A3ID: An Automatic and Interpretable Implicit Interference Detection Method for Smart Home via Knowledge Graph

Ding Xiao, Qianyu Wang, Ming Cai, Zhaohui Zhu, and Weiming Zhao

***Abstract*—The smart home brings together devices, the cloud, data, and people to make home living more comfortable and safer. Trigger–action programming enables users to connect smart devices using if-this-then-that (IFTTT)-style rules. With the increasing number of devices in smart home systems, multiple running rules that act on actuators in contradictory ways may cause unexpected and unpredictable interference problems, which can put residents and their belongings at risk. Previous studies have considered explicit interference problems related to multiple rules targeting a single actuator, whereas implicit interference (interference across different actuators) detection is still challenging and not yet well studied owing to the effort-intensive and time-consuming annotation work of obtaining device information. The lack of knowledge about devices is a critical reason that affects the accuracy and efficiency in implicit interference detection. In this article, we propose A3ID, an automatic detection method for implicit interference based on knowledge graphs. Using natural language processing (NLP) techniques and a lexical database, A3ID can extract knowledge of devices from a knowledge graph, including functionality, effect, and scope. Then, it analyzes and detects interferences among the different devices semantically in three steps, without human intervention. Furthermore, it provides user-friendly explanations in a well-designed structure to specify possible reasons for the implicit interference problems. Our experiment on 11 859 IFTTT-style rules shows that A3ID outperforms state-of-the-art methods by more than 33% in the F1-score for the detection of implicit interference. Moreover, evaluations on an extended data set for devices from ConceptNet (a knowledge graph) and five smart home systems suggest that A3ID also has favorable performance with other devices not limited to the smart home domain.**

***Index Terms*—Interference detection, knowledge graph, natural language processing (NLP), smart home.**

I. Introduction

THE SMART home provides residents with a safe, comfortable, and energy-saving home environment. Most industrial solutions that enable home automation and smart home adopt the trigger–action programming paradigm, where users set rules to associate triggering sensor events and triggered actuator actions [1]. These rules are also known as if-this-then-that (IFTTT)-style rules.

Manuscript received August 2, 2019; revised September 14, 2019 and November 16, 2019; accepted December 6, 2019. Date of publication December 11, 2019; date of current version March 12, 2020. This work was supported by the National Natural Science Foundation of China under Grant 51775496 and Grant 51875503. *(Corresponding author: Ming Cai.)*

The authors are with the Department of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China (e-mail: derll.xiao@gmail.com; w_morpheus@zju.edu.cn; cm@zju.edu.cn; 21621138@zju.edu.cn; 21621305@zju.edu.cn).

Digital Object Identifier 10.1109/JIOT.2019.2959063

When facing the growth of connected devices in smart homes, multiple simultaneously running rules can result in interference problems, especially with the actuators [2]. For instance, interference occurs when multiple rules command one or more actuators to act in contradictory ways or have interfering effects on the environment. Such interference problems lead to critical situations by putting home residents at risk [3]. Actually, multiple complaints have been posted on smart home platform communities about occasional (sometimes frequent) glitches like random, unexpected, or unpredictable interference problems among actuators. Therefore, how to detect interference accurately and efficiently in dynamic environments with massive heterogeneous devices has become a crucial challenge.

In general, two types of interference can be identified based on different scenarios. One is explicit interference, which occurs when multiple rules act on the same actuator in different ways. For example, explicit interference is caused when one rule demands to open the window and another rule demands to close the window. The other is implicit interference. This occurs when two or more rules simultaneously target multiple actuators that have contradictory effects on a shared environment property or on related physical quantities. For instance, one rule commands the heater to warm the room, while another rule demands to turn on the air conditioning (AC) in cooling mode. This produces an implicit interference on the effect of temperature.

Extensive industrial efforts have been made to deal with interference and conflict in the smart home. For example, the Samsung smart home platform (SmartThings) [4] adopts asynchronous execution and an eventual consistency model to maximize availability and performance with a rather weak consistency. This means that SmartThings tolerates inconsistency over a period of time when contradictory commands are issued on the same actuator. Therefore, on the one hand, eventual consistency helps the system run most efficiently by offering low communication delays and high availability [5]. On the other hand, it faces a greater risk of conflict. In the Microsoft Home Operating System (HomeOS) [6], users should give each configuration a priority number. If there is a conflict, the configuration with the highest priority takes precedence. Amazon AWS IoT resolves conflict by using a version-based optimistic locking. Every request of actuation should be supplied with a version. When the current version of the request matches the supplied version, the actuation is accepted and the version of request is incremented. Therefore, if a device is actuated simultaneously, the later request will be rejected due to a version mismatch. However, these methods only consider explicit interference problems related to runtime conflict resolution.

The academic communities have studied the problem of formally verifying the system against interference problems [1], [3], [7]–[10]. For example, Shehata *et al.* [8] proposed a semiformal method, called IRIS, to detect conflicts between policies in a smart home system. The core component within IRIS is an interaction taxonomy that provides guidelines on when two policies are in conflict. Aloulou *et al.* [11] proposed an approach to detect conflicts between rules using an ontological model and rule-based reasoning. It provides consistency-checking capabilities to detect and suggest missing sensors and find overlaps between rules under certain conditions. Liang *et al.* [12] presented a platform called SIFT. It combines path-bounded model-checking techniques with symbolic execution to verify conflicts or policy violations within home automation applications. By repeatedly executing each enumerated path in the rule set, SIFT is able to search conflict-triggering execution paths and identify the circumstances that lead to each conflict.

So far, very little research has been performed to address the problem of implicit interference. Specifically, DepSys [13] detects conflicts across devices by considering their impacts on the environment. It requires developers to specify the app's effect using XML tags and conducts conflict detection for every pair of devices at the app installation time. Sun *et al.* [14] proposed a formal rule model (called UTEA) to detect conflicts in smart building systems. Sensors and actuators are mapped into physical environment entities with fine-grained annotations. UTEA can detect the conflict between two rules and the contradiction among multiple rules. In practice, the necessity of manual annotations and human intervention to manage conflicts is one of the major difficulties of these approaches.

Comparatively, the detection and resolution of explicit interferences are well studied, whereas implicit interference issues still face several key challenges, as follows.

1) Effort-Intensive Annotations: State-of-the-art solutions for implicit interference detection require developers to specify the impacts of each device on the environment and what is considered as interference. The labeling and modeling processes are effort intensive and time consuming, and therefore, vulnerable to human errors. Moreover, these methods demand that developers should use the same terminology and be consistent when specifying device information. Subtle differences might give rise to false positives or negatives.

2) Low Recall Rate: Most smart devices are multifunctional, providing various capabilities and producing different effects (e.g., temperature, humidity, and ventilation). For example, a window can be used not only for ventilation but also for letting light in and protecting from the elements. Under such a situation, developers are likely to label only parts of the device's capabilities and effects, which may prevent a detector from identifying the implicit interference and lead to a low recall rate of interference detection.

3) Lack of Feedback: When an interference is detected, the available methods offer little feedback informing users about the problems. Therefore, it is unclear to a nonexpert why the system has failed in such a situation. To avoid further unintended harm, users require ways of understanding what is causing the interference in an intuitive manner, and they might even be offered ways of correcting the problem.

As mentioned above, the lack of knowledge about a device is a critical reason that affects the accuracy and efficiency in implicit interference detection. Therefore, we introduce the knowledge graph as a source of device information. By using natural language processing (NLP) techniques (e.g., grammar parse tree, POS tagging) and a lexical database (WordNet), we present a novel detection method for implicit interference called A3ID to automate the detection process. Specifically, A3ID proposes a more autonomous way than the state-of-the-art solutions by extracting actuator usage information (e.g., functionality, effect, and scope) from a knowledge graph using NLP and semantically detecting interference across devices without human intervention. Moreover, A3ID specifies key pieces of information in a well-designed structure as evidence of possible reasons for the interference problems to provide assistance for users in fixing them.

To summarize, this article presents four major contributions.

1) To the best of our knowledge, A3ID is the first method that utilizes NLP to extract knowledge of devices from knowledge graphs and identify the interferences semantically, so as to automate the detection process and provide user-friendly explanations that specify possible reasons for the interference problems.
2) Using a lexical database and semantic similarity computation based on word embeddings, we design three types of microtasks, namely, scope analysis, effect evaluation, and polarity calculation, to analyze the effects of each device and identify the interference among devices. In addition, we propose a method to select representative functionalities for interference explanations.
3) We devise two algorithms to reduce false positives of interference detection. The closed scope detection algorithm is proposed to determine the functionality scope of actuators in the prevention of misjudgment on uncorrelated actuators. In addition, we design a bidirectional word sense disambiguation (WSD) algorithm to pinpoint whether two words have antonym relations in the given context in an effort to determine the contradictory effects among devices correctly.
4) We conduct experiments on 11 859 IFTTT-style rules, five smart home systems (Samsung SmartThings, Apple HomeKit, Microsoft LoT, Xiaomi, and ZigBee) and ConceptNet. The results show that the F1-score reaches 0.83, which outperforms state-of-the-art methods by more than 33% for the detection of implicit interference.

The remainder of this article is organized as follows. Section II provides preliminary knowledge on implicit interference detection. Related works are reviewed in Section III. Section IV describes the framework of our

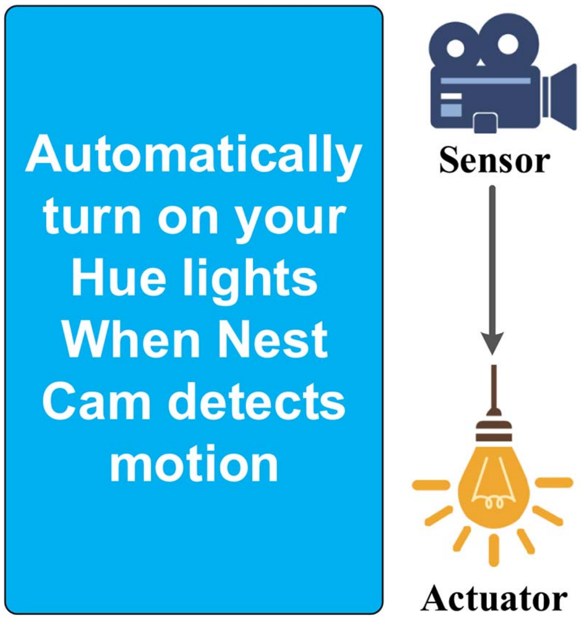

Automatically turn on your Hue lights When Nest Cam detects motion
Sensor
Actuator

Fig. 1. IFTTT sample rule.

proposed method, A3ID. In Sections V–VII, we precisely describe our techniques and methods in the three major parts (knowledge extraction, interference detection, and interference explanation) of A3ID, respectively. Evaluations are conducted in Section VIII, and we compare our method with a few related interference detection methods in terms of precision rate, accuracy rate, recall rate, and F1-score. Finally, we conclude this article in Section IX.

II. Preliminary Knowledge on Implicit Interference Detection

In this section, we introduce the background knowledge of implicit interference detection. We divide the content into two sections: the first section briefly describes the trigger–action programming in the smart home and the second one provides a concrete example of implicit interference based on the IFTTT-style rules.

A. *Trigger–Action Programming in Smart Home*

Smart homes constitute a branch of ubiquitous computing that aims at incorporating smartness into dwellings for comfort, healthcare, safety, security, and energy conservation [15]. One of the technologies that enable home automation provides users with tools to program their environment [16], [17]. This programming often takes the form “if trigger, then action” and is named trigger–action programming. The first notable attempt to build a trigger–action programming system is CAMP [18]. More recently, the website IFTTT [19] enables users to engage in trigger–action programming rules with smart devices. For a better understanding of trigger–action rules, let us examine a sample rule from IFTTT in Fig. 1, which reads “if Nest camera detects motion, then turn on Philips Hue.” This rule connects the Nest camera sensor with the Philip Hue actuator and provides convenience for users when they arrive home at night. With the trigger–action programming approaches, users are provided with flexible access to apply and manage smart devices in a smart home. Furthermore, users are not limited to linking one trigger to one action. They can build flows that involve many smart devices, conditional circumstances, and successive triggers and actions.

TABLE I
Rule Sets for Demonstrating Interference

Rule No.	Rule description
R1	Close your window at sunset.
R2	Open your window at sunset.
R3	Turn on your heater when you arrive home.
R4	Turn on air conditioning in cooling mode when you arrive home.

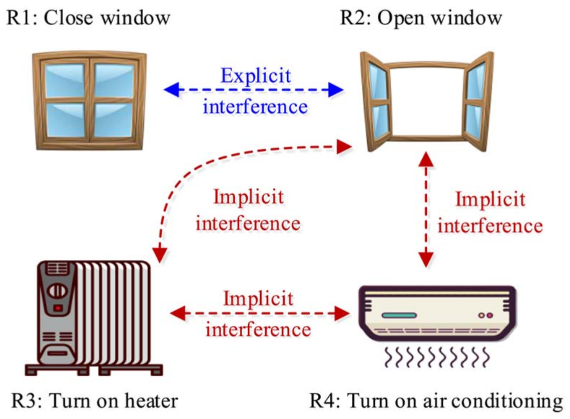


Fig. 2. Example of interference problems.

B. *Implicit Interference Detection Example*

Owing to the different concerns and management strategies of each scenario, interference problems, such as competition, conflict, and contradiction among smart devices frequently occur, especially with actuators. For instance, Table I lists four trigger–action rules and Fig. 2 demonstrates interference problems among these concurrently running rules.

As can be observed in Table I, R1 is in conflict with R2 because the two rules try to control a single actuator to perform different actions. We define it as explicit interference. Such problems are obvious and can be easily detected by determining the actuator types and trigger conditions.

R3 conflicts with R4 because the heater and AC have opposite effects on the environment: a heater increases the temperature of a room, while AC aims to decrease that temperature. We define this type of interference as implicit interference.

Meanwhile, there is another group of possible interference, R2 and R4. Imagine the following scene: the window has been opened because of the sunset, and then, as you arrive home, the AC is turned on. According to our common sense knowledge, the cooling effect of AC is significantly reduced with the window open. Therefore, R2 and R4 also have implicit interference.

Consequently, as the number of rules increases, the possibility and variety of implicit interference raise accordingly. Moreover, some types of implicit interference are difficult to identify even with manual labeling, and they are not easy to understand without providing sufficient explanations. Hence, the motivation of our approach, A3ID, is described as follows.

1) *Identifying Implicit Interference With Common Sense Knowledge:* We believe that common sense knowledge is important for improving the recall rate of implicit interference detection, helping identify implicit interference more thoroughly.
2) *Requiring Less Annotations:* In the state-of-the-art works, developers have to formally specify a massive amount of information for each device about its impacts on the environment, which is essential for implicit interference detection. It significantly complicates the procedure of development and maintenance. Thus, we design our method with the aim to reduce the demand for manual labeling.
3) *Providing Appropriate Explanations:* The contradictory rule pair, R2 and R4, may cause confusion to users if insufficient explanations are provided. Hence, our method has the motivation of providing explanations when implicit interference is identified.

III. Related Works

A. Smart Home Platforms

Considerable efforts have been devoted in the industry to the development of different smart home platforms, including Samsung SmartThings [4], Amazon AWS IoT, Google Smart Home [20], Apple HomeKit [21], Microsoft LoT [22], Xiaomi [23], and Home Assistant (an open-source home automation platform) [24]. These systems, which are vertically designed, are capable of tracking and controlling smart devices at home and facilitate end users to customize their smart devices using trigger–action programming.

There have been several attempts to manage interference problems and conflicting situations. For example, Samsung SmartThings adopts asynchronous execution and an eventually consistent model when facing interference. Microsoft Azure IoT, HomeOS [6], and EdgeOS$_H$ [25] employ priority to resolve conflicts if multiple configurations target the same device. Amazon AWS IoT utilizes a version-based optimistic locking mechanism to avoid interference, while Home Assistant uses Python's asyncio module to support asynchronous programming without any locks [24]. However, these approaches only focus on runtime conflict resolution using a concurrency control mechanism, and they lack efficient methods for identifying current or potential conflicts.

B. Conflict Detection

Conflict detection seeks to identify potential conflicts and resolve them before their actual occurrence. This strategy aims to ensure conflict-free execution. Such approaches can be implemented at different stages of the application lifecycle, but essentially at deploy time.

Several studies have considered applying various formal models (e.g., the rule-based model, ontological model, and Petri net) and reasoning algorithms to verify structural errors (e.g., redundancy, inconsistency, and circularity) among the IFTTT-style rules [1], [3], [7], [8], [10]–[12]. For example, Shehata *et al.* [8] proposed a semiformal method for detecting interactions between polices in the smart home. The work translates policies into event-based representations to detect interactions using predefined guidelines. Hu *et al.* [26] provided a semantic Web-based method for automatic detection of interactions for user policies. The method represents the user policy with a semantic Web-based model and performs automated reasoning over the Web ontology language (OWL). Carreira *et al.* [27] introduced a formal framework based on constraint solving to detect conflicts. The method formulates a conflict detection problem as a constraint solving problem (MiniZinc model) of determining whether an assignment exists. If the model is unsatisfiable, it suggests that a conflict exists. Yagita *et al.* [28] presented an approach to managing conflicts at installation time using the model-checking technique. This article defines a Kripke structure to represent the statuses of sensors and actuators and creates the temporal logic formulas accordingly. The checker will in turn be responsible for the detection of application conflicts triggered by conditions based on events.

As discussed earlier, these efforts require well-formed formal expressions, typically relying on domain experts to model the structure or create the expression. Moreover, the above methods only take into account the direct effects (explicit interference) of the actuators and cannot detect possible conflicts related to indirect effects (implicit interference) [28].

So far, very little research has been performed to address the problem of implicit conflicts. Notably, DepSys [13] and UTEA [14] propose approaches to detect conflicting situations among actuations triggered by multiple rules concurrently. However, they place the burden of specifying each application intents and impacts of the environment on the app's developers. Therefore, A3ID proposes the use of NLP techniques to extract knowledge of devices from knowledge graphs and detect interferences across devices without human intervention. Moreover, A3ID provides reasonable explanations of the interference problems so that users can take action.

IV. Proposed Method Framework

By combining the management information in the smart home system and knowledge of devices in the knowledge graph, A3ID is able to identify implicit interference among actuators automatically. The detection consists of the following three steps: 1) knowledge extraction; 2) interference detection; and 3) interference explanation. The workflow of A3ID is depicted in Fig. 3.

1) When a new device joins the smart home, A3ID will query the system to collect the device type and then use it to acquire knowledge of the device from the knowledge graph through the Web API. POS tagging

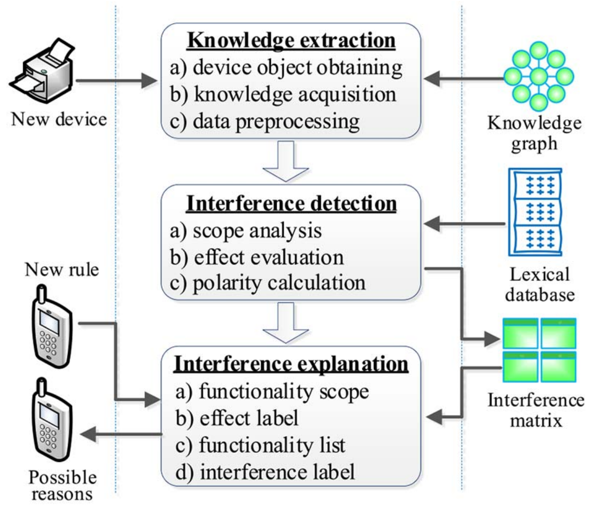


Fig. 3. Workflow of A3ID.

and lemmatization are utilized to preprocess the data to speed up the analysis time (Section V).

2) Three types of microtasks: a) scope analysis; b) effect evaluation; and c) polarity calculation, are employed to detect implicit interference using a lexical database and semantic similarity computation (Section VI). Scope analysis is used to depict the scope of effect of devices, and a device with a closed scope of effect can be free of interference detection. Effect evaluation is introduced to evaluate the effect of a device on the environment and find contradictory environmental effects among devices. Polarity calculation is used for detecting the antonymy relations between two functionality descriptions to identify conflicting operations. An interference matrix is created to store the results of interference detection.

3) When users issue new trigger–action rules, A3ID will extract actuator types from the rule engine in the smart home system and retrieve the results of interference detection from the interference matrix. Explanations and useful feedback will be provided in a well-designed structure to specify possible reasons for the interference problems (Section VII).

It is worth mentioning that A3ID is based on an event-driven mechanism, where the control flow is triggered and determined by external events (e.g., device registration, device unregistration, rule creation, and rule modification). The event-driven method is flexible, scalable, and loosely coupled to integrate with existing smart home platforms, such as Samsung SmartThings, Amazon AWS IoT, and Home Assistant by customizing event types and their arguments. Therefore, A3ID in partnership with the commercial off-the-shelf (COTS) products can serve residents with the conflict-free operation.

V. Knowledge Extraction

Knowledge graphs encode structured information of entities and their relations [29]. The basic elements of knowledge graphs are entities and edges: 1) entities are divided into head entities and tail entities, both of which have different

TABLE II
Overview of Seven Popular Knowledge Graphs

Knowledge graph	# of devices	AVG # of descriptions	Web API (Y/N)
wikiData	33	1.06	Y
WordNet 3.1	31	1.68	Y
ConceptNet 5.6	36	9.69	Y
BabelNet 4.0	34	1.56	Y
Nell	35	-	Y
Microsoft Concept Graph v1	37	2.05	Y
Google Knowledge Graph v1	14	0.79	Y

attributes and types and 2) edges are relations between head and tail entities. For example, in the triple *window* $\xrightarrow{UsedFor}$ *let light in a building*, the head entity is *window*, the tail entity is *let light in a building*, and the edge is *UsedFor*. From the above, we can obtain the knowledge: a window is used for letting light in a building. Even on the condition of the same edge name, a head entity can have multiple relations with different tail entities. For instance, a window is not only *UsedFor* letting light in a building but also *UsedFor* letting fresh air in.

Established with millions of relations, the knowledge graph performs as a knowledge base with the capacity of semantic processing and open interconnection to facilitate a wide range of intelligent information services, such as search, question–answering, and personalized recommendation. Using NLP, A3ID is able to extract knowledge of devices from the knowledge graph and detect implicit interferences among actuators by comprehending the semantic content of the control, especially some hidden conflict scenarios relating to different actuators with opposite effects.

A. *Acquiring Knowledge*

As it is crucial to choose a suitable knowledge graph with an extensive range of device information to support interference detection, we set up three criteria for selecting the knowledge graph: 1) wide coverage of device categories; 2) detailed functionality descriptions about devices; and 3) online Web API support. Thereafter, we use 38 commonly used actuators (e.g., AC, thermostat, and lock) collected from smart home systems as data sample to investigate seven popular knowledge graphs (wikiData [30], WordNet [31], ConceptNet [32], BabelNet [33], NELL [34], Microsoft Concept Graph [35], and Google Knowledge Graph [36]). As presented in Table II, ConceptNet outperforms the other candidates in the overall evaluation. Although Microsoft Concept Graph covers more device categories, ConcepNet has far more detailed descriptions for each device. Therefore, we adopt ConceptNet to acquire knowledge of devices.

ConceptNet is a knowledge graph that connects words and phrases of natural language with labeled, weighted edges [32]. Its knowledge is collected from several sources, such as expert-created resources and crowdsourcing. ConceptNet provides a REST API through api.conceptnet.io where developers can get data from ConceptNet in the JSON-LD format. The API is read only, and knowledge can be retrieved using an HTTP

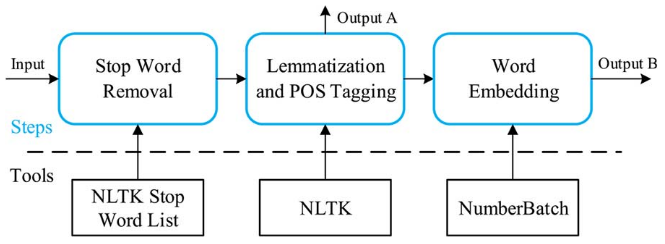


Fig. 4. Workflow of data preprocessing.

GET command. ConceptNet aligns its knowledge resources on 36 relations, among which four edges are recognized as the source of device information, namely, *UsedFor*, *CapableOf*, *DerivedFrom*, and *AtLocation*. For example, A3ID can obtain functionality descriptions of devices through *UsedFor* and location information through *AtLocation*. It is worth mentioning that when an entry (device) in ConceptNet does not contain *UsedFor*, we choose *CapableOf* instead. If a device has neither of the above two relations, *DerivedFrom* is considered as an alternative option. For example, the humidifier does not contain *UsedFor* or *CapableOf* in ConceptNet, but it has *DerivedFrom* with the word "humidify," which we choose as the functionality of humidifier.

B. Data Preprocessing

With respect to data preprocessing on functionality descriptions extracted from the knowledge graph, we perform stop word removal, structural completion, lemmatization, and POS tagging. The workflow is shown in Fig. 4.

1) Stop Word Removal: Stop words are words that are commonly used and have little importance of meaning. For example, articles (e.g., "the" and "a"), prepositions (e.g., "of" and "to"), pronouns (e.g., "that" and "this"), and causative verbs (e.g., "let" and "keep") are mostly in stop word lists. These stop words are redundant and even obstacles in NLP tasks. Therefore, we filter out stop words on the basis of the stop list in natural language tool kit (NLTK) v. 3.3 [37].

2) Lemmatization and POS Tagging: We utilize POS tagging (mark up a word in a text as corresponding to a particular part of the speech) and lemmatization (determine the lemma of a word based on its intended meaning) to preprocess these short texts. Specifically, NLTK v. 3.3 is employed as an NLP tool for POS tagging and lemmatization. However, POS tagging performs poorly when processing short texts lacking subjects and predicates. We complete these short texts based on their relations with subjects in ConceptNet. For example, AC has the functionality "cool your home." We can complete it as "AC can cool your home." This can significantly improve the accuracy of POS tagging.

After the above two steps, the preprocessed functionality descriptions (denoted as "Output A" in Fig. 4) can be applied to most of the following tasks. For the cosine similarity calculation (introduced in Section VI) and interference explanation (introduced in Section VII) tasks, word embedding is an indispensable step in data preprocessing.

3) Word Embedding: Word embedding is an important model to map the vocabulary to vectors of real numbers for quantifying semantic similarities between words. NumberBatch [38] is a set of word embeddings that merge the information from ConceptNet and combine embeddings produced by Glove and Word2Vec. NumberBatch is compatible for text processing on the corpus from ConceptNet. Therefore, NumberBatch is used as the word embedding tool in the following tasks, such as cosine similarity and clustering. After word embedding, the preprocessed functionality descriptions are converted to vector representations of sentences, which are also referred to sentence representations (denoted as "Output B" in Fig. 4).

VI. Interference Detection

This section introduces the core strategies of A3ID for interference detection. We design three types of microtasks, including scope analysis, effect evaluation, and polarity calculation, to detect interference among ConceptNet entities based on the sentence representations we acquired from the previous section.

A. Scope Analysis

We find that the comparison of functionalities of each pair of devices cannot alone fully understand their effect. For instance, in ConceptNet, dryer is *CapableOf* "dry clothing" and washing machine is *UsedFor* "washing clothes." Although a dryer and a washing machine have opposite functionalities, they would not generate any interference because the functionalities of such actuators only impact things inside themselves. Thus, we regard these actuators as devices with "closed scope of effect" attributes. By filtering out devices with closed scope of effect at first, we can avoid numerous misjudgments of interference. Hence, we analyze the scope of effect as the first step to exclude these devices from subsequent detection processes.

By observing the functionalities of the devices with closed scope of effect, we find that the objects the devices act on can be found in *AtLocation* of these devices in ConceptNet. For example, the washing machine has the functionality of washing clothes and "clothes" can be found in *AtLocation* of the washing machine. This information can be extracted by *UsedFor* and *AtLocation*, respectively, in ConceptNet. For *AtLocation*, as the edge is asymmetric and directed, it means the head entity is located in the tail entity. For instance, in the *AtLocation* that connects "clothing" and "washing machine," clothing is the head entity and washing machine is the tail entity. It represents that clothing is located in "washing machine." In this way, we can identify the objects located in devices. For each device, we collect the objects located in the device from *AtLocation* (clothing is located in the washing machine) and extract entities from *UsedFor* (washing machine is used for washing clothes). We find a simple way to extract entities from *UsedFor* and *AtLocation*. Because the functionality descriptions are constructed by a verb phrase (VP), the nouns in these short texts are always entities. For instance, the entity "clothes" is a noun in the functionality description

Algorithm 1 Closed Scope Detection

Input: The actuator, A; The entity set extracted from *UsedFor* of A actuator, $U = \{u_1, u_2, ...\}$; The entity set extracted from *AtLocation* of A actuator, $L = \{l_1, l_2, ...\}$; The cosine similarity threshold, θ.

Output: Whether the effect scope of A is closed scope, *isClosedScope*.

```
1:  $flag \leftarrow 0$
2:  for $u_i \in U$ do
3:      for $l_j \in L$ do
4:          if $cosineSimilarity(u_i, l_j) \geq \theta$ then
5:              $flag \leftarrow 1$
6:              break
7:          end if
8:      end for
9:      if $flag = 1$ then
10:         break
11:     end if
12: end for
13: if $flag = 1$ then
14:     $is\ ClosedScope \leftarrow True$
15: else
16:     $is\ ClosedScope \leftarrow False$
17: end if
```

"washing clothes." Next, by using NumberBatch, we measure the cosine similarity coefficient between the entities we extracted from these two relations one by one. The cosine similarity coefficient is defined as follows:

$$\text{Cosine}\big(W_i, W_j\big) = \frac{W_i \cdot W_j}{\|W_i\| \|W_j\|} \tag{1}$$

where W_i and W_j represent two word vectors of entities. Words that occur in a similar context tend to have high cosine similarity. If any pair of entities has the same words or their cosine similarity is higher than a threshold θ (0.6), we consider these two words as similar objects and determine the device with a closed scope of effect. For instance, "pants" are located in dryer and dryer is used for "dry wet clothes." In addition, pants and clothes are actually the same objects. Therefore, the dryer is recognized as a device with a closed scope of effect. In the following processes, we filter out these devices from further interference detection.

The detailed algorithm is described in Algorithm 1. As for the computational complexity, let m and n be the number of relations in *UsedFor* and *AtLocation*, respectively; then, the computational complexity of Algorithm 1 is $O(mn)$. Given that m and n are in practice not very large (usually dozens) and the algorithm has the possibility to break the loop in advance, the algorithm in actual operation is not time consuming.

B. Effect Evaluation

In previous works on implicit interference detection, the environmental effects (such as temperature and humidity) are regarded as the major factors to judge whether the implicit interference exists: actuators impacting on the same environmental effects have a high possibility of implicit interference [13], [14]. For example, both the AC and heater can influence the same environmental effect, temperature. If they change the temperature in opposite directions, implicit interference occurs. Therefore, previous methods require researchers to annotate the related environmental effects on each actuator. These environmental effects are generally classified into ten groups: 1) temperature; 2) humidity; 3) ventilation; 4) lighting; 5) entertainment; 6) energy; 7) water; 8) gas; 9) health; and 10) security. Our method has the ability to detect the above implicit interference automatically, without manual annotations of environmental effects, because we can extract the environmental effects caused by actuators from the knowledge graph (from *UsedFor* or *isCapleOf*).

Furthermore, during this article we find a new interference situation that previous works have never taken into consideration: implicit interference still has the probability to exist when two actuators influence different environmental effects. In this article, this is referred to as multienvironmental implicit interference. According to our further exploration, we find that this type of implicit interference only exists between actuators labeled with temperature and ventilation and also with humidity and ventilation. As temperature and humidity are two important factors of air, ventilation can change these two factors. For instance, a window and an AC may have implicit interference because ventilation (influenced by the window) mitigates the impact of temperature (influenced by AC).

However, the multienvironmental implicit interference cannot be detected automatically on the aforementioned detection method because it requires more in-depth knowledge, which is not yet included in the knowledge graph. Hence, to address the aforementioned problem, we propose a new method to pinpoint devices that have impacts on humidity, temperature, or ventilation automatically. For example, the window and AC should be labeled with the ventilation and temperature tags, respectively. In the end, we measure the cosine similarity coefficient between the entities extracted from *UsedFor* and the three environment variable words (temperature, humidity, and ventilation). If the similarity value is greater than the threshold θ (0.5), the device will be assigned the corresponding tag. In the above example, AC has the functionality of "keep the room cold" and "removing moisture from the air." The cosine similarity between "cold" and "temperature" is equal to 0.504 and the cosine similarity between "moisture" and "humidity" is equal to 0.842. Therefore, AC is labeled as "temperature" and "humidity." The window has the functionality of "ventilation and view" and it is labeled as "ventilation." Consequently, AC has implicit interference with the window. After analyzing the environmental effect, we have identified some hidden interference pairs relating to different actuators with contradictory environmental effects.

C. Polarity Calculation

1) Polarity of Functionalities: Furthermore, we use polarity to measure the relationship between two functionality descriptions through *UsedFor*. When the two short texts express opposite functionalities, the polarity between them is specified as a negative value (−1). Otherwise, their polarity is a positive value (1). According to the polarity, we can determine their interference relationship. For example, the fan has the functionality of "cooling off," and the heater is used for "warming." The polarity between these two functionalities is −1, and therefore, implicit interference may occur between fan and heater.

In ConceptNet, functionality descriptions of devices are constructed by a VP, and the core vocabularies that express the effects of devices are verbs and adjectives, such as "cool" in "cool the room," and "warm" in "keep warm." Therefore, to specify the polarity between the two functionality descriptions, we need to identify the antonymy among the verbs and adjectives in these texts. We utilize WordNet, a lexical database, to collect antonyms of these words. The polarity between the two texts containing antonym relations is specified as −1.

Therefore, the interference relationship between two functionality descriptions can be defined as

$$P(S, S') = (-1)^{(\eta+\eta'+c(S,S'))} \tag{2}$$

where S and S' represent two functionality descriptions, and η and η' indicate the number of negations in them, respectively. $c(S, S')$ represents the number of negative values (−1) that appear in any pair of words between two functionality descriptions, and it is defined by the following equation:

$$c(S, S') = \sum_{i}^{m}\sum_{j}^{n} \frac{\left|p\left(w_i, w_j'\right)\right| - p\left(w_i, w_j'\right)}{2} \tag{3}$$

where w_i and w_j are words in S and S', respectively, and $p(w_i, w_j') \in \{-1, 0, 1\}$ represents the polarity value of the words w_i and w_j. We filter out positive sentence polarity values according to (3), because a positive sentence polarity value cannot help find interference in our method.

2) Bidirectional Word Sense Disambiguation: Problems still remain in the above method because determining the antonymy of two words without considering their semantics always produces inaccurate results. For example, the door has the functionality of "entering or exiting an area" and the keyboard has the functionality of "entering text." In WordNet, "enter" and "exit" are a pair of antonyms, but they are irrelevant in the above situation. As a result, when a word has multiple meanings, it is necessary to find out the exact meaning of the word within the context of a specific scenario. One solution to this problem is to use the WSD technology. In A3ID, we choose pywsd 1.0 [39], the Python implementation of WSD technologies based on WordNet, to specify the corresponding "synset" ("synset" in WordNet represents a meaning of a word) of each word in WordNet. Additionally, when identifying antonyms in WordNet, the WSD processes ought to be bidirectional to promote the precision and accuracy rate.

The whole algorithm of bidirectional WSD (BWSD) is described in Algorithm 2. Suppose that m and n represent the average number of words in "synsets" and antonym lists, respectively. In our method, synsets and antonym lists of words are preprocessed in alphabetical order. Thus, we apply the binary search algorithm in our method. Consequently, the computational complexity of Algorithm 2 is $O(\log_2 m \cdot \log_2 n)$.

Algorithm 2 Bidirectional Word Sense Disambiguation

```
Input: Basic word sense disambiguation function, $wsd()$;
    Lemmatization function, $lem()$.
Output: Whether the two words have antonym relations,
    $haveAntonym$.
1: $flag \leftarrow 0$
2: $synset1 \leftarrow wsd(sen1, word1)$
3: $synset2 \leftarrow wsd(sen2, word2)$
4: if $lem(word2) \in lem(antonymList(synset1))$ then
5:     if $lem(word1) \in lem(antonymList(synset2))$ then
6:         $flag \leftarrow 1$
7:     end if
8: end if
9: if $flag = 1$ then
10:    $haveAntonym \leftarrow True$
11: else
12:    $haveAntonym \leftarrow False$
13: end if
```

3) Characterizing Smart Home-Specific Semantics: The antonyms collected from WordNet may be inappropriate in smart home scenarios because word semantics can substantially change across contexts. For example, "light" and "dark" are antonyms suitable for smart home scenarios, whereas "light" and "heavy" are not (and could even cause several misjudgments).

To solve this problem, we adopt a method called *Semaxis* proposed by An *et al.* [40] to capture smart home-specific word semantics. We extract all the antonyms from *Antonym* in ConceptNet and then manually select 127 pairs of them that are appropriate in smart home scenarios. These 127 pairs of antonyms are named as fundamental antonyms in this article. We calculate the average vectors in each pair of fundamental antonyms. These average vectors compose the fundamental antonym vectors helping to judge whether the pair of antonyms are appropriate: once we extract a pair of antonyms from the sentences automatically, we measure the average vector of these antonyms with the fundamental antonym vectors by using the cosine similarity coefficient. If the maximum value is greater than a threshold θ (0.7), we regard this pair of antonyms as effective antonyms; otherwise, we abandon them.

Through the above three types of microtasks (scope analysis, effect evaluation, and polarity calculation), we have identified the interference relationships between each pair of devices. Then, an incidence matrix (called interference matrix) is constructed with the devices as edges to store the results of interference detection along with intermediate data (e.g., scope of effect, environmental effect label, and antonym relations), which can serve as facts and evidence for identification of the implicit interference.

VII. Interference Explanation

When users have created new trigger–action rules, A3ID can extract actuator types from the rule engine module in a smart

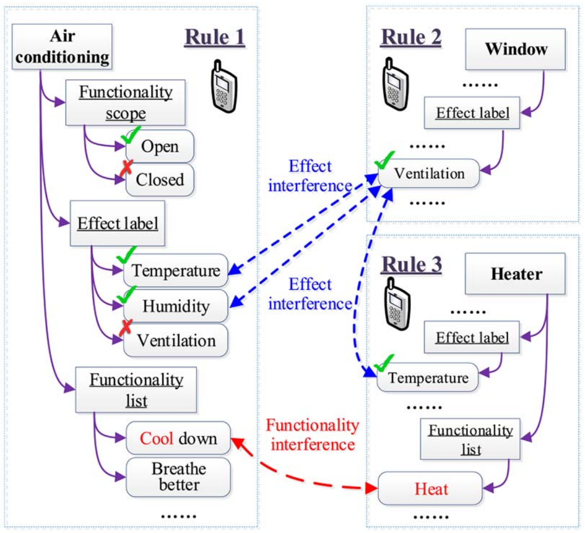
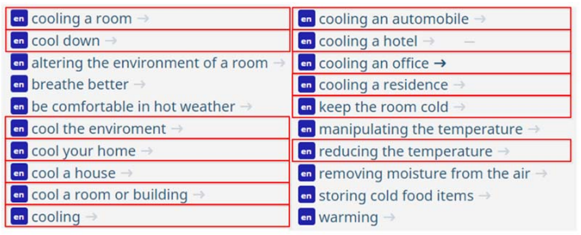


Fig. 5. Example of interference explanations.



en cooling a room →	en cooling an automobile →
en cool down →	en cooling a hotel →
en altering the environment of a room →	en cooling an office →
en breathe better →	en cooling a residence →
en be comfortable in hot weather →	en keep the room cold →
en cool the enviroment →	en manipulating the temperature →
en cool your home →	en reducing the temperature →
en cool a house →	en removing moisture from the air →
en cool a room or building →	en storing cold food items →
en cooling →	en warming →

Fig. 6. Functionalities of AC in conceptnet.

home platform, and then retrieve the results of interference detection in an interference matrix (described in Section V). However, only with the interference results, it is still unclear to users or developers why the system has such an interference problem and what causes the interference.

As a result, A3ID provides user-friendly explanations to specify the reasons for interference problems along with the implications of the identification of interference. On the one hand, instead of facing an unexpected or unpredictable failure, users are informed by A3ID about the reason for interference. Reasonable explanations assist users in taking the corresponding measures to prevent such interference. On the other hand, explanations can also help developers to diagnose the system and conduct optimization.

A3ID organizes key pieces of information in a well-designed structure as evidence of possible reasons for interference problems. The explanations of the interference consist of the following four parts.

4) Functionality Scope: The functionality scope is used for reflecting an actuator's scope of effect. It can be divided into two types: 1) open and 2) closed. Actuators with closed scope of effect would not have interference with other actuators. Only the actuators with open scope are possibly causing interference with others.

5) Effect Label: The effect label is used for depicting an actuator's effect on the environment. It consists of three types of labels: 1) temperature; 2) humidity; and 3) ventilation. According to prior knowledge, those actuators affecting the temperature or humidity may interfere with actuators that have ventilation to the open air.

6) Functionality List: Most actuators are multifunctional, providing various capabilities, which may create conflicting operations among actuators. The functionality list aims to inform users of all the functionalities of an actuator in a clear and concise way by providing its representative functionalities.

7) Interference Label: The interference label is used for presenting the cause of interference based on the facts about actuators provided by the above three types of information. The causes of interference are divided into two types: 1) effect interference and 2) functionality interference. In terms of effect interference, we indicate the contradictory environmental effects. As for functionality interference, we point out the antonym relations between the two functionality descriptions.

Fig. 5 shows an example of three interference explanations among three rules. A3ID provides the functionality scope, effect label, and functionality list according to the actuator type. For Rule 1 and Rule 2, the interference label marked by the blue dotted lines indicates that the interference between AC and window is caused by the contradictory environmental effect between temperature and ventilation and between humidity and ventilation. For Rule 1 and Rule 3, the interference label points out a functionality interference between AC and heater. AC has a functionality of "cool down," which is contrary to the functionality of "heat" in heater. Meanwhile, the label indicates the antonymy between "cool" and "heat." Moreover, the interference label also shows the effect of interference between Rule 2 and Rule 3. The effect of interference is caused by the contradictory environmental effect between ventilation and temperature.

Among the four sets of explanations, three of them (functionality scope, effect label, and interference label) can be obtained from the interference matrix. Then, we focus on the processes of selecting representative functionalities for the functionality list.

The functionality descriptions about AC in ConceptNet are shown in Fig. 6. Among them, 13 items marked by red frames express the similar functionality as decreasing the temperature or cooling. The reason is that ConceptNet has grown to include knowledge from crowdsourced resources. However, the redundant information prevents users from gaining a clear understanding of the functionalities of an actuator. Therefore, we try to select representative functionalities of actuators through clustering.

Without labeled training data and a specified number of categories, we need an unsupervised algorithm for clustering. The density-based spatial clustering of applications with noise (DBSCAN) [41] is a density-based cluster algorithm. DBSCAN does not require one to specify the number of clusters and is robust for outliers, which are also necessary in this article. Hence, we choose the DBSCAN algorithm. Because the functionality descriptions of devices in ConceptNet are constructed as phrases rather than words, we compute the phrase vector in a weighted method. Verbs are more able to reflect the device actions, such as "cooling the room" and

TABLE III
FUNCTIONALITY CLUSTER (AC)

Cluster No.	Functionality List
1	**cool down**, cooling a room, cool your home, cool the environment, cool a house, cooling a hotel, cooling an automobile, cooling an office, cooling a residence, cool a room or building, keep the room cold, cooling, reducing the temperature, storing cold food items, removing the moisture from the air, manipulating the temperature, warming
2	**altering the environment of a room**
3	**breathe better**
4	**be comfortable in hot weather**

TABLE IV
FUNCTIONALITY CATEGORY WITH POLARITY CALCULATION (AC)

Cluster No.	Functionality List
1	**cool down**, cooling a room, cool your home, cool the environment, cool a house, cooling, cooling an automobile, cooling a hotel, cool a room or building, cooling an office, cooling a residence, keep the room cold, storing cold food items
2	**reducing the temperature**, manipulating the temperature
3	**removing the moisture from the air**
4	**warming**
5	**altering the environment of a room**
6	**breathe better**
7	**be comfortable in hot weather**

"warming." In this article, we assign a 0.6 weight to the verb, and 0.4 is divided into other words.

For the selection of representative functionalities, first, we compute the average phrase vector of all the functionality descriptions in the category, then we compare the cosine similarity of each functionality sentence representation with the average sentence representation, and select the highest one as the representative functionality. Taking AC as an example, we present the categories after clustering in Table III and the phrases shown in bold are the selected representative functionalities. Unfortunately, the result shows that this clustering method based on the phrase vector does not achieve satisfactory results. For example, "warming" and "cooling down," which are obviously contradictory, are classified into the same category. One possible reason is that word embedding reflects the position distribution of vocabularies in documents rather than the antonym relationship between them. In the above example, "cool" and "warm" are antonyms but their cosine similarity has a high level (0.491).

To distribute antonyms to different categories, we use WordNet similarly. For each cluster classified above, we compute the polarity between each two functionalities using the polarity calculation method (described in Section V) to divide the opposite functionalities into different categories. The result of applying the antonym distribution method on AC clusters is presented in Table IV. In comparison to the results presented in Table III, we find in the consequences that the contradictory functionality "warming" is assigned to another category. Moreover, we find that the performance of antonym distribution is significant because the functionality of "removing moisture" has been separated. The reason is that our method treats "temperature" and "moisture" as contradictory functionalities automatically.

VIII. EVALUATION

In this section, we conduct three groups of experiments to evaluate the performance of our proposed method. The first group is the rule-based evaluation that verifies the performance of our method in comparison to related methods. The second one is the device extension evaluation in which we expand the quantity of actuators to test the robustness and stability of our method. Finally, we design an ablation experiment to verify the effectiveness of each step in our method. Consequently, we conduct experiments on 11 859 IFTTT-style rules, five smart home systems (Samsung SmartThings, Apple HomeKit, Microsoft LoT, Xiaomi, and ZigBee), and ConceptNet.

A. Rule-Based Evaluation

In A3ID, we collect the functionality of actuators from ConceptNet 5.6 and use the 60-D pretrained word vectors from NumberBatch 17.06. We also adopt WordNet 3.1 to detect the antonym relations of actuators.

1) Evaluation Setup: IFTTT is a popular trigger–action programming platform, on which users can automate more than 400 services of Web applications and IoT devices. Mi *et al.* [42] collected a six-month IFTTT dataset from November 2016 to April 2017, which contains 957 actuators, 1490 triggers, and approximately 320 000 rules. Therefore, we adopt the public dataset of the IFTTT-style rules from [42]. Compared to November 2016, the quantities of triggers, actions, and rules increase, respectively, by 31%, 27%, and 19% on April 2017; therefore, we select trigger–action rules from the data of April 2017. In this dataset, each rule is related to a specific actuator, which is noted in the attribute "actionChannelTitle."

However, the actuator names in the selected dataset are different from the ones in ConceptNet entries, which is a multisource and heterogeneous problem. Therefore, based on the information noted in the attribute "actionChannelTitle," we design a mapping table to convert the actuator names in the selected dataset to the ones in ConceptNet automatically because the names in both sets are enumerable. For example, for the rule "Every hour turn your A/C on," the "A/C" is automatically converted to "air conditioning," which is one of the entries in ConceptNet, based on the mapping table.

In this evaluation work, we aim to compare A3ID with DepSys [13] and UTEA [14] in the same situation. However, the datasets and codes of DepSys and UTEA are not publicly available. Hence, we implement the implicit interference detection methods of DepSys and UTEA according to the designs provided in their papers. Thus, we can evaluate the performance of A3ID, DepSys, and UTEA in the same environment.

TABLE V
ANNOTATIONS ON DETECTION METHODS

Annotation type	DepSys	UTEA	A3ID
Environmental effects	✓	✓	-
Variation trend	✓	-	-
Variation target range	-	✓	-
Annotation complexity	Medium	High	-

Both DepSys and UTEA need manual annotations on ten environmental effects to detect implicit interference. DepSys needs the types and variation trends of environmental effects (e.g., temperature increase or decrease) under the influence of rules, and UTEA needs the type and target range of environmental effects (e.g., temperature between 20 °C–25 °C) in the same situation. Different from DepSys and UTEA, A3ID requires no manual annotations on environmental effects, even for detection of the multienvironmental implicit interference, because the method of judgment is automatically performed inside A3ID. The detailed information of annotations on these three methods is presented in Table V.

For simulating a real living environment, we select ten floor plans of houses from the House Plans website [43] to deploy actuators and configure rules virtually. These ten floor plans include an apartment, bungalow house, contemporary house, cottage house, country house, craftsman, farmhouse, modern house, ranch house, and traditional house. The configuration is divided into two steps as follows.

1) We deploy actuators in each room for every floor plan. Considering the actual condition, the number of actuators we deploy in the same room is less than 20.
2) We bind the rules to the corresponding actuators in each room. In consideration of similarity with the real situation, the number of rules we configure on each actuator is between 3 and 10, while the specific rules are randomly selected by the program for every actuator. Therefore, different actuators in the same category may have different combinations of rules. For example, A, B, C, and D are different rules related to windows. Two different windows deployed in the same room may be configured with different rule sets, one may be A, B, and C and the other is B, C, and D. Thus, we could have many different rule combinations to configure in each actuator.

We design a simple experiment to study the relation between the number of implicit interferences and the quantity of rules. Fig. 7 shows the average value of implicit interference according to the quantity of rules (randomly picked by the program) in a single room. We find that the number of interferences has a sharp increase when the quantity of rules in a room is above 100. In our evaluation, we limit the number of rules in a single room to 90–110, which proves to be appropriate. We configure 22 types of actuators and 11 859 rules in ten floor plans.

Table VI includes the overall configuration of rules in floor plans. The rule number is the sum of rules configured in every single room. The rule pairs refer to any two rules configured on different actuators in the same room. Implicit interference exists in the rule pairs that influence different actuators toward contradictory results. Despite that the number of rule pairs is large, the actual number of implicit interference is relatively small because a considerable part of actuators does not interfere with each other. Therefore, we invite three developers with expertise in home automation to label the implicit interference following a three-step guideline. First, they list every pair of actuators that may cause implicit interference with each other. Second, they examine every rule from each side of the actuator pair and then label whether or not each rule pair has an implicit interference. Finally, they label other rule pairs with "negative" for implicit interference. In total, the actual number of implicit interferences is 2343.

The rule configuration data are imbalanced and rely on two assumptions: 1) the actual number of implicit interferences (2343 in total) is far smaller than the total number of rule pairs (691 620 in total) and 2) different types of actuators have imbalanced amounts of rules provided on the IFTTT platform. Whereas, our method does not use machine learning algorithms, which are prone to be influenced by imbalanced data problems. Instead, we use a rule-based matching algorithm, and every implicit interference is determined based on antonym relations. Hence, our method is not affected by imbalanced data. Besides, given that the experiment is a simulation test that should be as authentic as possible, we maintain the status quo of data.

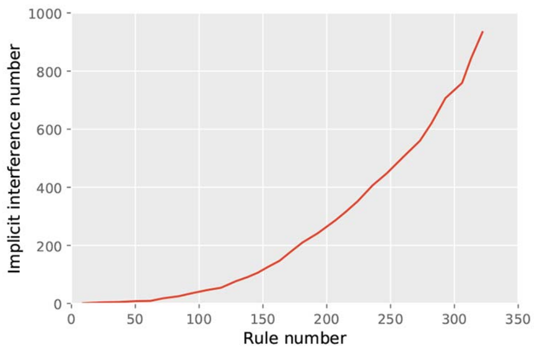


Fig. 7. Relations of rule and interference numbers in a single room.

TABLE VI
OVERALL CONFIGURATION OF RULES IN FLOOR PLANS

Floor plans	# of rooms	# of rules	# of rule pairs
Apartment	5	563	38842
Bungalow House	10	1040	62946
Contemporary House	12	1185	68461
Cottage House	10	953	52317
Country House	13	1264	75658
Craftsman	15	1458	85779
Farmhouse	17	1523	81229
Modern House	12	1437	79423
Ranch House	13	1229	70158
Traditional House	12	1207	76807
In Total	**119**	**11859**	**691620**

TABLE VII
CONFUSION MATRICES OF (a) DEPSYS, (b) UTEA, AND (c) A3ID

(a)

n=691620	Predicted: positive	Predicted: negative
Actual: positive	TP=200	FN=2143
Actual: negative	FP=0	TN=689277

(b)

n=691620	Predicted: positive	Predicted: negative
Actual: positive	TP=494	FN=1849
Actual: negative	FP=0	TN=689277

(c)

n=691620	Predicted: positive	Predicted: negative
Actual: positive	TP=1945	FN=398
Actual: negative	FP=64	TN=689213

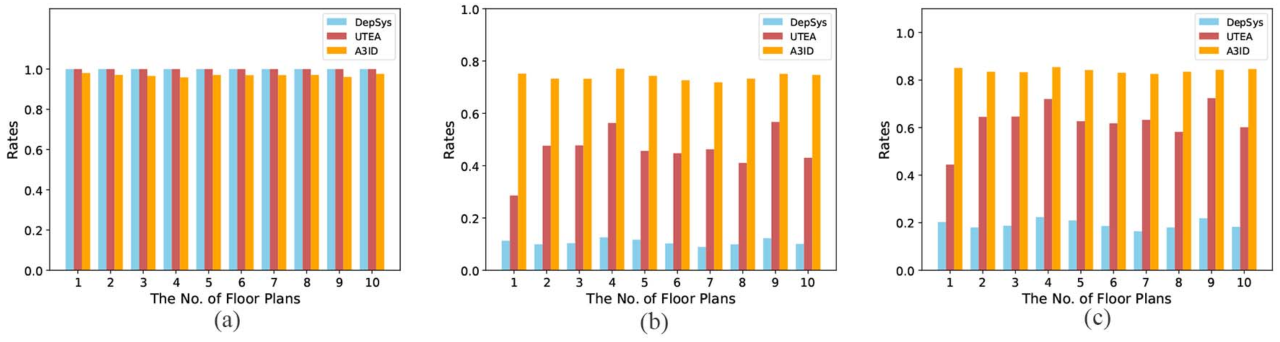


Fig. 8. Performance on ten floor plans. (a) Precision rate. (b) Recall rate. (c) F1-score.

TABLE VIII
OVERALL PERFORMANCE

Methods	Precision	Accuracy	Recall	F1 score
DepSys	**1**	0.9969	0.105	0.191
UTEA	**1**	0.9973	0.462	0.632
A3ID (ours)	0.969	**0.9993**	**0.738**	**0.838**

2) Overall Performance: Table VII presents the confusion matrices of DepSys, UTEA, and A3ID. n represents the total amount of rule pairs. TP, FN, FP, and TN represent true positives, false negatives, false positives, and true negatives, respectively. We use the precision rate (PR), accuracy rate (AR), recall rate (RR), and F1-score (FS) to evaluate the performance of our method (A3ID), DepSys, and UTEA. They are defined in the following equations:

$$\mathrm{PR} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} \tag{4}$$

$$\mathrm{AR} = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN}} \tag{5}$$

$$\mathrm{RR} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} \tag{6}$$

$$\mathrm{FS} = 2\cdot\frac{\mathrm{PR}\cdot\mathrm{RR}}{\mathrm{PR}+\mathrm{RR}}. \tag{7}$$

We regard the overall results of these three methods as the universal set. The implicit interference detection results of ten floor plans are shown in Fig. 8. We find that the precision rate, recall rate, and F1-score of the three methods are relatively stable in each floor plan, which is configured with independent combinations of actuators and rules. The overall performances of DepSys, UTEA, and our method are listed in Table VIII.

In the aspect of the recall rate, we find that our method outperforms DepSys and UTEA by a substantial margin. The performance has a 27.6% increase over that of UTEA and a 63.3% increase over that of DepSys, respectively. In terms of precision rate, our method is slightly worse than the other by 3.1% at most, and the accuracy rate of our method is the highest with 0.9993. Overall, our method achieves an F1-score of 0.838, which is 0.647 higher than that of DepSys and 0.206 higher than that of UTEA.

3) Analysis: We analyze the false negative and false positive scenarios in DepSys, UTEA, and our method based on the program outputs. The details of the results are discussed as follows.

False Positive Scenarios: Our method has 64 groups of false positives, because some interferences cannot be confirmed only on the basis of static rules. These interference problems could be detected combined with the running status of related actuators. For example, we cannot confirm whether there exists interference between the actions "turn on the air conditioning" and "turn on the heater," as we do not have enough information about whether the AC is in cooling or heating mode at the moment. In our method without any manual annotations, after connecting the rule "turn on the air conditioning" to the functionality of AC, the rule activates a functionality about cooling automatically. Thus, we consider that these two rules have interference in our method by mistake. DepSys and UTEA only detect interference among the labeled rules; hence, we can avoid false positives by not labeling the rules mentioned above. Despite these false positives, the precision rate of our method still remains at 96.9%. However, in some circumstances which require more strict precision rates, our method is not adequate without manual annotations.

False Negative Scenarios: According to Fig. 8(c), the recall rate of our method has much new upgradation compared to DepSys and UTEA. We can automatically detect numerous interferences (e.g., the interferences between windows and AC and between humidifiers and windows) that can be easily ignored during manual annotations. Although we constrain the trigger conditions, interference still occurs frequently when the quantity of rules is large. Meanwhile, our method still has 26.2% of undetected interferences. These undetected interference problems are also unrecognizable in DepSys, while only UTEA can detect them. They have the common feature that the targets of the action part are rated to a specific range. For example, rule A activates the AC heating mode, keeping the temperature of the room between 20 °C–25 °C when the temperature is below 20 °C , while rule B commands the heater to warm the room over 28 °C. In this situation, rule A and rule B have interference because their targets of action are contradictory. As the manual annotations of DepSys are related to the variation trend (up or down, warmer or colder), it cannot detect this type of interference as rule A and rule B both make the room warmer. The manual annotations of UTEA are adequate because they record the target range of environment variations. Our method cannot detect such interference without appropriate manual annotations.

B. Extended Experiments

In practice, any device may have implicit interference with other devices. What is the performance of A3ID in a different situation with more complicated species of devices? To answer this question, we design the device extension evaluation to verify the effectiveness of our method. The purpose of this section is to verify the effectiveness and performance of A3ID on the condition of a wider variety of actuators or devices; therefore, we perform the experiments with A3ID only. Given that these extended devices (e.g., socket, sprinkler) have sufficient functionality descriptions in knowledge graphs but lack the corresponding rules, we set the interference detection of actuators without the rule configuration, so as to judge whether interference exists between any pair of devices on the assumption that they are working at the same time. We design two ways of extension as follows.

Extension 1: We extend our 22 actuator types collected from IFTTT to 38 types based on actuator data from smart home systems. We collect actuator data from five different smart home systems: 1) Samsung SmartThings [4]; 2) Apple HomeKit [21]; 3) Microsoft LoT [22]; 4) Xiaomi [23]; and 5) ZigBee [44] according to their specifications and developer documents. The device number of each type in Extension 1 is shown in Fig. 9. For example, the actuator data of SmartThings is from the "capability reference" chapter of the developer documentation. The data source of HomeKit is from the service type of accessory objects in "HomeKit Developer's Guide," and the data of ZigBee is from the "device specification" part in "ZigBee Home Automation Public Application Profile (version 1.2)." The actuator types of LoT are collected based on their open-source code. As for Xiaomi, we collect the actuator list from its online developer documents.

Extension 2: We extend 22 actuators from the IFTTT rules to 99 types based on the device terms in ConceptNet, which are not limited to the smart home field. Therefore, we remove the step of characterizing smart home-specific semantics, which

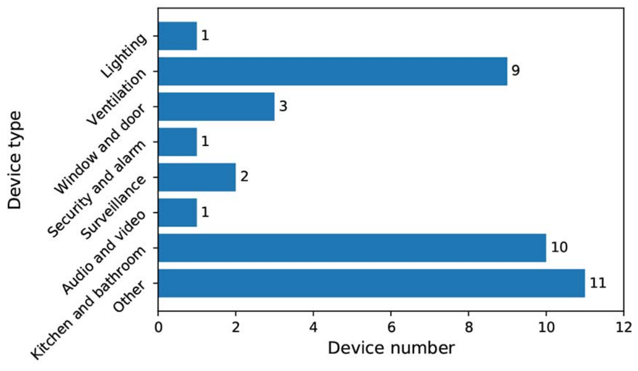


Fig. 9. Device number of each type in Extension 1.

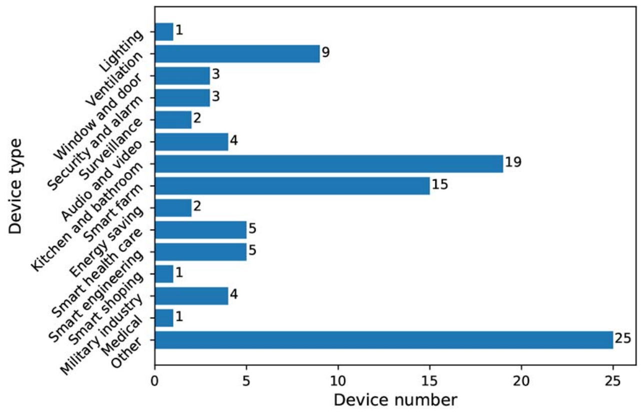


Fig. 10. Device number of each type in Extension 2.

was introduced in Section VI-C. From ConceptNet, we collect those terms that have *IsA* edge with the "device" entity. For example, "centrifuge" *IsA* "device." The device number of each type in Extension 2 is shown in Fig. 10.

The results of experiments on the extended datasets are presented in Table IX. Without considering complex conditions of rules, the performance of A3ID on the original dataset (Table IX) is better than the experiment on the rule-based evaluation (Table VIII). The experiment on Extension 1 shows the robustness and stability of our method with a small amount of increment on actuators. In Extension 2, the situation is different as we expand the dataset to a certain extent: the precision rate, recall rate, and F1-score all degrade, while the accuracy rate has been slightly increased. Extension 2 contains more implicit interferences than the other two datasets, causing more false positives and false negatives. Therefore, the precision rate, recall rate, and F1-score all decrease. While at the same time, the number of true negatives increases even more. Thus, the accuracy rate raises as a consequence.

C. Ablation Experiment

We design an ablation experiment to evaluate the performance of each component of our method. For comparison, we remove the scope analysis, effect evaluation, and

TABLE IX
PERFORMANCE OF A3ID ON EXTENDED DATASETS

Datasets	# of actuators	Precision	Accuracy	Recall	F1 score
Original	22	1	0.9957	0.750	0.857
Extension 1	38	1	0.9957	0.750	0.857
Extension 2	99	0.909	0.9988	0.667	0.769

TABLE X
PERFORMANCE OF ABLATION METHODS AND A3ID

Methods	Precision	Accuracy	Recall	F1 score
Minus scope, effect, and BWSD	0.085	0.9792	0.533	0.147
Minus effect and BWSD	0.138	0.9882	0.533	0.219
Minus BWSD	0.212	0.9891	**0.933**	0.346
A3ID	**0.909**	**0.9988**	0.667	**0.769**

BWSD components from A3ID in turn to perform the experiment. The dataset we choose in this experiment is Extension 2 from Section VIII-B. Because the dataset of Extension 2 is not limited to the smart home field, for all methods we remove the step of characterizing smart home-specific semantics, which was introduced in Section VI-C. The results are listed in Table X, which proves that A3ID performs better than any ablation method on the F1-score.

We find that the scope analysis leads to the better precision rate and accuracy rate. Effect evaluation upgrades our method on precision rate, accuracy rate, and recall rate. Although BWSD makes an extraordinary progress on the precision rate, it causes degradation on the recall rate at the same time. In A3ID, BWSD serves as a tradeoff between the precision rate and the recall rate. The reason of the recall rate degradation is that the Python WSD tool pywsd fails to give accurate "Synset" in some circumstances. For example, the minus BWSD method detects the interference between lighter and fire extinguisher, while after BWSD the interference is ignored by mistake. We find that after BWSD the word "light" in the functionality description "lighting the cigarettes" is mapped erroneously to the WordNet "Synset" (adj. unaccented, light, weak | "used of vowels or syllables").

IX. CONCLUSION

In this article, we have proposed the A3ID method, which combines NLP techniques and a lexical database to automatically detect implicit interferences with high accuracy and efficiency. It addresses one key obstacle in the interference detection area: lack of knowledge about a device. A3ID utilizes a knowledge graph to extract knowledge of devices and uses NLP to identify the interference among different devices semantically through three types of microtasks (scope analysis, effect evaluation, and polarity calculation) without human intervention, thus avoiding the intensive efforts for specifying device information. We have presented the detailed design and conducted extensive experiments on 11 859 IFTTT-style rules, five smart home systems, and ConceptNet. The results demonstrate that A3ID outperforms state-of-the-art methods by more than 33% in the F1-score for the detection of implicit interference and provides useful feedback specifying the reasons for interference problems.

REFERENCES

[1] C.-J. M. Liang *et al.*, "Systematically debugging IoT control system correctness for building automation," in *Proc. 3rd ACM Int. Conf. Syst. Energy Efficient Built Environ.*, Palo Alto, CA, USA, 2016, pp. 133–142.
[2] J. A. Stankovic, "Research directions for the Internet of Things," *IEEE Internet Things J.*, vol. 1, no. 1, pp. 3–9, Feb. 2014.
[3] R. B. Hadj, C. Hamon, S. Chollet, G. Vega, and P. Lalanda, "Context-based conflict management in pervasive platforms," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PerCom Workshops)*, 2017, pp. 250–255.
[4] *SmartThings*. Accessed: May 2019. [Online]. Available: https://www.smartthings.com/
[5] W. Vogels, "Eventually consistent," *Commun. ACM*, vol. 52, no. 1, pp. 40–44, 2009.
[6] C. Dixon *et al.*, "An operating system for the home," in *Proc. 9th USENIX Conf. Netw. Syst. Design Implement.*, San Jose, CA, USA, 2012, p. 25.
[7] J. Zhang, J. Moyne, and D. Tilbury, "Verification of ECA rule based management and control systems," in *Proc. IEEE Int. Conf. Autom. Sci. Eng.*, Arlington, VA, USA, 2008, pp. 1–7.
[8] M. Shehata, A. Eberlein, and A. Fapojuwo, "Using semi-formal methods for detecting interactions among smart homes policies," *Sci. Comput. Program.*, vol. 67, nos. 2–3, pp. 125–161, 2007.
[9] Y. Sun, T.-Y. Wu, X. Li, and M. Guizani, "A rule verification system for smart buildings," *IEEE Trans. Emerg. Topics Comput.*, vol. 5, no. 3, pp. 367–379, Jul.–Sep. 2017.
[10] Z. Lin, T.-Y. Wu, Y. Sun, J. Xu, and M. S. Obaidat, "A TAS-model-based algorithm for rule redundancy detection and scene scheduling in smart home systems," *IEEE Syst. J.*, vol. 12, no. 3, pp. 3018–3029, Sep. 2018.
[11] H. Aloulou, R. Endelin, M. Mokhtari, B. Abdulrazak, F. Kaddachi, and J. Bellmunt, "Detecting inconsistencies in rule-based reasoning for ambient intelligence," in *Proc. 21st Int. Conf. Eng. Complex Comput. Syst. (ICECCS)*, 2016, pp. 235–240.
[12] C.-J. M. Liang *et al.*, "SIFT: Building an Internet of safe things," in *Proc. 14th Int. Conf. Inf. Process. Sensor Netw.*, Seattle, WA, USA, 2015, pp. 298–309.
[13] S. Munir and J. A. Stankovic, "DepSys: Dependency aware integration of cyber-physical systems for smart homes," in *Proc. ACM/IEEE Int. Conf. Cyber Phys. Syst. (ICCPS)*, Berlin, Germany, 2014, pp. 127–138.
[14] Y. Sun, X. Wang, H. Luo, and X. Li, "Conflict detection scheme based on formal rule model for smart building systems," *IEEE Trans. Human–Mach. Syst.*, vol. 45, no. 2, pp. 215–227, Apr. 2015.
[15] M. R. Alam, M. B. I. Reaz, and M. A. M. Ali, "A review of smart homes—Past, present, and future," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 42, no. 6, pp. 1190–1203, Nov. 2012.
[16] M. García-Herranz, P. A. Haya, and X. Alamán, "Towards a ubiquitous end-user programming system for smart spaces," *J. Univ. Comput. Sci.*, vol. 16, no. 12, pp. 1633–1649, 2010.
[17] B. Ur, E. McManus, M. P. Y. Ho, and M. L. Littman, "Practical trigger–action programming in the smart home," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, Toronto, ON, Canada, 2014, pp. 803–812.
[18] K. N. Truong, E. M. Huang, and G. D. Abowd, "CAMP: A magnetic poetry interface for end-user programming of capture applications for the home," in *Proc. Int. Conf. Ubiquitous Comput.*, 2004, pp. 143–160.
[19] *IFTTT Helps Your Apps and Devices Work Together*, IFTTT, San Francisco, CA, USA. Accessed: May 2019. [Online]. Available: https://ifttt.com
[20] *Overview|Actions on Google Smart Home*. Accessed: May 2019. [Online]. Available: https://developers.google.com/actions/smarthome/
[21] *iOS—Home*. Accessed: May 2019. [Online]. Available: https://www.apple.com/ios/home/
[22] *Security Challenges the IoT Industry? Lab of Things*. Accessed: May 2019. [Online]. Available: http://www.lab-of-things.com/
[23] *Xiaomi U.S.* Accessed: May 2019. [Online]. Available: https://xiaomi-mi.us/mi-smart-home/
[24] *Home Assistant*. Accessed: May 2019. [Online]. Available: https://www.home-assistant.io/
[25] J. Cao, L. Xu, R. Abdallah, and W. Shi, "EdgeOS$_H$: A home operating system for Internet of everything," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Atlanta, GA, USA, 2017, pp. 1756–1764.

[26] H. Hu *et al.*, "Semantic Web-based policy interaction detection method with rules in smart home for detecting interactions among user policies," *IET Commun.*, vol. 5, no. 17, pp. 2451–2460, Nov. 2011.

[27] P. Carreira, S. Resendes, and A. C. Santos, "Towards automatic conflict detection in home and building automation systems," *Pervasive Mobile Comput.*, vol. 12, pp. 37–57, Jun. 2014.

[28] M. Yagita, F. Ishikawa, and S. Honiden, "An application conflict detection and resolution system for smart homes," in *Proc. 1st Int. Workshop Softw. Eng. Smart Cyber Phys. Syst.*, Florence, Italy, 2015, pp. 33–39.

[29] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity and relation embeddings for knowledge graph completion," in *Proc. 29th AAAI Conf. Artif. Intell.*, Austin, TX, USA, 2015, pp. 2181–2187.

[30] D. Vrandečić and M. Krötzsch, "Wikidata: A free collaborative knowledge base," *Commun. ACM*, vol. 57, no. 10, pp. 78–85, 2014.

[31] G. Miller, *WordNet: An Electronic Lexical Database*. Cambridge, U.K.: MIT Press, 1998.

[32] R. Speer, J. Chin, and C. Havasi, "ConceptNet 5.5: An open multilingual graph of general knowledge," in *Proc. 31st AAAI Conf. Artif. Intell.*, San Francisco, CA, USA, 2017, pp. 4444–4451.

[33] R. Navigli and S. P. Ponzetto, "BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network," *Artif. Intell.*, vol. 193, pp. 217–250, Dec. 2012.

[34] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka, and T. M. Mitchell, "Toward an architecture for never-ending language learning," in *Proc. 24th AAAI Conf. Artif. Intell.*, Atlanta, GA, USA, 2010, pp. 1306–1313.

[35] W. Wu, H. Li, H. Wang, and K. Q. Zhu, "Probase: A probabilistic taxonomy for text understanding," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, Scottsdale, AZ, USA, 2012, pp. 481–492.

[36] *Google Knowledge Graph Search API|Knowledge Graph Search API*. Accessed: May 2019. [Online]. Available: https://developers.google.com/knowledge-graph/

[37] *Natural Language Toolkit 3.4.1 Documentation*. Accessed: May 2019. [Online]. Available: https://www.nltk.org/

[38] R. Speer and J. Chin, "An ensemble method to produce high-quality word embeddings," *CoRR*, vol. abs/1604.01692, 2016. [Online]. Available: http://arxiv.org/abs/1604.01692

[39] Alvations. (May 2019). *Python Implementations of Word Sense Disambiguation (WSD) Technologies: Alvations/pywsd*. Accessed: Jan. 3, 2014. [Online]. Available: https://github.com/alvations/pywsd

[40] J. An, H. Kwak, and Y. Y. Ahn, "Semaxis: A lightweight framework to characterize domain-specific word semantics beyond sentiment," in *Proc. 56th Annu. Meeting Assoc. Comput. Linguist.*, vol. 1, 2018, pp. 2450–2461.

[41] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," *Knowl. Disc. Data Min.*, vol. 96, no. 34, pp. 226–231, 1996.

[42] X. Mi, F. Qian, Y. Zhang, and X. Wang, "An empirical characterization of IFTTT: Ecosystem, usage, and performance," in *Proc. Internet Meas. Conf.*, London, U.K., 2017, pp. 398–404.

[43] *Houseplans*. Accessed: May 2019. [Online]. Available: https://www.houseplans.com/

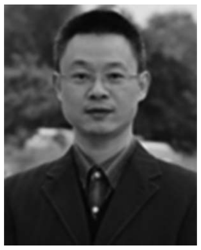
[44] *ZigBee Alliance*. Accessed: May 2019. [Online]. Available: https://www.zigbee.org/

Ding Xiao received the B.S. degree in computer science and technology from Zhejiang University, Hangzhou, China, in 2014, where he is currently pursuing the Ph.D. degree with the College of Computer Science and Technology.

Since 2014, he has been with the AI-Empowered System Software Laboratory, Zhejiang University. His research interests include Internet of Things, natural language processing, deep learning, and mobile application system.

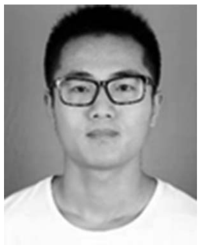
Qianyu Wang received the B.E. degree from Tianjin University, Tianjin, China, in 2016. He is currently pursuing the master's degree with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China.

He is with the AI-Empowered System Software Laboratory, Zhejiang University, where he is currently involved in Internet of Things and natural language processing.



Ming Cai received the B.S. degree in electrical engineering, and the M.S. and Ph.D. degrees in computer science from Zhejiang University, Hangzhou, China, in 1996, 1999, and 2002, respectively.

He is currently working as an Associate Professor with the College of Computer Science and Technology, Zhejiang University. He has published more than 60 research articles in international conferences and refereed journals. He is currently leading some research project supported by National Natural Science Foundation of China, Beijing, China. His research interests include Internet of Things, natural language processing, embedded system, and mobile application system.



Zhaohui Zhu received the B.S. degree in computer science and technology from Nanjing Tech University, Nanjing, China, in 2015, and the M.S. degree in computer science and technology from Zhejiang University, Hangzhou, China, in 2019.

His research interests include Internet of Things, natural language processing, and knowledge graph.

Weiming Zhao received the M.S. degree from the College of Computer Science and Technology, Zhejiang University, Hangzhou, China, in 2019.

His research interests include Internet of Things, natural language processing, and knowledge graph.