rIoT: Enabling Seamless Context-Aware Automation in the Internet of Things

Jie Hua[†], Chenguang Liu[†], Tomasz Kalbarczyk[†], Catherine Wright[‡], Gruia-Catalin Roman[‡], Christine Julien[†]
[†]Department of Electrical and Computer Engineering, University of Texas at Austin
{mich94hj, liuchg, tkalbar, c.julien}@utexas.edu
[‡]Department of Computer Science, University of New Mexico
{wrightc, gcroman}@unm.edu

***Abstract*—Advances in mobile computing capabilities and an increasing number of Internet of Things (IoT) devices have enriched the possibilities of the IoT but have also increased the cognitive load required of IoT users. Existing context-aware systems provide various levels of automation in the IoT. Many of these systems adaptively take decisions on how to provide services based on assumptions made *a priori*. The approaches are difficult to personalize to an individual's dynamic environment, and thus today's smart IoT spaces often demand complex and specialized interactions with the user in order to provide tailored services. We propose rIoT, a framework for seamless and personalized automation of human-device interaction in the IoT. rIoT leverages existing technologies to operate across heterogeneous devices and networks to provide a one-stop solution for device interaction in the IoT. We show how rIoT exploits similarities between contexts and employs a decision-tree like method to adaptively capture a user's preferences from a small number of interactions with the IoT space. We measure the performance of rIoT on two real-world data sets and a real mobile device in terms of accuracy, learning speed, and latency in comparison to two state-of-the-art machine learning algorithms.**

***Index Terms*—pervasive computing; smart environments; device discovery and selection;**

I. INTRODUCTION

With recent technology advances made in the Internet-of-Things (IoT), there is a growing number of smart devices helping to build the many smart-* scenarios that people have long envisioned [40]. In scenarios like smart-homes and smart-offices, the plethora of these new devices has created many possibilities for automating daily tasks. At the same time, new challenges arise; a particular challenge to note is that applications demand responsive and intelligent approaches to leverage context [10] in IoT environments. In this work, we address a fundamental piece of this challenge: automating human-device interaction, by asking a simple yet unsolved question: *how can contextual information be leveraged to make IoT device interaction more seamless and personalized*?

To make *seamlessness* and *personalization* concrete, consider a smart home system, embedded with sensors and actuators. Smart lights adjust lighting based on indoor illumination; a smart coffee maker automatically starts coffee when the user wakes up. While these individual applications enable some automation, they do not achieve the full vision of disappearing computer [40]. One gap that remains is directly related to abstractions for user interaction chosen by manufacturers [2], [7]. Simply put, interactions are not *seamless*. At setup, a user usually needs to connect the devices based on mac addresses, name each device, and remember them. To interact with devices, the user either scripts the behavior in advance in arbitrary *computer-friendly* languages (e.g., IFTTT [29], Hydra [12]) or must recall the name defined at setup and issue commands like "set the light over the stove to bright". Neither the scripted behavior nor the tailored commands provide a *seamless* interaction paradigm. We argue that a truly *seamless* IoT world will allow the user to interact with devices using simple and generic instructions like "turn the light on".

On the other hand, a key selling point of IoT applications is the *personalization* they enable by allowing users to customize the configuration with personal preferences. While such features are common, they are often limited and contrived [1]. For instance, although the smart coffee machine may allow the user to configure a customized time to start the coffee machine every day, such "personalization" is under the assumption that the interaction related to this device is based on time. If a user wants to start coffee after returning home from jogging, which may or may not happen every day, the user cannot benefit from the "smartness" of the coffee machine. Personalization in modern IoT systems should not require a user to express her preference via manufacturer defined assumptions.

Our solution to *seamlessness* and *personalization* is through context-awareness. Significant work has been done in context-awareness over the past decade [14], [31], [36], supporting better collecting, processing, and reasoning about *context*. In the IoT, a user's context can include any information that describes the user's situation, from location and time to ambient conditions or the presence of others [11]. We focus on utilizing collected contextual information to predict the device and associated service(s) that a user needs when the user makes a simple generic request (e.g., "turn on the light"). Unlike existing solutions, we respond immediately to users-supplied negative feedback and re-attempt the action.

We propose, **r**IoT, a framework enabling **r**esponsive and **r**einforcing automation in IoT service personalization. **r**IoT enables context-aware automation by providing a *seamless* and *low effort* approach to personalizing how IoT services are chosen to support a given request from a user. In contrast to the common IoT application workflow in which users must exert non-negligible efforts on the process of configuring, labeling,

and specifying a device before using it, **r**IoT incorporates a context learning algorithm that automatically adjusts based on user intentions and the environment. This learning is a continuous process that adaptively evolves a learned model as users change their interaction preferences or the environment changes. **r**IoT does not rely on *a priori* knowledge and learns a user's intentions only from the history of user interactions.

In summary, **r**IoT leverages rich contextual information to enable increased automation of user-oriented IoT device interaction. Our key research contributions are:

- We propose a context-aware learning framework, **r**IoT, for user-oriented IoT device selection.
- We incorporate user configurable context abstractions to enable personalization at per device level.
- We devise a context-aware algorithm that learns a user's interaction pattern with no *a priori* knowledge about the device, space, and user.
- We quantitatively evaluate **r**IoT using two real-world data sets. We show that **r**IoT has high accuracy and can quickly recover from environmental dynamics.

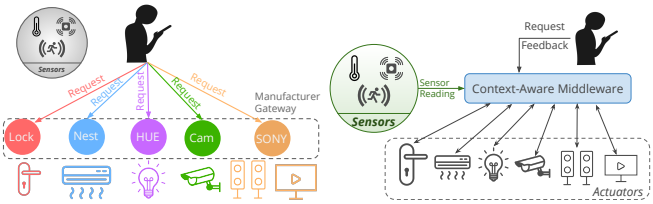
In Section II, we present an overview of the related work and key preliminaries of our proposed approach. We then present an overview of **r**IoT and its position in an IoT deployment. Section IV presents **r**IoT in detail, including the underpinning learning algorithms. We evaluate **r**IoT in Section V, comparing it to two alternative learning algorithms in the context of real-world contextual data. Section VI concludes.

II. Motivation and Related Work

We use state of the art middleware in the IoT to motivate the gap that **r**IoT fills. IoT devices fall into two categories: *sensors* and *actuators*. Users make requests to actuators to take some action (e.g., turn on the lights, adjust the volume, etc.), while sensors passively collect contextual information. A device can take on both roles simultaneously, e.g., a thermostat can both sense temperature and actuate the temperature set point.

Motivating Application Scenario. Alice is a smart home enthusiast who owns several IoT actuators: a smart lock, lights, security cameras, and a stereo system. She is an early adopter who purchases solutions as they become available, so her devices are from five different manufacturers. Alice also has a networked sensor system throughout her home, provided by yet another manufacturer. The stereo system is her favorite because it supports sophisticated collaboration among all of her speakers to provide ideal sound quality. Alice must figure out how to control all of the devices to satisfy her needs but minimize her overhead in interacting with them.

Existing Middleware Architectures for the IoT. Fig. 1 captures the architectures of the two primary control options available to Alice, given today's current technologies. The first, in Fig. 1a, is a *manufacturer-oriented* view in which Alice controls actuators through different manufacturer gateways and their (proprietary) applications. The obvious advantage is that manufacturers can provide comprehensive services for their devices. For Alice, this means she can enjoy the features



(a) Manufacturer oriented **(b)** Middleware oriented

Fig. 1: Existing human-device interaction in the IoT

that achieve ideal sound quality from the stereo system. On the other hand, Alice has to navigate steep and diverse learning curves associated with each of the manufacturers. Although some manufacturers allow users to define personalized automation based on primitive context information like time, they cannot leverage sensor data provided by other manufacturers [6], and as a result, they fail to fully respond to a user's more subtle intentions due to lack of context [41]. In other words, these systems' approaches to "personalization" do not truly reflect the user.

Fig. 1b shows another option in which Alice can employ a general-purpose IoT middleware as a sort of universal gateway (e.g., IFTTT [29], Hydra [12]). The advantage is that Alice only needs to learn one control language that can also leverage contextual data collected by diverse sensors. The disadvantage is that Alice has to define all the interaction patterns by herself using some script language defined by the middleware. Even with current context-aware automation solutions [3], [4], [26], since the control interfaces are designed by a third-party, some device features may not be supported. For example, it may not be feasible for a third-party framework to coordinate multiple speakers to provide manufacture designed sound effect.

Context-Awareness in the IoT. An obvious pain point is the inability of existing middleware to internalize an expressive and complete notion of *context*, a need that has been identified in both the research community [36] and in the industry [9]. Existing work incorporating context-awareness into IoT-like applications adopts a semantic approach [17], [24], [38], [39], where context-awareness relies on a pre-defined ontology characterizing devices, users, and their relationships. In contrast, providing users seamless experiences requires an approach that does not rely on a user having *a priori* knowledge about how IoT devices affect the space in which they are located. This is necessary to ensure the approach is suitable for new spaces a user encounters for the first time or for spaces in which the devices or environment are dynamic.

CA4IOT [31] is a context-aware architecture that selects sensors to provide context based on a *likelihood* index that captures the weighted Euclidean distance from the user's context and the context of the sensors. However, context reasoning is based only on a pre-defined static distance function with fixed contextual inputs. Probabilistic and association based solutions [27], [23] provide efficient activity sensing and fluid device interaction, while other approaches use Hidden Markov Models (HMMs) to model context-awareness [4], [5], [25], [34]. These approaches either require a list of pre-defined "sit-

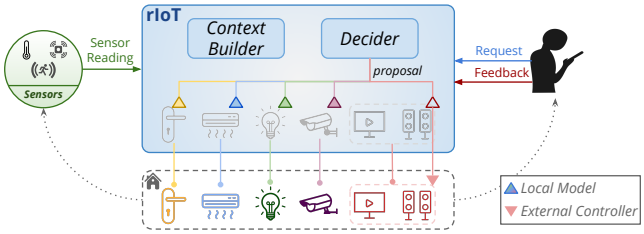


Fig. 2: The overview of the **r**IoT framework

uations" to which they are restricted, or they make assumptions restricting the context and the environment. More recently, deep learning has provided context-aware activity recognition, interaction prediction, and smart space automation [15], [28], [33]. Despite their promise, these approaches require very large data sets for training, which makes them not suitable for personalized approaches in which data is small.

The aforementioned approaches inspire our work. We target a more general space with diverse devices that may dynamically change. The key challenges of *interactive machines* in general [37], articulates a gap between what systems can sense about the context and the user's actual intentions. That is, no matter how many sensors we use to capture context, gaps will exist in the system's knowledge. Therefore, unlike existing solutions, we emphasize that user feedback should be explicitly included in the decision-making process.

System Support for rIoT. Efficiently collecting context has been well studied. Through multi-device cooperation, continuous monitoring systems like CoMon [20] and Remora [18] enable context generated by sensors to be consumed by applications executing on nearby smartphones. Self-organized context neighborhoods [22], [21] built using low-end sensors have negligible communication overhead. It is exactly because of the availability of these cost-effective continuous sensing systems that **r**IoT's vision of IoT personalization can seamlessly incorporate expressive context in an IoT enabled space.

We rely on existing solutions to provide connectivity among heterogeneous IoT devices. The web-of-things [13] makes devices available as web services and thus accessible through a canonical interface. Lightweight solutions [35] opportunistically discover surrounding devices and control them through users' personal devices. In this work, we focus on how to utilize context to better select and control these devices.

III. An Overview of rIoT

In this section, we overview **r**IoT's core contributions and define its underlying key concepts. We describe our algorithms in detail in the following section. Our work targets smart spaces that contain multiple rooms equipped with IoT devices. There may be one or more users sharing the space, however we assume that requests from different users are compatible with each other (e.g., we assume that two users never simultaneously request different actions on the same devices).

In Fig. 1, we identified a trade-off between user-oriented personalization and manufacturer-oriented features. We argue that it is important to enable personalization yet retain the full capabilities of devices. As shown in Fig. 2, **r**IoT inserts itself between applications and IoT devices to allow applications to leverage context to automatically determine which devices and what actions on those devices best match a user's needs and expectations. **r**IoT encapsulates a *context builder* that collects and abstracts sensor readings into high-level, usable context. **r**IoT's *decider* uses context information, knowledge about available IoT devices, and knowledge about the user's prior interactions to choose (i) the best device to fulfill a user's request and (ii) the best action to take on that device.

We assume that users' *requests* for IoT devices to take *actions* may be of varying levels of detail. At one end of the spectrum, a user may ask for a specific device to take a specific action (e.g., "turn off the kitchen light"). At the other end, the user might simply ask for the IoT to "act". In this case, **r**IoT needs to determine which action on which device is most likely to satisfy the request. There are a variety of requests in between; for example, given the request "turn on the light", **r**IoT knows that the right action is to "turn on" and that the *type* of device is "light", but must determine which light device to act on. While we support all levels of specificity, in this paper, we focus primarily on the least specified, i.e., situations in which the user simply says "act", and **r**IoT must select the combination of device and action that best satisfy the user.

rIoT learns a *local utility model* for each IoT device; this model (f_d) captures the likelihood that a given action on that device is the "best" action to take given a snapshot of the context at the moment that the user makes a request. Conceptually, each device proposes the action on that device that has the highest utility in the given context. **r**IoT's *decider* compiles all of the devices' proposals and selects the one with the highest overall utility. Given **r**IoT's choice of action, each device receives *implicit* feedback (i.e., if the device's proposal was selected, the device receives positive feedback; otherwise, the device receives negative feedback). Thus a device can learn about the utility of its actions in the context of other co-located devices. In addition, once the action is taken, **r**IoT allows the user to provide *explicit* feedback to reinforce (either positively or negatively) **r**IoT's selection. The feedback is incorporated into the device's utility model, allowing it to learn over time based on the user's interactions in the space.

rIoT's architecture also allows applications to maintain access to manufacturer-specific actions. In such situations, **r**IoT controls the devices as a *system* through an *external* controller as depicted to the right of Fig. 2. Rather than the individual devices proposing an action, this controller proposes for all devices it controls. This allows exposing manufacturer-specific actions as part of the **r**IoT decision framework.
We next define some terms that we use throughout the paper.

Definition 1. (context $\underline{c}$) Practically, a context c is any single piece of numerical or categorical data and can be raw sensor reading like *temperature* or *illumination level*, or an abstract value derived from raw data, e.g., *isAtHome*, *Cooking*.

Definition 2. (context snapshot $\underline{C_t}$) We define C_t as a vector of context values $c_{t,i}$ that describe the user's situation at time

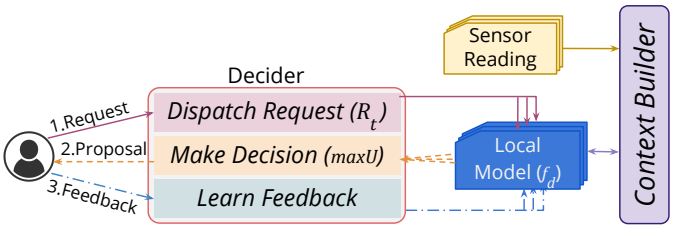


Fig. 3: The overview of data flow in **r**IoT

t, i.e., $C_t = (c_{t,0}, \ldots, c_{t,i}, \ldots, c_{t,n})$. We assume that the i^{th} element of any snapshot is always the same type of context; c_0 is always the user's identity.

Definition 3. (device $\underline{d}$) A device is an actuator that can be discovered and controlled through a virtual controller.

Definition 4. (device class $\underline{T}$) A class T is a set of devices that have the same type, and therefore the same action interface, e.g., $d \in T_{light}$. We assume a hierarchy of classes. For example, a dimmable light is also a light, i.e., $T_{dimmable} \subset T_{light}$.

Definition 5. (action $\underline{a}$) An action is performed by a device, e.g., *turnOn*, *turnOff*, etc. $\underline{A}$ is the set of all actions; A_d is the set of actions device d can perform. We assume A_d is finite.

Definition 6. (request $\underline{R}$) A request R_t made by the user at time t is a pair of class and action, both of which are optional. Specifically, $R_t = \langle T, a \rangle$ indicates that the user wants a device $d \in T$ to do action a. A request's fields can be blank, i.e., $R = \langle \perp, \perp \rangle$, which indicates that the user requests the IoT to "act", or have only one of the two fields, e.g., $R = \langle light, \perp \rangle$ indicates a request for some light to take some action.

Formally, our problem statement is:

> *Given a user's request R_t at time t and a snapshot of the context C_t at the same time t, output a tuple $\langle d, a \rangle$ that specifies the action a to be taken on device d to best satisfy the request R_t*

IV. Context-Aware Decision Models in **r**IoT

We now describe the components and processes that allow us to fill in the architecture in Fig. 2. We then describe **r**IoT's contextually-driven learning algorithm in detail.

A. *rIoT Approach*

As described in the overview, **r**IoT learns a local utility model for each IoT device. Conceptually, these models "belong" to the devices themselves, but, as Fig. 2 shows, the models are part of **r**IoT. The only exceptions are *external* models, which may contain manufacturer-proprietary information; in these cases, the **r**IoT local model is a proxy for the external model, which resides under the manufacturer's purview.

Fig. 3 shows the flow of requests, context, and decisions in **r**IoT. The user (at the left) makes requests to **r**IoT's *decider*, which resolves them using input from the local utility models. These models in turn rely on context snapshots generated by the *context builder*. Given a decision, the user may accept the proposal (providing implicit positive feedback) or reject it (providing explicit negative feedback). This information is used to update the local models. In the case of negative feedback, the *decider* repeats the decision process and makes a new proposal to satisfy the original request.

1) Building Context Snapshots: The *context builder* translates raw sensor data into contextual data, which the local utility models use for learning. We rely on four generic context abstractions: (1) the *numeric context* allows the *context builder* to capture standard numerical values, e.g., temperature, pressure; (2) the *cyclic numeric context* captures context types that are numerical but "roll over" on some predictable schedule, e.g., time, day of the week; (3) the *N-dimensional vector context* captures context values that are represented by a tuple of values, e.g., location coordinates; and (4) the *categorical context* captures labeled values, e.g., human activity, binary data. Depending on the type of context and the available sensors, the *context builder* assembles the higher level values out of the raw sensor data. **r**IoT leverages existing work in context construction to implement the *context builder*.

2) Context Distance Functions: The devices' local utility models will propose actions to take in a given context based on the feedback they have received about prior actions in the same or similar contexts. To judge the similarity of two contexts, **r**IoT relies on context distance functions. We first define $dist(c, c')$, or the distance between two contexts (e.g., the distance between two locations, the distance between two temperature values, etc.). Primitive context types typically have easily defined distance function (e.g., geometric distance, absolute value, cyclical distance). **r**IoT makes one additional constraint on any $dist(c, c')$, i.e., that the distance is normalized to the range $[0, 1]$. With this simple definition of contextual distance, we build a distance function $dist(C_a, C_b, W)$ that captures the distance between two context snapshots.

Definition 7. (context snapshot distance $\underline{dist(C_a, C_b, W)}$) This distance is computed using the Manhattan distance [19]; the vector W weights the elemental values of the snapshots:

$$dist(C_a, C_b, W) = \sum_{i=0}^{n} w_i \times dist(c_{a,i}, c_{b,i})$$

$$\text{where, } C_a = (c_{a,0}, c_{a,1}, \ldots, c_{a,n}), C_b = (c_{b,0}, c_{b,1}, \ldots, c_{b,n}),$$
$$W = (w_0, w_1, \ldots, w_n), (0 < w_i < 1)$$

Because the weight vector W is an input to the function, each local model can use a different distance function for context snapshots, enabling personalization. For instance, some users may have a strict daily routine based on the clock, in which case a difference in time means more for this user than other context types. An interaction with lights is more likely based on location, while requests for a remote camera may depend more on suspicious sounds or movements.

Because context values can be continuous, it can be useful to discretize context snapshots into buckets. When we do so, we need to ask whether a bucket "contains" a context snapshot. We define $contain(c^l, c^u, c)$ for the first three elemental context types as $c^l \leq c \leq c^u$ and as $c \in (c^l \cup c^u)$ for categorical context. We next extend this to context snapshots:

Definition 8. (context snapshot contains $\underline{contain(C_a, C_b, C_x)}$) This function simply requires the contain function for all of the elements of the snapshot to be true:

$$contain(C_a, C_b, C_x) = \begin{cases} \text{FALSE, if } \exists i, \text{ s.t. } contain(c_{ai}, c_{bi}, c_{xi}) = \text{FALSE} \\ \text{TRUE, otherwise} \end{cases}$$

3) Defining and Using Local Utility Models: **r**IoT's local utility models capture the suitability of devices' actions to requests from users in certain context states.

Definition 9. (local model $\underline{f_d}$) $f_d \colon C \times R \to A \times \mathbb{R}$ maps a request and context snapshot onto an action the device can take and a *utility value*, $u \in [0, 1]$. The utility captures the likelihood that the action is the "right" one given the request and context. $f_d(C, R)$ results in a proposal $P_d = \langle d, a, u \rangle$.

When a user requests an action, **r**IoT captures the context and requests a proposal from each device's local utility model. **r**IoT's objective is to output the final decision $P_{R_t, C_t} = \langle d, a, u \rangle$, which is the winning proposal, i.e., the proposal with the maximum utility across all of the devices' proposals.

rIoT's key challenge is therefore how to compute the f_d models for each device d. Because we do not want to make any assumptions about or place any constraints on the environment in which **r**IoT operates, our approach is to fit the f_d models using each user's history of interactions with each device, leveraging similarities among the contexts of the interactions.

B. Context Learning in rIoT

Imagine that Alice has four IoT lighting devices in her home. Depending on the context, a "turn light on" request may indicate a desire to turn on any one of these lights. For instance, when Alice awakens at 6:30am and says "turn light on", she intends to illuminate the bedroom. While cooking in the kitchen (regardless of the time of day), a "turn light on" request should control the kitchen light. And when Alice is reading anywhere in her home, "turn light on" should affect the light closest to her current location. Alice's context snapshots may contain, for example, time (captured as a cyclic numeric context), Alice's location (captured, in a coordinate system), and activity (categorical context).

The goal of **r**IoT is to learn each local utility function $f_d(C, R)$ based on the user's interactions and feedback. However, learning this function directly can be challenging. Context can be continuous, and the target function may be non-linear and non-continuous. Therefore, rather than learning $f_d(C, R)$ directly, we introduce a piecewise function set as an approximation, based on a concept we call *state*:

Definition 10. (state $\underline{S}$) A state is defined by three context snapshots, $S = (C_{min}, C_{max}, C_{mid})$. We use two functions to compare states: $contain(S, C_x) \equiv contain(C_{min}, C_{max}, C_x)$, determines whether a state contains a given context snapshot, while $dist(S, C_x, W) \equiv dist(C_{mid}, C_x, W)$ computes how far a context snapshot is from the state. We define the *radius* of a state as $r_S \equiv dist(C_{max}, C_{min})/2$.

Algorithm 1: Computing the Local Model

```
S: set of known states, initially empty
f_d(C, R): set of piecewise local functions, initially empty
Function onReceiveRequest: R = ⟨T, a⟩, C_t
    if (T ≠ ⊥ ∧ d ∉ T) or (a_req ≠ ⊥ ∧ a_req ∉ A_d) then
        return ⟨a_req, 0⟩
    end
    if a = ⊥ then let A_R ← A_d else let A_R ← {a}
    if ∄S_i ∈ S, s.t. contain(S_i, C_t) = TRUE then
        S_new ← (C_t − r, C_t + r, C_t)
        if S = ∅ then ∀a ∈ A_d, f̂_{d,a}(S_new) ← 0.5
        else ∀a ∈ A_d, initialize f̂_{d,a}(S_new) from neighborhood
        S ← S ∪ {S_new}
        f_d(C, R) ← f_d(C, R) ∪ {f̂_{d,a}(S_new) : a ∈ A_d}
    end
    let S_R ← {S_i : contain(S_i, C_t) = TRUE}
    let u_max ← ⟨max_{a∈A_R, S_i∈S_R} : f̂_{d,a}(S_i)⟩
    let a_max ← a s.t. f̂_{d,a}(S_i) = u_max
    return P_d = ⟨d, a_max, u_max⟩
end
Function onFeedback: P = ⟨d, a, u⟩, C_t, feedback
    for S_i ∈ S s.t. contain(S_i, C_t) = TRUE do
        if feedback is positive then
            f̂_{d,a}(S_i) ← f̂_{d,a}(S_i) + reward
        else
            f̂_{d,a}(S_i) ← f̂_{d,a}(S_i) − reward
        end
    end
end
```

The set of states that discretizes the space of context snapshots is not defined *a priori* but are learned over time. The learned states need not cover the entire space of possible context snapshots, and different devices can have different relevant states. In our scenario, Alice's lights may learn states defined by ranges of time and activity labels; her living room lights may not learn anything about states in the very early morning, though her bedroom lights will.

We use this concept to approximate each device's local utility model by combining a utility learned for each state.

Definition 11. (local utility of a state, $\underline{\hat{f}_{d,a}(S)}$) Each function $\hat{f}_{d,a} = u \colon S \to \mathbb{R}$ captures the utility of taking action a on d when the context is contained by S. The default value is 0.5 which means the likelihood action a is a good choice is 50%.

Given a request $R = \langle T, a_{req} \rangle$ in which T is a (potentially empty: $T = \perp$) device type and a_{req} is a (potentially empty: $a_{req} = \perp$) requested action, device d's local utility function $f_{d,a}(C, R)$ can then be approximated as:

$$A_R = \begin{cases} A_d, \text{ if } a_{req} = \perp \\ a_{req}, \text{ otherwise} \end{cases}$$

$$u_{max} = \langle \max a, S : contain(S, C) \wedge a \in A_R :: \hat{f}_{d,a}(S) \rangle$$

$$f_{d,a}(C, \langle T, a_{req} \rangle) = \begin{cases} \langle a_{max}, u_{max} \rangle, \text{if} (T = \perp \vee d \in T) \wedge \\ \qquad (a_{req} = \perp \vee a_{req} \in A_d) \\ \langle a_{req}, 0 \rangle, \text{ otherwise} \end{cases}$$

Algorithm 1 shows our approach. When receiving a request

$R = \langle T, a \rangle$, if C_t is not "contained" in any states, **r**IoT will create a state with a default radius r around C_t (Line 9). This new state's utility is the default (Line 10) or initialized based on other "nearby" states (Line 11; explained in detail below). Once C_t is "contained" in a known state, **r**IoT will output the action that has the highest utility of all actions and is compatible with the request R (Lines 15-18). When receiving feedback from the user, **r**IoT takes the prior proposal P and the prior context, C_t, and updates the local model. The $reward$ is computed using the *Sigmoid* function; we translate the utility from $(0, 1)$ to $(-\infty, +\infty)$, increment or decrement it by a constant $reward$, and then translate it back to $(0, 1)$. This adjusts the utility more slowly when it is close to 0 or 1.

1) Initializing Models from Nearby States: We assume that a user will have similar behaviors in similar contexts [31]. Therefore, when initializing a new state S, **r**IoT computes the initial utility values based on utility values for nearby states. For example, if Alice's actions routinely trigger the bedroom light at 6:30am, even the first time she requests a light at 7:00am, it is likely that she also wants the bedroom light. To capture this "nearness" in **r**IoT, the first time a user makes a request in a new state, we use the k learned states that are closest to the new context and use their learned models as a basis for the new state's utility value; a state's contribution is weighted by its distance to the new context. More formally:

$$\hat{f}_{d,a}(S_{new}) = 0.5 + \frac{1}{k} \sum_{S_i \in \mathbf{S}_{kNN}} \frac{\hat{f}_{d,a}(S_i) - 0.5}{dist(S_i, C_{new}, W_d)/r_{S_{new}} + 1}$$

Since 0.5 is the default value, we first shift the utility value to it and weight it with the distance plus one to make sure that the derived utility value is closer to 0.5 than the original value to prevent initializing the new state with too much confidence.

Although the assumption that behaviors in "nearby" states are likely to be similar is valid in general, this assumption can sometimes be misleading, for example, when there is a wall between two locations. In such a case, the utility value we "borrow" from the neighborhood can be very wrong. Therefore, we add a flag to any new state S_{new}. If the user gives negative feedback immediately following initialization, we abandon the initialized utility value and reset it to 0.5.

2) Decision-Tree Like State Splitting: **r**IoT *splits* states as its models learn more nuanced behaviors of users. We might initially learn the difference in behaviors between morning and afternoon; over time, differences between early morning and late morning might become apparent. Our approach is inspired by the splitting used in the ID3 decision tree algorithm [32]. To detect candidate states for splitting, we define *entropy* using the feedback the device has received for actions taken in each state. Conceptually, entropy captures how internally different the feedback is for a given state. If a state has an even mix of positive and negative feedback, it gives "wishy-washy" proposals; it therefore makes sense to look for a way to split the state into two that are more internally consistent.

Definition 12. (feedback cache $\underline{\Phi_S}$) For each state, a device stores a feedback cache $\Phi_S = \{(C_1, P_1, b_1), (C_2, P_2, b_2), \ldots\}$.

Algorithm 2: State Splitting

```
20 Function ONFEEDBACK: P = ⟨d, a, u⟩, C_t, feedback
21     for S_i ∈ S s.t. contain(S_i, C_t) = TRUE do
           ... as in Algorithm 1...
27         Φ ← Φ ∪ {(C_t, P, feedback)}
28         if E_{S_i} > threshold then
29             Ψ ← {(S_1, S_2) : all pairs of S_1 and S_2 split S_i
                 along every context value}
30             maxGain ← max_{(S_1,S_2)∈Ψ} (E_{S_i} − (E_{S_1} + E_{S_2}))
31             if maxGain > requiredGain then
32                 Split S_i into S_1, S_2
33             end
34         end
35     end
36 end
```

Each element $\phi_i \in \Phi_S$ contains the context in which an action was recommended (C_i), the proposal that was made by the device (P_i), and the (boolean) feedback (b_i).

Definition 13. (entropy $\underline{E_S}$) To compute entropy, we first compute the fraction of positive (and negative) feedback for each action $a \in A_d$:

$$p_{pos}(S, a) = \frac{|\{\phi : \phi \in \Phi_S \wedge \phi(a) = a \wedge \phi(b) = 1\}|}{|\{\phi : \phi \in \Phi_S \wedge \phi(a) = a\}|}$$

We use $\phi(a)$ to indicate the action associated with the proposal in ϕ and $\phi(b)$ to indicate the boolean feedback associated with ϕ. The value of $p_{neg}(S, a)$ is defined similarly but constraints $\phi(b)$ to be 0. We then compute the entropy of state S as:

$$E_S = \max_{a \in A_d} \{ - p_{pos}(S, a) \log(p_{pos}(S, a)) - p_{neg}(S, a) \log(p_{neg}(S, a)) \}$$

Intuitively, the more similar p_{pos} and p_{neg} are, the more "indecisive" the state is, and the higher the entropy value.

Algorithm 2 shows how we extend the ONFEEDBACK function to split states. After a new piece of feedback is incorporated into the local model and added to the feedback cache, the new entropy of the state is computed. If the entropy exceeds a *threshold*, then **r**IoT determines whether a split of the state S_i is preferable. Specifically, **r**IoT mimics the ID3 algorithm, using context as the attribute for splitting [32]. In **r**IoT, each split will be based on a single context. We compute all pairs (S_1, S_2) that split S_i according to context value. For categorical context, we simply split along all possible categories. For continuous contexts, we use a constant number of split points for each context value, splitting the range (C_{min}, C_{max}) evenly, given the target number of split points (Line 29). For each pair (S_1, S_2), we compute the information gain of splitting, which is defined as the entropy of the original state (S_i) minus the sum of the entropies of the two new states after splitting (Line 30); we choose the split (S_1, S_2) that has the maximal information gain (*maxGain*) as the final candidate. To avoid overfitting, we use a second threshold, *requiredGain*, as a required lower bound for information gain to achieve before we split (Line 31).

V. Evaluation

In this section, we evaluate **r**IoT on two real-world data sets [8][1]. We sought to answer the following questions:

1) How does **r**IoT perform on (noisy) real-world data?
2) How sensitive is **r**IoT to the specificity of context types?
3) How do request details assist **r**IoT in decision making?
4) How resilient is **r**IoT to dynamics?
5) Is it feasible for **r**IoT to run on mobile devices?

In our experiments, we compare the performance of **r**IoT's algorithms to two machine learning approaches: random forest (RF) [16] and multi-layer perceptron (MLP) [30].

A. *rIoT in the Real World*

In our first experiment, we use a data set (HH118) generated from seven months of interactions of an adult living in a smart home. The data set includes data from 18 infrared motion sensors which each generate a binary output indicating whether a human is present. From the data set, we derived context values from the raw sensor readings:

- **current location** [categorical context] – a label associated with the most recently activated motion sensor
- **previous location** [categorical context] – a label associated with the most recently de-activated motion sensor
- **second of the day** [cyclic numeric context] – the time

The home contained 10 light devices. We used the dataset's light on/off events to construct requests in **r**IoT. All such generated requests are the least specified type $R = \langle \perp, \perp \rangle$, meaning the user simply requests the environment to "act" without providing a device type or action to take, however since there are only light devices in this experiment, the request is the same with or without device type. We compared **r**IoT with MLP, RF, and a baseline that always selects the light closest to the user's location. For ground truth, we used the light that was actually chosen by the user in the dataset.

Among the alternatives, only **r**IoT supports re-selection after negative feedback. Therefore, we use a metric, **First Decision Accuracy (FDA)** that evaluates the success of the first decision made by each approach; we define **FDA** as *the percentage of requests that are satisfied by the first decision.*

Fig. 4 shows the results. For **r**IoT, we compute the accumulated **FDA** at each request. For RF and MLP, we re-train the model after every 100 requests. The hyper-parameters for RF and MLP are tuned during each training process. **r**IoT not only has the overall best **FDA** at 82.35%, but it also learns significantly faster than the alternatives. **r**IoT requires 2059 fewer requests (about 50 days) than RF to reach 80% accuracy and 863 fewer requests (about 20 days) than MLP to reach 70%. In addition, **r**IoT provides immediate correction if the user gives negative feedback; in this experiment, 98.5% of the requests are satisfied by the first two decisions in **r**IoT.

B. *rIoT's Sensitivity to Context and Request Detail*

We next experiment with different contexts and requests to show that when the user provides more information, **r**IoT can opportunistically improve its performance.

[1]The data sets (HH118, HH107) are at http://casas.wsu.edu/datasets/.

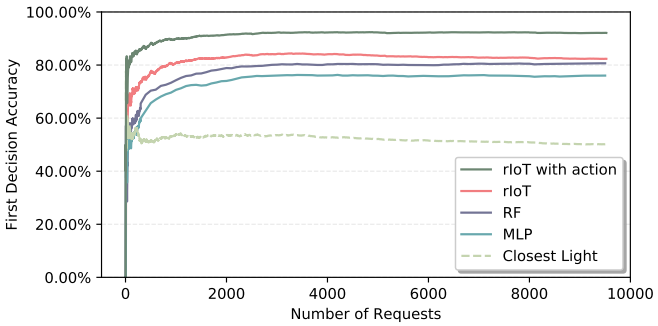


Fig. 4: *Experiments 1 and 2.* The figure shows the learning curves for each approach on the sequence of 9523 requests we derived from the data set. The curve "**r**IoT with action" is the result when the request states the action ("on" or "off") to take. **r**IoT learns faster and has higher overall accuracy.

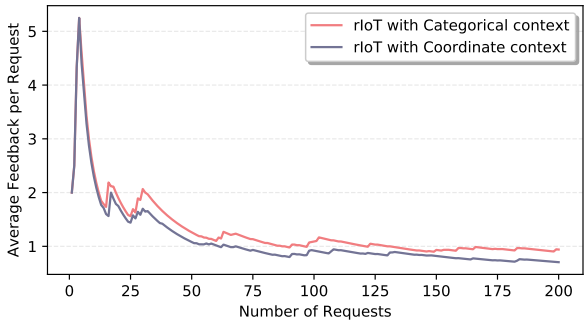


Fig. 5: *Experiment 3.* This figure shows the impact of context specificity on **r**IoT by contrasting the use of categorical location context versus coordinate context. The user needs to give fewer negative pieces of feedback for coordinate context because **r**IoT is able to learn from similar requests.

1) Request Specificity: We first demonstrate how providing more detailed requests assists **r**IoT. In particular, when the user specifies the particular action to take (e.g., "on" or "off") **r**IoT's achieved **FDA** improves (see Fig. 4). With the specificity of the request, the overall **FDA** is 92.10%, compared to 82.35% when the specific action is not provided.

2) Representing the User's Location: Although motion sensor readings can categorically label the location of the user, this sacrifices the ability to compare locations, which in turn hinders **r**IoT's ability to leverage similarities between contexts. Therefore, we evaluate the performance of **r**IoT in the same setup as above but with location represented as a two-dimensional coordinate rather than a label. We compute the **Average Feedback per Request (AFR)**, defined as *the average number of negative feedbacks the user gives per request* before **r**IoT selects the right device.

Fig. 5 shows the results for the first 200 requests. In the first several requests, the two options have the same **AFR** because **r**IoT has not yet learned from feedback. However, after 11 requests, **r**IoT with coordinate context initialized new states based on information from similar contexts. Quantitatively, **r**IoT with categorical context requires the user to give 39 more pieces of feedback in the first 200 requests, which increases

the user's overhead and frustration in employing **r**IoT.

C. *rIoT's Resilience to Dynamics*

Our next goal was to determine how **r**IoT responds to changes in the IoT deployment. We generated simulated contexts and requests so that we could inject uncertainty in sensor readings and changes in IoT devices. We simulated a user interacting with lights, cameras, and speakers based on the real activities in our second dataset (HH117). We created 6393 requests from 26 different daily activities performed over one month by two real people; the labeled activities included sleeping, bathing, cooking, reading, working, watching TV, etc. We created ground truth of how each user interacts with devices during these activities. For example, we assumed that, when the user is engaged in the bathing activity, the user will turn on the bathroom light or when watching TV in the living room, the user will turn on the living room speaker. We similarly assigned each activity to a location context based on the particular activity. If an activity had multiple possible locations (i.e., reading happens both in the living room and bedroom), we used the motion sensor reading in the original dataset to decide which room is the true location for the instance of the activity. We built each context snapshot using the time of the activity and a random x-y coordinate sampled from the room in which the activity was occurring. To simulate habitual interaction patterns, we added three activity types:

- **Wake up**: the *wake up* activity was added after every *sleep* activity. Our simulated user desired the bedroom speaker to play soft music upon waking.
- **Evening news**: the *evening news* activity was added every day the user was home at 18:00. Our simulated user desired to turn on the living room speakers no matter where he was located or what else he was doing.
- **Doorbell**: we added a randomly generated *doorbell*; when this event occurred, our simulated user would desire the doorbell camera to be turned on.

All the requests specified the device type that the user wants to use and related action, e.g. $R = \langle light, turnOn \rangle$; **r**IoT's task was therefore to determine the most appropriate device.

Before considering dynamics, we compared the overall **FDA** of the three approaches; the accuracies of **r**IoT, MLP, and RF were 98.92%, 84.06%, and 84.12% respectively. With no artificial inaccuracies in the contextual data, **r**IoT very closely captures all of the users' interaction patterns.

To test responsiveness to dynamics, we simulated the situation when the user moves the devices in her home by switching the dining room light and the doorway light midway through the experiment, after all algorithms have initially converged. We use the average of the **FDA** of the previous 30 requests to represent the instantaneous accuracy at this request; recall that the **FDA** is the probability that this request is satisfied by each algorithm's first decision. Fig. 6 shows the results.

The accuracy of all three approaches drops dramatically immediately following the switch. **r**IoT recovers significantly faster and restabilizes to a high accuracy. This quick recovery is because **r**IoT recomputes the utility value of the affected

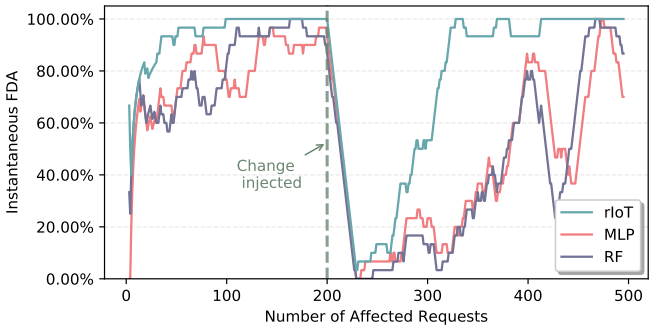


Fig. 6: *Experiment 4.* This figure demonstrates that **r**IoT can rebound from dramatic changes. The graph shows the FDA of **r**IoT, MLP, and RF when the user completely swaps two IoT devices in her environment after 200 requests. **r**IoT is much more agile in responding to the change.

models if it detects a disparity of between recent feedback and the previous learned model. Although it may seem faster, clearing the model and learning from scratch would harm the accuracy of unaffected requests. Further, there is no oracle that indicates that a dramatic change has occurred and thus a reset is in order. To the best of our knowledge, existing context-aware approaches do not consider environmental change explicitly, and thus must either retrain the entire model or wait until the new interactions dominate the old ones.

D. *rIoT's Feasibility on Mobile Devices*

We implemented **r**IoT on Android to demonstrate its feasibility on real IoT devices. All of our measurements are made using a Moto X (2nd Gen.) with Android 5.1 Lollipop.

In automating device interactions, **r**IoT does incur overhead. We measured both the response latency and the feedback latency in **r**IoT. The former is the time between a user issuing a request and **r**IoT responding with a proposal. The feedback latency is measured as the time between when **r**IoT receives a piece of feedback until it finishes updating the local utility models. We used the same requests and context snapshots as in Section V-A. To test how the latency is related to the number of contexts, we added random numerical contexts to the original contexts; the number of total contexts is increased from 3 to 33; measurements are averaged over all requests.

Fig. 7 shows the results. Response latency blocks the UI; it is below 400ms even with the maximal number of contexts. Incorporating feedback into the models can be scheduled in the background, but even so, the overhead is reasonable. Note that increasing the number of contexts does not always increase the latency; for non-random contexts (e.g., the binary types in Fig. 7), **r**IoT learns the model more quickly, so requests and feedback can be processed more efficiently.

VI. Conclusion and Future Work

In this paper, we demonstrated **r**IoT's ability to provide context-aware automation and personalized human-device interaction with no setup cost from the user. **r**IoT ensures user-oriented decision making by explicitly including and responding to a user's immediate feedback. We evaluated **r**IoT

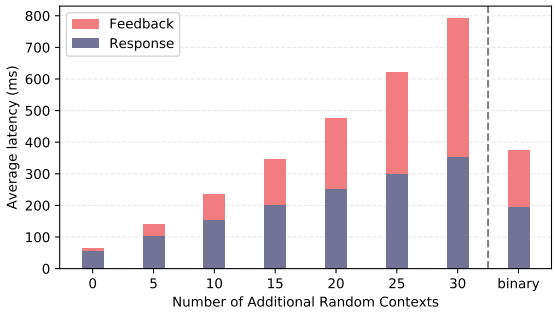


Fig. 7: *Experiment 5.* This figure shows the average response and feedback latency per request given an increasing number of contexts. The "binary" bar use 36 context attributes with only two possible values. In all cases, but especially for realistic numbers of contexts, **r**IoT is feasible on real devices.

on both real-world noisy data and simulated scenarios to show that **r**IoT performs well in terms of its accuracy and is resilient to environmental changes, which are common in the mobile IoT world. **r**IoT is also sufficiently lightweight and efficient to run on mobile devices. However, work with **r**IoT is not complete. For instance, our presentation of **r**IoT assumes that each request corresponds to one action on one device. In practice, a user may expect multiple devices to act together. An extension of **r**IoT could allow devices to coordinate to submit shared proposals. In addition, interfacing **r**IoT with existing smart home platforms like Google Smart Home[2] is also worth researching. In conclusion, **r**IoT opens new possibilities of an IoT world that is truly personalized, providing users seamless and intuitive interactions with the digitized world around them.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. CNS-1703497.

References

[1] M. Black et al. The Internet of Things: Can it find a foothold with mainstream audiences today. Technical report, Nielsen, 2014.
[2] A.J. Brush, M. Hazas, and J. Albrecht. Smart homes: Undeniable reality or always just around the corner? *IEEE Pervasive Comp.*, 17(1), 2018.
[3] L. Capra, W. Emmerich, and C. Mascolo. Carisma: Context-aware reflective middleware system for mobile applications. *IEEE Transactions on Software Engineering*, 29(10):929–945, 2003.
[4] P. Chahuara, F. Portet, and M. Vacher. Context-aware decision making under uncertainty for voice-based control of smart home. *Expert Systems with Applications*, 75:63–79, 2017.
[5] Y. Chen, J. Zhou, and M. Guo. A context-aware search system for internet of things based on hierarchical context model. *Telecommunication Systems*, 62(1):77–91, 2016.
[6] I. Chew et al. Smart lighting: The way forward? reviewing the past to shape the future. *Energy and Buildings*, 149:180–191, 2017.
[7] M. Clark, M. W. Newman, and P. Dutta. Devices and data and agents, oh my: How smart home abstractions prime end-user mental models. *Proc. of the ACM on IMWUT*, 1(3):44, 2017.
[8] D. Cook et al. Casas: A smart home in a box. *Computer*, 46(7), 2012.
[9] L. Da Xu, W. He, and S. Li. Internet of things in industries: A survey. *IEEE Trans. on Industrial Info.*, 10(4):2233–2243, 2014.
[10] A. K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001.
[11] P. Dourish. What we talk about when we talk about context. *Personal and Ubiquitous Computing*, 8(1):19–30, 2004.
[12] M. Eisenhauer, P. Rosengren, and P. Antolin. Hydra: A development platform for integrating wireless devices and sensors into ambient intelligence systems. In *The Internet of Things*, pages 367–373. 2010.
[13] H. Ferreira, E. D. Canedo, and R. T. de Sousa. IoT architecture to enable intercommunication through REST API and UPnP using IP, ZigBee and Arduino. In *Proc. of WiMob*, pages 53–60, 2013.
[14] D. Gil et al. Internet of things: A review of surveys based on context aware intelligent services. *Sensors*, 16(7):1069, 2016.
[15] N. Y. Hammerla, S. Halloran, and T. Plötz. Deep, convolutional, and recurrent models for human activity recognition using wearables. *arXiv preprint arXiv:1604.08880*, 2016.
[16] T. K. Ho. Random decision forests. In *Proc. of ICDAR*, 1995.
[17] Y.-W. Kao and S.-M. Yuan. User-configurable semantic home automation. *Comp. Standards & Interfaces*, 34(1):171–188, 2012.
[18] M. Keally et al. Remora: Sensing resource sharing among smartphone-based body sensor networks. In *Proc. of IWQoS*, pages 1–10, 2013.
[19] E. F. Krause. *Taxicab geometry: An adventure in non-Euclidean geometry*. Courier Corporation, 1986.
[20] Y. Lee et al. Comon+: A cooperative context monitoring system for multi-device personal sensing environments. *IEEE Trans. on Mobile Computing*, 15(8):1908–1924, 2016.
[21] C. Liu et al. Stacon: Self-stabilizing context neighborhood formobile iot devices. In *Proc. of IEEE PerCom Workshops*, 2019.
[22] C. Liu, C. Julien, and A.L. Murphy. PINCH: Self-organized context neighborhoods for smart environments. In *Proc. of IEEE SASO*, 2018.
[23] A. Luxey et al. Sprinkler: A probabilistic dissemination protocol to provide fluid user interaction in multi-device ecosystems. In *Proc. of PerCom*, pages 1–10, 2018.
[24] A. I. Maarala, X. Su, and J. Riekki. Semantic reasoning for context-aware internet of things applications. *IEEE IoT J.*, 4(2), 2017.
[25] A. Mannini and A. M. Sabatini. Machine learning methods for classifying human physical activity from on-body accelerometers. *Sensors*, 10(2):1154–1175, 2010.
[26] Z. Meng and J. Lu. A rule-based service customization strategy for smart home context-aware automation. *IEEE Transactions on Mobile Computing*, 15(3):558–571, 2016.
[27] S. Nath. Ace: exploiting correlation for energy-efficient and continuous context sensing. In *Proc. of MobiSys*, pages 29–42, 2012.
[28] F. Ordóñez and D. Roggen. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, 16(1):115, 2016.
[29] S. Ovadia. Automate the internet with "if this then that" (IFTTT). *Behavioral & Social Sciences Librarian*, 33(4):208–211, 2014.
[30] S. K. Pal and S. Mitra. Multilayer perceptron, fuzzy sets, and classification. *IEEE Trans. on Neural Networks*, 3(5):683–697, 1992.
[31] C. Perera et al. Context aware computing for the internet of things: A survey. *IEEE Communications Surveys & Tutorials*, 16(1), 2014.
[32] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.
[33] V. Radu and other. Multimodal deep learning for activity and context recognition. *Proc. of IMWUT*, 1(4):157, 2018.
[34] T. Ružić and A. Pižurica. Context-aware patch-based image inpainting using markov random field modeling. *IEEE Transactions on Image Processing*, 24(1):444–456, 2014.
[35] Y. Saputra et al. Warble: Programming abstractions for personalizing interactions in the internet of things. In *Proc. of MobileSoft*, 2019.
[36] O. B. Sezer, ErdoE.gan Dogdu, and A. M. Ozbayoglu. Context-aware computing, learning, and big data in internet of things: a survey. *IEEE Internet of Things Journal*, 5(1):1–27, 2018.
[37] L. Suchman. *Human-machine reconfigurations: Plans and situated actions*. Cambridge University Press, 2007.
[38] Y. Sun et al. Efficient rule engine for smart building systems. *IEEE Transactions on Computers*, 64(6):1658–1669, 2015.
[39] E. Wei and A. Chan. Campus: A middleware for automated context-aware adaptation decision making at run time. *Pervasive and Mobile Computing*, 9(1):35–56, 2013.
[40] M. Weiser. The computer for the 21 st century. *Scientific American*, 265(3):94–105, 1991.
[41] R. Yang and M. W. Newman. Learning from a learning thermostat: lessons for intelligent systems for the home. In *Proc. of UbiComp*, pages 93–102. ACM, 2013.

[2]https://developers.google.com/actions/smarthome/