Leveraging NLP to Enable Analysis of User Driven Routines

Ruhao (Tony) Tang
The College of William and Mary
Department of Computer Science
rtang@email.wm.edu

Abstract

Routines are trigger-action programs that user create when automating their home. While routine itself offers a wide variety of utilities with a simple conditional format: if *trigger* then *action*, misconfigured or conflicting routines can be potentially dangerous. In order to fully analyze home automation, we should not only be required to examine IoT apps, which are routine created by third-party developers, but should also analyze routines from users' perspective, also known as *user driven routine*. However, a direct analysis of user driven routines is non-trivial if routines are expressed in natural language, *i.e.*, without any constraints on what the user can express. In this paper, we propose an approach that utilize Natural Language Processing (NLP) to automatically transform user driven routines expressed in natural language into intermediate representation, which can later be automatically analyzed. We evaluate this approach with a dataset of 250 user driven routines from a concurrent work. Specifically, we extracted device names (*i.e.*, Light Bulb), device capabilities (*i.e.*, switch), and device variables (*i.e.*, on or off) from user driven routines. This approach produced an accuracy of 80.64% in one of the intermediate representations transformation. These results demonstrated that our approach can efficiently help identifying key device properties from *natural* user driven routines. Furthermore, we separately analyze our intermediate representations to provide additional insights on the characteristic and composition of routines from the end-users' perspective. Finally, we describe some challenges to this approach and propose several potential improvements for future work.

1 Introduction

Recent rapid growth of Internet of Things (IoT) technology has made home automation more attractive and more affordable. With smart home devices becoming increasingly more popular, over 20 billion of them are projected to be in use by 2020, [3] numerous smart-home platforms start to appear on the market. Much of the rise in popularity can be attributed to the tremendous utility offered by the smart devices, in which they were able to offer the convenience of automating their home. These devices range from small motion sensors to large digital house appliances such as refrigerator, and enable complex operations across devices (*e.g.*, "Turn on the security camera when a motion is detected while I am away from the house"). However, with increasing in convenience of home automation, the associated security risk is also increasing. In addition, automation of smart home opens the attack surface of cyber-physical environment, allowing adversaries to compromise and endanger users' safety without physical access to users' residence. For instance, an adversary can exploit a SSL vulnerability to compromise a door lock, and with that, gaining physical access to a place that is not normally accessible. Just recently, by exploiting the use of common factory default username and password in over 600,000 IoT devices, a historically large scale Distributed Denial of Service (DDoS) attack was launched by Mirai Malware trying to cause a massive Internet shortage. [8]

End-user programming is often done via a trigger-action framework provided by most Smart Home platforms. These platform vendors often provide an interactive user interface (UIs) that allows user to configure and customize their individual routines. Such design is in a stark contrast to application based platform like Android or iOS, where users depend entirely on applications build by third-party developers. These UIs allow users to create routines by tying together the capabilities of individual devices in the form of trigger-action chains, allowing users who do not have any programming experience to create such programs with ease. However, these user-driven home automations can be problematic since it introduces a set of unexplored scope that may not be apparent from just analyzing IoT apps built by platform vendors or third-party developers. This is illustrated by the dataset we used from a concurrent work which collected from the Computer Science department, in which 107 out of 250, 42.8%, routines created by 37 users cannot be represented by any of the IoT apps offered by the public SmartThings market. This perfectly demonstrates the existence of a mismatch between user needs and developer-provided functionalities. In addition, this mismatch highlights the unknown nature of user-driven routines and complicates the practical security analysis.

By enabling consumers to control and automate a diverse set of physical objects in their home, home automation platforms are offering a new level of functionality and convenience. While this convenience can be advantageous and beneficial, security flaws in the platform or any smart device can have severe consequences for the safety of users' physical environment. Prior work by Kafle et al. [4] has demonstrate the potential for the misuse of routines in Google's NEST platform by being able to perform a lateral privilege escalation attack. During this attack, they were able to first compromise a third-party app that only had access to a low-integrity device (*e.g.*, TP Link Kasa Switch), and using the compromised app to trigger an execution of a security sensitive routine, thereby *indirectly* change the state of a high-integrity device (*e.g.*, the NEST security camera).This finding motivates the need to assess the security of home automation using a holistic approach. Similarly, Celik et al. [1] proposed a static analysis system, Soteria, that can detect side-effects of concurrent execution of IoT apps. Soteria demonstrated that some IoT apps may accidentally trigger an execution of another app, thus leading unexpected and often harmful side-effects. In a similar fashion, IoTMon [2] has demonstrated how malicious applications could deliberately affect environment factors such as the temperature in order to trigger other IoT app.

Many similar prior works have addressed many IoT security problems by demonstrating the potential benefits to analyze not just IoT apps but also the execution of them. While the lessons and insights from prior work are valuable, they all suffer one common critical limitation: *one-sided perspective*. By analyzing solely on IoT apps from either platform vendors or third-party developers, prior work limited their analysis by relying on a potentially narrow and limited developer's perspective. Although this focus is certainly important, limiting focus to developer's perspective will undoubtedly prevent stakeholders prioritizing more realistic problems that can occur in home automation setup. This limitation motivates the main contribution of the paper, a method that enables analysis of user-driven routine to facilitate a natural perspective of home automation.

As mentioned previously, most prior work that assess the security of home automation draws insight from analyses of IoT apps, but the nature of such analyses is limited and can only be applied to existing routines that they are analyzing. To overcome this limitation, we have to examine user-driven routines. However, analyzing user-driven poses several challenges: (1) data collection of user-driven routines; (2) develop a systematic approach to analyze *natural* language. We were able to overcome the first challenge with a dataset of 250 routines from a concurrent study and propose a systematic approach that leverages NLP tools and techniques to transform user-driven routines into meaningful intermediate representation.

The contributions of this paper are as follows:

- We motivate the need for a holistic evaluation of home automation through the analysis of *user-driven routines*
- We develop a systematic approach that enables analysis of user-drive routines by converting them into intermediate representation.
- We analyze the result of these intermediate representation and provide insights to a series of question regarding the characteristics and compositions of user-drive routines.

The remainder of this paper proceeds as follows: In Section 2, we provide some background information on the dataset that we are using and the reason behind it. In Section 3, we describe the motivation behind the transformation of user-driven routines to intermediate representation using NLP. In Section 4, we describe the high-level design and details on each stages of our proposed approach, and our evaluation of these models follows in Section 5. Section 6 provides additional insights from the generated intermediate representations. Section 7 discusses the limitation of the models and possible improvement for future work. In Section 8, we highlight relevant related work, and Section 9 concludes.

2 Background

One of the main challenges of analyzing user-driven routines is obtaining them from existing or potential users. To achieve this, we utilize an existing dataset of 250 user-driven routines collected from a concurrent work. Participants were given the option to select any number of smart home devices from a list of 70 for their ideal smart home setup. In addition of receiving a device list, participants were provided with a device-capability list to guide with the creation of routines. Most importantly, participants were able to create routines for their smart home in the form of plain English text, which overcomes the first challenge for this study (*i.e.*, data collection of user-driven routines).

2.1 NLP Tools and Techniques

While the routine from our dataset is in a semi-structured format, parsing each one is still a non-trivial task. To analyze these phases, we rely on several existing natural language processing (NLP) tools and techniques. The first tool we tried to employ was NLTK, a Natural Language Toolkit written in Python [6]. NLTK is a leading platform for building Python programs to work with data related to human languages. In addition to have more than 50 corpora such as WordNet, NLTK also provides a suite of libraries for stemming, tagging, tokenization, semantic reasoning, and pre-processing.

It turns out that NLTK is excellent for pre-processing and tokenizing natural text, however, it is still lacking in correctly identifying the part of speech (POS) of each word. Part-of-Speech tagging is used to identify a word's part of speech based on its context and definition. For instance, the NLTK POS tagger tagged "thermostat" as a "JJ" (Adjective). This is problematic since "thermostat" is device that is quite commonly used as a device. This prompted the need to search for an alternative tool to perform POS Tagging.

Stanford CoreNLP, on the other hand, provides a more comprehensive library that does not only have a higher accuracy in POS tagging than NLTK, but also allows the parsing of syntactic dependencies of each word. This is particularly useful in that it helps us understand the grammatical structure of each sentence, and it is able to recognize different phases within a complex routine and pair subjects or objects with respect to the associated verb.

3 Motivation

In order to obtain unconstrained user-driven routines from users, participants provided routine in plain English text. While this approach provides maximum flexibility and reflects *direct* users' requirements, there are several complications that make *automatic* analysis of user-driven routines a non-trivial task. We can summarize the complications as the follows:

- **Complication 1:** Many desired behaviors for one smart home automation can involve more than one smart device. Specifically, some trigger or action or a combination of both might can incorporate multiple devices.
- **Complication 2:** In practice, users often combine multiple triggers in a large number of unique ways. For example, user can specify to turn off the security camera when user is at home AND it is morning.
- **Complication 3:** Since routines are written in plain text, key device attributes and state variables are often *implicit*. For example, one user created the following routine: If the temperature is low, then change the thermostat mode to cold. In this case, there is no references of a device name in the trigger part of the routine.

3.1 Motivating Examples

We introduce three noticeable samples of user-driven triggers for exposition and illustration (Table. 1). Each of these routines demonstrated one or more complications describe above. For example, the first user-driven trigger is:

(The presence sensor is not detected for more than 1 hour OR the time is between 1AM and 5:30AM) AND the door is unlocked.

To begin, this trigger requires multiple devices (Challenge 1) separated by conjunction "AND" and "OR". In addition, although the fact that the trigger "door is unlocked" is referring to door lock is obvious to us, a traditional script will not be able to recognize it since smart door does not 'exist', and the only commercially available parts relating to door are door lock, or doorbell.

Based on these observations, a general static analysis script without using NLP would not be *effective* in automatically gather and output the desired intermediate representations of user-driven routines. Therefore, we propose the following framework that leverages state-of-art NLP toolkits and techniques to perform the transformation of user-driven routine to intermediate representations. (Figure. 1)

4 Design and Implementation

In this section, we will describe our design choices of our approach of transforming user-driven routine to intermediate representation. Figure. 1 provides an overview of the three main stages.

Table 1: Examples of complex routine created by users.

No.	Trigger
1	(The presence sensor is not detected for more than 1 hour OR the time is between 1AM and 5:30AM) AND the door is unlocked
2	The blinds are closed AND the time is between 10AM and 6PM AND the motion sensor does not detect motion
3	The mode is Vacation AND Gas Sensor odorLevel reach "X" OR Water Leak Detector is wet OR carbonMonoxide Level is above "Y"

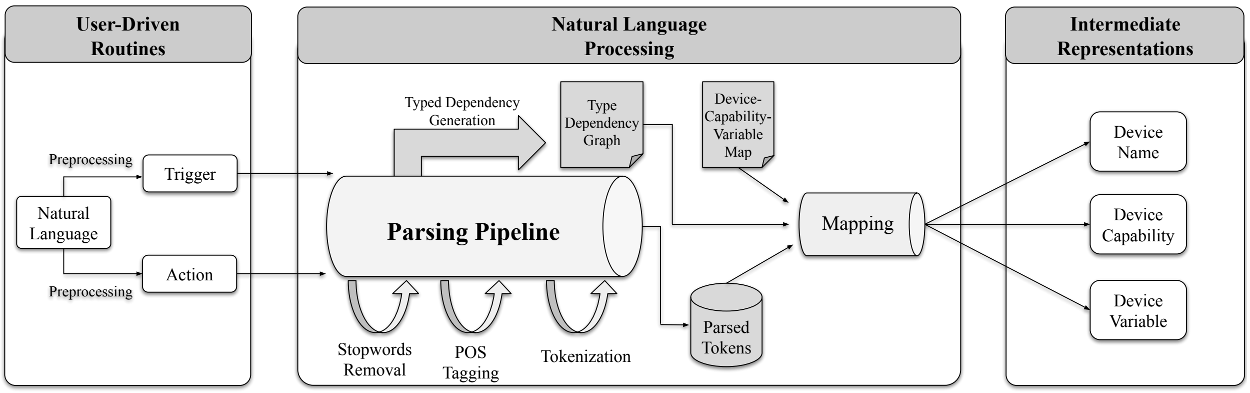

Figure 1: Overview of the framework for transforming user-driven routines into intermediate representation (IR)

4.1 Design choices

One of the other main goals of this project is to be able to generate intermediate representation *automatically*. In order to automate the process of converting user written routines to execution indicators for systematic testing, we use NLP techniques to extract devices information from human written routines. We extract and correlate the behaviors in three layers: (1) device name, (2) device capability, and (3) device variable (state).

As mentioned before, this approach consists of three main steps. In Step 1, we *pre-processes* user-driven routines through splitting them by "If" and "Then", these syntactic indicators were provided for participants when creating routines. Since both the trigger and action should each contain all the device information needed to generate the intermediate representation, we could treat them as the same problem and pass them into the same program following the same procedure.

Although triggers and actions might seem to be distinct at first glance, they share sufficient amount of similarity that can be consider equivalent in terms of being considered as input to our models (one for each intermediate representation type).

We can list the similarities as the follows:

- **Similarity 1:** Triggers and actions both have to specify the three characteristics of the devices in one way or another. For example: If air temperature is greater than 80, turn on the air conditioner. Both trigger and action have to specify some characteristic about the device.
- **Similarity 2:** Trigger are often expressed as an action, and the reverse is also true. For example, "turn on the light" can act both as the trigger of one routine and as the action of another. This demonstrates that there is not much difference in the essence of the phrase.

One other design decision we made is that we required participants of the survey to use conjunction "AND" or "OR" in all capitalized characters when combining the different trigger conditions. This is to help our models to recognize each distinct trigger condition more accurately without imposing much constraint on user.

Some triggers, such as Location Mode and Time, does not have a physical device associated with them. In those cases, we create a new device type called "Null" with the respective capabilities to incorporate them. By doing so, we were able to prevent ourselves to omit some virtual trigger conditions.

Following the pre-processing procedure in the above manner, we can obtain a list of triggers conditions and a list of actions. For the proof of concept, we implement this approach on the list of triggers. Applying this method to the list of actions is trivial, all we have to do is to replace the input from trigger conditions to the action phrases.

The processed triggers will then be pass into the next stage, the Parsing Pipeline, where the major part of the Natural Language Processing takes place. After trying out different part of speech taggers, we settled with Stanford CoreNLP POS Tagger [5] as it was able to perform better than the default NLTK POS Tagger.

The result of the Parsing Pipeline, combine with the addition of the device-capability-variable map and type dependency graph, will then be mapped into the resulting three intermediate representation.

4.2 Preprocessor

Our pre-processor accepts plain English text as the input and reduces the number of lexical tokens by eliminating unnecessary words and unrecognizable characters or words. In particular, the preprocessor performs the following pre-processing tasks:

- **Comparison Handling:** Many routines contain value comparisons as their trigger conditions. However, since there is no constraint to what user can input, we observed that a substantial number of participants uses comparison operators (*e.g.*, ">", "<", etc) to construct their routines. Therefore, in order to have NLP tools to recognize these important symbols, We replaces all instances of comparison operators into English plain text.
- **Stop words removal:** We follow standard NLP practices to pre-process our data, such as removing stop words (*e.g.*, "a", "this", "the") using NLTK library.
- **Whitelisting input:** Since there is no enforcement in our user input, we whitelisted the routines using regular expression filtering. We only accepted a narrow set of input such as characters and numbers, and punctuation symbols as these sentences would become our input to the NLP parser.

4.3 NLP Parser

Our NLP Parser accepts the pre-processed sentences and annotates each words within a sentence using standard NLP techniques. We perform the following tasks with the Stanford CoreNLP parser:

- **Name Entity Recognition:** We use the parser to identify named entities (*e.g.*, person names, organizations) in each routine. In addition, these entities are added to a dictionary so that we can perform additional natural language processing.
- **Part of Speech Tagging:** We use Stanford POS Tagger to process each word in a given routine. We see that often the case that nouns are more related to the device name than the others; for example, "smoke" is good indicator of a "smoke detector". While most cases are more complex than this example, we used this as one of our indicator for selecting the possible candidate for intermediate representation of Device Name. Similar approach can be used on identifying possible words that represents the Device Variables such as "on" or "off", which corresponds to part of speech "IN", preposition/subordinating conjunction. As for device capability, we noticed that they are generally associated with "VB", verb (*e.g.*, lock the door), would be mapped to door.lock.
- **Typed Dependency Generation:** Our more comprehensive analysis follows. Specifically, we use Stanford parser to annotate each routine with Stanford-typed dependency. One sample typed dependency graph generation of trigger condition is shown by Figure. 2. These information can be used later to help generating the intermediate representation mapping.
- **Tokenization:** When performing the above techniques, Stanford Parser would automatically tokenize each word in the sentence and output them as a tuple. This is important during the mapping phase since it saves the result of POS and type dependency generation with the associated word.

4.4 Intermediate Representation Mapping

From the result of the Parsing Pipeline, we were able to get a list of parsed tokens with valuable syntactic information associated with them. In order to create the three different types of intermediate representation, we would need to perform the mapping in three distinct ways. Before making different models, we created a device-capability-variable mapping with the all the devices mapped with its capabilities and with the possible variables / states that it can take. This mapping was done separately from the user-driven routines since it does not require any information from end-users. In order to create a model that can generate the desired intermediate representation, we would look at each token in a given routine, and examine its part of speech in addition to its typed dependency, and finally check for the device-capability-variable map as a way of confirmation.

5 Evaluation

The main goal of our evaluation is to measure the effectiveness of our models at extracting and transform key device characteristics into their respective intermediate representations. For this study, we focus on evaluating against the existing dataset of 250 user-driven routines. To test for the accuracy of

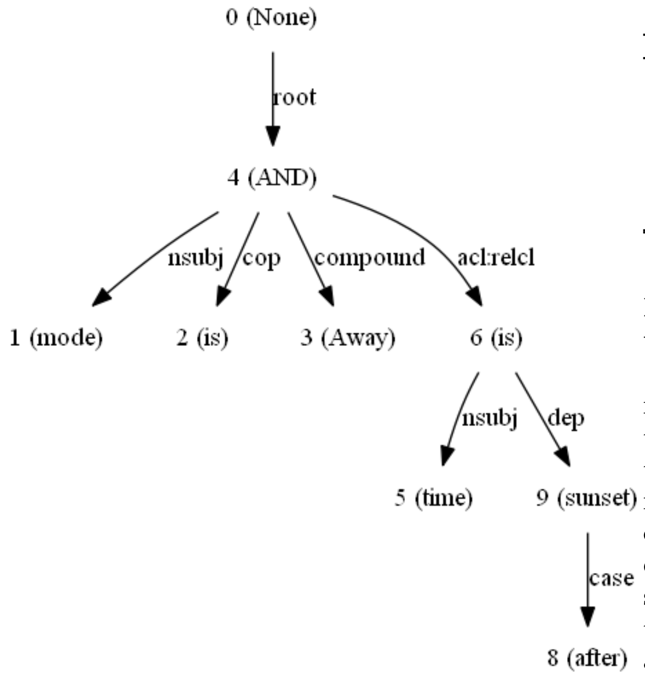

Figure 2: Typed Dependency Graph of a Sample User-Driven Trigger

the models we generated, We first manually went over each routine and produce the intermediate representations. Then, we compare the result of our manual effort with the intermediate representations generated by our models. In the end, our models were able to extract device name, device capability, and device variables with an accuracy of 80.64%, 58.04%, and 64.29% respectively. We were able to extract device name with the highest accuracy because device name often exist as is in the natural text from end-user, and we were able to extract them with minimum effort. This approach is still limited in the sense that it cannot contents that are not inside the given input. This is probably the reason that we are unable to have a decent accuracy fro device capability, which are often are not explicit.

6 Additional Insights

While the prior sections focused on the design and implementation of the generation of intermediate representation approach, this section describes several insights that can be drawn from the result of this approach with answers to a sereies of research qeustions (**RQ**). In order to get a more accurate grasp of the results, we use the manually generated intermediate representation.

Table 2: Top 8 most common routines

Routine	Count
Turn Off/On the Air Purifier based on air quality.	6
Change the Thermostat mode based on room temperature.	5
Change the AC mode based on the room temperature.	5
Change the AC mode based on the mode of the home.	4
Open/close the window Shades/Blinds based on the time.	4
Turn on the CoffeeMaker based on time of the day.	4
Sound the security alarm if smoke is detected.	3
Lock the door at Night.	3

RQ 1 - How diverse is our dataset, *i.e.*, how many unique routines?:

One of the core features of the intermediate representation is its ability to abstract and characterize the given user-driven routine. We know that a routine consists if triggers and actions. We can define a routine to be"unique" if it is consisted of different combinations of intermediate representation. In other words, if two routines end up having the same trigger and action intermediate representations, they are considered equivalent. By using this approach, we created a dictionary that maps each routine with an aggregation of all of its intermediate representations. For example, an simple routine:

If it is morning, then turn on the Coffee Maker.

would have the following aggregation:

’Morning + switch + on + Coffee Maker + switch + on’

To count the total number of unique routine, we simply count the number of unique aggregations of intermediate representation. In the end, we found that there are **220** unique user-driven routines, or 88%, in the dataset of 250. The approach we used is a overestimation that serve as the higher bound of measuring number of unique routines. An alternative method to perform the aggregation of intermediate representation is to only include the device names and device capabilities, without device variables. Intuition behind this is that device variables represents the state or the condition for the device's capability, as which does not offer much difference to a similar routine that uses the same device and capability.

RQ 2 - Top 8 most common routines:

Using similar strategy as before, we create a mapping of the aggregated intermediate representation of each routine and a count variables. After iterate though the entire dataset, we present the most common routine that is used in our dataset as shown in Table. 2. It is quite interesting to observe that Air quality and temperature are the most common factor that the user most wanted to control. This result is helpful to for critical stakeholders such as platform vendors, end-users, and security researchers

Table 3: Top 8 devices user want

Device Name	Count	%
Air Conditioner	32	86.49
Light Bulb	32	86.49
Thermostat	28	75.68
Security Camera	27	72.97
Security Alarm	26	70.27
Door Lock	25	67.57
Audio Player	23	62.16
Temperature Sensor	23	62.16

Table 4: Top 8 trigger conditions

Trigger	Count	%
locationMode+Null	27	72.97
switch+Morning	14	37.84
presence+Presence Sensor	13	35.14
switch+Night	12	32.43
temperature+Temperature Sensor	12	32.43
temperature+Thermostat	8	21.62
motion+Motion Sensor	7	18.91
airQuality+Air Purifier	6	12.77

Table 5: Top 8 actions

Action	Count	%
thermostat+Air Conditioner	28	75.68
switch+Light Bulb	20	54.05
alarm+Security Alarm	16	43.24
notification+Phone	12	32.43
lock+Door Lock	11	29.73
contact+Shades/Blinds	10	27.02
switch+Security Camera	9	24.32
thermostat+Thermostat	8	21.62

to prioritize analysis related to temperature control.

RQ 3 - What are the most common devices used by users?:

To get an insight on the devices being used by user, we look at the device name for each routine (both trigger and action) and increment the count once for each device being used by each user (we did not count multiple instances of the same device used by one user), as shown in Table. 3. This result is also interesting as it shows that, once again, user prefer to incorporate Air Conditioner(over 85%) / Thermostat control in their home automation, supporting the previous finding in **RQ 2**.

RQ 4 - What are the most common trigger conditions?:

We collect the number of triggers (aggregated device name and device capability of the intermediate representation), and we found out that most user (over 70%) would utilize "Mode" as the trigger for their automation, as shown in Table. 4

RQ 5 - What are the most common actions?:

Similar as the approach to **RQ 4**, we total the number of unique actions, based on the device name and device capability. And found out that Aic Conditioner is the most popular (over 75%) action to perform, as shown in Table. 5

RQ 6 - Can existing IoT marketplace apps satisfied most end-users' requirement? :

One of our core arguments for a paradigm shift from the predominant trend of analyzing marketplace IoT apps is that marketplace IoT apps are constrained by the limited perspective of third-party developers. Often times, end-users' direct requirement are not satisfied. Therefore, a key research question we ask is that can existing IoT marketplace apps satisfied most end-users' requirement? We conducted three experiments to assess our arguments. We first went over each routine using intermediate representation as a guide and manually determined whether if an existing IoT app can satisfy the end-user's requirements, we refer this as the "Manual" method. The manual method demonstrates that 107 out of 250, 42.8%, of the user-driven routines cannot be represented with one of public smart apps from SmartThings. Since many user-driven routines are similar and might effect the ratio, we filter the result based on two method, one focus on Device Name and Device Capability, with 47.62% cannot be represented by public IoT apps, while the other focus on all three intermediate representations, with 43.89% cannot be represented (Table. 6). For all three cases, there are more than 40% of user-driven routines that cannot be represented with one of public smart apps from SmartThings. This further calls for a paradigm shift from the traditional focus of solely on marketplace IoT apps.

7 Discussion

In addition to perform POS tagging, we could try to group words that are similar to each other by encode each word to numerical vectors representation. This can be achieved using Word2Vec, a state-of-art tool that used to produce word embedding. Word2Vec is a group of models, Specifically, two-layer neural networks that are trained to reconstruct linguistic contexts of words. It takes a corpus as input and produces a vector space, with each unique words in the corpus assigned with a corresponding vectors. Word2Vec was created by a group of researchers at Google led by Tomas Mikolov. Some advantages of Word2Vec over other rival tool such as WordNet is that it can better capture syntactic and semantic information and it can achieve a lower false positive rate than ESA (Explicit Semantic Analysis) Word2Vec also offer two types of model to produce representation of word: Continuous bag-of-word (CBOW) or continuous skip-gram. In CBOW model, the model predicts a word from a range

Table 6: Analysis of Public SmartThings Repo

Method		Count
Manual	RAW routines that can be represented with one of the public smart apps	143
	RAW routines that **CANNOT** be represented with one of the public smart apps	107
Device Name + Device Capability	Unique routines that can be represented with one of the public smart apps	99
	Unique routines that **CANNOT** be represented with one of the public smart apps	90
Device Name + Device Capability + Device Variable	Unique routines that can be represented with one of the public smart apps	124
	Unique routines that **CANNOT** be represented with one of the public smart apps	97

of surrounding words. On the other hand, a skip-gram model uses the current word to predict the surrounding windows of context words. This model is able to achieve this by assigning more weight to the neighboring words than more distant words. This technique will be useful in that we will be able to group device with its capabilities and states, and this will have a more predicting power than a standard POS tagging approach. We were unable to perform this techniques due to a limited number of data to train the model, but can be used for future work.

8 Related Work

In terms of leveraging NLP tools to perform analysis of the security of smart home automation, work by Tian et al. (SmartAuth) [8] and Ding et al. (IoTMon) [2] are similar to our paper. However, there are some key differences. First, we target different region of the home automation system. While both Tian and IoTmon are using NLP techniques to analyze IoT application, routines created by third-party developers, descriptions to identify physical channel in IoT platform, our goal is leveraging NLP techniques to analyze the *user-driven* routines. Second, we utilize the result of NLP for a different purpose. While they integrated NLP techniques to devise static analysis system to detect different security violations, we leverage NLP techniques for a quantitative analysis, and call for a motivation to prioritize routines that are used more common by real users. In addition, we investigated and propose a method that enables the analysis of user-driven routines, which are less common research area. The closest relevant work is by Surbatovich et al. [7]. They also focused on the idea of the need to examine **both** user created and developer-created If-This-Then-That (IFTTT) recipes, analogous to what an IoT app is. However, we differ in the problem scope and domain. IFTTT is a platform that connects external services like Instagram or Twitter to smart home devices. For example, user can create an IFTTT recipe that can turn on a smart light when the pizza delivery guy is on his way. This work by Surbatovich et al. focuses on exposing security and privacy ramifications of IFTTT recipes. In contrast, our goal is to enable analysis of user-driven routine in home automation, which limits our scope to events within users' home.

9 Conclusion

In this paper, we demonstrated and motivated the need for a way to enable then analysis of user-driven routines. We presented a systematic approach of extracting device names, device capabilities, and device state / condition using a combination of state-of-art NLP tools and techniques. We evaluated this approach with the manually labeled result of 250 routines from our user study. The result demonstrated that we were mostly successful in extracting device name from the natural language routines with a 80% accuracy. However, due to the fact that device capabilities and device condition / states are vague and often represented as complex phases, our straightforward approach were less accurate in extracting the corresponding device capabilities and device conditions. In future work, we can improve this kind of extraction with and potentially with another layer of extraction so that we can generate more accurate models, thus making generation of intermediate representation more representative when converting to IoT smart apps for further platform-wise testing.

Acknowledgments

I would like to express my sincere gratitude to my honors adviser, Professor Adwait Nadkarni, for his continuous support during my project. I would also like to thank Sunil Manandhar for his continuous support and valuable feedback during time of doubt. Lastly, I am grateful to my friends for their patience and encouragement that fueled me throughout the year.

References Cited

[1] Z. Berkay Celik, Patrick McDaniel, and Gang Tan. Soteria: Automated IoT Safety and Security Analysis. In *2018 USENIX Annual Technical Conference (USENIX ATC)*, pages 147–158, 2018.

[2] Wenbo Ding and Hongxin Hu. On the Safety of IoT Device Physical Interaction Control. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 832–846, October 2018.

[3] Gartner. Gartner Says 8.4 Billion Connected Things Will Be in Use in 2017, Up 31 Percent From 2016. https://www.gartner.com/newsroom/id/3598917, Accessed June 2018.

[4] Kaushal Kafle, Kevin Moran, Sunil Manandhar, Adwait Nadkarni, and Denys Poshyvanyk. A Study of Data Store-based Home Automation. In *Proceedings of the 9th ACM Conference on Data and Application Security and Privacy (CODASPY)*, March 2019. To Appear.

[5] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 55–60, 2014.

[6] Bird Steven, Edward Loper, and Ewan Klein. *Natural Language Processing with Python*. 2009.

[7] Milijana Surbatovich, Jassim Aljuraidan, Lujo Bauer, Anupam Das, and Limin Jia. Some Recipes Can Do More Than Spoil Your Appetite: Analyzing the Security and Privacy Risks of IFTTT Recipes. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1501–1510, April 2017.

[8] Yuan Tian, Nan Zhang, Yueh-Hsun Lin, XiaoFeng Wang, Blase Ur, XianZheng Guo, and Patrick Tague. Smartauth: User-centered authorization for the internet of things. In *Proceedings of the 26th USENIX Security Symposium*, 2017.