do not support templating (see line 11), so the buggy version fails.

D3. Wrong data retrieval or calculation means that some data used in an automation is wrongly accessed or calculated, causing automation fail to run normally. For instance, lines 2–6 in Listing 3 show a buggy snippet from a debugging thread [4], which defines an action to send a notification message via mobile phone. This action calls the service `notify.mobile_app_huawei_p20`, with a dynamically generated message to incorporate the reading of a temperature sensor `sensor.temperatur_gefrierschrank`. However, as users wrongly accessed the sensor value via `sensor.temperatur_gefrierschrank`, the automation does not run and an error message is produced: “*Error rendering data template: UndefinedError: 'sensor ist undefined'.*”

```

1 # Bug: Wrong way of retrieving the sensor state value
2 action:
3 - service: notify.mobile_app_huawei_p20
4   data:
5     title: Temperature Warning
6     message: 'Temperature is: {{ sensor.temperatur_gefrierschrank }}'
7 # Fix: sensor.temperatur_gefrierschrank => states("sensor.temperatur_gefrierschrank")
8 # Implementation: To include sensor value into the notification
9 action:
10 - device_id: c593faddfd2c2d134bc507567b588b2ae
11   domain: mobile_app
12   type: notify
13   message: Energy production ended. Energy produced today: {{states('sensor.solaredge_current_power')}}

```

Listing 3: Two YAML snippets to show the common challenge of “data retrieval or calculation” [4, 9]

```

1 # Bug: Wrong combination of predicates
2 condition:
3 - condition: sun
4   before: sunrise
5   after: sunset

```

Listing 4: A buggy snippet of “wrong predicates” (D5) [18]

D4. Wrong format means that a YAML file is malformed, by violating the formatting rules defined by either YAML or HA. Typical violations include wrong line indentation and wrong quote usage.

D5. Wrong predicates means automation fails because the predicates used in automation have flaws: the defined predicates or predicate combinations do not enable action execution as expected. For instance, Listing 4 shows a buggy condition that combines two predicates `before:sunrise` and `after:sunset`. The former predicate corresponds to the time period between midnight and sunrise, while the latter corresponds to the period between sunset and midnight. These two predicates should never get used together to define a period “*after the sunset of Day N and before the sunrise of Day (N+1)*”, because the two predicates are always interpreted as two separate periods: “*after the sunset of Day N and before the sunrise of Day N*”; no time point satisfies both predicates at the same time.

D6. Wrong condition type is similar to D2. There are multiple types of conditions. However, using some condition type (e.g., device condition) can limit automation expressiveness (e.g., no template allowed), making some logic infeasible.

D7. Missing predicates means one or more predicates are not checked before an automation executes certain action.

D8. Wrong usage of `wait_for_trigger`: Action `wait_for_trigger` allows automation to wait for a trigger being fired before doing anything. Some users messed it up with (1) another specialized action `wait_template`, which waits for a template to evaluate to true before running automation, or (2) the trigger segment.

D9. Flaky program behavior means two semantically equivalent implementations of the same logic have divergent outcomes: one automation succeeds and the other fails.

D10. Visual editor issue means a problematic YAML file is generated due to the usage of a visual editor in HA IDE.

D11. Wrong data specification for service calls means an action calls service(s) using wrong data, making the entire automation fail. Namely, if we treat service calls as analogous to method calls in Java programming, then wrongly specified service data is analogous to wrongly provided parameter values. For instance, Listing 5 shows a service call, which intends to make the Google speaker announce an audio message ```Dryer has started```. However, lines 7–8 are not needed by the service call. When both key-value pairs are provided, the automation fails and an error message is generated: *extra keys not allowed @data['cache_dir']*.

D12. Wrong action type is similar to D2 and D6. It happens when users wrongly select a type to implement some unsupported logic.

```

1 # Bug: A service call with wrongly provided data
2 action:
3   - service: tts.google_say
4     data:
5       entity_id: media_player.living_room_speaker
6       message: Dryer has started
7       cache: true
8       cache_dir: /tmp/tts

```

Listing 5: A buggy snippet of D11 [19]

D13. Wrong usage of states in action(s) means users try to define an action by specifying entity states. However, action can be only defined via interactions with services or events (e.g., service calls).

D14. Wrong loop structure means users wrongly use the loop construct to define a malformed automation.

D15. Wrong usage of wait_template means users have difficulty correctly using this action.

D16. Wrong trigger-condition-action structure means users wrongly define the trigger-condition-action structure to implement an automation, like defining an action without a trigger.

D17. Wrong selector used: Selectors are used to specify what values are acceptable by automation, or define how the input is shown in GUI. D19 is similar to D2, D6, and D12. Basically, when multiple alternative selectors are available, users have difficulty choosing the right one to specify the intended logic.

D18. Wrong control of the frequency/duration of service calls is similar to D2, D6, D12, and D17. There are multiple means to control the frequency or duration of service calls, and these means differ in expressiveness. If users choose the wrong mean, then they are unable to express the intended logic.

D19. Third-party integration issue means when an automation depends on a third-party

software for device control, users may have difficulty fixing issues due to that software usage.

The 52 implementation issues cover 13 distinct concepts. Three concepts are shared among segments (I1–I3).

I1. To retrieve or calculate data is similar to D3, because it also reflects users’ concerns on appropriate data access or calculation. For instance, lines 8–12 in Listing 3 show a correct snippet from an implementation thread [9], which defines an action to send a notification message via mobile phone. The message is formulated based on the value of a sensor that tracks energy production `sensor.solaredge_current_power`. This snippet was recommended because some users could not retrieve the sensor value. Note that I1 is different from D3, as I1 covers implementation questions while D3 is about debugging questions.

I2. To specify data for matching is similar to D1, as it also reflects users’ concerns on data specification for predicate-value matching. For instance, lines 8–15 in Listing 1 shows a snippet from an implementation thread [11]. The snippet defines a state trigger, which monitors four sensors (`sensor.pcX_printjob`), and starts automation when any sensor (1) changes its state and (2) the to-state is not `'unknown'`. This snippet was suggested because some users could not specify an unwanted state for the trigger.

I3. To define different service calls concerns about the correct way of calling distinct services based on the fired triggers, satisfied conditions, or collected data.

I4. To define varying trigger(s) means to define one or more changeable triggers (i.e., triggers with variables or templates), whose firing rules vary with the surrounding environment, time, or inputs.

I5. To define a group-based trigger(s) means to define triggers based on the values of a group of sensors.

I6. To define a selection-based trigger means to define a trigger, which fires based on which GUI button users click.

I7. To control frequency/duration of service calls is similar to D18, because it also concerns the correct way of controlling the frequency or duration of service calls.

I8. To define varying wait_for_trigger(s) is similar to D8, as it also focuses on the correct usage of `wait_for_trigger`.

I9. To stop a repetitive service call means when users call a service in a loop structure, they need help in defining the loop so that the service is called repetitively, and the loop condition is properly evaluated before automation terminates.

I10. To group entities means to group a set of entities (e.g., speakers) so that they work unanimously just like one entity.

I11. To call a service based on complex combinations of triggers and conditions means users have complex predicates/tests to meet before a service is called. Thus, they specify their automation needs and all predicates, asking for a well-structured automation that properly puts predicates in triggers and conditions.

Both *I12* and *I13* are self-explanatory, so we skip the explanation for succinctness.

The nine optimization issues cover three unique concepts: to avoid unnecessary trigger firings, to efficiently define changeable triggers, and to reduce duplicated code in automation. One of the concepts is shared among segments (O1).

We use **segment-agnostic concepts** to refer to the concepts shared among distinct segments, and use **segment-specific concepts** to refer to the remaining. Fig. 5.3 shows one, three, and eight segment-agnostic concepts separately found in optimization, implementation, and debugging issues. They imply that users ask similar or related questions even

though they work on distinct segments. Additionally, across all categories, issues related to trigger cover in total five segment-specific concepts (i.e., D9, I4–I6, O2). Meanwhile, issues related to condition, action, and automation separately cover one, nine, and eight segment-specific concepts. These observations imply that the issues related to trigger and condition are less diverse than those related to action and automation; action and automation are harder to develop and maintain.

Finding 3: *One, three, and eight segment-agnostic concepts are separately identified in optimization, implementation, and debugging issues. This implies the wide existence of common challenges.*

Between the two categories—debugging and implementation, we observed similarity among identified concepts: (i) D1 vs. I2, (ii) D3 vs. I1, (iii) D8 vs. I8, (iv) D18 vs. I7, (v) D11 vs. I3. For instance, both D1 and I2 focus on “data specification for matching”; however, the former describes a root cause of recurring bugs and the latter describes a frequent implementation request. Between implementation and optimization, we also observed a commonality: to define varying triggers (I4 vs. O2). All such similarity or commonality implies that some concepts are challenging and popular; they confuse people no matter whether people implement, debug, or optimize automations.

To facilitate understanding, we use representative examples to show two common challenges. Listing 1 has a buggy snippet from a debugging thread of D1, and a correct snippet from an implementation thread of I2. In the first snippet, users committed mistakes by misspelling states (e.g., 'On' instead of 'on'), and the automation never started. The second snippet was suggested because some users could not specify an unwanted state. Both snippets evidence that users have difficulty in specifying data for matching. Listing 3 shows another buggy snippet and another correct snippet. In the former one, users made mistakes when accessing a sensor value; the latter one was recommended because some users asked for help in getting

Table 5.1: The representative resolution strategies (i.e., each strategy occurs at least four times in our dataset)

Idx	Focus	The Relevant Technical Concept(s)	# of Issues Resolved	Strategy Content
R1	Quote usage	Wrong format	10	Add or remove quotes to properly use String literals, variables, and templates
R2	Indentation usage	Wrong format	6	Add or remove white spaces to properly align statements
R3	Picking a trigger type	Wrong trigger type	4	Replace the device trigger with a (numeric) state trigger
R4	Reading an entity's state	Wrong data retrieval or calculation, to retrieve or calculate data	9	Use template <code>{{states('ENTITY')}}</code> to read the state of ENTITY
R5	Specifying states for matching	Wrong data specification for matching, to specify data for matching	8	Look up all possible states of the entity in Developer Tools -> States, and use one of the states with case sensitivity.
R6	Different service calls based on fired triggers	To reduce duplicated code, to avoid unnecessary triggering, wrong data specification for matching	5	Define multiple triggers, specify a unique ID for each trigger, and call services differently based on the fired trigger.
R7	Different service calls based on entity values	To reduce duplicated code, to define different service calls, wrong data specification for matching, wrong loop structure	7	Define a dictionary to map entity values to distinct service calls or service data, and call services based on the retrieved entity values.
R8	Handling a group or list of same-typed entities	To reduce duplicated code, to define different service calls, to define varying trigger(s), wrong data retrieval or calculation, wrong algorithm design	5	Use <code>expand()</code> to enumerate a group or list of entities, and define filters to pick elements that satisfy certain requirements.

a sensor value. Both snippets imply that users have difficulty in retrieving or calculating data.

Finding 4: *Five concepts commonly exist in implementing, debugging, and optimization issues; these concepts are challenging no matter whether users implement, debug, or optimize automations.*

5.4.2 RQ2: Frequent Resolution Strategies

We realized that the 190 examined issues were resolved in diverse ways; no dominant strategy was applied to resolve the majority. However, we managed to identify some strategies repetitively applied to resolve multiple issues (≥ 4). Table 5.1 lists each strategy in terms of its focus, the relevant technical concept(s), the number of issues it addressed, and the strat-

egy content. We ranked the eight strategies in ascending order of the number of relevant concepts. This section explains all strategies in detail.

R1 corrects quote usage. It was frequently applied when askers posted YAML files with wrong formats. For instance, Listing 6 shows a representative bug and fix. The buggy snippet tries to set a counter `counter.configure` using the value of another counter `counter.aux_ac_pieza`, but the quote usage violates a domain-specific constraint in HA: when there are quotes inside and outside a template, differentiate the outer and inner ones by using different quote types. The buggy code nests single quotes. Thus, the fix is to replace the inner pair of single quotes with double quotes.

R2 corrects indentation. As with R1, this was also adopted to fix wrong formats. The basics of YAML syntax are block collections and mappings containing key-value pairs. Indentation is important for specifying relations among collections, mappings, and their items. For instance, the buggy version in Listing 7 tries to repetitively (1) check whether the door is open at night (between 23:30pm and 5am next day), and (2) send a notification if so. Such a logic can be realized with a `while`-loop, which encloses two blocks—a `while` block and a `sequence` block—with keyword `repeat`. However, as the buggy version did not indent the `sequence` block properly (see lines 10–11), the automation fails.

R3 replaces device triggers with triggers of `numeric_state` or `state`, because device triggers are too limited to express triggering events. For instance, the buggy version in Listing 2 defines a device trigger, to fire when sensor `binary_sensor.v4b01_motion` detects no motion

```
1 service: counter.configure
2 data_template:
3   value: '{{states('counter.aux_ac_pieza')}}'
4 # Bug: A pair of single quotes enclose another pair of single quotes
5 # Fix: Replace the inner quotes with double quotes, as below
6   value: '{{states('"counter.aux_ac_pieza'"')}}'
```

Listing 6: A bug and fix related to the quote usage (R1) [13]

for the number of minutes users specified via `input_number.timeout_offices`, a numeric input box in GUI. However, device triggers do not support templating (see line 11), so the buggy version fails. The fix is to replace that trigger with a state trigger (lines 14–20).

R4 is about state access. To correctly read the state of an entity (see Listing 3), people are suggested to call function `states(...)` with that entity’s ID, such as `states('sensor.solaredge_current_power')`.

R5 fixes misspelled states, by suggesting users to look up valid states of a given entity in Developers Tools of HA IDE and properly specify states. For example, `input_boolean.ecobee_fan_on_off` in Listing 1 is a boolean variable representing an input box in GUI, whose state value is 'on' or 'off'. Users must specify states accordingly.

R6 defines alternative triggers in one automation rule, and calls services differently depending on the fired trigger. It establishes correspondence between triggers and service calls by (1) assigning distinct IDs to triggers, and (2) referring to those IDs in service calls. For instance, suppose that an HA user wants to turn on a wifi bulb at 3pm and off at 5pm every day; the bulb should be also turned on whenever it gets online during 3pm-5pm after being offline, because the bulb sometimes goes offline at 2:59pm due to connectivity issues. To define both turning-on and turning-off behaviors of the bulb in the same automation

```
1 action:
2   - repeat:
3     while:
4       - condition: state
5         entity_id: binary_sensor.group_door_sensor_at_night
6         state: 'on'
7       - condition: time
8         before: '05:00:00'
9         after: '23:30:00'
10  sequence:
11    - service: notify.mobile_app_iphone
12  # Bug: The while-loop does not have "sequence" aligned with "while"
13  # Fix: To indent the "sequence" block
14    sequence:
15      - service: notify.mobile_app_iphone
```

Listing 7: A bug and fix about indentation (R2) [10]

```

1 - alias: TurnOnOffBulb
2   trigger:
3     - id: 'on'
4       platform: time
5       at: '15:00:00'
6     - id: 'off'
7       platform: time
8       at: '17:00:00'
9     - id: 'on'
10      platform: state
11      entity_id: light.mybulb
12      from: 'unavailable'
13 condition: "{{ today_at('15:00:00') <= now() < today_at('17:00:02') }}"
14 action:
15   - service: 'light.turn_{{ trigger.id }}'
16     target:
17       entity_id: light.mybulb

```

Listing 8: Different service calls based on fired triggers (R6) [5]

rule, lines 2–12 in Listing 8 define three alternative triggers, and assign two IDs to those triggers separately: 'on' and 'off'; line 15 calls the corresponding service (i.e., `light.turn_on` or `light.turn_off`) by composing a service name using the `trigger.id` value.

```

1 - id: '1631649587197'
2   alias: Alert - Bin collection tomorrow
3   description: ''
4   variables:
5     waste: >
6       {% set t = (now() + timedelta(days=1)).date() %}
7       {% set x =
8     { strptime(states('sensor.rubbish_bin_collection'), '%a %d %b %Y').date(): 'Rubbish',
9     ↳ strptime(states('sensor.recycling_bin_collection'), '%a %d %b %Y').date(): 'Recycling',
10    ↳ strptime(states('sensor.garden_bin_collection'), '%a %d %b %Y').date(): 'Garden' } %}
11     {{ x[t] if t in x.keys() else 'nothing' }}
12   trigger:
13     - platform: time
14       at: '20:00:00'
15   condition: "{{ waste != 'nothing' }}"
16   action:
17     - service: notify.admin_devices
18       data:
19         message: "{{ waste }} bin collection tomorrow!"
20   mode: single

```

Listing 9: Different service calls based on entity values (R7) [1]

R7 reads values of multiple entities (e.g., sensors), and calls services differently depending on those values. It typically defines a dictionary to map entity values with service calls or data. Given an entity value, it looks up the dictionary to take an action. For instance, Listing 9

uses three self-defined sensors to pull bin collection dates (rubbish, recycling, garden) from a website, and sends notification messages the night before a collection is due. In the automation, the Jinja variable `x` is defined as a dictionary (line 7), to map pulled dates of bin collection to bin types. The Jinja variable `t` holds the date of the next day (line 6). Variable `waste` is initialized as `'nothing'` if the next day's date `t` does not match any bin collection date, or as a bin type if `t` finds a match (line 9). Finally, the service is called with the value set to `waste` (line 17).

R8 defines pipelines to uniformly handle a group or list of same-typed entities. Each pipeline typically starts with the Jinja function call `expand()` to enumerate all elements in a group/list, and then invokes Jinja built-in filters (e.g., `map()`) to refine or process elements. For instance, suppose that an HA user wants to set all Google speakers at home to the same volume, by referring to the most recent volume setting among all speakers. Listing 10 satisfies the automation need through three steps. First, to identify all turned-on speakers, it enumerates speakers (lines 4–7), rejects turned-off entities (line 7), gathers values of attribute `'entity_id'` (line 8), and lists those values. Second, to acquire the most recent volume setting, it enumerates all volume sensors (lines 10–13), sorts them in descending order of their latest change timestamps (line 14), gathers values of attribute `'state'` (i.e., timestamps), gets the first one in that list (i.e., the most recent one), and converts the value to a floating-point number. Third, the automation sets all turned-on speakers to the same volume.

Notice that R1–R3 address debugging issues; R4–R5 resolve debugging and implementation issues; R6 fixes optimization and debugging issues; R7–R8 handle issues of all three categories. These strategies provide two insights for future tool support of smart home development. First, users repetitively commit certain mistakes, and such mistakes present clear bug patterns; it is promising to create new tools to detect and fix repetitive bugs. Second, some users cannot (1) define diverse actions depending on the fired triggers or entity values,

```

1 action:
2   - variables:
3     speakers_on: >
4       {{ expand("media_player.google_home_bedroom",
5         "media_player.google_nestmini_office",
6         "media_player.google_nesthub_living_room",
7         "media_player.google_nestmini_kitchen")| rejectattr('state', 'eq', 'off')
8         | map(attribute='entity_id') | list }}
9     recent_volume: >
10      {{ expand("sensor.volume_google_speaker_bedroom",
11        "sensor.volume_google_speaker_office",
12        "sensor.volume_google_speaker_living_room",
13        "sensor.volume_google_speaker_kitchen" )
14        | sort(attribute='last_changed', reverse=true)
15        | map(attribute='state') | first | float(0) }}
16   - service: media_player.volume_set
17   data:
18     volume_level: "{{{ recent_volume }}}"
19   target:
20     entity_id: "{{{ speakers_on }}}"

```

Listing 10: Handling a group or list of same-typed entities (R8) [3]

or (2) define uniform processing for elements in a group/list, although such automation needs are not uncommon. Therefore, it will be helpful to create tools, which generate automation implementations to at least partially satisfy those needs.

Finding 5: *The eight resolution strategies imply that domain experts addressed recurring issues by following certain principles; it is promising to automate such principles.*

5.4.3 RQ3: Current Debugging Tool Support

To study the existing tools for debugging YAML files or YAML-based automation configuration, we searched online with keywords “YAML validator” and found five publicly available tools: YAML Validator by Code Beautify [33], YAML Lint [83], YAML Checker [54], YAML Validator by JSON formatter [63], and ONLINEYAMLTTOOLS [119]. Additionally, as HA provides an IDE to help automation development, we also included its checker into the tool list for evaluation. As shown in Table 5.2, for simplicity, we assigned a unique ID to each tool, and will consistently use T1–T6 to refer to these tools. In our experiments, **because we found no tool to suggest any bug fix, this section focuses on our results about**

tools’ capabilities of bug detection.

5.4.3.1 Metrics for Automatic Bug Detection

We defined three metrics to evaluate the bug detection capabilities of tools:

Recall (R) measures among all known bugs, how many of them are reported by a tool:

$$R = \frac{\# \text{ of known bugs reported}}{\text{Total } \# \text{ of known bugs}} \quad (5.1)$$

To compute the recall of a tool, we first intersected the tool-reported bugs with known 129 bugs. Then we computed the count ratio between this intersection and the known bug set.

Precision (P) measures among all bugs reported by a tool, how many of them are real bugs:

$$P = \frac{\# \text{ of true bugs reported}}{\text{Total } \# \text{ of bugs reported}} \quad (5.2)$$

The 129-bug set does not include all bugs existent in given YAML files/snippets, as developers sometimes omitted discussion on trivial bugs. To compute a tool’s precision, we manually inspected all bug reports. If a report B_r clearly describes a known bug, we consider B_r a true positive. Otherwise, if B_r is confusing, we applied the tool which originally output B_r also to the developer-fixed version. If (1) that tool reported nothing for the fixed version or (2) our domain knowledge confirms the bug reported by B_r , we consider B_r a true positive; otherwise, B_r is a false positive.

F-score (F) combines P with R , to measure the overall accuracy of bug detection as below:

$$F = \frac{2 \times P \times R}{P + R} \quad (5.3)$$

Table 5.2: The bug detection capabilities of current tools

ID	Tool Name	# of Bugs Reported		P	R	F
		Correct	Incorrect			
T1	YAML Validator by Code Beautify [33]	15	3	83%	9%	15%
T2	YAML Lint [83]	14	3	82%	7%	13%
T3	YAML Checker [54]	15	5	75%	8%	14%
T4	YAML Validator by JSON formatter [63]	15	4	79%	8%	14%
T5	ONLINEYAML TOOLS [119]	15	4	79%	8%	14%
T6	HA IDE Checker [7]	15	1	94%	8%	14%

F is the harmonic mean of P and R . All three metrics have their values vary within $[0, 1]$.

The higher, the better.

5.4.3.2 Tool Effectiveness

As shown in Table 5.2, the tools behaved similarly to each other. All tools achieved high precision (75%–94%), low recall (7%–9%), and low F scores (13%–15%). They reported bugs in 14 common YAML files/snippets, and detected no bug in 106 common files/snippets. High precision means that the six tools report bugs with low false positives. Namely, if a tool reports a bug for a given YAML file/snippet, the file or snippet is very likely to be buggy. Meanwhile, low recall means that these tools poorly reveal known bugs. Among the 129 bugs we distilled from HA discussion, only 9–11 bugs were covered by the reports generated by each tool; the combination of all tools’ outputs only revealed 14 known bugs. Due to such low recall rates, the measured F values are also low.

Among the tools, T6 got the highest precision; T1 got the highest recall and F-score; T4 and T5 acquired identical values for all metrics. Between T4 and T5, we found a significant overlap in tool-generated bug reports; both tools described all errors in almost identical ways. This interesting observation implies that T4 and T5 may share the same core implementation.

Table 5.3: The confusing tool-generated reports for Listing 6

ID	Error Message
T1	Error: Unexpected characters near “counter.aux_ac_pieza’)}}’”. Line : 3 value: ‘{{states(‘counter.aux_ac_pieza’)}}’
T2	Unexpected scalar at node end at line 3, column 21
T3	bad indentation of a mapping entry (3:21)
T4	Error: can not read an implicit mapping pair; a colon is missed at line 3, column 46
T5	YAMLError: can not read an implicit mapping pair; a colon is missed at line 3, column 46
T6	bad indentation of a mapping entry (3:21)

Additionally, we once hypothesized T6 to outperform other tools in all metrics, because T6 was specially designed to reveal HA-related issues while all other tools are general YAML file checkers. Surprisingly, we found T6 to work similarly with other tools, and only outperformed the other tools in precision.

Finding 6: *For bug detection, existing tools achieved high precision, low recall, and low F scores.*

5.4.3.3 Characterization of Tool-Generated Bug Reports

Among the bug reports output by different tools, we observed two phenomena. First, *the bug reports are purely about wrong formats*. These issues may involve wrong quote usage, bad indentation, and wrong tag (i.e., token) usage. However, as mentioned in Section 5.4.1, the 129 known bugs cover 19 distinct concepts; “wrong format” is just one of them, and its bugs seem easier to detect because formatting rarely requires for advanced program analysis. Unfortunately, existing tools cannot detect other types of bugs. Even within the 20 known bugs of “Wrong format”, we still observed 11 cases missed by all tools. For three cases, T3–T5 failed to interpret the legal notation “!” and incorrectly reported it as wrong tag usage.

Finding 7: *Existing tools only revealed a relatively simple type of bugs—wrong format; even for the detection of such bugs, current tools suffer from significant false positive and false negative issues.*

Second, *the error messages in many bug reports are very confusing.* They may incorrectly pinpoint the bug location, wrongly describe an error, or provide meaningless hints that mislead users. For instance, Listing 6 shows a buggy snippet to misuse quotes at line 3. Essentially, single quotes should not be nested. However, T1 and T2 output confusing messages to complain about unexpected characters/scalar at line 3 (see Table 5.3), without clarifying which character/scalar is unexpected. T3 and T6 reported bad indentation, but no indentation issue exists at all. T4 and T5 mentioned a missing colon, which does not reflect the wrongly used quotes, either. Among the 109 bug reports we examined in total, 45 reports contain confusing error messages. T1 produced the biggest number of confusing reports—12, while T3–T6 output the fewest—6.

Finding 8: *In our experiment, 41% (45/109) of the examined bug reports are confusing, which evidences the need of improving the error-reporting mechanism.*

5.5 Threats to Validity

Threats to External Validity. All our observations are based on the experimental dataset. Although the study analyzes 190 threads in-depth, the sample size, while substantial, may not fully represent the diverse range of experiences and challenges encountered by all users. In the future, we will add more data to our dataset, and conduct further research as well as analysis to draw more generalized conclusions about automation configuration issues in smart homes.

Threats to Internal Validity. Our manual analysis for the collected data and tool outputs is

subject to human bias, and is limited to our domain knowledge. To mitigate the problem, we had two authors independently examine the data in multiple iterations. When they disagreed upon certain data labels or classification criteria, they had discussion until coming to an agreement and enforced the same labeling/classification mechanism in the next iteration.

5.6 Lessons Learned

Below are actionable items we learned from this study.

For Tool Developers and SE Researchers: Our study shows that HA users do not have sufficient domain knowledge or good tool support for automation development. Tool developers and SE researchers can build tools to (1) better detect or fix bugs in user-defined automations or (2) generate automation implementation from scratch. In particular, we suggest three future directions.

First, syntax checkers and fixers. Users often use quotes and indentation in wrong ways (Sections 5.4.1 and 5.4.2), and the current tool support is poor (Section 5.4.3). HA defines a set of grammar rules on top of the basic YAML rules (e.g., `while-loop`). To enforce these domain-specific rules, future work can create parsers to analyze YAML files, locate HA-specific keywords as well as structures, comprehend automation rules based on the extracted information, and create syntax trees. In the tree-creation procedure, parsers can detect malformed content by reporting any violation of grammar rules; they can further automate or suggest fixes by observing grammar rules and program context.

Second, semantic checkers and fixers. As described by Section 5.4.2, HA defines semantic rules on segment-specific concepts (e.g., device triggers do not support templating), and segment-agnostic ones (e.g., sensor states should be accessed via the function `state(...)`). To

enforce these rules, future work can create analyzers to traverse the parsing trees mentioned above, gather semantic information like the definition or usage of variables/templates/entities, compare the gathered information with predefined bug patterns, and report a bug for each found pattern-match. The analyzers can also automate or suggest fixes based on predefined bug-fixing patterns, or data analysis of correct automations.

Third, generators of automation configurations. As mentioned in Section 5.4.2, some users have similar automation needs (e.g., to call distinct services based on the fired triggers), and the code to satisfy those needs share commonality (e.g., defining and using trigger IDs). Future work can use data-driven approaches like machine learning and large language models (LLMs) to (1) infer the correspondence between automation needs and YAML code, and (2) generate YAML code given automation specifications. Such tools can be used in combination with the bug detectors and fixers mentioned above, to iteratively refine or optimize automations.

For HA Developers: Users are often confused about the correct usage of some program constructs, built-in functions, or templates (Section 5.4.1). The error messages generated by existing HA checker seem not quite helpful (Section 5.4.3). To help users better adopt HA and further broaden the platform’s impact, HA developers may need to improve documentation, and combine existing concrete YAML examples with a more systematic and comprehensive concept explanation. They may also need to enhance the error-reporting mechanism, to clarify the root causes or even fixes of reported bugs.

For HA Users: Carefully read the available HA documentation to follow best practices and avoid the well-known pitfalls. When asking questions on HA, provide all relevant information (e.g., automation specification, unsuccessful trials, error messages, and help requests) to benefit most from the community wisdom.

5.7 Summary

With the growing prevalence of smart homes, we believe that they will become crucial to lower utility costs, improve people’s lives, and protect people’s properties. Wrongly configured home automation can waste utilities, jeopardize people’s lives, and compromise home safety/security. Our study characterizes the challenges and opportunities in HA-based smart homes, enlightening future research to improve end-user programming and smart-home quality.

Data Availability

Our data is available at <https://figshare.com/s/7aa8ea9f4af98c371114>.

Chapter 6

Conclusion

This dissertation has explored and contributed to the advancement of smart home technology through three distinct yet interconnected projects. First, we developed Building Data Lite (BDL), a cost-effective, and scalable IoT-based infrastructure for collecting and visualizing indoor environmental data. The system was successfully deployed in a real-world case-study to gather an extensive dataset. Second, we leveraged machine learning techniques to streamline the process of building occupant persona generation, demonstrating the potential for data-driven approaches to enhance human-centered building design. Lastly, we conducted an empirical study on smart home automation challenges, particularly focusing on the complexities of YAML-based configuration, revealing significant areas for improvement in existing tools and identifying future research opportunities. The combination of these projects provides a comprehensive framework for addressing critical challenges in smart home development and offers valuable insights for future innovations in this rapidly evolving field.

In the following sections, we will provide a detailed overview of each of these three projects, summarizing the key findings and their implications for the smart home domain. Finally, we will outline potential directions for future work, highlighting opportunities to build upon the foundation established by these projects to further enhance the capabilities and usability of smart home technologies.

6.1 Summary

The three major areas of smart home research are sensing, data utilization, and automation. This research addresses and tries to mitigate the challenges present in all three areas. To address the issue of customized data collection for researchers of different backgrounds, we presented the Building Data Lite system, an open-source IoT-based data collection infrastructure that allows no-code sensing node customization. It is a cost-effective, distributed, scalable, and portable indoor environmental data collection system. In a case study, we evaluated its functionalities by deploying 48 sensing nodes and the system has accumulated over 2 million data points for future research.

Following the development of the data collection infrastructure, this research focuses on the utilization of such data to understand the occupants and their preferences. We utilized machine learning tools to accelerate the process of building occupant persona generation. The current approaches to developing building occupant personas face a major obstruction of manual data processing and analysis. This work proposes and evaluates a machine learning-based semi-automated approach to generate building occupant personas. It leverages the Residential Energy Consumption Dataset from the United States Energy Institute Administration with state-of-the-art machine learning techniques for the prediction of 16 occupant characteristics, such as age, education, and, thermal comfort. The models achieved an average accuracy of 61% and accuracy over 90% for attributes including the number of occupants in the household, their age group, and preferred usage of heating or cooling equipment. The results of the study show the feasibility of using machine learning techniques for the development of building occupant persona to minimize human effort.

The third and last focus of this research is the empirical study to better understand automation configuration issues faced by the open-source developer community for the development of automation scripts in Home Assistant, a popular open-source smart home system. This piece of work investigates the issues related to YAML-based automation configuration by mining the Home Assistant Community forum for relevant discussion threads. It categorizes these issues into three major categories: implementation, optimization, and debugging. It highlights common challenges and resolution strategies. Notably, 68% of the examined issues pertain to debugging, yet existing tools can detect a maximum of 14 issues and resolve none, indicating limited support for users. This research underscores the need for improved tools and sheds light on future directions for smart home development.

6.2 Future Works

The findings presented in this dissertation lead to several directions for future research and development. This section sheds light on a few of the potential challenges we identified that can be explored in future works.

The Building Date Lite (BDL) system could be further enhanced by incorporating additional features, deploying more nodes across diverse environments, and conducting detailed analysis of the collected data to uncover new insights. This would not only improve the system's functionality but also expand its applicability to a broader range of research scenarios.

Conducting a detailed analysis of the data collected by BDL is another crucial direction for future work. The current research has demonstrated the system's ability to collect large volumes of data, but there is still much to be learned from this dataset. Future research

could focus on the exploration and utilization of the collected data, such as using machine learning algorithms to identify patterns or trends, or applying statistical techniques to assess the relationships between different environmental variables. This could lead to new insights into how indoor environments affect occupant health, comfort, and productivity, and could inform the design of more effective smart home systems. Furthermore, sharing the BDL dataset with the broader research community could enable collaborative research and accelerate the development of new technologies and applications in the field of smart home research.

A key challenge in automating occupant persona development is the need to incorporate a broader range of occupant characteristics. The current approach focuses on a limited set of attributes, such as age, education, and thermal comfort. However, there are many other factors that can influence occupant behavior and preferences, such as lifestyle, and health conditions. Incorporating additional characteristics into the machine learning models could lead to more accurate and personalized occupant personas, enabling smart home systems to better meet the needs of diverse populations. This would require the collection of more detailed data on occupant behavior and preferences, as well as the development of new algorithms capable of processing and analyzing this data.

Another important direction for future work is the development of more robust machine learning models for occupant persona development. The models used in the current research achieved a moderate level of accuracy, but there is still room for improvement. Future research could explore the use of advanced techniques such as deep learning, which has shown great potential in other areas of artificial intelligence. Deep learning models are capable of processing large volumes of data and identifying complex patterns, making them well-

suited for the task of occupant persona development. Additionally, future work could focus on developing models that can continuously learn and adapt based on new data, enabling smart home systems to respond in real-time to changes in occupant behavior and preferences.

One potential direction for future work is the design of an automation configuration bug detection plugin for the Home Assistant IDE. This plugin could be designed to improve the identification of and diagnosis of common issues in YAML-based configuration scripts, providing developers with real-time feedback and suggestions for resolving these issues. The plugin could also include features for optimizing automation scripts, such as identifying redundant or conflicting rules, and providing recommendations for improving efficiency and performance. By reducing the time and effort required to configure smart home automations, this tool could significantly enhance the usability and reliability of smart home systems, making them more accessible to a broader range of users.

Integrating real-time environmental sensing and occupant profile data with smart home systems is another promising direction for future research. In the current study, the focus was on collecting and analyzing environmental data, and developing occupant personas based on static datasets. However, there is significant potential to combine these two areas of research, creating smart home systems that can dynamically adjust based on real-time data from both the environment and the occupants. For example, a smart home system could use data from sensors to monitor indoor air quality, and combine this with occupant profile data to adjust ventilation, heating, or cooling systems in real-time, ensuring optimal comfort and health for the occupants. This would represent a significant advancement in smart home technology, enabling the creation of more personalized and adaptive living environments.

6.3 Impact

Our tools and research have implications for the research community and developers of smart home systems. Researchers of different backgrounds can use our tool to create customized sensing systems to collect the data they require in their work without the need for complex coding and setup using the Building Data Lite (BDL) system. The work on building occupant persona development can aid the building simulation and development community in better understanding the needs of the occupants while designing future homes. It will enlighten them with a deep understanding of the needs of the future residents of the housing they design and develop. Developers of smart home systems, like Home Assistant, can be aware of the automation configuration issues and take necessary precautions or look for similar resolution techniques according to their situation following the findings of our empirical study on smart home automation configuration issues. The development of an IDE plugin can improve such automation issue detection and resolution tasks significantly.

Bibliography

- [1] Automation - reference condition template variable in action notification. <https://community.home-assistant.io/t/automation-reference-condition-template-variable-in-action-notification/339858>, 2021.
- [2] Jinja. <https://palletsprojects.com/p/jinja/>, 2022.
- [3] Automation Action Templating. <https://community.home-assistant.io/t/automation-action-templating/409680>, 2022.
- [4] Notify - How to display sensor value in message. <https://community.home-assistant.io/t/notify-how-to-display-sensor-value-in-message/376646>, 2022.
- [5] Exceptions on automation. <https://community.home-assistant.io/t/exceptions-on-automation/409661>, 2022.
- [6] Time Before and After condition, I'm going crazy. <https://community.home-assistant.io/t/time-before-and-after-condition-i-m-going-crazy/416170>, 2022.
- [7] Home Assistant. <https://www.home-assistant.io>, 2022.
- [8] Automation - house thermostat based on phone location - not working. <https://community.home-assistant.io/t/automation-house-thermostat-based-on-phone-location-not-working/401942>, 2022.
- [9] Putting information from sensor into mobile notification. <https://community.home-assistant.io/t/putting-information-from-sensor-into-mobile-notification/264000>, 2022.

- [10] Automation door alarm use template. <https://community.home-assistant.io/t/automation-door-alarm-use-template/439744>, 2022.
- [11] Condition on trigger.entity_id. <https://community.home-assistant.io/t/condition-on-trigger-entity-id/442838>, 2022.
- [12] Boolean triggering script. <https://community.home-assistant.io/t/boolean-triggering-script/404465>, 2022.
- [13] Set counter value equals to other counter in automation. <https://community.home-assistant.io/t/set-counter-value-equals-to-other-counter-in-automation/442485>, 2022.
- [14] Strategies for dealing with restarts in automations. <https://community.home-assistant.io/t/strategies-for-dealing-with-restarts-in-automations/400922>, 2022.
- [15] Script Syntax - Home Assistant. <https://www.home-assistant.io/docs/scripts/>, 2022.
- [16] With 21.1% CAGR, Smart Home Market Worth USD 380.52 Billion in 2028. <https://www.globenewswire.com/en/news-release/2022/06/28/2470249/0/en/With-21-1-CAGR-Smart-Home-Market-Worth-USD-380-52-Billion-in-2028.html>, 2022.
- [17] Cant use template in for field of trigger automation. <https://community.home-assistant.io/t/cant-use-template-in-for-field-of-trigger-automation/400475>, 2022.
- [18] Lights on only after sunset and before sunrise. <https://community.home-assistant.io/t/lights-on-only-after-sunset-and-before-sunrise/307761>, 2022.
- [19] My automation generates an error message. <https://community.home-assistant.io/t/my-automation-generates-an-error-message/434970>, 2022.
- [20] How Popular Is Home Assistant Compared To The Competition? <https://whatsmarthome.com/how-popular-is-home-assistant/>, 2023.

- [21] If starting again what smart home platform would people go with. https://www.reddit.com/r/smarthome/comments/16sk3qg/if_starting_again_what_smart_home_platform_would/, 2023.
- [22] The best home automation systems: Compare SmartThings, Apple HomeKit, Amazon Alexa, and more. <https://www.zdnet.com/home-and-office/smart-home/best-home-automation-system/>, 2023.
- [23] Automation actions - Home Assistant. <https://www.home-assistant.io/docs/automation/action/>, 2024.
- [24] The global smart home market size was valued at \$80.21 billion in 2022 & is projected to grow from \$93.98 billion in 2023 to \$338.28 billion by 2030. <https://www.fortunebusinessinsights.com/industry-reports/smart-home-market-101900>, 2024.
- [25] GitHub. <https://github.com>, 2024.
- [26] Home Assistant Analytics. <https://analytics.home-assistant.io>, 2024.
- [27] Home Assistant Community. <https://community.home-assistant.io>, 2024.
- [28] IFTTT - Automation for business and home. <https://ifttt.com>, 2024.
- [29] Tags - Home Assistant Community. <https://community.home-assistant.io/tags>, 2024.
- [30] Automation Trigger - Home Assistant. <https://www.home-assistant.io/docs/automation/trigger/>, 2024.
- [31] YAML. <https://yaml.org>, 2024.
- [32] Automation Conditions - Home Assistant. <https://www.home-assistant.io/docs/automation/condition/>, 2024.

- [33] CodeBeautify 2023. *CodeBeautify - YAML Validator*, Accessed October 11, 2023. URL <https://codebeautify.org/yaml-validator>.
- [34] U.S. Energy Information Administration. *Residential Energy Consumption Survey Data 2015*, accessed May 31, 2022. URL <https://www.eia.gov/consumption/residential/data/2015>.
- [35] Philip Agee, Xinghua Gao, Frederick Paige, Andrew McCoy, and Brian Kleiner. A human-centred approach to smart housing. *Building Research & Information*, 49(1): 84–99, 2021.
- [36] Shohin Ahelerooff, Xun Xu, Yuqian Lu, Mauricio Aristizabal, Juan Pablo Velásquez, Benjamin Joa, and Yesid Valencia. Iot-enabled smart appliances under industry 4.0: A case study. *Advanced Engineering Informatics*, 43:101043, 2020. ISSN 1474-0346. doi: <https://doi.org/10.1016/j.aei.2020.101043>. URL <https://www.sciencedirect.com/science/article/pii/S1474034620300124>.
- [37] Shohin Ahelerooff, Xun Xu, Ray Y Zhong, and Yuqian Lu. Digital twin as a service (dtaas) in industry 4.0: An architecture reference model. *Advanced Engineering Informatics*, 47:101225, 2021.
- [38] Hamed S Alavi, Elizabeth F Churchill, Mikael Wiberg, Denis Lalanne, Peter Dalsgaard, Ava Fatah gen Schieck, and Yvonne Rogers. Introduction to human-building interaction (hbi) interfacing hci with architecture and urban design, 2019.
- [39] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monroe. Sok: Security evaluation of home-based iot deployments. In *2019 IEEE symposium on security and privacy (sp)*, pages 1362–1380. IEEE, 2019.
- [40] Abhilash Alva, KR Shailesh, and S Tanuja. Design and implementation of video based

- occupancy sensing for adaptive lighting applications. In *2012 International Conference on Emerging Trends in Electrical Engineering and Energy Management (ICETEEEM)*, pages 478–482. IEEE, 2012.
- [41] Kadir Amasyali and Nora M El-Gohary. Real data-driven occupant-behavior optimization for reduced energy consumption and improved comfort. *Applied Energy*, 302: 117276, 2021.
- [42] Sheik Murad Hassan Anik. *BDL Project Repository*, accessed October 1, 2021. URL https://github.com/anik801/data_collection.
- [43] Sheik Murad Hassan Anik, Xinghua Gao, Na Meng, Philip R Agee, and Andrew P McCoy. A cost-effective, scalable, and portable iot data infrastructure for indoor environment sensing. *Journal of Building Engineering*, page 104027, 2022.
- [44] AOSONG. Temperature and Humidity Module, DHT11. *Datasheet*, 2010. ISSN 1098-6596. doi: 10.1093/acprof:oso/9780195179477.003.0005.
- [45] John Bell, John Mabb, Veronica Garcia Hansen, Robert Bergman, and Lidia Morawska. Occupant health and productivity: an australian perspective. In *Summary Book of The CIB 2003 International Conference on Smart and Sustainable Built Environment (SASBE 2003)*, pages 1–8. Queensland University of Technology, 2003.
- [46] Birgitta Berglund, Belt Brunekreef, H Knöppe, T Lindvall, Marco Maroni, Lars Mølhave, and P Skov. Effects of indoor air pollution on human health. *Indoor air*, 2(1): 2–25, 1992.
- [47] Somchai Biansoongnarn, Boonyang Plungkang, and Sriwichai Susuk. Development of low cost vibration sensor network for early warning system of landslides. *Energy Procedia*, 89:417–420, 2016.

- [48] Will Brackenbury, Abhimanyu Deora, Jillian Ritchey, Jason Vallee, Weijia He, Guan Wang, Michael L. Littman, and Blase Ur. How users interpret bugs in trigger-action programming. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, pages 1–12, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359702. doi: 10.1145/3290605.3300782. URL <https://doi.org/10.1145/3290605.3300782>.
- [49] Eric Brangier and Corinne Bornet. Persona: a method to produce representations focused on consumers' needs. In *Human factors and ergonomics in consumer product design*, pages 37–61. CRC Press, 2011.
- [50] Alex H Buckman, Martin Mayfield, and Stephen BM Beck. What is a smart building? *Smart and Sustainable Built Environment*, 2014.
- [51] Giuseppina Buttitta, William JN Turner, Olivier Neu, and Donal P Finn. Development of occupancy-integrated archetypes: Use of data mining clustering techniques to embed occupant behaviour profiles in archetypes. *Energy and Buildings*, 198:84–99, 2019.
- [52] Salvatore Carlucci, Marilena De Simone, Steven K Firth, Mikkel B Kjærgaard, Romana Markovic, Mohammad Saiedur Rahaman, Masab Khalid Annaqeeb, Silvia Biandrate, Anooshmita Das, Jakub Wladyslaw Dziedzic, et al. Modeling occupant behavior in buildings. *Building and Environment*, 174:106768, 2020.
- [53] Z Berkay Celik, Patrick McDaniel, and Gang Tan. Soteria: Automated {IoT} safety and security analysis. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 147–158, 2018.
- [54] YAML Checker. *YAML Checker*, Accessed October 11, 2023. URL <https://yamlchecker.com/>.

- [55] James Coleman, Eric Teitelbaum, Hongshan Guo, Jake Read, and Forrest Meggers. Examining architectural air and temperature with novel sensing techniques. *Energy Procedia*, 122:1135–1140, 2017.
- [56] Drury B Crawley, Linda K Lawrie, Frederick C Winkelmann, Walter F Buhl, Y Joe Huang, Curtis O Pedersen, Richard K Strand, Richard J Liesen, Daniel E Fisher, Michael J Witte, et al. Energyplus: creating a new-generation building energy simulation program. *Energy and buildings*, 33(4):319–331, 2001.
- [57] Marilena De Simone. Modeling occupant behavior and quantifying its impact on building energy use. *Frontiers in Sustainable Cities*, page 62.
- [58] Zhipeng Deng and Qingyan Chen. Artificial neural network models using thermal sensations and occupants’ behavior for predicting thermal comfort. *Energy and Buildings*, 174:587–602, 2018.
- [59] Dalibor D Dvorski. Installing, configuring, and developing with xampp. *Skills Canada*, 2007.
- [60] Tobore Ekwevugbe, Neil Brown, Vijay Pakka, and Denis Fan. Real-time building occupancy sensing using neural-network based sensor network. In *2013 7th IEEE international conference on digital ecosystems and technologies (DEST)*, pages 114–119. IEEE, 2013.
- [61] Sheikh Ferdoush and Xinrong Li. Wireless sensor network system design using raspberry pi and arduino for environmental monitoring applications. *Procedia Computer Science*, 34:103–110, 2014.
- [62] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. Security analysis of emerging

- smart home applications. In *2016 IEEE symposium on security and privacy (SP)*, pages 636–654. IEEE, 2016.
- [63] JSON Formatter. *JSON Formatter - YAML Validator*, Accessed October 11, 2023. URL <https://jsonformatter.org/yaml-validator>.
- [64] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *ICML*, 1996.
- [65] Lirong Fu, Shouling Ji, Kangjie Lu, Peiyu Liu, Xuhong Zhang, Yuxuan Duan, Zihui Zhang, Wenzhi Chen, and Yanjun Wu. Cpscan: Detecting bugs caused by code pruning in iot kernels. *CCS '21*, pages 794–810, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384544. doi: 10.1145/3460120.3484738. URL <https://doi.org/10.1145/3460120.3484738>.
- [66] Weijia He, Jesse Martinez, Roshni Padhi, Lefan Zhang, and Blase Ur. When smart devices are stupid: Negative experiences using home smart devices. In *2019 IEEE Security and Privacy Workshops (SPW)*, pages 150–155, 2019. doi: 10.1109/SPW.2019.00036.
- [67] Brawner Brian L Heyasa and R Galarpe Van Ryan Kristopher. Initial development and testing of microcontroller-mq2 gas sensorfor university air quality monitoring. *IOSR Journal of Electrical and Electronics Engineering (IOSR-JEEE)*, 12(3):47–53, 2017.
- [68] Mohammad Hossin and Md Nasir Sulaiman. A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process*, 5(2):1, 2015.
- [69] Bing Huang, Dipankar Chaki, Athman Bouguettaya, and Kwok-Yan Lam. A survey

- on conflict detection in iot-based smart homes. *ACM Comput. Surv.*, 56(5), nov 2023. ISSN 0360-0300. doi: 10.1145/3629517. URL <https://doi.org/10.1145/3629517>.
- [70] Terence KL Hui, R Simon Sherratt, and Daniel Díaz Sánchez. Major requirements for building smart homes in smart cities based on internet of things technologies. *Future Generation Computer Systems*, 76:358–369, 2017.
- [71] Mengda Jia, Ravi S Srinivasan, Robert Ries, and Gnana Bharathy. A framework of occupant behavior modeling and data sensing for improving building energy simulation. In *Proceedings of the Symposium on Simulation for Architecture and Urban Design*, pages 1–8, 2018.
- [72] John D Kelleher, Brian Mac Namee, and Aoife D’arcy. *Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies*. MIT press, 2020.
- [73] Emilien Kenler and Federico Razzoli. *MariaDB Essentials*. Packt Publishing Ltd, 2015.
- [74] Neil E Klepeis, William C Nelson, Wayne R Ott, John P Robinson, Andy M Tsang, Paul Switzer, Joseph V Behar, Stephen C Hern, and William H Engelmann. The national human activity pattern survey (nhaps): a resource for assessing exposure to environmental pollutants. *Journal of Exposure Science & Environmental Epidemiology*, 11(3):231–252, 2001.
- [75] Sotiris B Kotsiantis, Ioannis D Zaharakis, and Panayiotis E Pintelas. Machine learning: a review of classification and combining techniques. *Artificial Intelligence Review*, 26(3):159–190, 2006.
- [76] Karthik Krishnamurthi, Suraj Thapa, Lokesh Kothari, and Arun Prakash. Arduino

- based weather monitoring system. *International Journal of Engineering Research and General Science*, 3(2):452–458, 2015.
- [77] Shiu Kumar and Seong Ro Lee. Android based smart home system with control via bluetooth and internet connectivity. In *The 18th IEEE International Symposium on Consumer Electronics (ISCE 2014)*, pages 1–2. IEEE, 2014.
- [78] Jorma Laaksonen and Erkki Oja. Classification with learning k-nearest neighbors. In *Proceedings of International Conference on Neural Networks (ICNN’96)*, volume 3, pages 1480–1483. IEEE, 1996.
- [79] Xinyi Li, Lei Zhang, and Xipeng Shen. Diac: An inter-app conflicts detector for open iot systems. *ACM Trans. Embed. Comput. Syst.*, 19(6), oct 2020. ISSN 1539-9087. doi: 10.1145/3391895. URL <https://doi.org/10.1145/3391895>.
- [80] Hongliang Liang, Qian Zhao, Yuying Wang, and Haifeng Liu. Understanding and detecting performance and security bugs in iot oses. In *2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 413–418. IEEE, 2016.
- [81] Zhiying Lin, Runwei Yang, Kaishu Li, Guozhong Yi, Zhiyong Li, Jinglin Guo, Zhou Zhang, Peng Junxiang, Yawei Liu, Songtao Qi, et al. Establishment of age group classification for risk stratification in glioma patients. *BMC neurology*, 20(1):1–11, 2020.
- [82] Lucy Linder, Damien Vionnet, Jean-Philippe Bacher, and Jean Hennebert. Big building data-a big data platform for smart buildings. *Energy Procedia*, 122:589–594, 2017.
- [83] YAML Lint. *YAML Lint*, Accessed October 11, 2023. URL <https://www.yamllint.com/>.

- [84] Purdy Lounge. *OpenHAB Vs Home Assistant: Detailed Comparison By An Expert in 2022*, Retrieved September 12, 2022. URL <https://purdylounge.com/openhab-vs-home-assistant/>.
- [85] Md Anam Mahmud, Kyle Bates, Trent Wood, Ahmed Abdelgawad, and Kumar Yelamathi. A complete internet of things (iot) platform for structural health monitoring (shm). In *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, pages 275–279. IEEE, 2018.
- [86] Amir Makhshari and Ali Mesbah. Iot bugs and development challenges. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 460–472. IEEE, 2021.
- [87] Jeetika Malik, Ronita Bardhan, Tianzhen Hong, and Mary Ann Piette. Developing occupant archetypes within urban low-income housing: A case study in mumbai, india. In *Building Simulation*, volume 15, pages 1661–1683. Springer, 2022.
- [88] Gonçalo Marques and Rui Pitarma. A cost-effective air quality supervision solution for enhanced living environments through the internet of things. *Electronics*, 8(2):170, 2019.
- [89] Robert J Meyers, Eric D Williams, and H Scott Matthews. Scoping the potential of monitoring and control technologies to reduce energy use in homes. *Energy and Buildings*, 42(5):563–569, 2010.
- [90] Daniel Minoli, Kazem Sohraby, and Benedict Occhiogrosso. Iot considerations, requirements, and architectures for smart buildings—energy optimization and next-generation building management systems. *IEEE Internet of Things Journal*, 4(1):269–283, 2017.
- [91] Tom M Mitchell. Artificial neural networks. *Machine learning*, 45:81–127, 1997.

- [92] ZhengChun Mo. Intelligent buildings and intelligent agents-a human-centered framework for building controls. In *Proceedings of the 41st SICE Annual Conference. SICE 2002.*, volume 5, pages 3151–3156. IEEE, 2002.
- [93] Audris Mockus, Roy T Fielding, and James Herbsleb. A case study of open source software development: the apache server. In *Proceedings of the 22nd international conference on Software engineering*, pages 263–272, 2000.
- [94] Igor Mujan, Aleksandar S Anđelković, Vladimir Munćan, Miroslav Kljajić, and Dragan Ružić. Influence of indoor environmental quality on human health and productivity-a review. *Journal of cleaner production*, 217:646–657, 2019.
- [95] OpenAI. Chatgpt (version 3.5). <https://chat.openai.com>, 2023.
- [96] Marco A Ortiz and Philomena M Bluysen. Developing home occupant archetypes: First results of mixed-methods study to understand occupant comfort behaviours and energy use in homes. *Building and Environment*, 163:106331, 2019.
- [97] Maja Pantic. Introduction to machine learning & case-based reasoning. *London: Imperial College*, 2005.
- [98] Aakash Parmar, Rakesh Katariya, and Vatsal Patel. A review on random forest: An ensemble classifier. In *International Conference on Intelligent Data Communication Technologies and Internet of Things*, pages 758–763. Springer, 2018.
- [99] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

- [100] Yuzhen Peng, Adam Rysanek, Zoltán Nagy, and Arno Schlüter. Using machine learning techniques for occupancy-prediction-based cooling control in office buildings. *Applied energy*, 211:1343–1358, 2018.
- [101] Pimoroni. *Enviro for Raspberry Pi*, accessed October 1, 2021. URL <https://www.pi-shop.ch/enviro-for-raspberry-pi>.
- [102] ReadyOne. Building automation system cost per square foot, 2017. URL <https://ready.one/building-automation-per-square-foot/>.
- [103] D Robotics. Dht11 humidity & temperature sensor. *DHT11 Datasheet*, 2010.
- [104] Arnab Kumar Saha, Sachet Sircar, Priyasha Chatterjee, Souvik Dutta, Anwesha Mitra, Aiswarya Chatterjee, Soumyo Priyo Chattopadhyay, and Himadri Nath Saha. A raspberry pi controlled cloud based air and sound pollution monitoring system with temperature and humidity sensing. In *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 607–611. IEEE, 2018.
- [105] Robert E Schapire. Explaining adaboost. In *Empirical inference*, pages 37–52. Springer, 2013.
- [106] Michael Schiefer. Smart home definition and security threats. In *2015 ninth international conference on IT security incident management & IT forensics*, pages 114–118. IEEE, 2015.
- [107] Chirag Shah. *A Hands-On Introduction to Data Science*. Cambridge University Press, 2020.
- [108] SmartThings. *SmartThings*, 2024. URL <https://www.smartthings.com/>.

- [109] Ying Song, Fubing Mao, and Qing Liu. Human comfort in indoor environment: a review on assessment criteria, data collection and data analysis methods. *IEEE Access*, 7:119774–119786, 2019.
- [110] Statista. Smart home worldwide - market outlook, 2024. URL <https://www.statista.com/outlook/cmo/smart-home/worldwide>. Accessed: 2024-08-10.
- [111] Husni Teja Sukmana, Muhammad Ghiyast Farisi, and Dewi Khairani. Prototype utilization of pir motion sensor for real time surveillance system and web-enabled lamp automation. In *2015 IEEE Asia Pacific Conference on Wireless and Mobile (AP-WiMob)*, pages 183–187. IEEE, 2015.
- [112] Philip H Swain and Hans Hauska. The decision tree classifier: Design and potential. *IEEE Transactions on Geoscience Electronics*, 15(3):142–147, 1977.
- [113] Aslam Forhad Symon, Nazia Hassan, Humayun Rashid, Iftekhhar Uddin Ahmed, and SM Taslim Reza. Design and development of a smart baby monitoring system based on raspberry pi and pi camera. In *2017 4th International Conference on Advances in Electrical Engineering (ICAEE)*, pages 117–122. IEEE, 2017.
- [114] Ihsan A Taha and Hamzah M Marhoon. Implementation of controlled robot for fire detection and extinguish to closed areas based on arduino. *Telkomnika*, 16(2):654–664, 2018.
- [115] Shun Takai and Kosuke Ishii. A use of subjective clustering to support affinity diagram results in customer needs analysis. *Concurrent Engineering*, 18(2):101–109, 2010.
- [116] Hao Tang, Yong Ding, and Brett Singer. Interactions and comprehensive effect of indoor environmental quality factors on occupant satisfaction. *Building and Environment*, 167:106462, 2020.

- [117] Kwok Wai Tham. Indoor air quality and its effects on humans—a review of challenges and developments in the last 30 years. *Energy and Buildings*, 130:637–650, 2016.
- [118] Alaa Tharwat. Classification assessment methods. *Applied computing and informatics*, 17(1):168–192, 2021.
- [119] Online YAML Tools. *Online YAML Tools*, Accessed October 11, 2023. URL <https://onlineyamltools.com/validate-yaml>.
- [120] Rahmadi Trimananda, Seyed Amir Hossein Aqajari, Jason Chuang, Brian Demsky, Guoqing Harry Xu, and Shan Lu. Understanding and automatically detecting conflicting interactions between smart home iot applications. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1215–1227, 2020.
- [121] D Uk. Humidity & temperature sensor. *DHT11 Datasheet*, page 9, 2010.
- [122] Tao Wang, Kangkang Zhang, Wei Chen, Wensheng Dou, Jiaxin Zhu, Jun Wei, and Tao Huang. Understanding device integration bugs in smart home system. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 429–441, 2022.
- [123] Sholom M Weiss and Ioannis Kapouleas. An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. In *IJCAI*, volume 89, pages 781–787. Citeseer, 1989.
- [124] Petros Xanthopoulos, Panos M Pardalos, and Theodore B Trafalis. Linear discriminant analysis. In *Robust data mining*, pages 27–33. Springer, 2013.
- [125] Fu Xiao, Le-Tian Sha, Zai-Ping Yuan, and Ru-Chuan Wang. Vulhunter: A discovery for unknown bugs based on analysis for known patches in industry internet of things.

- IEEE Transactions on Emerging Topics in Computing*, 8(2):267–279, 2020. doi: 10.1109/TETC.2017.2754103.
- [126] Anuradha Yarlagadda, JVR Murthy, and MHM Krishna Prasad. A novel method for human age group classification based on correlation fractal dimension of facial edges. *Journal of king saud university-computer and information sciences*, 27(4):468–476, 2015.
- [127] Ye Yuan, Liji Wu, and Xiangmin Zhang. Gini-impurity index analysis. *IEEE Transactions on Information Forensics and Security*, 16:3154–3169, 2021.
- [128] Nurul Azma Zakaria, Zaheera Zainal Abidin, Norharyati Harum, Low Chen Hau, Nabeel Salih Ali, and Fairul Azni Jafar. Wireless internet of things-based air quality device for smart pollution monitoring. *Int. J. Adv. Comput. Sci. Appl*, 9:65–69, 2018.
- [129] Yongli Zhang. Support vector machine classification algorithm and its application. In *International conference on information computing and applications*, pages 179–186. Springer, 2012.
- [130] Yang Zhongguo, Li Hongqi, Sikandar Ali, and Ao Yile. Choosing classification algorithms and its optimum parameters based on data set characteristics. *Journal of Computers*, 28(5):26–38, 2017.
- [131] Wei Zhou, Chen Cao, Dongdong Huo, Kai Cheng, Lan Zhang, Le Guan, Tao Liu, Yan Jia, Yaowen Zheng, Yuqing Zhang, Limin Sun, Yazhe Wang, and Peng Liu. Reviewing iot security via logic bugs in iot platforms and systems. *IEEE Internet of Things Journal*, 8(14):11621–11639, 2021. doi: 10.1109/JIOT.2021.3059457.