University of Salerno

Department of Computer Science

Dottorato di Ricerca in Informatica
Curriculum Internet of Things and Smart Technologies
XXXV Ciclo

Tesi di Dottorato / Ph.D. Thesis

Security and Privacy Concerns within Smart Environments: User-Centered Approaches for Defining Secure Trigger-Action Programs

Bernardo BREVE

Supervisor:	**Prof. Vincenzo DEUFEMIA**
PhD Program Director:	**Prof. Andrea DE LUCIA**

A.A 2021/2022

Bernardo Breve
Security and Privacy Concerns within Smart Environments:
User-Centered Approaches for Defining Secure Trigger-Action Programs
Ph.D. Thesis
Reviewers: Prof. Giuseppe Desolda and Prof. Autilia Vitiello
Supervisor: Prof. Vincenzo Deufemia
Co-Advisors: Prof. Giuseppe Polese and Prof. Loredana Caruccio
Ph.D. Program Director: Prof. Andrea De Lucia

University of Salerno
Data Science and Artificial Intelligence Systems Laboratory
Department of Computer Science
Via Giovanni Paolo II, 132
84084 Fisciano (SA), Italy

Agli studenti universitari.
Non è una gara, è un viaggio.
Godetevelo.

ABSTRACT

The advent of the Internet of Things (IoT) paradigm has launched a new world of opportunities, bringing with it a new understanding of objects and technology that, today, is all around us. In fact, especially in domestic environments, there is a real digital overhaul of the objects that users interact with on a daily basis. Thus, devices such as lights, TVs, cameras, locks, and electrical outlets are expanding their set of physical and technical characteristics expected from such devices with the addition of "smart" functionality, made possible by providing such devices with Internet connectivity. The activation and management of these devices, therefore, can be coordinated remotely, via smartphone, or through voice assistants such as Amazon Alexa or Google Home. Being devices eternally connected to the Internet, they never stop collecting, processing and sending data of the environment in which they are placed. For example, a temperature sensor can at any time send updates on the state of the environment, directly to the user's smartphone, or a camera can provide, at any time, a live image of the apartment. In addition, such devices can be configured to provide true automation that is triggered in response to the occurrence of certain conditions. This aspect is precisely among the most important of those offered by IoT technology; in fact, the interoperability that can be established between different IoT devices and/or communication with different web services is capable of radically simplifying the everyday life of users who make use of them.

Recently, we have witnessed the emergence of several platforms specifically designed to simplify the definition, by the end-users, of automation, based on the interaction between devices and services in a smart environment. Among these platforms, the most popular ones are those that provide a reinterpretation of the Trigger-Action paradigm, i.e., the ability to define automatisms by specifying the event (or trigger) and the action. The first component establishes the type of event for which the automation is triggered if conditions are met. Instead, the second

component refers to the operation to be performed to complete the automation. Such platforms take the name of Trigger-Action Platforms (TAPs).

The wide spread of IFTTT and other TAPs raised the question of the enormous security risks to smart environments and the privacy of end users interacting with the platforms, which might be caused by the defined behaviors. This is partly due to the high level of abstraction that TAPs provide and that very often gives little emphasis to security and privacy issues. In addition, the low level of technical knowledge that the average TAP user has, however, would not allow them to approach such issues and understand their severity.

This thesis aims to address a common problem in trigger-action platforms (TAPs) - the inability for users to create behaviors that actively protect their smart environment. To solve this issue, we propose integrating the capabilities of a newly created IoT device, called the *Intrusion Defender* (ID), into an existing TAP. The ID is capable of monitoring network traffic throughout the smart environment for unusual patterns that may indicate a cyber attack. Additionally, by providing an appropriate level of abstraction, the events detected by the ID are presented to the user in a more understandable format, allowing them to create rules that respond to, for example, a Denial-of-Service attack.

Additionally, we propose an NLP-based solution that can automatically identify any potential security and privacy risks associated with the trigger-action rules defined by the user. We achieved this by utilizing the capabilities of transfer learning models that incorporate the transformer architecture to achieve precise outcomes even with a limited amount of data for training. Specifically, we employed the Bidirectional Encoder Representations from Transformers (BERT) model developed by Google for identifying risks. The results from our experiments demonstrated the reliability and precision of the model we trained in classifying risks into three different damage classes, particularly when compared to other similar methods in the field.

To further reinforce the understanding of these risks, we use natural language models to generate example scenarios. In particular, our risk identification model is paired with a component that explains why the rule is activated through a possible scenario. We

fine-tune the model through prompts, which are virtual tokens embedded in a continuous space, to aid the model in understanding the rule's purpose. Our evaluations of the trained models prove that our approach is effective in producing plausible and contextually appropriate justifications.

ABSTRACT

L'avvento del paradigma dell'Internet of Things (IoT) ha dato vita ad un nuovo mondo di opportunità, portando con sé una nuova comprensione degli oggetti e della tecnologia che, oggi, ci circonda. In effetti, soprattutto negli ambienti domestici, c'è una vera e propria rivoluzione digitale degli oggetti con cui gli utenti interagiscono quotidianamente. Pertanto, dispositivi come luci, televisori, telecamere, serrature e prese elettriche stanno espandendo il loro insieme di caratteristiche fisiche e tecniche attese da tali dispositivi con l'aggiunta di funzionalità "intelligenti", rese possibili dalla fornitura di tali dispositivi con la connettività Internet. L'attivazione e la gestione di questi dispositivi, quindi, possono essere coordinate in modo remoto, tramite smartphone o tramite assistenti vocali come Amazon Alexa o Google Home. Essendo i dispositivi eternamente connessi a Internet, non smettono mai di raccogliere, elaborare e inviare dati dell'ambiente in cui sono collocati. Ad esempio, un sensore di temperatura può inviare in qualsiasi momento aggiornamenti sullo stato dell'ambiente, direttamente sul smartphone dell'utente, oppure una telecamera può fornire, in qualsiasi momento, un'immagine in diretta dell'appartamento. Inoltre, tali dispositivi possono essere configurati per fornire un'automazione vera e propria che viene attivata in risposta all'occorrenza di determinate condizioni. Questo aspetto è proprio tra quelli più importanti offerti dalla tecnologia IoT; infatti, l'interoperabilità che può essere stabilita tra diversi dispositivi IoT e/o la comunicazione con diversi servizi web è in grado di semplificare radicalmente la vita quotidiana degli utenti che ne fanno uso.

Recentemente, abbiamo assistito alla comparsa di diverse piattaforme specificamente progettate per semplificare la definizione, da parte degli utenti finali, dell'automazione, basata sull'interazione tra dispositivi e servizi in un ambiente intelligente. Tra queste piattaforme, quelle più popolari sono quelle che forniscono una rielaborazione del paradigma Trigger-Action, ovvero la capacità di definire automatismi specificando l'evento (o trigger) e

l'azione. Il primo componente stabilisce il tipo di evento per il quale l'automazione viene attivata se le condizioni sono soddisfatte. Invece, il secondo componente si riferisce all'operazione da eseguire per completare l'automazione. Tali piattaforme prendono il nome di piattaforme Trigger-Action (TAP).

La diffusione capillare di IFTTT e altre piattaforme TAP ha sollevato il problema dei rischi di sicurezza enormi per gli ambienti smart e la privacy degli utenti finali che interagiscono con le piattaforme, che potrebbero essere causati dai comportamenti definiti. Ciò è dovuto in parte al alto livello di astrazione che le TAP forniscono e che molto spesso dà poca importanza alle questioni di sicurezza e privacy. Inoltre, il basso livello di conoscenza tecnica che ha in media un utente TAP, tuttavia, non gli permetterebbe di affrontare tali questioni e di comprenderne la gravità.

In questa tesi, proponiamo una soluzione ad un comune difetto presente in tutte le piattaforme trigger-action (TAP), ovvero la mancanza di capacità per gli utenti di definire comportamenti che difendono attivamente il loro ambiente smart. Per risolvere questo problema, integriamo la funzionalità offerta da un dispositivo IoT creato ad hoc, chiamato Intrusion Defender (ID), all'interno di una TAP esistente. L'ID è in grado di monitorare il traffico di rete in tutto l'ambiente smart per rilevare schemi anomali che possono indicare un attacco di sicurezza informatica in corso, e attraverso l'introduzione di un adeguato livello di astrazione, gli eventi catturati dall'ID vengono presentati all'utente finale in modo più comprensibile, al fine di consentirgli di definire una regola le cui azioni si verificano in risposta ad un attacco di negazione del servizio, ad esempio.

Inoltre, proponiamo una soluzione basata su NLP in grado di identificare automaticamente eventuali rischi di sicurezza e privacy associati alle regole trigger-action definite dall'utente. Abbiamo fatto ciò sfruttando il potenziale dei modelli di transfer learning che sfruttano l'architettura transformer per ottenere risultati precisi anche con un insieme di dati limitato disponibile per la fase di addestramento. In particolare, abbiamo utilizzato il modello Bidirectional Encoder Representations from Transformers (BERT) presentato da Google per eseguire i compiti di identificazione dei rischi. Gli esperimenti condotti hanno dimostrato la affidabilità e la precisione del modello addestrato nell'identificazione dei rischi, classificandoli in 3 diverse classi di danni, soprattutto quando confrontati con soluzioni simili presenti in letteratura.

Infine, per rafforzare ulteriormente la comprensione di questi rischi, utilizziamo i modelli di linguaggio naturale per generare scenari di esempio. Abbiamo abbinato il modello per l'identificazione dei rischi con un componente di spiegabilità, che presenta uno scenario plausibile per l'attivazione della regola. Abbiamo utilizzato il fine-tuning basato su prompt, con prompt morbidi costituiti da token virtuali incorporati in uno spazio continuo, per guidare il modello nella deduzione del contesto di esecuzione della regola. Le valutazioni condotte sui modelli addestrati hanno dimostrato l'efficacia della nostra soluzione nella generazione di giustificazioni plausibili e adeguate al contesto.

CONTENTS

III Conclusion

LIST OF FIGURES

LIST OF TABLES

LISTINGS

1 INTRODUCTION

IoT-based applications are built by programming a set of IoT devices to communicate with each other and perform certain tasks, e.g., voice-controlled cameras and remote-controlled door locks. To help users define interoperability behaviors between different smart devices and web services, several platforms have been defined and commercialized [85, 89]. Among them, the most popular are the *Trigger-Action IoT Platforms* (TAPs) [152], which empower users to define custom behaviors by means of conditional rules [50, 71].

Unfortunately, the enormous spread of IoT devices, the considerable variety of data that such devices are capable of acquiring and processing, along with the user base (commonly characterized by inexperienced and non-practicing technology personalities), make IoT ecosystems a particularly attractive target for malicious characters [169]. In fact, one of the largest and most damaging cyberattacks in history involves Internet of Things devices: in fact, the Mirai botnet attack involved more than 600 thousand IoT devices that were infected and turned into pawns in a large-scale Distributed-Denial-of-Service (DDoS) attack, which saturated DNS providers Dyn, Lonestar Cell, and Liberian Telecom with requests, sending offline all sites that relied on their services [4]. Furthermore, even the use of the TAPs themselves can pose risks to the user. The definition of behavior through such platforms may lead to the formation of harmful automations. Therefore, the development of trigger-action rules should also be closely monitored, ideally through automated methods to ensure that the user is not causing harm to their smart environment or compromising their privacy.

This thesis proposes a series of ad-hoc designed solutions for monitoring, reporting, and notifying security and privacy issues connected to the behaviors definable through TAPs. In particular, the solutions leverage state-of-the-art technologies and concepts, such as automated tools for network intrusion detection or artificial intelligence models for natural language processing, with the addition of appropriate abstractions and automation, so as to enable end-users to both specify automation behaviors through TAPs that can be triggered in response, for example, to a security violation. Furthermore, the interaction with TAPs has been made more safe proof with the addition of automatic approaches for the identification and justification of any risks that might arise if a certain rule created by means of a TAP is deployed into the smart environment.

1.1 IOT TECHNOLOGY

Internet of Things (IoT) [10] is an emerging technological approach that is transforming the way we live. In the IoT vision, "smart" physical objects are networked together, able to interact and communicate with each other, with human beings and/or with the environment, to exchange data and information "sensed" about the environment. They are also able to react autonomously to events in the physical world and influence them by running processes that trigger actions and perform service functions. In Gartner's Hype Cycle 2017, IoT was considered entering the "peak of inflated expectations" meaning that the technology is now overcoming the initial enthusiasm of the novelty effect and has entered a more mature stage. Gartner indicated that this IoT technology is likely going to enter the phase of mainstream adoption in 2 to 5 years.

Thanks to this technology, new classes of disruptive interactive applications are emerging that have the potential to empower users and the quality of their life in many different fields. The need of controlling the increasing number of devices, and of interacting with them in a meaningful way is of paramount importance in order to drive the development of this technology toward socially-sustainable goals [129] as well as to prevent new forms of digital divide [63].

IoT technology weaves itself into "the fabric of everyday life" [155], which raises new challenges in terms of modeling interaction and shaping the behavior of devices consistently with the characteristics of physical objects and spaces, the actual context of use, the application domain, and the characteristics and needs of the target user. The power of IoT technologies is directly related to their capability of addressing the needs of every single end-user or class of end-users, which in turn raises the need of empowering them by offering means of customizing, if not building from scratch, IoT applications autonomously. Different approaches to designing, customizing, tailoring, and evolving interaction-rich IoT systems have been experimented and are being investigated (see, for example, the TOCHI special issue [105], which includes articles of researchers involved in this project). Yet, it is well recognized that the so-called computational thinking (i.e., the set of knowledge and cognitive skills that are involved in computer programming) is the new literacy of the digital age [127, 160] and may offer a new methodological ground to address the above issues.

The effort of configuring, controlling, and adapting interactive IoT ecosystems might be alleviated by providing end-users with carefully designed environments that allow them to "tailor" to the needs and desires of the applications they use. Yet, in order to develop usable and useful tools, it is important to ground their design on the cognitive principles of computational thinking [127], to understand the impact of other constructs such as personality [128], to leverage meaningful mental models as a solid base for the interface metaphors [108], as well as to know relevant mental theories that might foster or hinder the understanding of the task [28].

1.2 END-USER DEVELOPMENT

Users are becoming more proficient with ICT technologies and are able to create basic applications on their own. However, without the training and expertise of professional software developers, it is unrealistic to expect them to use traditional programming environments and methodologies. End-User Development (EUD) refers to the techniques, tools, and methods that enable non-professional software developers to create, modify or extend software systems [95]. Recent years have seen some investigation into topics related to EUD, but until recently EUD primarily focused on desktop-based applications such as spreadsheets that were not adaptable to changing usage contexts. The introduction of

IoT brings additional challenges, such as designing applications that can respond to events generated by various combinations of sensors, objects, services, devices, and people.

In order to effectively support the EUD of IoT applications, it is crucial to take into account the varying context of the users, technology, environment, and social aspects. Factors such as the user's tasks, preferences, and emotional state, the devices and modalities supported, and the environment's light, noise, and location all play a role in determining how the application should react to contextual events. Only the end-users themselves know the most appropriate ways in which their applications should respond to these events. To reach a wide range of users, it is important to provide a user-friendly interface that is easy to learn, while also allowing for customization and complexity in the software. This approach will enable faster tailoring, improved control over application functionality, and an overall enhanced user experience.

It is important to recognize that the foundation of EUD is the concept of tailorability, which refers to the ability for end-users to customize software to meet their specific goals and needs in various usage contexts (such as at work or at home). To achieve EUD, a shift in design paradigm is necessary, moving away from user-centered and participatory design towards meta-design, which refers to "design for designers" [64]. Under meta-design, professional developers do not design the final product as in traditional design, but instead create open software that various stakeholders, including end-users, can adapt to meet their unique requirements.

Providing end-users with the necessary tools to develop IoT applications is a way to ensure that their needs are met, and it also encourages innovation from end-users rather than just software companies. EUD focuses on the use of simple, easy-to-understand abstractions and metaphors to reduce the complexity of handling the various devices, sensors, actuators, and services involved in IoT-related domains. This approach allows for greater creativity and participation from end-users in the development process. Using meaningful abstractions in EUD, such as relevant concepts, metaphors, programming styles, vocabularies, and intuitive notations, can facilitate communication and understanding between different stakeholders, including professional software developers, domain experts, and users. Ultimately, EUD for IoT empowers users to have more control, confidence, and personalized support in their interactions with devices and smart things, both at home and in learning environments. In a field as diverse as IoT, EUD is crucial in meeting the wide range of requirements for end-use developers.

The EUD area has explored the use of rule-based solutions, which involve specifying system behavior through a series of if-then statements. This approach led to the development of Trigger-Action Platforms or Systems (TAPs or TASs). One early example of this is iCAP [52], which allowed for the creation of if-then rules (a.k.a. Trigger-Action rules or Event-Condition-Action rules) to personalize access to home appliances. Recently, rule-based approaches have gained popularity in the IoT space, as they allow end-users to easily understand and configure the behavior of their applications. These ECA rules can address various aspects of a smart environment, such as the example in Fig. 1.1, where the action of opening shutters in the living room is triggered by a temperature change in the house above 25 degrees Celsius. High-level abstraction in ECA rule creation helps users to effectively and efficiently create their rules without needing to understand technical details [42].

However, even if rule specification could seem simpler than specifying a block of code, such approaches can become difficult for non-programmer users when complex rules have to be expressed. The correct formulation of logical expressions implies knowledge of some key concepts (e.g. Boolean operators, the priority of operators) that may not always be intuitive for them. Some approaches do not even support event composition at all. Therefore, further effort in enabling end users to specify rules combining multiple triggers and actions should be pursued because this would provide them with the possibility to indicate more flexible behaviors [50, 71]. As highlighted in [81], rule-based approaches, and in particular trigger-action rules, could raise some ambiguity in their interpretation due to potential discrepancies in end-users' mental models.

In [108], an established theory of mental models is used to guide the design of interfaces for TAPs so that people can easily

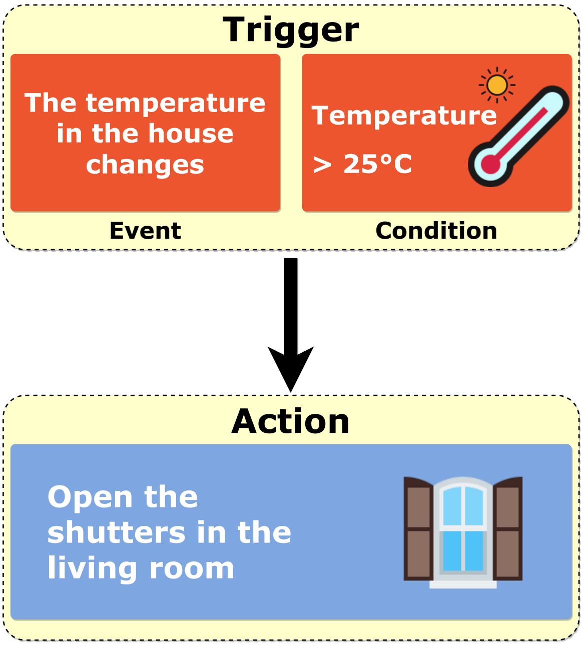

Trigger
The temperature in the house changes
Temperature > 25°C
Event
Condition
Action
Open the shutters in the living room

Figure 1.1: An example of ECA rule.

comprehend and manipulate logical expressions. According to such theory, people find it easier to conceptualize logical statements as a disjunction of conjunctions (an OR of ANDs), as opposed to other logically equivalent forms. Thus, the authors propose a paradigm to facilitate the specification of complex logical expressions that however is still far from providing general solutions.

Another interesting aspect (yet not addressed in the EUD area) is how people can test and possibly assess whether the modified/created behavior of the application actually results in the expected one. This need is especially relevant in IoT domains, where incorrect behavior of applications or actuators can eventually have safety-critical consequences (e.g., in the elderly assistance domain and in the home domain). If we consider rule-based approaches, a way to reduce the likelihood of errors is to allow users to simulate the conditions and the events that can trigger a rule and the effects that they will bring about. However, most TAPs environments do not include debugging aids [47] since non-professional end-users find debugging especially difficult. Therefore, another important area for further investigation is devising debugging mechanisms that are adequate for end users.

1.3 AIMS AND CHALLENGES

A significant issue that is often overlooked in the TAP domain is the misalignment between an end-users' mental model, influenced by their lack of computational thinking, and the potential security and privacy risks that can arise [55]. This is particularly relevant in the context of IoT devices, which have the capability to collect and make available large amounts of sensitive data, leading to an increased number of privacy and safety concerns. For instance, in a smart home setting, data about the individuals living in the house can be tracked through the monitoring of the cyber and physical activities of connected domestic devices, assisted living systems, or smart meters. This can potentially make the smart home environment and the information stored in its back-end vulnerable to malicious attacks. In addition, the low level of computational thinking that characterizes the typical user of TAPs and in general the users of IoT ecosystems often leads to risks caused by their own actions. It is not uncommon for inexperienced users to define trigger-action rules that introduce inconsistencies or even real flaws within the smart environment [23, 39, 138]. For example, a user who wishes to play music from their smartphone as soon as their Bluetooth earphones connect may define the rule: "`If a Bluetooth device is connected to the smartphone, then the smartphone plays the audio files contained in a folder`". This rule implies an important privacy disclosure as there may be a scenario where the selected folder contains personal audio files, and the smartphone connects to a Bluetooth speaker. The rule in Fig. 1.1 could also raise risks for the users. In this scenario, the opening of shutters is ruled only by the internal temperature of the smart house. If the user forgets that the rule is active, shutters might open on hot summer days when the house is empty, providing an entry point for thieves. Furthermore, open windows represent a risk factor for unsupervised children.

Despite this, a significant number of users are not motivated to engage with technical issues and do not fully understand the potential risks associated with ECA rules [73, 133, 157]. Several studies have shown that many ECA rules can be potentially dangerous and pose privacy and security risks [138], while the users often have little concern for these risks and do not feel responsible for them, often assuming that the companies producing IoT devices should guarantee their privacy [171]. However, there has been limited research on automatically identifying harmful ECA rules. For example, Surbatovich *et al.* used an information-flow lattice to analyze potential secrecy or integrity violations in ECA rules [138]. Paci *et al.* proposed two approaches based on information flow analysis to detect rules that unintentionally violate users' privacy by sharing private photos [118]. Other studies have focused on identifying undesired behaviors caused by the chain execution of rules [107, 162], such as actions triggered automatically without user intervention.

Furthermore, simply identifying and alerting users to the presence of security and privacy risks may not be enough for them to fully understand the gravity of these risks [162]. Therefore, it is crucial to provide clear and comprehensible explanations of the risks associated with a rule in order to increase user trust. This can be accomplished through the use of *Explainable Security* (XSec), a concept derived from the field of *eXplainable Artificial Intelligence* [74], which aims to make the reasoning behind a classification model's output transparent. Despite the importance of this issue being acknowledged in the literature, there are currently few practical approaches for providing users with explanations about security and privacy issues. One notable example is the work of Xiao *et al.* who proposed a module for detecting conflicts between automation rules and providing explanations that include the scope of the affected actuators, the effects on the environment, the functionalities provided, and the type of interference produced [162]. This allows users to understand the nature of the problem and its causes.

1.4 THESIS CONTRIBUTIONS

To provide viable solutions to the challenges introduced in Section 1.3 respectively, we make three major contributions in this thesis:

- **Definition of security-events triggered rules**
- **Identification of harmful rules at design-time**
- **Explanation of the risks raised by harmful rules**

In particular, the contributions made in this thesis aim at providing an answer to the following research question:

RQo."Can we support end-users in dealing with security and privacy concerns within smart environments?"

1.4.1 *Definition of security-events triggered rules*

We proposed an approach to allow end users to create rules that are triggered by security events [25]. To achieve this, we integrated the functionality offered by a custom-made IoT device, called Intrusion Defender (ID), within an existent trigger-action platform.

The ID is designed to monitor network traffic throughout the smart environment for detecting anomalous patterns by employing a signature-based approach. The match of a signature with a known cybersecurity attack may indicate an ongoing risk for the smart environment, generating a security event. Through the use of an appropriate level of abstraction, the events captured by the ID are presented to the end user in a more understandable and user-friendly manner, summarizing the initial set of 35 cyberattacks into 6 categories thanks to a card-sorting study. This enables the user to define a rule whose action occurs in response to one of these categories of cyberattacks, for example, a Denial-of-Service attack.

A user study was conducted on a pool of 20 participants to evaluate the effectiveness of this approach. The results of the study demonstrated that thanks to the abstraction provided by the creation paradigm of the TAP, enables even an inexperienced user to define rules to safeguard their smart environment in the same way as a more experienced individual in the cybersecurity domain would. Overall, this solution offers a more efficient and user-friendly approach to securing smart environments.

1.4.2 *Identification of harmful rules at design-time*

To mitigate the risk of defining trigger-action rules that threaten both the security and privacy of the smart environment and its users, we presented a model for identifying different types of harm associated with trigger-action rules [23]. The model is based on BERT and trained on a dataset of over 80,000 rules defined through the IFTTT platform. We managed to label the whole dataset by employing a novel strategy of labeling, combining a minimal manual effort with the advantages of semi-supervised learning approaches and an ensemble strategy. The model is able to classify 3 different types of risks by semantically analyzing the various components of a rule: personal harm, physical harm, and cybersecurity harm, with an average accuracy of 92%. This is achieved by using BERT's pre-training on a large corpus of text, which allows the model to understand the meaning of the words and phrases in the context of the rule, and thus make accurate predictions.

Our model is compared to other solutions proposed in the literature, such as one that uses information flow analysis and another that uses recurrent neural network architecture. The results of the comparison reveal that our BERT-based model, considering semantic properties, is able to discriminate more accurately between different types of risks. This is because our model's ability to understand the meaning of the words and phrases in the context of the rule, allows it to make more accurate predictions, and thus more effectively identify different types of harm.

1.4.3 *Explanation of the risks associated with harmful rules*

We paired the model for identifying risks associated with trigger-action rules with an explainability component, which presents a possible scenario, known as justifications, to which the rule's activation could lead [24]. The aim of this component is to provide a clear understanding of the risks related to the activation of a certain rule, and what could be the potential consequences of its execution.

To generate justifications, we used a method called prompt-based fine-tuning, experimenting with both hard and soft prompts. The hard prompts involve providing specific examples of justifications for the model, whereas the soft prompts involve providing more general guidance on how to construct the justifications. We find that the use of both types of prompts allows the model to deduce the context of the rule's execution in a better way. In fact, we mapped the services related to both the trigger and the action through a novel strategy comprising the generation of their embeddings by leveraging the Skip-gram architecture. By providing virtual tokens as prompts, the model is able to understand the underlying context and the relationships between different services, which enables a more effective selection of tokens to use in constructing the justification related to the identified harm. Additionally, the use of virtual tokens allows the model to generalize the justifications to other rules and environments, making the system more robust and adaptable.

We evaluated the performance of the proposed solutions using an automatic metric commonly used in natural language generation tasks, called the Bilingual Evaluation Understudy with Representations from Transformers (BLEURT), which has been shown to have a high correlation with human ratings. The evaluation of the results revealed an overall high BLEURT score value, with an average score of 0.412 and 0.416 for the QA Prompt and Span Prompt respectively, for the test set considered. This indicates that the proposed system is able to generate justifications that are coherent, accurate, and easy to understand, making it an effective tool for ensuring the security and privacy of smart environments.

1.5 THESIS OUTLINE

Overall, the document is divided into three main parts.

The first part provides an overview of trigger-action platform concepts and the rules that can be created through this paradigm.

It also examines commercially deployed solutions, such as If-This-Then-That (IFTTT), which is the most widely used trigger-action platform globally. The first part also delves into threat and risk models that an end-user may encounter when interacting with IoT ecosystems. Additionally, it includes empirical studies conducted on the IFTTT platform as well as a recent study that resulted in a dataset of rules created following that paradigm. The part concludes with the introduction of an artificial intelligence model that can classify the degree of semantic consistency between the behavior stated by a rule and the behavior enacted.

The second part focuses on end-user oriented solutions. Specifically, it discusses an approach to enable the end-user to define, via trigger-action platforms, a new type of rule, specifically, rules that relate to smart environment security risk scenarios. This approach allows the user to customize countermeasures to events that may compromise their privacy or the security of IoT devices connected to their environment. The second part also includes a natural language processing model that can process the components that form a trigger-action rule, extract its semantic information, and identify whether such a rule presents security risks to the smart environment and the privacy of the user who defined it. Finally, the second part presents an approach that, by utilizing natural language models, is able to identify inherent risks associated with trigger-action rules and provide a classification justification, thus facilitating the end-user's understanding of the risk's severity.

The third and final part concludes with a summary of the presented work and discusses potential future developments.

Part I

CONTEXT & BACKGROUND

2 TRIGGER-ACTION PROGRAMMING PLATFORMS

The Internet of Things (IoT) connects various objects, allowing them to exchange data, respond to real-world events, and activate services [10]. This technology offers many opportunities and capabilities, but also poses risks and potential problems. Unauthorized parties can monitor users' interactions with these objects, and their misuse can lead to unintended consequences and activate services that do not meet the needs of the end users.

Recently, there has been a growing interest in developing tools that empower end-users to control and customize the behavior of IoT devices in smart environments where they live and work. Researchers have highlighted the benefits of utilizing personalization techniques to combine IoT devices, such as sensors and actuators, with other software services [105]. However, the diverse nature of smart objects presents significant challenges in creating methodologies and tools that can effectively empower end-users to make the most of this technology [140]. End-User Development (EUD) is well-suited to addressing this problem [95], as it focuses on designing tools that enable end-users to compose and create their own applications, as well as support their work practices [46].

In relation to the EUD of IoT systems, event-driven and rule-based approaches have gained popularity because they allow end users to easily understand the contextual events and the corresponding behavior of their applications [144]. Various web tools have adopted the trigger-action rule paradigm to address the problem of Task Automation (TA). Specifically, Task Automation Systems (TASs) or Trigger-Action Platforms (TAPs) are software tools that are available as websites or mobile apps, and they assist non-technical users in configuring the behavior of smart objects within a smart environment. Typically, configuration is done through either *i*) visually illustrating the interaction between objects (for example, by creating graphs that show how events and data parameters flow among the different objects to achieve synchronization) or *ii*) visually defining Event-Condition-Action (ECA) rules, often through a step-by-step interface that guides users through the rule definition process. Tools such as Node-RED or Microsoft Flow offer graph-based notations that enable a high degree of customization, but they may not align with the mental model of most end users, particularly those without technical expertise [100].

2.1 TRIGGER-ACTION RULES

Given the nature of the involved devices, smart interactive spaces and their components are intrinsically event-driven. Different task-automation tools today enable configuring and synchronizing the behavior of IoT devices and Web services by means of event-driven programming [44, 50]. In this paradigm, the execution flow of an IoT system is determined by the occurrence of events, such as user actions, sensor outputs, or messages from other services. The behavior of the involved resources is then specified by means of trigger-action rules: these are chains of conditional statements that automatically trigger actions that change the status of synchronized resources when specific events occur. More specifically [103]:

- An **event** represents a state change that a resource of the IoT system is able to detect. Events bring parameters with themselves, which express the values sensed for variables the events refer to, and can be exploited to express conditions over the rule activation. State changes leading to new events can occur in data produced by the surrounding environment (e.g., the temperature in a room), they can be generated by actions of the users interacting with the system, or can refer to changes in data provided by some software services.

- A **condition** is a state of a specific contextual aspect that lasts for a period of time.

- A **trigger** corresponds to either an event or a condition or their composition through Boolean operators (AND/OR).

- As reactions to the sensed events and/or the occurrence of a condition, **actions** refer to functions that a resource of the IoT system can execute through its actuators and that therefore are able to produce state changes at different levels: on IoT devices installed environment, on smart objects, on software services managing the system.

Given these elements, trigger-action rules adhere to a common schema consisting of two parts:

- a cause(s) part, dedicated to specifying trigger(s), which defines the event(s) and/or the condition(s) that activate the execution of the rule;
- an effect(s) part, dedicated to specifying action(s) to carry out when the rule is verified.

Therefore, a rule is verified when any event(s) occur and/or any condition(s) are satisfied in its trigger part. It is executed when it is verified and the action part is performed.

2.2 STATE-OF-THE-ART ON TAPS

Event-driven software architectures have been studied for many years to tackle the design of active systems across various domains, including active databases, workflow design, and context-aware applications [57]. This architectural pattern can be implemented whenever the components of a system have to be coordinated based on the production, detection, and consumption of certain events. In this design, the trigger-action rules play a crucial role. These rules enable the specification of active behavior by identifying an event that triggers the rule, a condition that activates an action, and the action itself, which involves performing an operation on data or functions exposed.

The work in [29] provides a comprehensive review of literature from 2010 to 2016 on trigger action. The authors identified several tools for smart home control and analyzed them based on some characteristics and design principles for smart home control, identifying three tools suitable for an additional investigation. The paper conducted an experimental study involving real users to compare the three selected tools, aiming to identify the interaction mechanisms that end-users find more usable for managing smart home rules. The study's findings led to the derivation of implications for the design of future tools to support non-technical people to install, configure and use the devices of a smart home. Overall, the paper's methodology of combining a literature review with an experimental study involving real users contributes to the validity and reliability of the study's results and provides valuable insights for designers and developers of smart home systems.

Event-driven, rule-based approaches for configuring behavior in IoT applications are becoming increasingly popular due to their ease of use for end-users. These approaches enable end-users to comprehend the context of events and how they impact the behavior of their applications [144]. Various web tools have been developed to address the problem of Task Automation (TA) by utilizing the trigger-action rule paradigm. These tools help to synchronize the behavior of smart objects and services [16, 44]. Table 2.1 summarizes the key features of some well-known tools for trigger-action rule specification. Some of these tools are user-friendly and feature wizard procedures to guide users through the composition process.

Several tools utilize wizard procedures to aid users in defining data-flow chains among predefined and custom services. elastic.io offers a catalog of web services primarily focused on business applications such as Magento and SAP, and allows for the registration of custom services to access and control smart objects [67]. Similarly, Zapier allows for the composition of web services and smart objects by proposing a wizard to specify one event and one action in a "basic rule", which can be later extended with additional events and actions [167]. Filters on triggering events can also be added to further control rule activation. itDuzzit is another web tool with a composition paradigm similar to Zapier, but with the limitation of only being able to contain one trigger and one action. The wizard approach is also used by WigWag, a commercial web tool specifically designed for the automation of smart environments [158], which is also available as a mobile app.

Name	Service Type	Composition Paradigm	License	Execution Device	Target Users	Rule Type
Atooma	Device functions Web services Smart Objects	Wizard	Free Payware	Mobile	Non-tech.	IF TriggerS Do ActionS

AutomateIt	Devices functions	Item selection	Free
			Payware
				Mobile	Non-tech.	IF TriggerS
						THEN ActionS

Bip.io	Web services Custom	Wired	Free Payware	PC	Non-tech. technical	Complex process

Context-Dependent Authoring	Web services
	Smart Objects
		Item selection	Research project	PC	Non-tech.
					technical
						IF TriggerS
						THEN ActionS

Elastic.io	Web services Custom	Wizard	Payware	PC	Non-tech. technical	Data flow chain service -> service

IFTTT	Web services
	Smart Objects
		Wizard	Free
			Payware
				PC
				Mobile
					Non-tech.	IF Trigger
						THEN ActionS

itDuzzit	Web services	Wizard	Free Payware	PC	Non-tech.	IF Trigger
						THEN Action

Node-RED	Web services Smart Objects Custom	Wired Programming	Free	PC	Technical	Complex process
Spacebrew	Web services Smart Objects Custom	Wired Programming	Free	PC	Non-tech.	Publishers -> Subscribers

Tasker	Devices functions	Wizard	Free
			Payware
				Mobile	Technical	IF TriggerS
						THEN ActionS

We Wired Web	Web services Smart Objects	Item selection	Free Payware	PC	Non-tech.	When TriggerS THEN ActionS
WigWag	Smart Objects	Wizard	Payware	PC Mobile	Non-tech	When TriggerS THEN ActionS

Zapier	Web service
	Custom
		Wizard	Free Payware	PC	Non-tech	IF Trigger
						THEN ActionS

Zipato	Smart Objects	Building blocks	Payware	PC Mobile	Technical	Complex process

Table 2.1: Task-automation tools and their characteristics [29]

Other alternative composition approaches have been suggested. One such example is the We Wired Web, where rule creation takes place within a web page that is divided into two sections. The left panel is used to specify the elements that trigger the rule, while the right panel is used to define the actions that occur when the rule is triggered [5]. This approach also allows for the use of filters in the definition of triggering events, similar to Zapier.

Bip.io presents a unique approach by utilizing the graph metaphor for connecting Web services represented as nodes. Users are guided through the process of defining trigger and action properties when nodes representing Web services are connected with arrows. Similarly, Node-RED, a Web tool for composing both smart objects and Web services [84], also adopts a graph-based representation. However, Node-RED is geared towards professional users as it offers advanced rule customization options through nodes representing control statements, JavaScript functions, and debugging procedures.

Spacebrew combines the concepts used in We Wired Web and Node-RED [136] to offer a unique experience. It allows for the creation of rules within a workspace that is divided into two sections: the left panel for configuring services that publish events and the right panel for configuring services that perform actions in response to those events. These services can be connected in a wired manner. For non-technical users, Spacebrew also offers the Zipato paradigm, which is a web tool that utilizes graphical widgets to represent typical programming language constructs, such as "when-then," "while," "if," and logical operators. These widgets can be combined to create automation rules for tasks [173].

TA services can also be accessed through mobile devices, such as the Atooma app for Android [9]. This app allows for the combination of device functions, web services, and connected smart objects. The rules follow an "If-Do" paradigm, and multiple triggers and actions can be included in each rule, with the ability to add filters similar to those found in Zapier. Other apps such as AutomateIt and Tasker also support creating rules for mobile device functions and apps. The ECCE toolkit aims to support the creation of "ecologies" of smart objects by connecting them to a server for setup and management [19]. It uses an XML-based language to describe the properties of each object and generates the necessary code for their behavior on the platform server. The behavior of these connected devices can then be synchronized through a web interface for the end-user. However, this toolkit does not currently support specifying further conditions to constrain rule activation.

The manipulation of physical objects was studied as a means of defining smart object behaviors. One example of this is AutoHAN, which utilizes special cubes as "one-button remote controls" for interacting with the functions of home appliances. Each cube is dedicated to a specific function, such as Play/Pause, and users can associate these functions by placing a cube against the front of an appliance. The language's expressive power is achieved through the combination of these functions, by placing multiple cubes together and instructing AutoHAN to store the configuration. This configuration can then be used to schedule home automation processes. Another system, SiteView, also utilizes the manipulation of physical objects to create ECA rules [18]. Conditions are set by placing physical objects in a designated "condition composer area", and actions are programmed by placing objects representing appliances in a "world-in-miniature area", which is a small-scale model of the active environment. Users can view the rule representation in a "rule display" and simulate the results in an "environment display".

An example of an alternate application of user-defined event-action rules can be found in [71]. The environment described in the study allows for easy editing of a wide range of context-aware rules, including various options for triggers and actions. Additionally, it also includes features such as the ability to reuse and share rules.

2.3 IFTTT

IFTTT, short for "If This, Then That", is a platform that was established in 2010 by Linded Tibbets and Jesse Tane. It utilizes a visual interface that enables users to create automation rules, referred to as *IFTTT applets*, by specifying and configuring two primary components: the *trigger* and the *action*. The trigger defines the event or events that initiate the execution of the applet, while the action corresponds to the operation performed once the applet is triggered.

2.3.1 *IFTTT applet components*

The key components that define an IFTTT applet are as follows [174]:

- **Triggers:** triggers are events on the service. Some example triggers are "*any new post*" or "*New photo added to album*" or "textitany new movement";

- **Trigger fields:** allow users to enter filters or modifiers for a given trigger. For example, when a user selects the trigger "*New photo added to album*", they will be asked for the name of the album;

- **Ingredients:** are the attributes emitted by a given trigger. For the trigger *"New photo added to album"*, there would be, for example, the ingredients for *"PhotoURL"* and *"TakenAt"*;

- **Query:** a query provides a way to request additional data that a trigger does not include. For example, when triggering the Google Calendar service *"New Event Added"* you might need information about the event participants. The query *"Participant List"* returns the desired data;

- **Query fields:** are like fields in a form and are often populated with the ingredients of a trigger. For example, the query *"Participant List"* contains fields for "calendar ID" and *"event ID"*;

- **Action:** such as, for example, sending data or creating resources on the service. Some example actions include *"Create post"* or *"Upload photo"* or *"Turn on lights"*;

- **Action fields:**. when a user creates an applet, he assembles these ingredients into the action fields of the action. Action fields are like form fields. For the action *"Create post"*, there would be an action field for *"Title"* and another for *"Body post"*;

A user, when creating a rule can also enter:

- a **title** representative of the IFTTT applet created;

- a **description** detailing the operation of the rule created.

In these fields, the user enters text content written in natural language, on which no filtering or checking is performed by the official IFTTT.com site.

To create an applet, the user must first select the service, also known as a *channel*, associated with the trigger component, such as a social network or an IoT device producer's cloud service. Once the channel is chosen, the user can then select a trigger from a range of options, completing the first section of the applet. The process is then repeated for the action section, where the user can select the channel and its corresponding action. Depending on the chosen trigger or action, the user may need to provide additional information, known as *fields*, to complete the definition of the applet's behavior. These fields can include information such as the name of the folder to upload a file to on Google Drive or which parameters tracked by a Fitbit device should be recorded. Finally, the user can give the applet a *title* and *description* to easily remember its purpose and make it understandable to other users. Every applet created is automatically made available on the IFTTT platform for other users to activate without having to create a new one.

IFTTT was born as a free platform available to all users. Completion of the registration process gave users complete access to the huge catalog of applets created and shared by other users, as well as the ability to create an unlimited amount of applets as needed. In the late 2020s, however, the exponential growth of users involving the platform prompted the creators to introduce a revenue plan through monthly subscriptions, which not only introduced new features but also guaranteed the unlocking of certain limitations that characterized the free version. Specifically, starting in 2021, IFTTT's plans became three [175]:

- **IFTTT Free**:
  - Ability to create (and activate) up to a maximum of 3 Applets.
  - High latency for the applets (See Section 4.1.3 for an explanation).
  - Unlimited Applets run.
  - Free mobile app access.
  - Simple no-code integrations.
- **IFTTT Pro**, priced at 2.5€ (2.72$ per month):
  - Ability to create (and activate) up to a maximum of 20 Applets.
  - Low latency for the applets.
  - Multi-action Applets.
  - Customer support.
- **IFTTT Pro+**, priced at 5€ (5.44$ per month):

- Ability to create (and activate) unlimited Applets.
- Everything in Pro.
- Connect multiple accounts.
- Use queries and filter code.
- Developer tools.
- Prioritized customer support.

2.4 OPEN CHALLENGES

Tools such as IFTTT, Zapier, and Atooma offer a user-friendly visual approach for defining rules for ECA systems, which can be easily understood by most users. However, the notation used in these tools has limited expressive power and does not allow for the synchronization of complex requirements for smart objects. As a result, while the ease of use of these tools presents many possibilities for the user-centered design of smart environments, their limited expressive power limits their effectiveness. In particular, addressing security and privacy concerns is nearly impossible, creating tension with the need to protect smart objects from vulnerabilities [96].

Ur *et al.* explored the relationship between the expressivity and usability of TAP by conducting a study to investigate how average end users could handle flexible trigger-action rules in the home domain [144]. According to their findings, average users were able to effectively manipulate multiple triggers and actions to create rules. However, further research is needed to understand the attitudes of users towards rules that are similar but differ slightly in their triggers, such as a simple check or a state change. As highlighted by Huang and Cakmak [81], distinguishing between the semantics of relevant elements can be challenging, with users struggling to understand the difference between events and conditions or between different types of actions, such as extended actions, which automatically revert back to their original state after a certain period of time, and sustained actions, which do not. Therefore, it is crucial for EUD tools to assist users in correctly assigning roles to different rule elements.

Additionally, a smart environment may also present various security and privacy concerns. For instance, data exchanged among smart objects can be intercepted by malicious actors for fraudulent activities [96]. Due to their small size, smart objects often have limited hardware capabilities, making it difficult to implement advanced security protocols or protective measures within the devices. Furthermore, smart objects are vulnerable to physical tampering, wardriving, malicious software, and side-channel attacks. It is crucial to provide TASs with tools that can assist users in managing the security of their smart environments.

In conclusion, the state of the art on TAPs (or TASs) highlighted how trigger-action rules, and the paradigms defined for their creation embedded within the platforms, provide important abstraction elements for defining automatisms. However, there are still several features that, if introduced, could benefit users in a major way. Among others, there is a need to provide users with proactive features that reduce the likelihood of generating errors and inconsistencies when creating rules, be they bugs [22, 103] or security and privacy vulnerabilities [34], that may inevitably plague the smart environments, as we will see in more detail in the next chapter, motivating the pursuit of approaches that can respond to **RQ0**. Thus, the focus of this thesis is addressing the issues of security and privacy that IoT devices face. To address these challenges, the approach taken is to allow the end user to establish trigger-action rules that can be activated in response to potential security threats in the smart environment.

3 SECURITY AND PRIVACY ISSUES IN TAPS

In this chapter, we will provide considerations concerning security and privacy issues related to the Internet of Things and Trigger-Action Platforms. Indeed, in the literature, security-privacy issues related to trigger-action platforms and rules that are achievable through their interfaces have received important attention and recognition [1, 36, 152]. Studies have pointed out the actual presence of several types of errors and inconsistencies that can plague trigger-action rules, generating serious vulnerabilities that can go unnoticed, especially in light of inexperienced users who are very often totally unaware of the risks [34].

3.1 CLASSIFICATION OF THREATS

The survey produced by Chen et al. [34] classified the variety of inconsistencies that can be derived from rules and TAPs into two macro-categories, i.e., 1) logical errors in rules that lead IoT devices to behave in unexpected ways, going on to cause serious risks to security and the physical world, and 2) threats that arise directly from TAPs, where the rule management and execution processes are incorrectly implemented.

Logical errors were then further distinguished by the authors into three categories:

- **Rule prevention:** defines that scenario that occurs when the action of one rule prevents the trigger of another rule from being activated. For instance, rule 1 states that smart outlets should be turned off if no one is home in order to save energy, and rule 2 states that the smart pet feeder should be turned on if it is 10 AM. If no one is home at 10AM and the pet feeder is powered by one of the smart outlets, the pet will not be fed.

- **Rule collision:** defines a scenario where the actions of two rules are executed simultaneously but whose outcomes turn out to be conflicting. For instance, if rule 1 instructs "if the kitchen sensor detects smoke, open the window" and rule 2 instructs "if it is dark outside, close the window," while something is burning in the oven and it is nighttime, the window could not open properly due to the conflict between the two rules.

- **Unexpected rule chain:** defines a scenario in which one rule may inadvertently cause another rule to be triggered. For instance, if rule 1 states, "If the temperature in my room is below 20°C, put on the heater in my room," and rule 2 states, "If the temperature in my room is over 25°C, open the window,", then the temperature may increase after the first rule is executed, triggering rule 2 and causing the window to open, leading to a scenario where the smart environment has both a window open and the heather on, implying a considerable waste of electricity.

3.1.1 *Threat model*

Regarding the attack surfaces produced by using the trigger-action programming paradigm, Chen et al. identified the threat model by selecting five possible attack channels.

1. **Network sniffing:** by monitoring the state of a network and the traffic of IoT devices connected to it, it might be possible for an attacker, for example, to detect the state of connected devices, identify patterns in order to predict user routines or even acquire personal user data by manipulating the communication between the TAP and the IoT device.

2. **Privacy escalation attack:** an over-privileged vulnerability in the TAP allows an attacker to take advantage of accidentally granted authorization to access IoT devices that the attacker is not meant to access.

3. **Malicious applet:** some TAPs can offer a sophisticated feature called filter code. A TAP applet's filter code allows premium/paid users to change the triggers and actions while the applet is in use. A malicious TAP applet might

be created by an attacker with the intention of passively leaking sensitive user data or manipulating the settings of IoT devices when the end-user uses the malicious applet released by the attacker.

4. **Rule logical attack:** an attacker who could be able to manipulate the trigger-action rules created by the victim (for example, via IFTTT and SmartThings) can introduce a known logical error to injure the end-user, destroy the physical environment, or aid in a crime in the real world.

5. **Unintentional rule logical attack:** legit users of the TAP may unwittingly turn into an attacker if the rules they create have a logical flaw that might pose serious risks to the user or harm the physical environment where the IoT device is installed.

3.2 OPEN CHALLENGES

A smart interactive space should provide mechanisms to mitigate privacy and security concerns, such as guiding users while setting up the environment, and monitoring any possible security and privacy leak that might be generated by the user. The users should be warned in a simple and intuitive way, explaining where the security issue might be generated from and what procedure should be followed to quickly solve the issue, leaving aside some technical details having only the purpose of further confusing the unaware user. The EUD environment should (i) enable end-users to intuitively specify security and privacy requirements, (ii) monitor security and privacy leak, warning the users when this happens to provide suggestions for possible solutions, and (iii) use methods for increasing users' trust in EUD tools.

ENABLING THE SPECIFICATION OF SECURITY AND PRIVACY REQUIREMENTS According to [168], IoT environments should enhance users' technology threat models to effectively manage privacy and security risks, and communicate best practices for smart spaces. A TAP should assist users in editing rules, providing feedback on potential privacy or security violations, and preventing the deployment of such rules in the smart environment.

One example of this is the use of security lattices, as described by Surbatovich *et al.* in [138], to understand how certain statements within IFTTT recipes can lead to potential security risks. Additionally, EUD environments should support end-users in managing access control and authentication for smart objects. He *et al.* suggest using a capability-centric model to fit access control and authentication mechanisms to the various capabilities and features of smart objects [78]. This approach allows for more specific and restricted access, such as only allowing a user to turn on/off a device or update software, rather than giving complete access. Furthermore, contextual factors, such as the user's proximity to the smart space, should also impact access to smart objects' capabilities.

MONITORING SECURITY AND PRIVACY LEAK The security and privacy of IoT devices continues to be a significant concern, as they not only gather personal data such as names and phone numbers, but also have the ability to track user behaviors. Therefore, EUD environments should integrate mechanisms to protect users from potential security threats communicating best practices to fix or reduce them. For example, [88] analyzed the password policies of several Web applications and their vulnerability to default password. EUD environments should be able to provide feedback on these kinds of vulnerabilities and suggestions on how to mitigate their risks. Several privacy issues are raised when data acquired by smart objects are exchanged over the internet. For instance, [112] highlighted an important aspect of privacy issues, focusing on the analysis of smart meters. They showed that without knowledge about family information living in a house, it is possible to extrapolate enough data smart meters for discovering daily routine. Authors proposed a privacy-enhancing smart meter, which, among others, is based on a remote utility server, applying Zero-Knowledge protocols that allow for the preservation of the data acquired by the smart meters, without compromising user's privacy and still being able to accomplish their goals. EUD environments should adopt similar techniques for impeding data to be leaked and inferred by malicious external users.

INCREASING USER TRUST TOWARDS THE TAP Another important aspect of TAPs relies on the necessity to let users feel safe while interacting with the tool. Indeed, for being able to manage and synchronize the environment, TAPs require full access to IoT devices. In this scenario, users might feel unsafe granting complete access to the objects and, consequently, not encouraged in using the TAP. [142] focused on protecting users from the so-called smart apps, i.e., third-party mobile applications performing automatic operations on IoT devices. In order to exploit the devices, smart apps require users to grant permissions, such as the ability to control a lock. Tian et al. proposed to automatically analyze the permissions required by smart apps in order to inform the users whether they are truly necessary for the purposes of the app declared in the app store. Similarly, in the EUD context, user trust could be increased by informing users in a transparent manner on how the authorizations will be exploited by the TAPs.

In conclusion, the literature has shown that the critical security and privacy issues that plague smart environments urgently need to be addressed with ad hoc solutions. This urgency gains further value when one takes into account the type of users who make use of both TAPs and IoT devices, namely, inexperienced users without sufficient technological background to make them aware of the many risks they face. Therefore, this thesis presents contributions whose ultimate goal is to support end users in their approach toward the security and privacy issues that plague smart environments. Empowering them to both proactively defend the environment itself and monitor the process of interaction with TAPs by preventing them from introducing vulnerabilities themselves through the rules they create.

4 AN EMPIRICAL ANALYSIS OF IFTTT APPLETS

In Chapter 2, we focused on TAPs and highlighted the IFTTT platform as the most widely used due to its extensive collection of pre-existing rules, also known as "applets". In this chapter, we will delve into the results of our empirical analysis on IFTTT applets. This was made possible by the availability of datasets that include all the features of the applets, which were directly obtained from the IFTTT platform.

We will begin by discussing previous research on applets found in early versions of the IFTTT platform. Then, we will present a preliminary study on more recent applets, along with the updated method used to collect data from the IFTTT platform. Finally, we will introduce an automated approach for verifying the semantic consistency between the trigger-action components of an applet and its natural language description provided by its creator.

4.1 REVIEWS OF EXISTING STUDIES ON IFTTT

In this section, we will revise the studies published in the literature aiming at analyzing the IFTTT platform and its users.

4.1.1 *Practical trigger-action programming in the smart home*

One of the earliest studies found in the literature is the one conducted by Ur *et al.* in 2014 [144]. The authors utilized a catalog from the IFTTT platform from 2013, which consisted of 67,169 recipes (the name by which applets were identified back then), to conduct three different studies. In the first study, they evaluated the effectiveness of using a trigger-action approach for defining personalized behaviors involving physical smart devices. In the second study, the authors analyzed users' preferences towards specific trigger-action combinations by utilizing the 67k+ recipes scraped from the IFTTT platform. Lastly, in the third study, the authors recruited 226 participants to assess the ease of use of the interaction paradigm proposed by IFTTT.

FIRST STUDY The study was developed through a survey carried out on 318 workers at Amazon's Mechanical Turk (MTurk) asking them to imagine that they own a home with smart devices that are connected to the Internet and can be given instructions on how they can behave. At this point, participants had to imagine 5 things they would prefer the house to do entirely autonomously.

The intent of this study was to test whether what participants desired was naturally expressible through the trigger-action paradigm. The results confirmed this assumption, showing that 62.6 percent of the behaviors sent by participants were about defining functionality centered on automatism involving the smart environment. Moreover, of these behaviors, 77.9% could have been realized by leveraging the IFTTT platform.

SECOND STUDY The second study involved 67,169 written recipes, scraped directly from IFTTT.com, publicly shared by 35,295 different creators on the platform as of June 20, 2013. By considering 6 different channels associated with the control of physical devices, the objective of this study was to verify, despite the vast availability of recipes on the platform, users preferred to define new, perhaps redundant, behaviors rather than serve to search and activate recipes already on the platform. Once again, the study demonstrates that the most popular dozen channels were frequently coupled with other channels in recipes that involved physical triggers or actions, suggesting that users could find it simpler to express a relevant combination than to look for it in a big list. According to the Good-Turing [68] calculation, the likelihood of producing a combination for the triggers and the actions that didn't show up in this sample is around 11% and 9%, respectively. This outcome indicates that around 1 in 10 searches would return no results if users were only permitted to use recipes from that scrape. This outcome demonstrates that even a considerably large library of IFTTT recipes, will not include all potential combinations of triggers and actions.

THIRD STUDY The third study, again, involved a user study involving 226 Amazon MTurk workers in a test lasting a total of 30 minutes, during which participants were randomly assigned to use one of two types of user interfaces. The first one, similar to the IFTTT platform, allowed the definition of behaviors characterized by a single trigger and a single action. The second one, although identical at a first glance, allowed the definition of rules that supported multiple triggers and multiple actions. Each participant was asked to complete 10 tasks identified with letters A through J. Both interfaces allowed tasks A-F to be solved, while tasks I-J could only be solved through the multi-trigger and multi-action interface. Finally, tasks G and H were impossible to solve with either interface. In any case, users could skip to the next task through a dedicated button.

The tasks were evaluated in terms of success rates, the time required for task completion, and satisfaction in performing the tasks verified through liker-scale responses to questions. The results showed that the type of interface assigned, the participant's gender, and prior programming expertise, did not influence the success at solving Tasks A through F or as well as to determine that some tasks (G-H) could not be solved. Overall, the task completion rate was very high, close to 80%. An important effect uncovered is that of "learning", which means that the success rate of a single task is influenced by the number of tasks previously completed. As might be expected, the age of the participants had an impact on success rates. A similar argument applies when considering the time taken to complete a task, where only age proved to be a discriminating factor in the time taken to complete, while the other factors did not influence the time. Again, there was a learning effect, with participants taking less and less time in task resolution as they completed tasks. Last but not least, responses to the questionnaires report that participants concurred that creating recipes was simple and intuitive, and they would be willing to create recipes on a daily basis.

4.1.2 *Trigger-Action Programming in the Wild: An Analysis of 200,000 IFTTT Recipes*

Ur *et al.* deepened the analysis on IFTTT 2 years later, in 2015, to assess the type of recipes created by IFTTT users [145]. To do so, they collected the totality of applets shared publicly on the IFTTT platform as of September 6, 2015. These recipes were then the target of an empirical evaluation designed to verify the type of recipes created, their creators, how they are described, and to estimate the growth of the platform compared to the previous study.

THE DATASET To collect the applets, the authors used the Selenium framework to create a Web scraper for this purpose. Each publicly available recipe on IFTTT was assigned a unique ID or numeric identity. In fact, each recipe was accessible via a URL in the form *https://ifttt.com/recipes/ID*. The IDs appear to have been assigned consecutively, starting with the number 1. The creator then analyzed each result to extract the main components presented on the recipe page. The terms used to identify the various extracted components of the recipes are as follows:

- **ID**: the unique identifier of the recipe
- **Trigger Channel**: the device or the service offering a set of available triggers.
- **Trigger**: describes the actual event that triggers the recipe.
- **Action Channel**: the device or the service offering a set of available actions.
- **Action**: describes the actual action to be performed.
- **Author**: the user who created the recipe.
- **Date**: the date the recipe was created.
- **Adoptions**: the number of users who have added that recipe to “my recipes”.
- **Description**: a textual description provided by the creator of the recipe which serves as the recipe’s title.

The final dataset comprises a total of 224,590 recipes created by 106,452 different creators, such a dataset has been released publicly and made available for research on trigger-action programming. Unfortunately as of today, the dataset is no more available for download.

ANALYSIS In comparison with the dataset collected in the publication in [144], the authors found a significant increase in each statistic. In fact, compared with two years earlier, the number of triggers and actions tripled, as did the number of recipes created and their creators. The most significant stat, however, is in the number of adoptions recorded, which increased tenfold with respect to 2013.

Through a manual evaluation of the description assigned to recipes, the authors identified a few different description styles. Some of them stated explicitly what the recipe was ("Sends post to Twitter"), while others were more implicit ("Tweet latest post"). There were descriptions omitting the "then" ("If new post, send to Twitter"), and others that instead of utilizing the if-then format, use the when-then structure. Arrows were also used by authors to indicate relationships (such as "Post -> Twitter").

An interesting result was the identification of a certain degree of redundancy in the choice of triggers and actions. In fact, although the number of triggers and actions on IFTTT allowed for up to 282,624 combinations, the analysis performed by the creators showed that only 6% of these (15,961 unique combinations) were actually used by creators.

Finally, the authors pointed out that although the number of recipes posted on the IFTTT platform has been growing exponentially, at the rate of 50% per year, many of the newly submitted recipes were duplicates of recipes already in IFTTT's catalog, that is, recipes characterized by the same combination of trigger and action. This finding supports what was discovered in [144], from which it was already inferred that it appeared almost more immediate for users to define a new recipe than to go searching within the catalog for one that met their needs.

4.1.3 *An Empirical Characterization of IFTTT: Ecosystem, Usage, and Performance*

As of today, the last published study on the IFTTT platform applets was in 2017, authored by Mi et al. [109]. In that study, in addition to conducting an empirical analysis on a dataset consisting of more than 320,000 recipes (which had since been renamed applets), the authors developed a test environment to monitor the entire process of applets from creation to execution, analyzing not only what happens in the front-end but also in the back-end.

IFTTT ECOSYSTEM ANALYSIS In order to conduct a more detailed analysis of what goes on behind the scenes of an IFTTT applet, the authors have built a testing environment, which comprises also their own IFTTT service, supporting a set of triggers and actions for both IoT devices and web applications, which is charged with managing both IoT devices and third-party web services. This allowed the authors to intercept and monitor the interactions among the entities involved, i.e., the devices and/or the web services, within the IFTTT ecosystem.

Among the aspects evaluated by the authors is Trigger-to-Action (T2A) Latency, which is a metric that evaluates the time taken between the occurrence of the condition satisfying the trigger to the execution of the specified action. To perform this evaluation, the authors first considered 7 popular applets (identified as $A1 \dots A7$), measuring the T2A Latency on the test environment. The results of this first experiment showed that over a 3-day execution period, running each applet 50 times a day, the T2A Latency for A_1 through A_4 applets averaged relatively high and yet highly variable latencies, ranging from a minimum of 1 minute to a maximum of 15 minutes. For the $A5 \dots A7$ applets, on the other hand, whose major difference is that they used Amazon Alexa as a trigger, the T2A Latency turned out to be much lower on average, ranging from 1 second up to a maximum of 10 seconds. This would show that such a service might be treated in a special way by IFTTT, to which it would give higher priority.

To identify the cause of such high latency and variability, the authors considered applets again by involving them in three different experiments. In the first, the trigger service was replaced with the customs service created by the authors themselves, while in the second, both the trigger and the action were replaced with their own service. Finally, in the third and final experiment, the authors were going to replace the IFTTT engine entirely, that is, that module responsible for running the applet. Such a module has been replaced with a custom implementation following the IFTTT protocol but with a polling (update cycle) of 1 second. This last experiment was the one to demonstrate a significant decrease in T2A Latency, giving proof that the bottleneck lies in the IFTTT engine itself.

Further investigation had shown that such a high polling regime also causes several problems when a single trigger is activated multiple times, this causes the corresponding actions to be executed over a series of clusters, separated from each other by as much as several hundred seconds. In addition, polling also negatively affects concurrent applet execution, i.e., cases in which the user defines multiple applets with the same trigger expecting that, when it is triggered, distinct actions will be executed simultaneously. As mentioned, due to the high and variable T2A Latency, the distance between the execution of the two actions turns out to be even more than 2 minutes.

Finally, it was pointed out that the IFTTT platform does not perform any checks on cases where several applets could be chained together going on to generate an infinite loop, i.e., cases in which an applet A triggers an applet B which in turn triggers A again. To face such an issue, the authors urge the need for runtime detection techniques.

EMPIRICAL ANALYSIS In order to evaluate the growth of the IFTTT platform both in terms of published applets and with respect to the number of users, the authors conducted an empirical study similar to what was done previously by Ur *et al.* in [145]. In fact, the authors collected a total of 320k applets from the platform and, through a web scrape process, retrieved the following information: applet name, description, trigger, trigger service (channel), action, action service (channel), and the add counter, i.e., the number of users who activated that applet. Applets were collected weekly during the months from November

2016 to April 2017. The 200GB of data represents the biggest dataset of IFTTT applets ever scraped with 30% more applets collected with respect to the study of [145]. Furthermore, such a dataset was shared and made available for research, and it is still available for download [1].

While comparing weekly applet collection sessions, the authors showed that over the 6-month period of the experiment, the platform grew steadily with an increase in trigger services, action, applets, and total add count of 11%, 31%, 27%, and 19%, respectively. Furthermore, an in-depth analysis of the most involved services showed that, compared to the 2016 study by Ur *et al.*, more than half of them involve IoT devices. Concerning the type of IoT-related triggers and actions, the authors pointed out that the top 3 of the most used ones are Alexa, Philips Hue, and Fitbit, while on the other hand, non-IoT-related triggers include Social Networks, online services, RSS feeds, and time/location services. On the contrary, non-IoT-related actions focused more on notification services via notification or email or posting on social networks.

Finally, the authors ranked the applets by their add count, showing that the top 1% of applets contribute in 84.1% of cases. A final consideration comes from the fact that although service providers themselves may also publish their own applets, 98% of the applets shared publicly on the platform, and contributing to the top 1% in 18% of all applets activated, are created by users.

4.2 A RECENT EMPIRICAL STUDY OF IFTTT APPLETS

The study by Mi *et al.* [109] represents the latest published empirical evaluation of IFTTT, as well as the latest dataset of applets scraped from the platform and made available for the research.

With the intent not only to verify the evolution of the platform after more than 5 years, but also to provide researchers with an additional, more updated dataset on which to carry out their studies, we have carried out a new applet web scraping process, directly from IFTTT.com. A further novelty that characterizes this new scraping process is its compatibility with the

[1] https://xianghang.me/IFTTT_measurement/

new multi-trigger and multi-action paradigm introduced with IFTTT Pro (see Section 2.3), as well as the fact that it represents a workaround solution to restrictions introduced by IFTTT precisely to limit the scraping of applets. In this section, we first present in detail the approach devised for scraping applets from the IFTTT platform and then provide preliminary results of an empirical study performed on the new dataset.

4.2.1 *The scraping approach*

This section outlines the approach carried out for collecting applets from IFTTT. In fact, unlike up to 2017, the platform has changed the way individual applets are referenced within the site.

The need for a new approach to applet scraping is driven by a change in the way individual applets are referenced within the website. Whereas until 2017, in fact, applets (recipes) were uniquely identified by an ID assigned consecutively and incrementally, which allowed the scraping process to be automated by accessing *https://ifttt.com/recipes (or applets)/ID* and incrementally increasing the ID, after 2017 the URL format for applets became *https://ifttt.com/applets/ID-*title of the applet** where ID now consists of 8 alphanumeric characters. An example is the following URL, which references an Instagram applet that automatically saves photos to a Dropbox folder: `https://ifttt.com/applets/rd4EyWXx-autosave-your-instagram-photos-to-dropbox`.

This design choice made scraping applets following the previously adopted state-of-the-art approach, virtually impossible. Thus, we approached the scraping of applets by leveraging the IFTTT's sitemaps.

A sitemap is a file that classifies and identifies all of a website's pages that we want search engines to index. As its name suggests, a sitemap is a map of a website.

Like the table of contents of a book, the sitemap may be thought of as the website's table of contents. In reality, the sitemap gives a general overview of the content's organization to make it simpler for users and search engines to locate.

Listing 4.1: An example of a sitemap from IFTTT.com

```
<url><loc>https://ifttt.com/pinboard</loc><lastmod>
    2021-08-12T09:34:36-07:00</lastmod><changefreq>weekly</
    changefreq><priority>0.9</priority></url>
<url><loc>https://ifttt.com/pocket</loc><lastmod>2021-08-12
    T09:34:36-07:00</lastmod><changefreq>weekly</changefreq><
    priority>0.9</priority></url>
<url><loc>https://ifttt.com/evernote</loc><lastmod>
    2021-08-12T09:34:36-07:00</lastmod><changefreq>weekly</
    changefreq><priority>0.9</priority></url>
<url><loc>https://ifttt.com/instagram</loc><lastmod>
    2021-10-13T15:21:12-07:00</lastmod><changefreq>weekly</
    changefreq><priority>0.9</priority></url>
[...]
<url><loc>https://ifttt.com/applets/5203-notify-your-das-
    keyboard-when-your-android-phone-s-battery-is-low</loc><
    lastmod>2018-08-02T12:00:51-07:00</lastmod><changefreq>
    weekly</changefreq><priority>0.7</priority></url>
<url><loc>https://ifttt.com/applets/5210-get-the-weather-
    forecast-every-day-at-7-00-am</loc><lastmod>2017-05-30
    T17:08:49-07:00</lastmod><changefreq>weekly</changefreq><
    priority>0.7</priority></url>
<url><loc>https://ifttt.com/applets/5211-automatically-
    create-a-discover-weekly-archive</loc><lastmod>2017-01-20
    T12:03:04-08:00</lastmod><changefreq>weekly</changefreq><
    priority>0.7</priority></url>
<url><loc>https://ifttt.com/applets/5212-save-your-
    instagrams-to-google-drive</loc><lastmod>2017-04-20
    T17:17:16-07:00</lastmod><changefreq>weekly</changefreq><
    priority>0.7</priority></url>
```

Listing 4.1 shows an example of a sitemap, which provides links to both the pages of some services and the link that redirects directly to the applet page.

IFTTT sitemaps are updated monthly and are basically a collection of the featured applets that the platform would like to have indexed by search engines. This obviously implies that some applets, over time, may be repeated, reasoning that the scraping approach will have to account for any duplicates.

As one would imagine, this type of approach does not allow exhaustively and non-stop scraping of applets from the IFTTT platform, instead only applets present in the sitemap can be obtained and 1 time per month. However, to date, this represents the only way to acquire applets from IFTTT.

The scraping session considered 5 different sitemaps, acquired over a period of 5 months, and which in total allowed the collection of 55256 applets.

In order to do so, we parsed the sitemaps, isolating only the links that reference pages of the applet. Once the applet page is reached, the scraping process is identical to what was done by Ur et al. and Mi et al. in [109, 144, 145].

We categorized the information concerning the applets into 5 different files:

ACTIONLIST E TRIGGERLIST: These two files, as the name implies, contain information about the triggers and actions for each service available on the platform; in addition, for each, fields related to the variables are stored. The actions dataset has ***2508*** elements, the triggers dataset has ***3029*** elements. A comprehensive list of the feature extracted is provided in Table 4.1.

CHANNELLIST: This is the file related to ***774*** services. It includes details such as the name and ID of the service, with its description. A comprehensive list of the feature extracted is provided in Table 4.2.

CHANNELDETAILLIST: Similar to the previous file, except that, for each service, the associated actions and triggers are listed. A comprehensive list of the feature extracted is provided in Table 4.3.

APPLETS: The applet dataset, which counts ***55256*** applets, indicates, in addition to information such as name and description, the elements of which an applet is composed (actions and triggers), the number of times the applet has been installed, and the name of its creator. A comprehensive list of the feature extracted is provided in Table 4.4.

Note that, in contrast to previously published datasets, whenever an applet was created through IFTTT Pro, through which it is possible to define behaviors through a concatenation of trig-

NAME	DESCRIPTION
actionId (triggerId)	Numerical identifier of the action (trigger)
actionUrl (triggerUrl)	URL of the action (trigger) page
actionTitle (triggerTitle)	Action (trigger) name
actionDesc (triggerDesc)	Action (trigger) description
actionChannelName (triggerChannelName)	Name of the service associated with the action (trigger)
actionChannelId (triggerChannelId)	ID of the service associated with the action (trigger)
actionChannelUrl (triggerChannelUrl)	URL of the service associated with the action (trigger)
actionFieldList (triggerFieldList)	Parameters of the action (trigger)

Table 4.1: Features in the actionList (triggerList) file.

gers and actions the features related to the triggers and actions of each applet present a collection of triggers (or actions). This possibility is supported by our scraping approach.

The applets scraped from the IFTTT platform have been collected in a GitHub repository and made publicly available for research[2].

4.2.2 *Analysis results*

This section presents the preliminary results of a statistical comparison carried out between the dataset we collected (from now on IFT22) and the dataset published in [109].

Specifically, data were acquired every month from October 2021 to March 2022, obtaining a sufficient amount of data in order to purpose of inferring a correct estimate of the data characteristics. In contrast, the dataset presented in [109] was constructed by applying a crawling process from November 2016 to April 2017.

[2] https://github.com/buonleandro/IFTTT-Spider

NAME	DESCRIPTION
channelId	Numerical service identifier
channelName	Service name
channelDesc	Service description
channelUrl	URL of the service page
channelImgUrl	URL of the service icon
lastModified	Date of last service change

Table 4.2: Features in the channelList file.

NAME	DESCRIPTION
name	Service name
id	Numerical service identifier
url	URL of the service page
desc	Service description
specName	Service reference
specUrl	Service official page
type	Service category
suggestedApplet	Suggested applet IDs associated with the service
suggestedChannels	Related suggested services
triggerList	Triggers provided by the service
actionList	Actions provided by the service
lastModified	Date of last service change

Table 4.3: Informazioni detailChannelDataset.

As can be seen from Table 4.5, the number of installed applets and the number of creators is lower than in the dataset presented in [109] (−82.73% and −82.52%, respectively); this is because, excluding private ones, many applets have been removed and consequently creators who do not have published applets were not involved in the count.

In recent years starting in 2020, moreover, there are no duplicate applet links among the various platform sitemaps, an event that occurred in previous years, resulting in a reduction in the number of applets counted.

NAME	DESCRIPTION
title	Applet title
desc	Applet description
triggers	List of triggers used by applets
actions	List of actions used by applets
favoritesCount	Number of additions to favorites
addCount	Number of applet installations
creatorName	Name of the applet creator
creatorURL	URL of the creator's profile
URL	URL of the applet page
dateCreated	Date of applet creation

Table 4.4: Features of the applet

STATISTICS	IFT22	DATASET [109]
#Applet	55256	320k
#Services	774	408
#Triggers	3029	1490
#Actions	2508	957
#Adds	31, 6M	24M
#Authors	23, 6k	135k
#Snapshot	5, once a month	25, once a week

Table 4.5: Comparison between our dataset and the one from [109].

Instead, the line *additions* refers to the number of applet installations by platform users. In total they increased by 31.67%, reaching almost 32 *million*.

In general, from Table 4.5, it can be seen that services and triggers show an increase of about **double** compared to the previous dataset (+89.71% and +103.29%, respectively) while shares increased by 162.10%

SERVICES Each of the 774 services offers management functions related to home automation devices, automobiles, wearables, web services, etc.. Because of the great variety in the type of services, each of them has been assigned a **category**. The next table shows the distribution of services in the various categories, comparing it with [109].

CATEGORY	% SERVICES IFT22	% SERVICES IN [109]
Smart devices	49.10%	37.70%
IoT Hubs/Integration solutions	8.10%	9.30%
Wearable devices	2.79%	2.70%
Automobilies	0.42%	2.00%
Smartphone applications	4.60%	3.70%
Cloud storage	1.25%	2.50%
Online services	17.10%	8.80%
RSS and recommendation systems	3.49%	2.20%
Personal data managers and schedulers	1.53%	10.30%
Social networking, blogging, sharing platforms	4.60%	5.60%
Messaging, team collaboration, VoIP	5.20%	4.70%
Time & location	1.12%	1.20%
Mail services	0.70%	1.00%
Other	-	8.30%

Table 4.6: Comparison of the distribution of services across categories.

Table 4.6 shows the difference in the distribution of services differs between the two datasets. For example, no service in the proposed dataset was associated with the category *Other* (present instead in [109]), where there are all those services that do not find an exact location in the other classes.

In addition, it can be seen that in the dataset presented in [109], after the category **Smart devices** (which contains the most services overall in both datasets), the second category that includes most of the services is **Personal data managers and schedulers**; in contrast, in the proposed dataset the second category with which the most services were associated is **Online services**, a sign that in recent years there has been a greater need to need to connect web services with other services and/or devices.

Compared to the total number of services in the proposed dataset in [109], **148** were removed and **514** new ones were added. Table 4.7 shows the number of services added to each category.

As can be seen, the amount of services related to IoT devices is predominant compared to other categories. This phenomenon was also evidenced in the study conducted in [109], a sign that the platform is mainly popular among smart device manufacturers. In fact, more services were added in the category related to smart devices and in the category related to web applications (this shows that the platform is mainly used for the development of applets for IoT devices and for web services).

CATEGORY	# SERVICES ADDED	# SERVICES REMOVED
Smart devices	249	54
IoT Hubs/Integration solutions	58	14
Wearable devices	16	5
Automobilies	3	6
Smartphone applications	17	5
Cloud storage	5	2
Online services	95	40
RSS and recommendation systems	23	2
Personal data managers and schedulers	7	2
Social networking, blogging, sharing platforms	8	3
Messaging, team collaboration, VoIP	26	8
Time & location	6	2
Mail services	1	1

Table 4.7: Amount of services added compared to the dataset in [109].

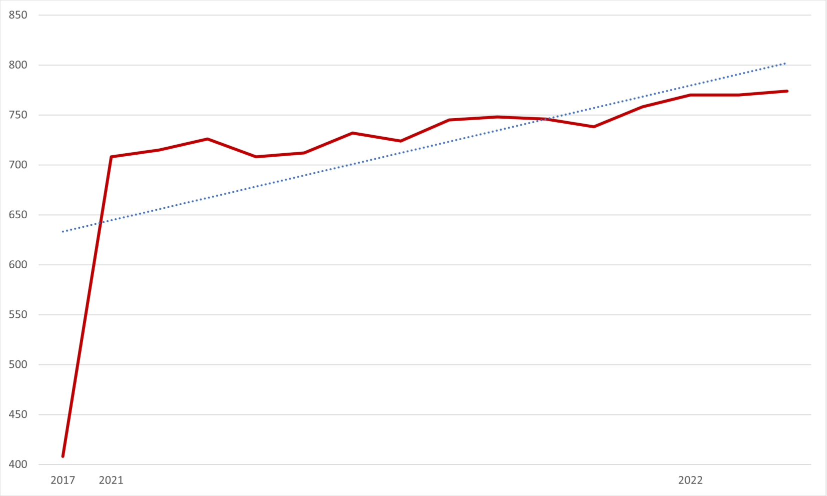


Figure 4.1: Evolution of services until March 2022.

Figure 4.1 shows the trend in the number of services over time, from the year of publication of [109] to the date of the last data collection with respect to the proposed dataset. The red line is associated with the number of services for each month, while the dashed line represents the **trend** of the data trend, indicating steady growth in the number of services over time.

TRIGGERS AND ACTIONS Hand in hand with the increase in services, the number of triggers and actions has also grown in recent years.

For each category of services, trigger and action count values were collected. In the following tables, each column represents the distribution of trigger additions (i.e., the count, in percentage, of additions of applets whose triggers belong to a service within each category) and actions.

CATEGORY	% AC TRIGGER IFT22	% AC TRIGGER [109]
Smart devices	20.79%	6.40%
IoT Hubs/Integration solutions	0.86%	0.80%
Wearable devices	0.98%	1.60%
Automobilies	0.34%	0.50%
Smartphone applications	7.42%	11.00%
Cloud storage	0.82%	0.60%
Online services	17.00%	20.00%
RSS and recommendation systems	5.90%	9.80%
Personal data managers and schedulers	7.70%	11.20%
Social networking, blogging, sharing platforms	23.74%	17,70%
Messaging, team collaboration, VoIP	2.72%	0.80%
Time & location	10.63%	14.10%
Mail services	1.10%	4.40%
Other	-	1.30%

Table 4.8: Comparison of the distribution of triggers addCounts.

As shown in Table 4.8, the most commonly used triggers in the definition of applets belong to the category related to **Smart devices**, in contrast to [109], where the most popular triggers were those related to **Web services**. Smart devices most involved triggers include ***Amazon Alexa*** and ***Google Assistant***, both of which are most associated with applets whose actions include the service ***Philips HUE*** -Philips smart devices for lighting- (163 and 80 applets, respectively).

In contrast, from the 4.9 table, it can be seen that the most frequently used actions are also part of the **Smart devices** category, whereas previously the most frequently used were those related to the **Personal Data Managers and Schedulers** category. The smart devices most involved in actions, on the other hand, include **textitAndroid Device** (a service associated with devices running the **Google** operating system) and ***Philips HUE***, which

CATEGORY	% AC ACTION IFT22	% AC ACTION [109]
Smart devices	25.22%	7.90%
IoT Hubs/Integration solutions	1.72%	1.00%
Wearable devices	0.47%	1.00%
Automobilies	0.00%	0.10%
Smartphone applications	12.71%	13.80%
Cloud storage	4.87%	13.60%
Online services	9.35%	1.90%
RSS and recommendation systems	0.68%	0.10%
Personal data managers and schedulers	10.00%	27.40%
Social networking, blogging, sharing platforms	19.23%	17,30%
Messaging, team collaboration, VoIP	5.23%	3.10%
Time & location	0.09%	0.00%
Mail services	10.43%	12.80%
Other	-	0.20%

Table 4.9: Comparison of the distribution of actions addCounts.

are most involved in applets whose triggers belong to the ***Date & Time*** service (360 and 291 applets, respectively).

Table 4.10 illustrates a ranking of the most commonly used services according to the actions and triggers they expose. Also, in Table 4.11, a list of the triggers and actions most commonly used in applet definition is presented.

TOP-10 TRIGGER SERVICES	TOP-10 ACTION SERVICES
Weather Underground (3, 48M)	Notifications (5, 09M)
Amazon Alexa (3, 06M)	Android Device (4, 15M)
Instagram (2, 70M)	Twitter (3, 06M)
Button Widget (2, 54M)	Email (2, 66M)
Space (2, 37M)	Google Sheets (2, 23M)
Facebook (2, 33M)	Philips Hue (2, 02M)
Location (2, 32M)	Google Calendar (1, 54M)
Google Assistant (1, 52M)	Dropbox (1, 48M)
Date & Time (1, 20M)	VoIP Calls (1, 03M)
Android SMS (1, 09M)	Phone Calls (US only) (912k)

Table 4.10: Trigger services and actions sorted by the total number of applet installations they take part in.

In addition, from the collected applets it is possible to analyze the interaction between the various services. By counting the occurrences of applets that have a trigger of a service belonging

TOP-10 TRIGGERS
Any new photo by you (Instagram)
Today's weather report (Weather Underground)
Image of the day by NASA (Space)
New SMS received matcher search (Android SMS)
Any new contact (iOS Contacts)
You enter an area (Location)
You enter or exit an area (Location)
Every day of the week at (Date & Time)
Tomorrow's forecast calls for (Weather Underground)
ISS passes over specific location (Space)
TOP-10 ACTIONS
Post a tweet image (Twitter)
Send a notification from the IFTTT app (Notifications)
Update device wallpaper (Android Device)
Set ringtone volume (Android Device)
Add row to spreadsheet (Google Sheets)
Quick add event (Google Calendar)
Mute ringtone (Android Device)
Send a notification from the IFTTT app (Notifications)
Update profile picture (Twitter)
Call my device (VoIP Calls)

Table 4.11: Triggers and actions sorted by the number of applet installations.

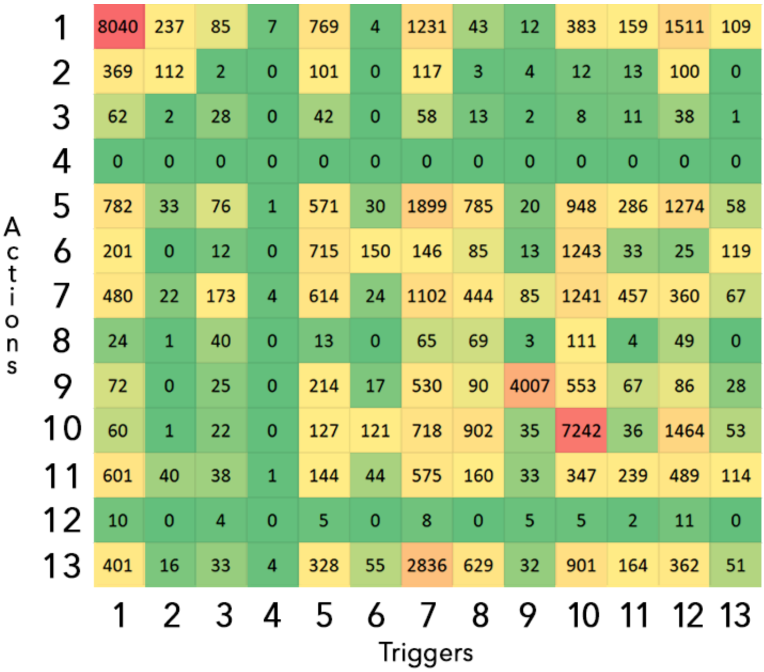
to a X category and an action of a service belonging to a Y category, it is possible to determine the pairs of categories most related to the act of building an applet.

Specifically, each coordinate frame (i, j) corresponds to the number of applets that have an action of a service of category i and a trigger of a service of category j.

Figure 4.2 illustrates a heatmap highlighting the interactions between the various categories.

Where:

1. Smart devices
2. IoT Hubs/Integration solutions
3. Wearable devices



	1	2	3	4	5	6	7	8	9	10	11	12	13
1	8040	237	85	7	769	4	1231	43	12	383	159	1511	109
2	369	112	2	0	101	0	117	3	4	12	13	100	0
3	62	2	28	0	42	0	58	13	2	8	11	38	1
4	0	0	0	0	0	0	0	0	0	0	0	0	0
5	782	33	76	1	571	30	1899	785	20	948	286	1274	58
6	201	0	12	0	715	150	146	85	13	1243	33	25	119
7	480	22	173	4	614	24	1102	444	85	1241	457	360	67
8	24	1	40	0	13	0	65	69	3	111	4	49	0
9	72	0	25	0	214	17	530	90	4007	553	67	86	28
10	60	1	22	0	127	121	718	902	35	7242	36	1464	53
11	601	40	38	1	144	44	575	160	33	347	239	489	114
12	10	0	4	0	5	0	8	0	5	5	2	11	0
13	401	16	33	4	328	55	2836	629	32	901	164	362	51

Figure 4.2: Heatmap of interactions between categories.

4. Automobiles
5. Smartphone apps
6. Cloud storage
7. Online services
8. RSS and recommendation systems
9. Personal data managers and schedulers
10. Social network, blogging, sharing platforms
11. Messaging, team collaboration, VoIP
12. Time & location
13. Mail services

For example, the applet

Mail me free games with Steam keys posted to /r/gamedeals is included in the cell count (13, 7), since it has an action associated with a category 13 service (Email) and a trigger associated with a category 7 service (Reddit).

Instead, the applet

Tell Alexa to adjust your Nest Thermostat to a certain temperature

It is included in the cell count (1,1) because the services associated with the trigger and action (Amazon Alexa and Nest Thermostat, respectively) belong both to category 1.

This evaluation demonstrated a strong correlation between specific trigger and action categories. In fact, there is a particularly high number of occurrences of applets in which a trigger belonging to the **Smart devices** category (1) is associated with an action belonging to the same category.

An example is the following applet:

If my smart camera detects turn the light on.

APPLET Figure 4.3 ordinates for the number of installations per applet. The x-axis illustrates the indices identifying applets, the y-axis illustrates the indices identifying the number of installations.

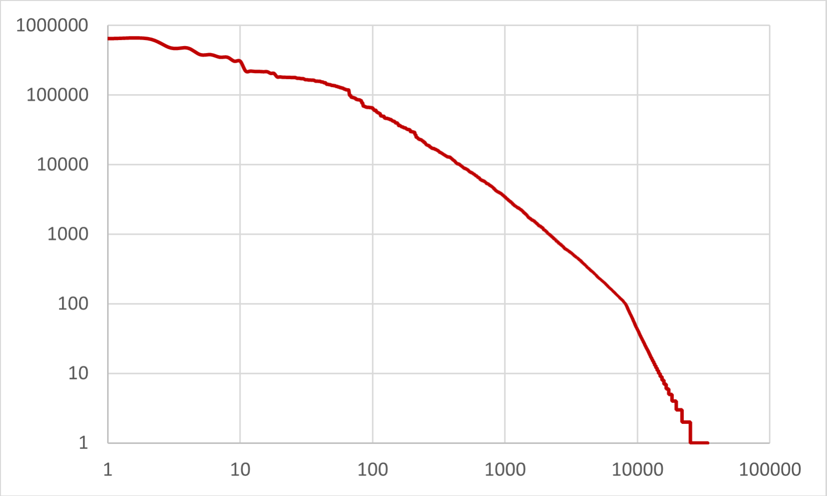


Figure 4.3: Number of installations per applet

The first 10% of the collected applets accounts for about 97.83% of the total number of installations. In general, **32087** applets have a total of maximum 2 installations while just **66** exceed *100000* installations.

Many applets help to link web services together. Moreover, from the number of installations, it can be seen that the use of applets for automating social networking channels is widespread.

As for the rest of the applets, that is, those that feature IoT devices in the trigger, action, or both, it can be seen that the platform provides applets that act as bridges to coordinate the various elements. In a smaller number, one can find applets created for testing purposes *(named "test," "test2," etc.)* or applets whose structures show remarkable similarities, with differences related only to titles or parameters.

The methodology adopted to identify differences between the proposed dataset and the dataset in [109] was also used to get an account of the number of new applets that the addition of the new services resulted in. Specifically, the new services contribute as many as **29700** applets, accounting for 68.16% of the installations (21.5M).

CREATORS Figure 4.4, shows the number of total installations recorded per creator. The x-axis illustrates the indices identifying the creators (order by number of installations), the y-axis illustrates the indices identifying the number of installations.

On the **IFTTT** platform, each user is free to create a personal account and define his or her own automations. Approximately **23600** creators (i.e., users who have at least one non-private applet published) have been identified; these, considering the portion of applets mentioned above, i.e., those that represent the first 10% of the collected applets, contribute a large number of installations (they represent approximately 27.58% in the first 10%).

Most creators contributed at most 2 applets. In contrast, creators who contributed 3 or more applets represent only 12.66% of all active users (**2988**).

Without taking into account the creator accounts corresponding to official services, such as **textitAmazon Alexa, Instagram, Facebook** or ***Google***, the total number of applet installations per user ranges from a minimum of **0** to a maximum of about **2.1 M** (with an average of **1337** total installations per user). The number of installations of a single applet, on the other hand, ranges from

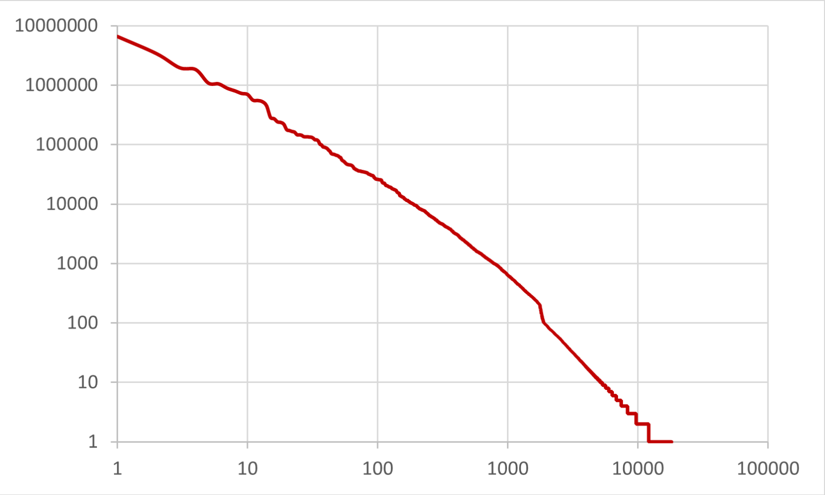


Figure 4.4: *Log* of the number of total installations per creator

a minimum of **0** to a maximum of **350647** (with an average of **589** installations per applet).

4.3 MODEL FOR AUTOMATIC SEMANTIC CONSISTENCY CHECKING

According to the IFTTT creation paradigm, when a user creates a new applet, the creator must specify a title and a description, in natural language, that summarize how the applet works. By reading these fields, a new user can more easily understand what an applet is for and decide whether or not to activate it on their device. However, on the part of IFTTT, there is no control over the content of the description and title entered by the user, so the creator could write anything, falsely describing the applet's behavior.

To this end, in this section, we present a model that can check whether there is some semantic consistency between the trigger-action components of an applet and its natural language description provided by its creator.

For example, consider the following description:

> "Create a link note on Evernote for my favorite tweets."

We define this description as consistent if the components that characterize the defined IFTTT rule are those in Table 4.12.

Trigger	New tweet liked by you
Trigger Channel	Twitter
Action	Create a new note
Action Channel	Evernote

Table 4.12: Examples of components for a consistent applet

The proposed approach attempts to evaluate the coherence of the description through a semantic approach based on the Bidirectional Encoder Representations from Transformers (BERT) model. The description is defined as consistent if it agrees with the description of the IFTTT rule synthesized from its components. The method manages to distinguish, within the description, trigger channels and action channels, what the first method fails to do.

The experimental analysis performed shows that the BERT-based model is more effective than other models implemented in the literature.

4.3.1 *Literature review*

This section presents the main studies carried out on the semantic analysis of IFTTT rules.

Studies carried out in the literature use "Language to code" approaches, i.e., which allow the components that characterize an IFTTT applet to be derived from its description. Other studies, on the other hand, attempt to simplify rule creation for the user by creating complex graphical interfaces designed specifically to enhance the user-experience of the user.

4.3.1.1 *From language to code: Using semantic parsers for IFTTT applets*

Programming computers with natural language would make it easier for inexperienced users to utilize modern technology [102]. Translating language to code often requires extensive manual work for constructing parsers or large amounts of training data. One area of interest is IFTTTs, An interesting subset of the possible program space are IFTTTs.

In [121] authors built a semantic parser that allows users to describe applets in natural language and automatically map them to executable code. For this work, 114,408 applet-description pairs were collected.

Playing a key role in achieving correct semantic derivation are ASTs (Abstract syntax tree).

In computer science, an AST, or simply syntax tree, is a tree representation of the abstract syntactic structure of text (often source code) written in a formal language.

Each node in the tree denotes a construct present in the text. Syntax is abstract in the sense that it does not represent every detail that appears in actual syntax, but rather only structural or content-related details. Similarly, a syntactic construct such as an $if - condition - then$ statement can be denoted by a single node with three branches. Once constructed, additional information is added to the AST by subsequent processing [159].

An example of AST is shown in Figure 4.5.

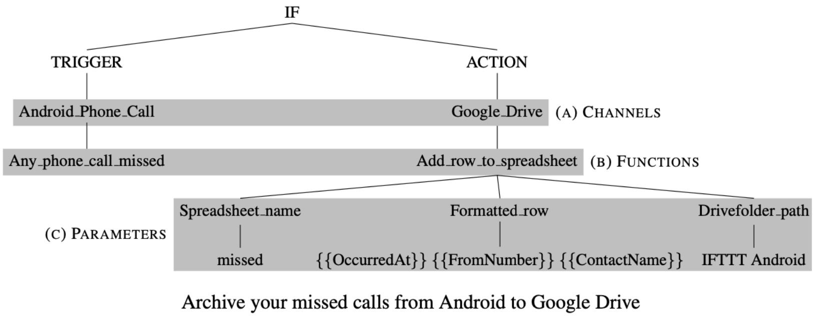


Figure 4.5: An example of AST

The constructed ASTs are given as input to a series of classifiers. The AST obtained as output from the classifier is given back as input to the classifier to improve its performance until a maximum number of iterations is reached or the desired performance is achieved.

The synthesis of the IFTTT rule from the natural language description is carried out through four steps:

- **program retrieval:** multiple users could potentially have similar needs and thus create similar or even identical programs. Given a new description, the closest description in a table of description-rule pairs is searched to return the associated program. Several text similarity metrics were used;

- **machine translation:** where data are segmented and assembled at runtime. First, each program is translated into an AST and then converted into flat sequences of tokens. The tokens are annotated with their similarity, i.e., the number of arguments or operands taken by the functions, which is useful for reconstructing the AST tree later. The collected set of tokens are converted into a well-formed program according to the formal grammar;

- **generalization without alignment:** this approach is to treat the source language as context, rather than as a hard constraint. The program, obtained from the previous step, is analyzed to extract its various instructions. For each construct in the formal grammar, a binary classifier intended to predict whether or not that construct corresponds to an instruction is trained. This classifier uses the general characteristics of the source sentence. This allows for context-dependent inference of instructions. Each instruction is assigned a probability by looking at each construct independently;

- **synchronous generation:** Previous methods focus on learning the distribution of instructions based on the input, without considering the syntax of the source sentence or any correspondence between the language syntax and the program structure. Unlike traditional systems, this approach does not use word alignment as a strict guideline. Instead, the phrasal correspondence is established during the training process to produce a semantic derivation, as shown in Figure 4.6.

This study illustrates an example of natural language translation into a program. The authors' best results were obtained by

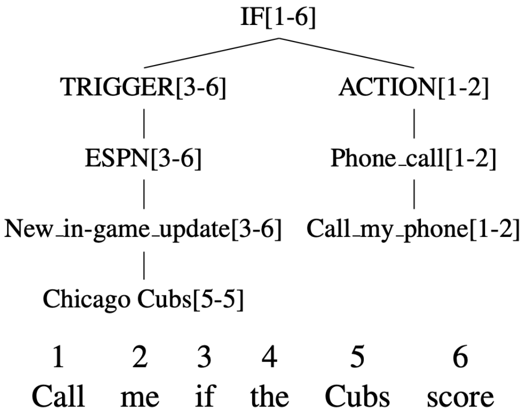


Figure 4.6: An example of semantic derivation.

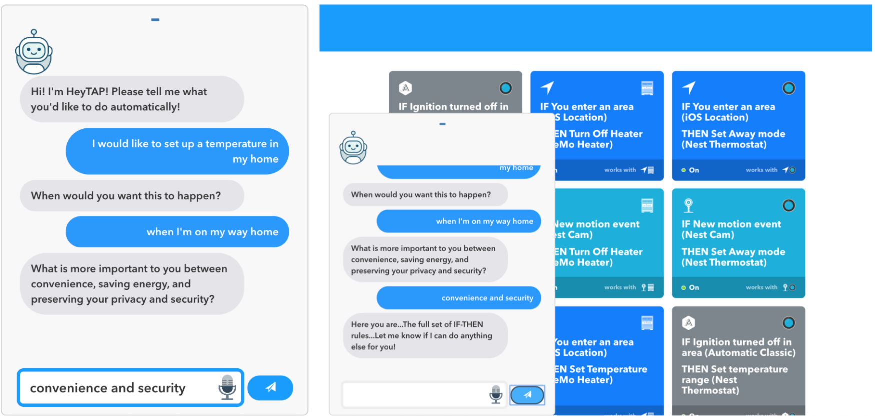
introducing the vaguely synchronous approach. In practical situations, however, many elements of the semantic representation may be implied only by description, rather than explicitly stated.

This work has considered only interactions in which the user describes a request and the system responds with an interpretation. An important next step would be to engage the user in an interactive dialogue to confirm and refine the user's intent and develop a fully functional correct program.

4.3.1.2 *HeyTAP: bridging the gap between user needs and technology in IF-THEN rules via conversation*

In [43] authors present the HeyTAP system, a conversational and semantic trigger-action programming platform that can map users' abstract needs to executable IF-THEN rules. By interacting with a conversational agent, the user communicates his or her customization intentions and preferences. The user's input, along with contextual and semantic information about available connected entities, is then used to recommend a set of IF-THEN rules that meet the user's needs.

Taking advantage of a multimodal interface, the user can interact with a conversational agent to communicate his or her personalization intentions for different contexts, for example, to personalize the temperature of his or her room when he or she is close to home as in Figure 4.7.



Hi! I'm HeyTAP! Please tell me what you'd like to do automatically!

I would like to set up a temperature in my home

When would you want this to happen?

when I'm on my way home

What is more important to you between convenience, saving energy, and preserving your privacy and security?

convenience and security

(a) Intentions and Preferences Input

my home

When would you want this to happen?

when I'm on my way home

What is more important to you between convenience, saving energy, and preserving your privacy and security?

convenience and security

Here you are...The full set of IF-THEN rules...Let me know if I can do anything else for you!

IF Ignition turned off in

IF You enter an area
S Location)
EN Turn Off Heater
eMo Heater)
works with

IF You enter an area (iOS Location)
THEN Set Away mode (Nest Thermostat)
On works with

lew motion event
st Cam)
EN Turn Off Heater
eMo Heater)
works with

IF New motion event (Nest Cam)
THEN Set Away mode (Nest Thermostat)
On works with

ou enter an area
S Location)
EN Set Temperature
eMo Heater)

IF Ignition turned off in area (Automatic Classic)
THEN Set temperature range (Nest Thermostat)

(b) Recommendations Selection

Figure 4.7: HeyTAP GUI [43].

By interacting with the agent, the user can also specify his preferences on how to achieve the goal of his personalization intention, such as convenience and preserving security. To model such concepts, the authors extended the EUPont [41] model, a semantic, ontology-based representation of IoT devices.

Figure 4.8 shows the HeyTAP architecture.

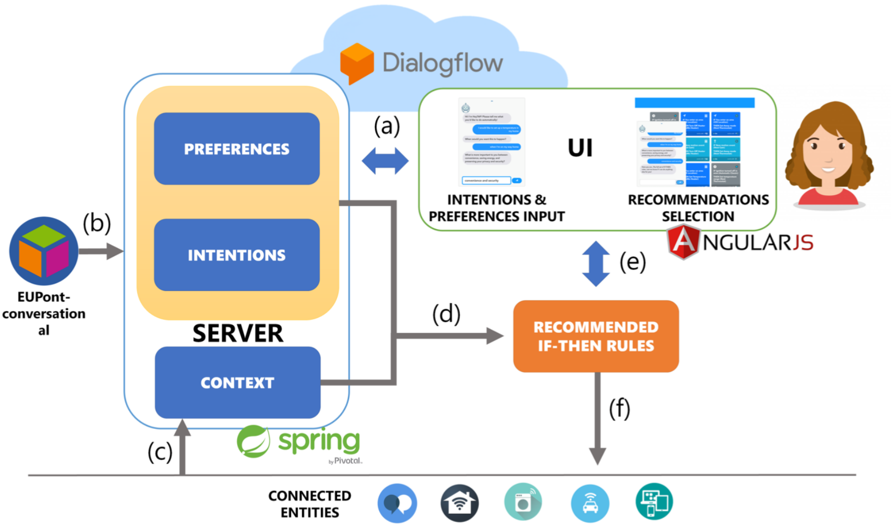


Figure 4.8: HeyTAP architecture [43].

The EUPont ontology was leveraged by HeyTAP to classify the triggers and actions offered by the user's connected entities in terms of the functionality provided and to model contextual information, e.g., user-owned devices and services and their location. In addition, classes and restrictions are added to automatically characterize triggers and actions based on user preferences, for example, to discriminate between behaviors that require energy and privacy-invasive behaviors. All this semantic information is used to suggest a set of IF-THEN rules that meet the user's needs, i.e., intentions and preferences. The user can eventually inspect the suggested rules in the multimodal interface and select one or more of them to personalize his or her connected entities.

HeyTAP helps users move from their abstract needs to IF-THEN rules involving real smart devices and online services and consists of two main phases, namely conversation and recommendation.

The user interacts with a conversational agent (a) to tell the system his personalization intentions for different contexts, along with his preferences, with different levels of abstraction. Leveraging a semantic model, i.e., EUPont-conversational (b), the server analyzes the user's input, along with contextual information about the available connected entities (c), to infer a set of IF-THEN rules that meets the user's needs (d). The recommended rules are displayed in the multimodal user interface to the user, who can decide (e) which rules should be instantiated and executed on real smart devices and online services (f).

The HeyTAP server uses user input, i.e., the user's intentions and preferences, together with contextual information about available connected entities (c), to infer a set of IF-THEN rules that include available and real connected entities, e.g., with triggers and IFTTT actions (d). The server analyzing the user action and trigger intentions extracts a set of appropriate IFTTT actions and triggers, respectively. In this step, it first extracts all actions and then filters them according to the available intention elements, i.e., functionality, category, entity, technology, where, and when. The same steps are then used to extract a set of triggers, which are combined with the retrieved actions to generate an initial set of IF-THEN rules. That set of IF-THEN rules is finally filtered by considering the user's preferences.

On the one hand, modern trigger-action programming platforms exploit highly technology-dependent representation models, thus making end-user customization of connected entities a complex task. On the other hand, using a higher level of abstraction requires an effective way to select the actual entities, triggers and actions with which to satisfy the user's abstract needs.

The results of an exploratory study of eight end-users confirm the effectiveness of the approach and show that HeyTAP can "successfully translate" users' abstract needs into IF-THEN rules that can be instantiated and executed by concurrent trigger-action programming platforms.

However, this study targeted only a limited number of users with a computer science background. A more valuable study would be to deploy HeyTAP, testing it with different types of users.

4.3.1.3 *InstructablesCrowd: creating IF-THEN rules for smartphones via conversations*

In [82] authors, they built a system called "InstructableCrowd", which allows users to schedule their own devices via conversation.

The user verbally expresses a problem to the system and a group of crowd workers collectively respond to schedule and return IF-THEN rules requested by the user.

The IF-THEN rules generated by Instructable Crowd link combinations of relevant sensors (e.g., location, weather, device acceleration, etc.) to useful effectors (e.g., text messages, device alarms, etc.).

InstructableCrowd is implemented as a conversational agent for android smartphones. The user is able to issue commands to the agent via voice or text as shown in Figure 4.9). The client side records the user's speech and sends it to the server, which in turn sends this speech to Google Automatic Speech Recognition; the user can also use text input to enter the command. InstructableCrowd adopts the *LIA* [11] framework, which uses a combinatorial categorical grammar (CCG) parser to parse the input text into a logical form and execute the corresponding commands to recognize the user's speech input. Once the user

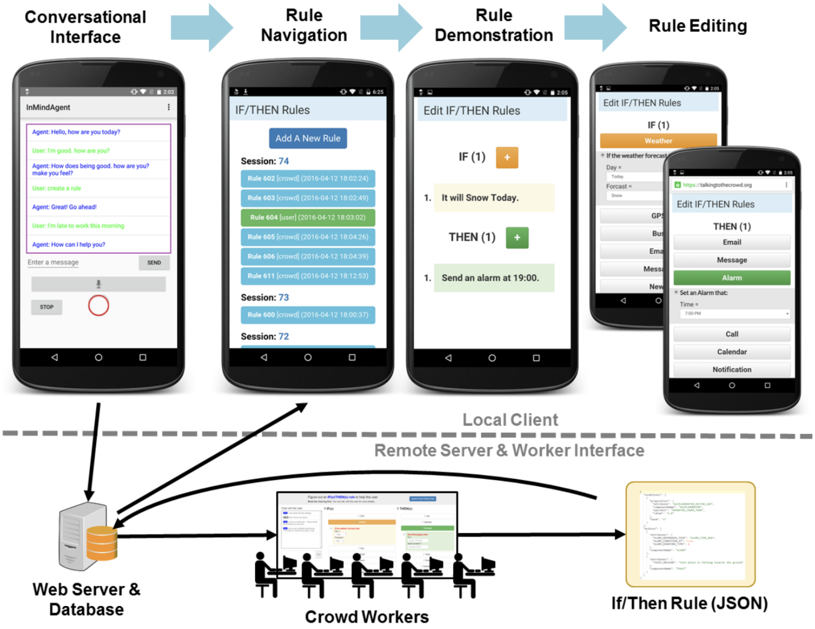

Conversational Interface
Rule Navigation
Rule Demonstration
Rule Editing
InMindAgent
Agent: Hello, how are you today?
User: I'm good, how are you?
Agent: How does being good. how are you? make you feel?
User: create a rule
Agent: Great! Go ahead!
User: I'm late to work this morning
Agent: How can I help you?
Enter a message
SEND
STOP
IF/THEN Rules
Add A New Rule
Session: 74
Rule 602 [crowd] (2016-04-12 18:02:24)
Rule 603 [crowd] (2016-04-12 18:02:49)
Rule 604 [user] (2016-04-12 18:03:02)
Rule 605 [crowd] (2016-04-12 18:04:26)
Rule 606 [crowd] (2016-04-12 18:04:39)
Rule 611 [crowd] (2016-04-12 18:12:53)
Session: 73
Rule 600 [crowd] (2016-04-12 18:00:37)
Session: 72
Edit IF/THEN Rules
IF (1)
1. It will Snow Today.
THEN (1)
1. Send an alarm at 19:00.
Edit IF/THEN Rules
IF (1)
Weather
THEN (1)
Email
Message
Alarm
Set an Alarm that:
Time =
7:00 PM
Call
Calendar
Notification
Local Client
Remote Server & Worker Interface
Web Server & Database
Crowd Workers
If/Then Rule (JSON)

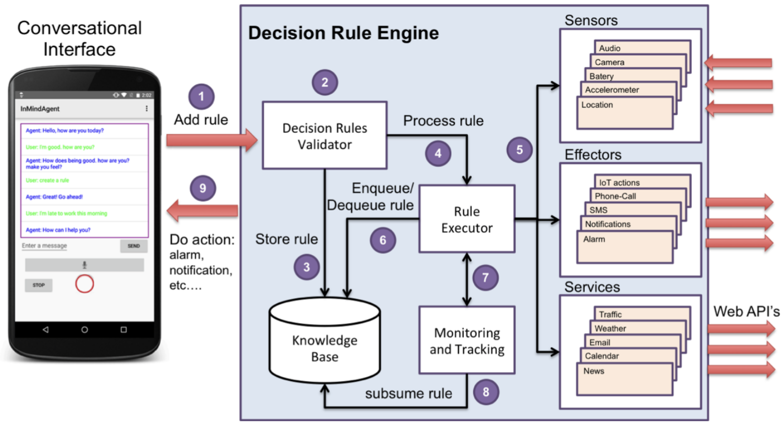
Figure 4.9: Functioning of InstructablesCrowd [82].

gives verbal commands such as "create a rule", *LIA* connects to InstructableCrowd and begins the rule creation process.

InstructableCrowd also provides an editing interface to allow the user to manually create new rules and edit them. All rules are grouped by conversation session in which the rule was created. Crowd generated rules are blue and user created or modified rules are green. The Decision Rule Engine is responsible for validating, storing, processing, and executing rules created by a crowd-worker or user. The Decision Rule Engine is composed of multiple modules that interact with each other to perform an action given a set of specific conditions that are true. These modules are interconnected as shown in Figure 4.10.

The workflow and how the components of the Decision Rule Engine cooperate over time to handle rules created by the user or crowd-workers can be described in the following six steps:

- **Decision Rule Validator:** After the user or crowd worker defines a new rule to be added (Step 1), this component validates the syntax of that rule based on the attributes and constraints of the sensors and effectors (Step 2);



InMindAgent

Agent: Hello, how are you today?

User: I'm good. how are you?

Agent: How does being good. how are you? make you feel?

User: create a rule

Agent: Great! Go ahead!

User: I'm late to work this morning

Agent: How can I help you?

Figure 4.10: Architecture of InstructablesCrowd [82].

- **Knowledge Base:** once the rule has been parsed and validated, it is stored in a knowledge base that can be accessed at any time by any component (Step 3). These rules are stored locally for performance and privacy reasons, so potentially confidential information contained in the rule is protected;

- **Rule executor:** after validation, the rule is immediately processed to determine whether it should be executed at that time (Step 4). If so, it invokes actions from the appropriate effectors (Step 5). Otherwise, it adds the rule to a queue so that it can be executed later when all its conditions are met. The rule executor periodically checks to see if each queued rule should be executed (Step 6);

- **Monitoring and tracking:** this module is responsible for monitoring the rule execution process (Step 7) and checks to see if there are any rules that are never triggered or conflict with each other (e.g., one rule intends to turn GPS on while the other intends to turn it off). When conflicts occur, the monitoring/tracking module temporarily subsumes the least relevant rule (i.e., the one that has been activated least frequently) and then the user is asked to confirm this subsumption decision (Step 8);

- **Integrated and external sensors/effectors:** in addition to the built-in sensors and effectors that are part of the operating system, such as GPS and SMS messages, some virtual sensors/effectors rely on external services, such as weather forecasts and news feeds. Finally, the user is always aware of the execution of the action through notifications, text messages, alarms, etc. (Phase 9).

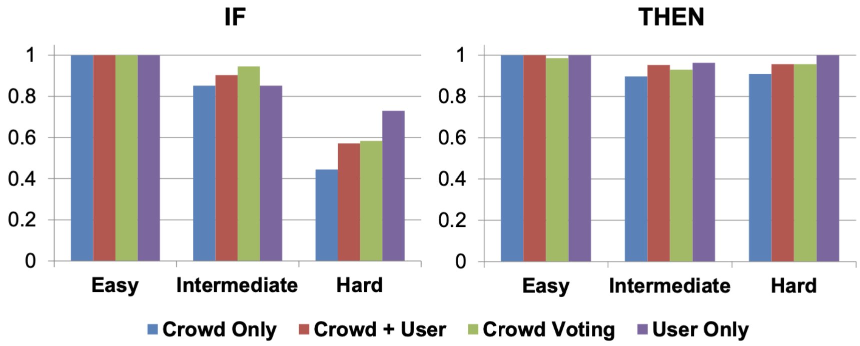


Figure 4.11: InstructablesCrowd Scoring [82].

While the THEN parts were not affected much, performance in the IF parts decreased as the scenarios became more complex. The "Crowd Voting" performed similarly or slightly better than "User Only" in the easy and intermediate rules, but worse in the more complex rules. These results indicate that the large number of sensors and effectors influences the level of difficulty in composing the rule.

This study allows us to highlight the increasing complexity of IFTTT applet components due to the large number of triggers, actions, and channels that increase over time and how much human intervention is required for consistency analysis of these components.

4.3.1.4 *Interactive semantic analysis for IF-THEN recipes by hierarchical reinforcement learning*

In most existing semantic parsers, text parsing and comprehension are performed in a single step.

With this study [164] authors propose a system that can translate the description of an IFTTT rule into an applet by exploiting hierarchical reinforcement learning. Semantic analysis becomes interactive; the agent can ask clarifying questions to resolve ambiguities through a multi-turn dialogue.

The agent interactively predicts the 4 components of an IF-THEN recipe, deciding to ask a question when the prediction probability of a recipe component is below a predefined threshold. An optimization framework also simultaneously improves the accuracy of analysis and reduce the number of questions.

This work is done through hierarchical reinforcement learning (HRL). The whole semantic analysis is decomposed into 4 subtasks or options namely, trigger prediction, trigger channel, action and action channel.

Four low-level policies are used one for each task. These policies are used to give rewards to the agent which are positive in case it makes correct predictions and negative if it makes incorrect predictions.

The rewards also stimulate the agent to predict a correct component with fewer questions. The agent chooses the order of subtasks to perform and moves to the next subtask only when the current one has been completed.

To move from one state to the next, the agent must perform an action, which may consist of predicting an IFTTT component or asking the user a clarification question. During training, the analysis is not interrupted even if one of the predictions is wrong to encourage the agent to predict as many correct components as possible.

In this experiment, the authors used different reinforcement learning algorithms to train the agent to recognize trigger, action, channel and function of an IFTTT rule, testing their work on different datasets and with different policy functions. With this study, the authors demonstrated that the use of reinforcement learning greatly improves, compared to other work done in the literature, the prediction of the components of an IFTTT rule.

The improvement is achieved more on applets that possess very vague and unclear descriptions because the agent by asking questions of the user is able to gain clarity on the individual components to be predicted. The model, in fact, on this type of recipe achieves an accuracy of 87%.

With the different policy features implemented by the authors, the agent asks fewer questions of the user and still achieves very good accuracy.

This work is not robust with respect to sentences that contain typos in the descriptions and does not consider any typos made by the user.

4.3.1.5 *Synthesizing If-Then programs through the mechanism of latent attention*

A fundamental problem for computational linguistics is translating natural language descriptions into executable programs. Over the past decade, there has been an increasing number of attempts to address this problem from both the natural language processing community and the programming language community [90]. In [97], researchers have focused on the description of IFTTT rules to try to predict from the latter the trigger and action of the rule.

To translate a natural language description into a program, they identified the words in the description that are the most relevant to predict the desired labels: trigger, action, and channel. For example, in the following description:

"Automatically save Instagram photos to your Dropbox folder"

The text "photo of Instagram" is the most relevant to predict the trigger. To acquire this information, they adapted the Latent Attention mechanism: they first calculate the importance weight of each token in the sentence and then produce a weighted sum of the embeddings of these tokens.

The weight of each token depends not only on the token itself, but also on the overall structure of the sentence.

The Latent Attention, or latent attention, mechanism was used to exploit these clues and calculate the latent weight for each token to determine which of them are most relevant in the sequence for the trigger or action.

The architecture of the Latent Attention model, implemented by the researchers, is presented in Figure 4.12. The model takes as input a sequence of symbols $x_1, ..., x_J$, each coming from a dictionary of N words where J is the maximum length of a

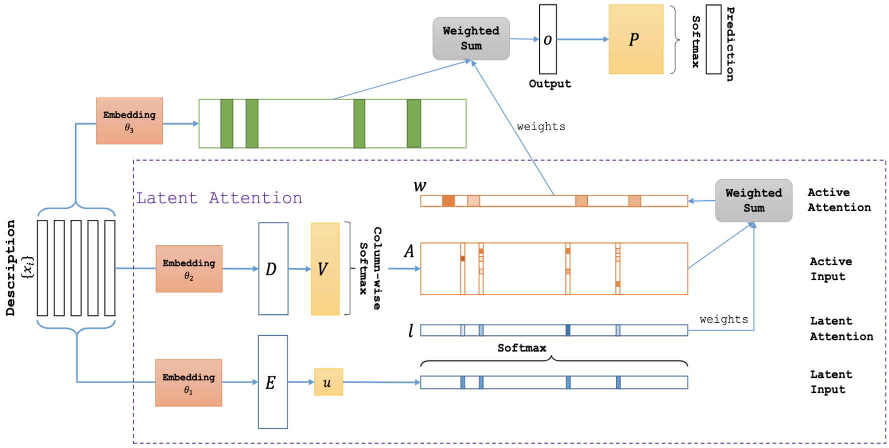


Figure 4.12: Architecture of the latent attention model [97].

description. The architecture has several components, detailed below.

Vocabulary. Each sentence is tokenized by splitting it on spaces and punctuation (e.g. “.:;!? "o:;”) and converted to lowercase characters. All punctuation symbols are still retained as tokens. In this mapping, typos in the text are not considered. The 4,000 most frequent tokens remain unchanged while all the rest are placed in a single token that is subsequently compressed. Another approach used is to take word embeddings, run it through a bidirectional LSTM using concatenation of two LSTM outputs at each time step as embeddings. This can take into account the context around a token by having the embeddings contain more information about the entire sequence as well as the individual token. Both approaches were used in the training phase.

Latent attention layer. Each symbol x_i is encoded as a $one-hot$ vector of N size. The input sequence X is embedded in a $d-dimensional$ embedding sequence using $E = Embed_{\theta_1}(X)$, where θ_1 is a set of parameters.

The output of the latent attention layer is computed as a standard softmax function on E in the following way:

$$l = softmax(uTEmbed_{\theta_1}(X)).$$

Where, l is the $J-dimensional$ output vector and u is a $d-dimensional$ trainable vector. The softmax function, or normalized exponential function, is a generalization of a logistic function that compresses an $k - dimensional$ z vector of arbitrary real values into an $k - dimensional$ $\sigma(z)$ vector of values in the interval $(0, 1)$ whose sum is 1. The function is calculated as follows:

$$\sigma(z) = \frac{e^z_j}{\sum k = 1^K} e^{z_k} \qquad per \;\; j = 1, .., K$$

Attention layer. The attention layer or active attention layer calculates the weight of each token based on its importance to the final prediction. These weights are called active weights. First X is embedded in D using another set of parameters 2, i.e., $D = Embed_{\theta_2(X)}$ is of size $d * J$. Next, for each token D_i, its active attention input A_i is computed through a softmax:

$$A_i = softmax(V \;\; D_i)$$

Here, A_i and D_i denote the i-th column vector of A and D respectively, and V is a trainable parameter matrix of dimension $J * d$. Note that $V \;\; D_i = (VD)_i$, is computed by performing a softmax per column on $(V \;\; D)$. Here, A is of dimension $J * J$. The active weights are calculated as the weighted sum of A_i, using the following formula:

$$w = \sum_{i=1}^{J} = l_i A_i$$

Output representation. A set of parameters is used to embed X into the inclusion matrix J, and the final output o, a $d - dimensional$ vector, is the weighted sum of embeddings obtained by the following formula:

$$o = Embed_{\theta_3}(X)w$$

Prevision The softmax function is used to make the final prediction: $\widehat{f} = softmax(P_o)$, where P is a matrix of parameters $d * M$ and M is the number of classes.

This complex model made by the researchers was trained with a very large number of manually validated and cleaned-up IFTTTs.

The realized architecture was used to build the following six different training models:

- **Dict:** model that uses the architecture described above with the latent attention and attention layers turned off and performs tokenization of input data by exploiting the constructed vocabulary;
- **Dict +A:** model that uses the architecture described above with only the attention layers active and performs input data tokenization by exploiting the constructed vocabulary;
- **Dict +LA:** model that uses the above architecture with the attention and latent attention layers active and performs input data tokenization by exploiting the constructed vocabulary;
- **BDLSTM:** model that uses the architecture described above with the latent attention and attention layers turned off and performs tokenization of input data with two-way recurrent networks;
- **BDLSTM+A**: model that uses the architecture described above with only the attention layers active and performs tokenization of input data with bidirectional recurrent networks;
- **BDLSTM+LA:** model that uses the architecture described above with the attention and latent attention layers active and performs input data tokenization with bidirectional recurrent networks.

In the various studies carried out, the researchers have performed several trainings by also testing an ensemble of the above models from which the following considerations emerged:

- latent attention consistently improves performance compared to standard attention architectures and no attention layer using both embedding methods;
- among the six architectures evaluated performance improves significantly with single model;
- when the ensemble of models is considered the tokenization of input data done with bidirectional recurrent networks

(BDLSTMs) performs better than that done using vocabularies, since BDLSTMs are able to encode the information of tokens and their surroundings.

Figure 4.13 illustrates the accuracy obtained by training the various models in channel prediction. Note that the best performance was obtained with the model **BDLSTM+LA** reaching 91% accuracy in predicting the channel of an IFTTT rule.

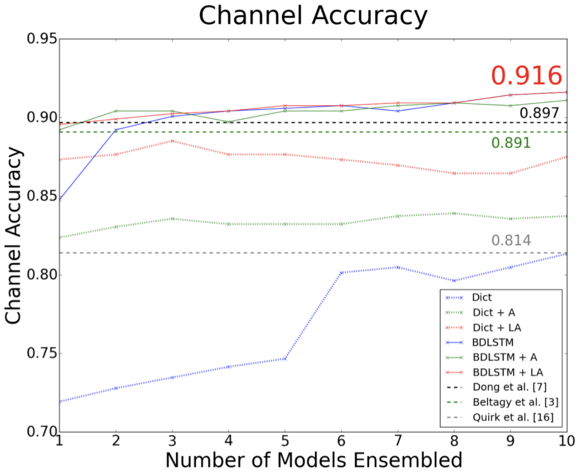


Figure 4.13: Latent Attention model: accuracy evaluated on channels [97].

4.3.2 *Methodology*

In this section, we discuss the proposed solution for evaluating the semantic consistency of an IFTTT applet. In particular, we evaluated the consistency of the description through a semantic approach based on Google's BERT. The description is defined as consistent if it agrees with the description of the IFTTT applets synthesized by considering the other components, i.e., Trigger, Trigger channel, Action, Action Channel.

4.3.2.1 *Pre-processing*

The rules considered for training the proposed approach are derived from the dataset of Mi et al. [109]. At a preliminary stage, the complete dataset went through a preprocessing phase during which the following steps were performed:

- **filtering of applets with English descriptions**: most of the descriptions in the dataset are in English. Therefore, this study focused only on descriptions written in English since it is not good practice to train a single classifier with data from different languages;
- **deletion of non-text characters**: any numeric or punctuation characters have been deleted in order to leave only text characters;
- **deletion of stop-words**: for each description, words that are used very frequently, which do not possess semantic meaning such as "of, are, the, it, is" were deleted.
- **normalization of text**: all words have been normalized to lowercase;

The resulting dataset then went first through a feature selection and construction phase, followed by with a labeling phase, leading the generation of training, validation and test sets we employed in our approach.

SYNTHESIZING A DESCRIPTION FROM THE ELEMENTS OF AN APPLET Since our ultimate goal is to evaluate the semantic consistency between the description of the applet provided by the creator and its actual behavior, defined on the basis of the choice of trigger and action components, we decided to train the model to classify the semantic coherence of an applet by going to compare the natural language description with one artificially synthesized from the applet's components. In fact, by taking into account the four "static" components of a rule, i.e., trigger, trigger channel, action, action channel, it is possible to generate a natural language description of how it works.

For example, considering an IFTTT applet having the following components:

- **triggerTitle**: "Any new SMS received"
- **triggerChannelTitle**: "Android SMS"

- **actionTitle**: “Send me an email”
- **actionChannelTitle**: “Email”

we can synthesize a description by exploiting the following pattern.

IF triggerTitle (triggerChannelTitle) THEN actionTitle (actionChannelTitle)

The synthesized description for such an applet thus becomes:

IF Any new SMS received (Android SMS) THEN Send me an email (Email)

DATASET LABELING From the applets resulting from the cleaning phase, we selected 11000 and proceeded to label them over a 3-week period. We built each set by considering the following features:

- **Author’s description**: the natural language description drafted by the creator of the applet at the time of its creation.
- **Synthesized IFTTT description**: The IFTTT description generated according to the approach described above.
- **Label**: Label indicating whether the description is consistent, inconsistent, or partially consistent. In particular, the label can have one of the following values:
  - “entailment”: Whether the description drafted by the developer is consistent with that summarized by the applet;
  - “contradiction”: Whether the description drafted by the developer is inconsistent with that summarized by the applet.

Finally, the labeled dataset was randomly divided into training, validation, and test sets with the distribution of applets shown in Table 4.13.

	# entailment applets	# contradiction applets
Training set	3000	3000
Validation set	450	400
Test set	1500	1000

Table 4.13: Subdivision of applets in the labeled dataset

4.3.2.2 *The BERT model*

The model trained for classifying the semantic consistency of an IFTTT applet is presented in Figure 4.14.

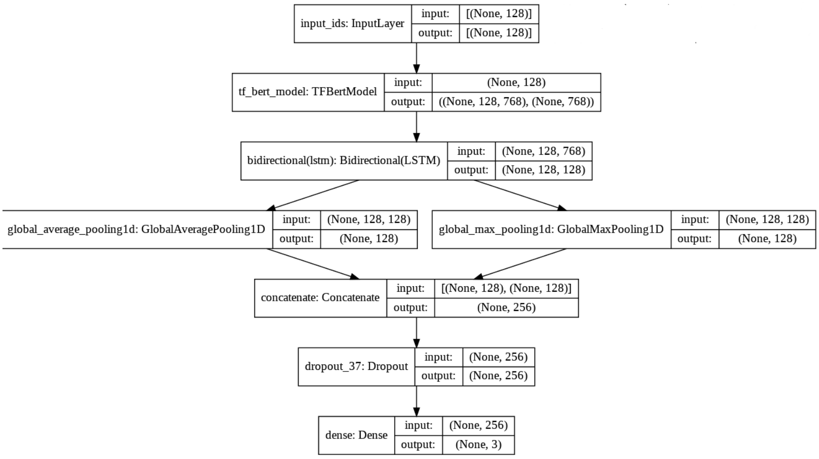


Figure 4.14: Trained BERT model

The dataset applets are fed to BERT's pre-trained model, which converts them into densely compressed vectors. They are called dense vectors because each element of the vector has a specific numerical value and the vector produced is a vector of dense type.

Each layer of the pre-trained BERT network outputs a series of dense vectors, thus capturing different levels of syntactic and semantic information. The result obtained from the BERT model is passed to an LSTM layer which stores the relationships present in the sequence passed as input. The output of this stage is scaled down through two layers of MaxPooling and concatenated into a single vector. Finally, the DropOut operation is performed before passing the data to the densely connected layers of the network, to prevent the system from going into overfitting. The network thus constructed was trained to teach the model to distinguish coherent from incoherent or neutral descriptions.

4.3.3 *Experimental evaluation*

The BERT model trained on the previously built training set was then evaluated on the test set by measuring its performance on the basis of precision, recall and F1-score metrics. Results are shown in Table 4.14

As we can see, the classification results are particularly high especially regarding the results of the "entailment" class. In any case, in general the global accuracy of the model is still sufficiently high, having returned an accuracy of 82% on the test set.

An example classification of the trained BERT model is shown below.

Given the following IFTTT applet:

- **triggerTitle**: "Any new post by you",
- **triggerChannelTitle**: "Instagram",
- **actionTitle**: "created a photo post",
- **actionChannelTitle**: "blogger"

Starting from the above applet, the following description is synthesized:

If any new post by you on (instagram) then create a photo post on (blogger).

The description written by the creator of the applet is as follows:

Use this Applet to automatically post your Instagram uploaded photos to a Blogger pager.

The description written by the creator of the IFTTT and the one synthesized starting from the applet's components are given as input to the BERT model which returns the following prediction as output:

	Precision	Recall	F1-Score
Entailment	0.97	0.98	0.89
Contradiction	0.78	0.75	0.82

Table 4.14: Classification performance on the test set

('entailment', ' 0.92%')

The classifier is able to recognize that the description associated with the IFTTT applet is 92% consistent with its actual behavior, making a correct prediction.

Although the two descriptions are formulated in different ways, the model is able to fully learn the semantic content of the description and carry out a correct classification.

An example of a case in which the creator has written an inconsistent description is shown below.

Given the following description of IFTTT:

Save your all twitter photo in your blog.

https://meet.google.com/uxt-vryu-bby?authuser=2 Giving input the summary description of the rule and the inconsistent description written by the developer, BERT makes the following prediction:

('contradiction', ' 0.67%')

Also in this case the model made an excellent prediction with a fairly high probability.

Part II

APPROACHES FOR SECURING IOT ECOSYSTEMS

5 EMPOWERING THE DEFINITION OF SECURITY-EVENTS TRIGGERED RULES

One aspect often neglected by current TASs, which is instead typical of IoT, is that “connected” smart objects are vulnerable in terms of security. These devices are an attractive target for external malicious attackers since they provide entry points to user’s online services and physical devices [115, 138]. Also, data can be arbitrarily manipulated by attackers and cause damage. This problem is amplified when end-users, not fully aware of these risks, put in communication devices by using TASs. In many situations, end users do not have sufficient skills in security and privacy, in particular when they deal with IoT devices [83]. In addition, they under-evaluate the importance of these aspects in defending their smart environments, thus they neglect countermeasures that might protect the security of their smart devices [3, 88].

In this chapter, we try to address this problem and present a visual paradigm that supports users, even those who do not possess technical skills in IoT and security, in securing their smart environments. The definition, implementation, and evaluation of the following proposal have been performed in conjunction with researchers from the University of Bari "Aldo Moro" and the Politecnico di Milano with different expertise in the area of Human-Computer Interaction with specific accents to usable security and privacy in IoT.

5.1 LITERATURE REVIEW

According to [168], IoT environments should improve users’ technology threat models, which enable the conscious management of privacy and security risks, and communicate best practices suitable for the smart spaces. Transferred into the EUD context, this recommendation implies that EUD environments should support users during the creation of rules, guiding possible privacy or security violations and ad-hoc elements in the rule definition language that can facilitate the definition and deployment of rules addressing these issues. Along this line of action, [138] defined security lattices to understand how the flow of statements within IFTTT recipes could lead to the generation of potential security risks. This is an interesting work, which suggests ways to extend TASs with debugging capabilities, but that, however, does not sufficiently empower the end users to configure the security of their smart environments. This is instead the goal of our work, which focuses on proper paradigms for the end users to become aware of possible security threats and be enabled to control them.

The EUD environment should also support end users in the management of access control and authentication to the smart objects. In this context, [78] suggested using a capability-centric model to fit the access control and authentication mechanisms to the variety of smart objects and features they can perform. For example, rather than allowing the users to have complete access to a smart object, a more adequate solution would be to configure the access schemes only on certain capabilities, such as turning on/off the device, update the software, and so on. This would also reduce the complexity of system configurations addressing security and privacy problems. In line with this recommendation, our work concentrated on identifying categories of attacks and related countermeasures that could make sense for the end users who have not sufficient knowledge in security aspects.

5.2 INTRUSION DEFENDER: A SMART OBJECT TO SECURE SMART ENVIRONMENTS

In order to extend TASs with security management capabilities, we built a smart object, called Intrusion Defender (ID), that monitors the network traffic of a private area network (PAN) to detect anomalous events. The events occurring on a PAN might originate from cyberattacks that a malicious individual could launch against a smart environment, e.g., a smart home, with the intent of stealing sensitive data acquired by smart objects or deactivating intrusion detection systems controlling the smart environment.

ID has been developed on top of Snort, a software monitoring and identifying network threats. It is an Open-Source Network Intrusion Detection System that monitors the packets of the network traffic, which travel to and from all smart objects in the smart environment. A network packet can carry data related to an exchange of information that takes place between a smart device and the Internet. Some features of this network traffic, e.g., the number of packets sent in a given time frame or the type of packets received and sent, provide important clues for identifying cyberattacks. By monitoring these features, which are called signatures, Snort can identify the intrusions by performing a fast comparison with the signatures contained within a database of pairs (attack, signature). Once an attack is identified, Snort generates an event reporting the main information about the attack. All the generated events are stored into a database.

To effectively defend a smart environment from external cyberattacks, Snort has to be configured to monitor the network locally, i.e., within the PAN. Indeed, it is not possible to remotely monitor the exchange of packets between smart devices of a PAN from outside the network. The individual smart devices communicate with the outside through the router, which masks the information generated by the smart objects. Thus, ID has been implemented as a smart object that acts as an intermediary in the communication between the smart devices and the router, by intercepting and analyzing the network traffic for possible network intrusions. This solution is also the safest from the data management point of view. In fact, a different solution would allow the smart devices to share their information externally so that the ID could retrieve this information remotely. However, this type of solution would expose the smart environment to an even higher risk since the communication could be intercepted by an external malicious attacker. Performing the network analysis locally avoids sharing more information than the one required by the smart devices to behave as they are designed to.

The ID has been installed on a Raspberry Pi, a particularly inexpensive single-board computer, small in size and low in energy consumption [26]. On the Raspberry Pi, therefore, the Snort software has been installed, the launch of which depends on the execution of some scripts, which contain the commands and pa-

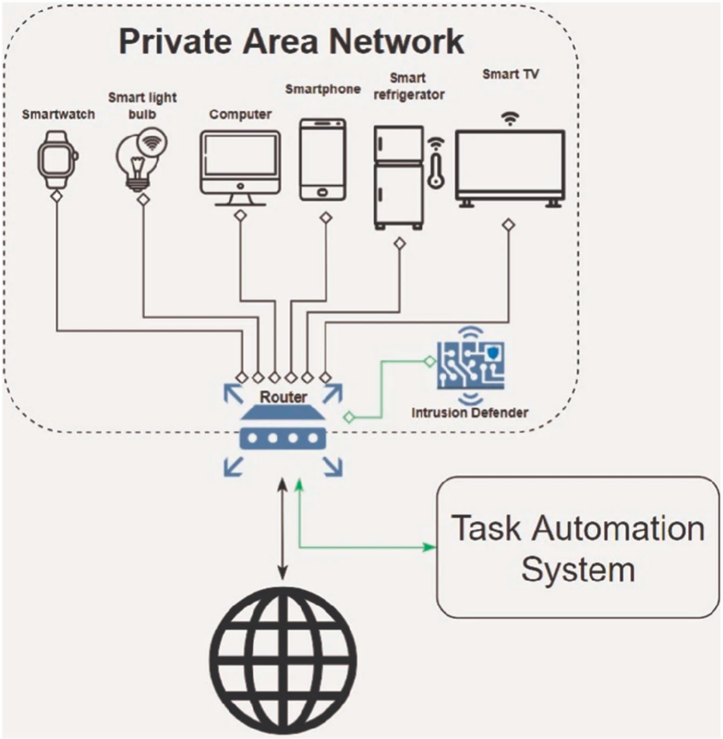


Figure 5.1: Architecture of the secured smart home environment

rameters necessary for Snort to know the type of monitoring that must be carried out. The board also contains databases storing both the signature repository that Snort uses for analysis and the archive of all the events associated with cyberattacks that Snort can detect. The database is then made available to the other components of the smart environment.

Fig. 5.1 summarizes the architecture of the smart environment after the installation of the Intrusion Defender. The previously established connections between the various smart objects and the router are preserved from the introduction of the ID within the PAN. Indeed, the ID behaves like any smart object connected to the smart home environment. Once connected, it analyzes the network packets by interacting with the router; when a cyberattack is detected, it sends the information about the identified event directly to the TAS.

Besides managing the detection of attacks at runtime, the TAS also provides the environment for configuring the rules defining countermeasures for security attacks. In our research, we adopted the EFESTO-5W platform, which offers a visual paradigm for ECA rule definition that has proved to be effective for different application domains [6–8, 13, 50]. In the following sections, we will illustrate the design process that eventually led to extending EFESTO-5W to support the definition of ECA rules covering ID events.

5.3 ABSTRACTING SECURITY-RELATED EVENTS

The configuration of the ID device requires IT and cybersecurity expertise. For example, it reports the detected cyberattacks in a log file or in a dashboard, in both cases targeted to experts. However, countermeasures to defend the PAN devices must be taken by the user, who has to be aware of the meaning and level of risk of cyberattacks. Also, countermeasures like switching off the attacked devices must be configured by developing, for example, scripts in programming languages such as JavaScript or Python. Therefore, users with no expertise in IT and cybersecurity, who are the majority of the actual users of smart environments, would be excluded from the effective use of the ID, or in general, from the use of similar technical devices for cyber defense.

To solve these problems, this article proposes a solution to configure the behavior of the ID by using Task Automation Systems, and in particular EFESTO-5W. The final goal is to make possible the creation of Event-Condition-Action (ECA) rules that can be triggered when an attack is detected by the ID, for example, by defining a rule like "IF the ID detects the attack X then switch off the attacked device". This research goal poses two main challenges. First, the ID is able to detect several attacks (35 in the current implementation), and this high number can overload the users with too much information. Second, the detected attacks refer to cybersecurity concepts (e.g., DDoS, man in the middle, etc.), which are too technical and complex for lay users.

To address the first challenge, a card sorting session was carried out with 11 IT and cybersecurity experts, to reduce the number of ID-detected attacks to be exposed to the users. All the possible attacks have been grouped based on their meaning and consequences on the attacked devices. For the identified groups, the event descriptions that the users can select when creating ECA rules have been designed by adopting the Communication-Human Information Processing (C-HIP) model [161]. C-HIP frames the most important activities and entities involved in the communication of a warning, and sets the foundation for structuring warning messages. Lastly, to make the description text clearer, we iteratively refined the descriptions of the events by using metrics for the evaluation of text comprehension, readability and sentiment. In the following, these activities are illustrated in detail.

5.3.1 *Card-sorting study to reduce information overload*

The definition of ECA rules by using TASs is the ground of the proposed solution. An example of ECA rules in smart homes might be "IF the motion sensor detects a person THEN switch on the smart lamp". Regardless of the visual metaphor implemented by a TAS, the definition of an ECA rule like this does not require technical IT knowledge but only knowing the smart objects' functionalities. However, when dealing with devices like the ID, the creation of an ECA rule could be more complex. Indeed, the events detected are exposed in the technical language, thus an ECA rule would be, for example, something like "IF the ID detects an unusual client port connection on the device X THEN switch off device X". The number of all the possible events can be high and sometimes the differences between different events are related to technical nuances which might not be meaningful for non-technical users. In other words, a simplification of the ID events is required, in relation to both their multiplicity and technical description. For this purpose, as a first step, we conducted a card sorting session to identify groups of events to be represented by a unique name. Card sorting is a low-tech approach used to generate a dendrogram (category tree) or folksonomy [137]. It is widely used during the design of interactive systems to optimize the information architecture, menu item organization, or navigation paths. HCI expert identifies key concepts and reports them on cards (e.g., Post-it notes). A group of users, individually, are required to arrange the cards according to their preferences and follow the study goal (e.g., structuring menu items). Typically, a number of 10/15 users is sufficient to obtain reliable results [117].

We recruited 11 IT and cybersecurity experts as study participants; two HCI experts then managed the study. Given the COVID-19 pandemic, the study was carried out remotely by using the kardSort platform [12]. Since the original ID events are often too short and meaningless, all of them were converted into a clearer description, so that participants can better understand their meaning and consequences of the detected attacks. For example, the original event "malware-cnc" was replaced with the description "Detected infected device sending system information to other infected devices".

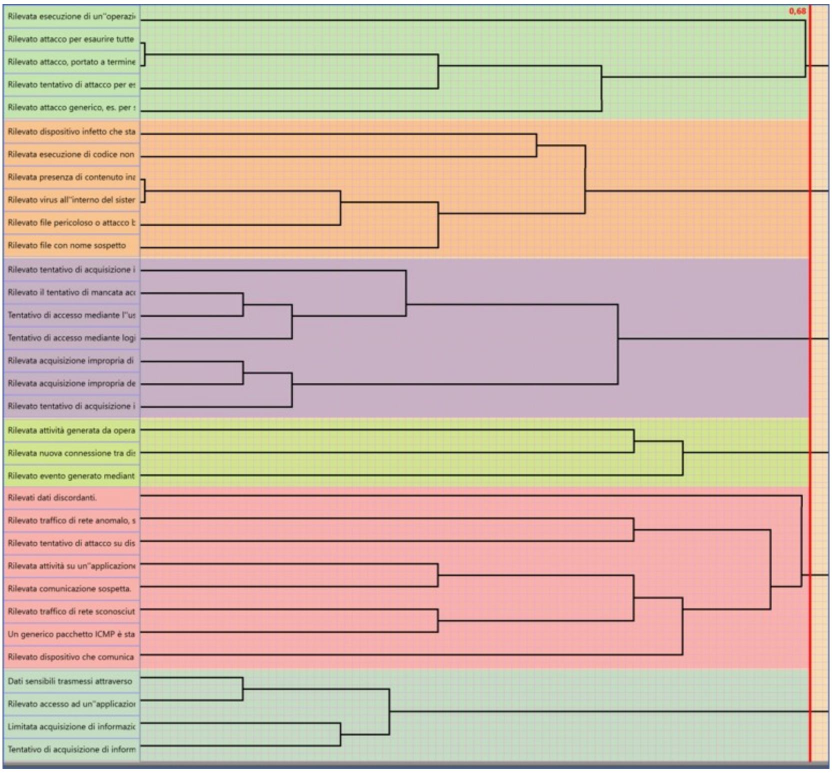


Figure 5.2: Process of grouping security events

The final 35 descriptions were registered as cards in the kardSort platform and the study was set up as "open", meaning that no predefined categories were defined but the users were completely free to arrange the cards. In addition, an introductory text was added in kardSort to instruct participants they had to create groups containing cards reporting similar attacks (e.g., DoS and DDoS) and/or having the same consequences (e.g., that tries to collapse a device). The study lasted 2 days and each participant spent around 20 minutes.

The results have been analyzed by using Casolysis [139]. This tool implements two methods for the analysis of open tests, namely Hierarchical Clustering (Single Link, Average Link, Complete Link) and Section Label Analysis (SLA). For our analysis, we used the Average Link since it typically performs better than other solutions [135]. Finally, a threshold for the similarity of the clustering algorithm was set to 0.68/1 to obtain a low number of categories that include attacks that can reasonably stay in the same group. Figure 5.2 depicts the process of grouping the original 35 events in the final 6 groups which are summarized in Table 5.1. The final groups represent the basis for the design of the final messages, as reported in the next section.

	Class	**Labels**
1	DOS	system-call-detect; denial-of-service; dataset successful-dos; attempted-dos; misc-attack

2	Malware	malware-cnc; shellcode-detect; inappropriate-content;
		trojan-activity; file-format; suspicious-filename-detect;

3	Privilege Escalation	attempted-user; unsuccessful-user; default-login-attempt; suspicious-login; successful-admin; successful-user; attempted-admin
4	Data Exfiltration	sdf; web-application-attack; successful-recon-limited; successful-recon-largescale; attempted-recon
5	Suspicious Connection	misc-activity; tcp-connection; non-standard-protocol

6	Suspicious Traffic	protocol-command-decode; bad-unknown; client-side-exploit;
		web-application-activity; string-detect; Unknown; network-scan;
		icmp-event; unusual-client-port-connection

Table 5.1: List of the final 6 groups and their original labels

5.3.2 *Designing event description for the ID*

Reducing the 35 ID events in 6 groups solved the problem of the information overload that users might have if using the original ID events. However, this phase did not solve the problem of the technical skills required to understand the ID events. Therefore, we designed, for each group, a short title, and an event description that explains the attack and its consequences. Since the ID event can be seen as a warning event that users adopt in the ECA rule definition, as the ground of this design phase we used the Communication-Human Information Processing (C-HIP) model, which defines the critical route and sets the foundation for structuring warning messages [161].

The C-HIP model summarizes the most important activities and entities involved in the communication of a warning. The model starts with a source delivering a warning through a channel to a receiver, who then takes it along with other stimuli (environmental or internal) that subject the message to a lot of distractions or distortions. It then identifies a set of steps between the delivery of a warning and the user's final behavior or response, which is usually based on the effect of the various processes such warnings had undergone. In [46] the authors also define a set of design guidelines related to the C-HIP model and present rules for descriptive text: i) Briefly describe the risk and consequences of not complying with advice; ii) Illustrate clearly how to avoid the risk; iii) Be transparent and avoid technical jargon where possible; iv) Be as brief as possible.

Based on prescriptions and guidelines of the C-HIP model, we designed the descriptions for the 6 ID events, generating them following a template purposely defined:

Title + *Hazard Identification* + *Effects of a successful attack*

The generation of these messages was carried out iteratively, considering metrics that perform static evaluations of the readability and sentiment of the event messages. Indeed, conveying the hazard messages to all users is not simple, as reported in [59, 79]. Readability metrics measure the degree to which a person can read, easily understand, and find interesting that text [49]. Among the most popular metrics, we used the Flesch Reading Ease formula, the Flesch-Kincaid Grade Level, and the SMOG formula [56, 66, 106]. The Flesch–Kincaid Reading indicates how difficult a text can be understood and it is measured in an interval between 1 and 100 [65]. The higher this score, the easier for a particular text to be read by the majority of people [56]. The Flesch–Kincaid Grade Level Test [65, 66] reflects the US education system needed to understand a text. It ranges from 0 to 18, where 0 indicates a basic level (learning to read a book) while 18

	Event message (title + description)	Flesch-Kincaid	Flesch Reading	SMOG Index	Sent.
1	Someone is trying to collapse a smart device down!	5.0	75.5	8.8	-0.74
	Someone is attacking one of your smart devices. This has the goal to make the device collapse.	5.1	73.8	10.1	0.00
2	Virus threat in a smart device!	2.5	87.9	3.1	0.83
	A virus has infected one of your smart devices. This virus can compromise your device and your privacy (e.g., steal your files and passwords).	6.8	67.8	11.2	-0.81
3	A hacker is breaking into a smart device!	3.8	82.4	3.1	0.62
	A non-authorized user has accessed one of your devices (or is trying to). If not stopped, this user may damage your device and steal your private data.	5.9	77.8	8.8	0.00
4	There's a danger of data theft from a smart device	3.7	86.7	3.1	-0.50
	Someone is trying to steal your private data on one of your smart devices. This can threaten your privacy (e.g., pictures/video stolen).	6.9	66.4	10.1	-0.92
5	A suspicious event has occurred in your network	6.7	61.2	8.8	0.00
	Someone is looking for vulnerabilities in your network. This event might reveal an incoming attack.	9.4	41.3	8.8	-0.85
6	Threats coming from outside your network	4.5	73.8	3.1	-0.69
	Suspicious activity is going on against your network. Someone could be trying to attack and access your network.	6.9	61.4	8.8	-0.87

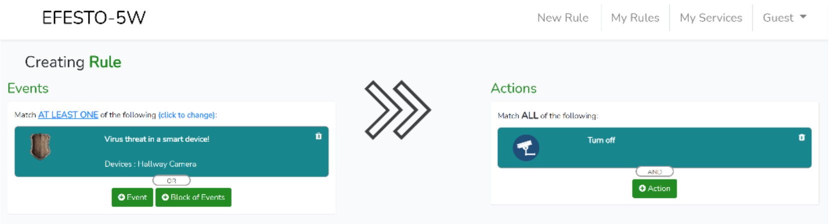
Table 5.2: Event messages and related readability and sentiment scores.

indicates an advanced level (an academic paper); the lower the easier it to read. The SMOG (Simple Measure of Gobbledygook) estimates the years of education a person needs to be able to comprehend a passage [106]. These three readability measures have been calculated by using the platform readable [125]. Regarding sentiment analysis, texts convey emotions which are key components to effectively communicate messages and to understand reactions to messages [141]. For warning messages, a negative valence is preferred to alert users about potential dangers. The text sentiment analysis was carried out by using the IBM Watson platform [30].

The resulting messages for the 6 ID events, as well as their final readability and sentiment scores, are reported in Table 5.2. The ID device and its events have been integrated into the EFESTO-5W platform. Figure 5.3 reports the visual representation in EFESTO-5W of an ECA rule to switch off a smart device (Hallway Camera in the example) when ID detects a virus threat in the smart device itself. The reported ECA rules refer to the six tasks administered in the evaluation study detailed in the next section.

5.4 EVALUATION

To verify if end users, even without technical skills in IT and cybersecurity, are able to defend their smart environments by defining ECA rules based on the ID device, we conducted a



Creating Rule

Events

Match AT LEAST ONE of the following (click to change):

Virus threat in a smart device!

Devices : Hallway Camera

Actions

Match ALL of the following:

Turn off

Figure 5.3: ECA rule created in EFESTO: it is triggered when a virus threat is detected in a smart device (Hallway Camera) and, in this case, executes the switch off of the attacked camera.

controlled experiment. More formally, the study aims to answer the following research questions:

RQ1. "Do the designed events enable the users to define ECA rules to protect a smart environment?"

RQ2. "Are there differences between expert and non-expert users in defining ECA rules based on the Intrusion Defender device to protect a smart environment?"

5.4.1 *Study design*

To answer the two research questions, we adopted a between-subject design, with users' skills as the independent variable. The two between-subject conditions are experts and non-experts. A total of 20 volunteers (6 females) were recruited. 9 of them were non-experts in IT and cybersecurity, while the remaining were mild experts (expertise assessed in a pre-test questionnaire). Their mean age was 27 years (SD = 7.04).

5.4.2 *Procedure*

Given the COVID-19 pandemic, the study was performed remotely. To facilitate the remote execution, a tool for remote user testing, eGLU-Box PA, was used [61]. Three evaluators (HCI experts) were involved. A total of 30 candidates among students, friends, family members and colleagues were contacted 5 days before the study by emails, SMS, and phone calls. Candidates were asked to fill in an online form providing demographic data, contact information and answer two questions on a self-

Task instructions	**ID Event**
T1. If a hacker is trying to switch off the lights in the house entrance and it's evening (between 7 pm and 4 am), then turn on the emergency lights outside and send a notification to your smartphone	1
T2. If a virus has infected your vacuum cleaner robot, then stop this robot	2
T3. If a hacker is trying to access your garage camera without your permission, then request to change your garage camera credentials by sending a notification to your email address	3
T4. If a hacker is trying to steal private data from your vacuum cleaner robot, then turn off this robot	4
T5. If a hacker is monitoring your network to try to identify vulnerabilities, then send a notification to your smartwatch	5
T6. If a hacker is attacking your network, then send a notification on your smartwatch	6

Table 5.3: List of tasks and the related ID event.

evaluation of IT and cybersecurity skills. 20 candidates agreed and were recruited as participants. The evaluators sent an email to all the participants with a link to start the test in eGLU-Box PA, the description of the technical requirements (PC, webcam, microphone, a browser like Chrome, Firefox, or Edge and a stable internet connection), and information on the test duration (around 20 minutes) so that participants could freely decide when to perform the tests without interruptions and disturbances.

After opening eGLU-Box PA, participants were asked to sign a digital consent form and, if they agreed, eGLU-Box PA checked the functioning of all the peripherals devices needed for data collection (microphone, webcam and screen recording). If no technical problems were detected, eGLU-Box PA showed a video, which asks participants to complete a set of tasks according to a scenario in which a person who lives in a smart home needs to configure the security of some devices. Then, eGLU-Box PA randomly administered six tasks one at a time. For each task, all the participants had a maximum of 5 minutes. Each task is designed to cover each of the 6 events. Table 5.3 reports the tasks and the associated ID event. After the task execution, eGLU-Box PA administered the SUS (System Usability Scale) and NASA-TLX questionnaires.

5.4.3 *Data collection and analysis*

To answer the two research questions, for each participant different quantitative and qualitative data were collected. Regarding the quantitative data, we gathered task completion time, task success (success, partial success and failure), SUS score for measuring perceived usability, and NASA-TLX index to assess the perceived workload [76, 77]. For SUS we also analyzed its subdimensions - Usability and Learnability [27, 92]. Regarding the qualitative data, significant comments made by the participants were annotated by reviewing the audio/video recordings. Independent t-test was used to analyze task completion times, SUS scores and NASA-TLX index (they did not violate the normal distribution, the assessment has been performed by using Shapiro–Wilk's test). Pearson Chi-Square was used to analyze success results (nominal values). An alpha level of .05 was used for all statistical tests.

5.4.4 *Results*

PERFORMANCE The participants' performance was evaluated by measuring the task time and the task success. Regarding the task time, participants spent an average of 112 seconds to complete a task (only successful or partially successful tasks were computed). In particular, each task required the following time: T1 = 219s, T2 = 69s, T3 = 143s, T4=66s, T5=85s, and T6=88s. The t-test revealed that no differences exist between experts and non-experts in performing all 6 tasks (p = 0.60). A detailed analysis revealed that no differences persist for T1 (p = 0.886), T3 (p = 0.55), T4 (p = 0.224), T5 (p = 0.106) and T6 (p = 0.181) while for T2 a significant difference emerged (p = 0.039). The analysis of the success rates revealed an overall positive performance of all the participants. The 6 tasks resulted in a 71% of success rate: T1 = 62.5%, T2=75%, T3=77.5%, T4=67.5%, T5=80%, and T6=67.5%. Pearson Chi-Square revealed that no differences exists between experts and non-experts ($\chi(1) = 2.209$, p = .331). No differences also emerged comparing task success of experts and non-experts for each task (T1: $\chi(1) = 2.229$, $p = .317$; T2: $\chi(1) = 3.300$, $p = .069$;

T3: $\chi(1) = 2.424$, $p = .298$; T4: $\chi(1) = 2.181$, $p = .336$; T5: $\chi(1) = 0.202$, $p = .904$; T6: $\chi(1) = 1.694$, $p = .429$).

PERCEIVED USABILITY The perceived usability was assessed by analyzing the results of the SUS and NASA-TLX questionnaires. The SUS scores highlighted that, in general, a good usability level in creating ECA rules with ID (SUS Score: $\overline{x}$ = 83.9, SD= 11.3; SUS Usability: $\overline{x}$ = 84.11, SD= 11.4; SUS Learnability: $\overline{x}$ = 83.8, SD = 20.7). No differences emerged in the SUS scores even in comparing experts and non-experts (SUS Score: p =.852; SUS Usability: p = .87, SUS Learnability: p=.394). The analysis of the NASA-TLX results shows that participants' workload was quite low ($\overline{x}$ = 24.91, SD = 12.34). No differences emerged in the workload of experts and non-experts (SUS Score: p =.852; SUS Usability: p = .87, SUS Learnability: p=.307).

5.5 DISCUSSION

Concerning RQ1, interesting and promising results emerged. Indeed, the overall task success rate is 71%, which can be considered as a positive result. T1 was the most complex since it required to set two actions in the rule. It resulted in the lowest rate (62.5%); however, the analysis of media recording revealed that only one user selected the wrong ID event while 6 users selected one or two wrong actions. The other tasks obtained a higher success rate but some users selected the wrong events: three users for T5, four users for T2, T3, T6, and six users for T4. Focusing on T2, T3, T4 and T6 it emerges that users selected the wrong event more frequently, and a deeper investigation is needed to clarify the users' difficulties in identifying the right ID event. The analysis of video recordings and further reflections on the event descriptions led us to hypothesize that some descriptions can appear too similar to users who do not have a profound knowledge of possible malicious attacks; therefore, they are in trouble guessing which is the right one. This would imply a revision of the event taxonomy. Even results on task time can be considered positive. Participants spent around 112 sec. creating the right ECA rule to defend the smart home. Only T1 required an average of 219 sec.; the analysis of the recorder media confirmed that the complexity of the task caused this higher time since two actions and temporal constraints had to be defined. These good performances have been confirmed by the perceived usability. Indeed, SUS scores (SUS Score: $\bar{x}$ = 83.9, SD = 11.3; SUS Usability: $\bar{x}$ = 84.11, SD = 11.4; SUS Learnability: $\bar{x}$ = 83.8, SD = 20.7) highlighted excellent usability: with respect to of one thousand studies reported in Bangor et al, SUS score above 68 are considered above the mean and scores above an 80.3 are in the top 10% of scores [15]. Similarly, NASATLX revealed that a low workload ($\bar{x}$ = 24.91) was required to accomplish the 6 tasks. With respect to RQ2, the results demonstrated that, in general, there are no differences between experts and non-experts. This result is very important since, typically, tools for advanced configurations are designed for professionals. This limitation is common, for example, for TASs that are created to suit the skills and mental models of technical users [148]. We can safely assume that thanks to the abstraction mechanisms designed for ID events, non-experts perform like experts. Of course, the study results are important because both users' performances and usability obtained positive results. The only emerged difference regards the task time spent for T2 since non-experts were slower. The analysis of the recorded media revealed that two users spent more time since they fixed the rule during their creation, in both cases because they selected the wrong action, i.e., switch off instead of stop. Finally, an interesting behavior has been observed three times in T6. Two non-experts and one expert created the rule by adding two alternative events, i.e., the ones titled "A suspicious event has occurred in your network" and "Threats coming from outside your network". Even if the right event is the second one, this task was considered as a partial success since the created rule produces the expected behavior. However, this aspect deserves more attention since it might be the symptoms that the two events confuse users. For example, the aggregation of the two events into a new one could be a possible solution. Indeed, the two events (5 = A suspicious event has occurred in your network; 6 = Threats coming from outside your network) might appear quite similar and their distinction can easily confuse the users.

In conclusion, the integration of the ID module into EFESTO-5W, a TAS that uses a visual, wizard-based approach, empowered users in the definition of security-triggered rules. While the study focused on the EFESTO-5W visual paradigm, the principles used for threat categorization and event descriptions can be applied to other TAS systems, such as IFTTT. In a comparative study, it was found that non-expert users were able to create rules that addressed security and privacy issues with a performance comparable to that of expert users. These results demonstrate that the proposal effectively supports end-users in the somewhat tedious tasks of securing their smart environment, and therefore positively provides an answer to the **RQ0**.

6 IDENTIFICATION OF HARMFUL ECA-RULES

Enabling end-users to define custom behaviors for their smart environment in a simple and intuitive fashion represents the main goal for a TAP platform. However, the usage of such tools by a non-expert user could easily lead to the generation of severe security and privacy flaws, risky for both the smart environment and mostly for the end-user's privacy. Therefore, it is essential to support end-users when creating ECA rules by providing them with information on the potential risks.

In this chapter, we will present solutions aiming at identifying security and privacy issues concealed within ECA rules [23].

To date, very little research has been performed to address the problem of automatically identifying risks concealed by ECA rules. In particular, Surbatovich *et al.* defined an information-flow lattice to analyze potential secrecy or integrity violations in ECA rules [138]. Paci *et al.* [118] proposed two approaches based on information flow analysis to detect rules that unintentionally violate users' privacy by sharing private photos. Other works focused on the automatic identification of undesired behaviors caused by rules' chain execution [107, 162], where a rule is automatically triggered by the action of another rule without user intervention. Instead, we propose to tackle such a problem by using Natural Language Processing (NLP) techniques, for semantically analyze the information of the rules, and infer how the type of trigger and action is related to the potential damage a rule might cause. In particular, the NLP models analyze the triggers and actions associated with the rules, and the natural language textual descriptions provided by the creators. The latter represent an important resource for understanding the behavior of the rules, as demonstrated by the effectiveness of NLP-based analyzers that exploit these descriptions to generate executable code [97], and to infer the context in which the devices are involved [101].

We evaluate the considered NLP-based models on the IFTTT (If-This-Then-That) platform, exploiting a set of crawled applets directly from the website and available on the Web [109]. We extract a set of 79,214 IFTTT applets and classify them into four categories based on the type of damage they might cause. We select 2,473 rules for manual classification and classify the remaining by using an ensemble of three semi-supervised learning techniques. Then, using the labeled dataset we train several classifiers on different combinations of rule's features. The experimental results demonstrate that the model based on BERT, a pre-trained language model released by Google [51], achieves the highest Precision and Recall scores. Furthermore, we compare our approach with a baseline system implemented by using the information flow approach, which only focuses on the analysis of the event and the action chosen for building the rule [118, 138].

The main contributions are the following:

- Define a process for automatic labeling ECA rules with respect to security and privacy risks. The proposed strategy encompasses the application of an ensemble method to the predictions provided by different semi-supervised learning techniques.

- Make a dataset of ECA rules, crawled from the IFTTT platform and labeled with respect to security and privacy risks, publicly available in the additional material accompanying this thesis, so that others can advance work in the area.

- Provide an automatic, NLP-based approach for semantic-based and context-aware identification of security and privacy risks underlying ECA rules.

- Implement and evaluate the proposed approach on a mainstream trigger-action platform, i.e., IFTTT, showing that among the considered NLP-based models, the BERT-based one gives higher accuracy when trained on all rule's features.

6.1 LITERATURE REVIEW

In this section, we discuss prior work for preserving the privacy and security of trigger-action IoT platforms and their users. We first discuss approaches that analyze and evaluate the potential risks of ECA rules, and then we review solutions that enhance platforms with mechanisms that prevent the disclosure of confidential information or malicious attacks.

6.1.1 *Analyzing Privacy and Security of ECA Rules*

The study presented in [138] is the first that analyzes the privacy and security risks of IFTTT applets. In particular, Surbatovich *et al.* analyzed a dataset consisting of 19,323 IFTTT applets with a multi-level lattice that associates security labels to IFTTT triggers and actions. Information flow analysis is then performed to determine if a rule could involve a secrecy or integrity violation. The results highlighted that around 50% of the analyzed rules are potentially unsafe. They also manually categorized a subset of randomly selected applets according to the potential issue they can cause. About 60% of them were involved in a violation. The study presented in [39] refined the previous analysis by taking into consideration two factors while marking applets as harmful or not, i.e., the contexts in which the rules are applied, or the users' privacy preferences. The user study with 28 IFTTT users and 732 applets revealed that, with respect to the prior work, the number of harmful applets was significantly reduced by considering users' opinions.

In [118], Paci *et al.* focused on privacy issues related to sharing images via IFTTT applets. In particular, they introduced two prototypes, one for providing users with warnings about the privacy risks they may incur at design-time (when an applet is created), and one for providing warnings at run-time (when an applet is running). The first prototype considers the audience of the information, namely the "visibility", to report a privacy violation. In particular, the tool verifies whether sensitive information flows from a private trigger to a public or restricted action. The second prototype considers the "sensitivity" of the shared data. To this end, the tool exploits the Google "Vision" API to evaluate the sensitivity of a photo, reporting a privacy violation if a sensitive data item flows from a private trigger to a public or restricted action.

Other approaches focused on the problem of identifying unexpected behaviors potentially caused by ECA rules. McCall *et al.* [107] proposed SafeTAP, a tool that verifies through model checking whether the behavior of a new rule is affected by the existing ones [21]. Xiao *et al.* proposed A3ID, a tool for detecting implicit rule interferences, which occur when two or more rules are triggered simultaneously, causing contradictory effects on the environment [162]. A3ID uses NLP techniques to extract smart devices' knowledge (e.g., functionality, effect, and scope) from knowledge graphs. iRULER is a system proposed by Wang *et al.* for detecting different interferences conditions between trigger-action rules, such as action loop, where a rule is activated cyclically, or condition block, where the condition of a rule is unsatisfiable [152]. To this end, iRULER uses Satisfiability Modulo Theories solving [113] and model checking by operating on an abstracted information flow model inferred with NLP techniques. IoTMon [54] and SafeChain [80] are systems for identifying harmful attack chains produced by a combination of ECA rules. ProvThings is a tool that tracks data provenance for the purpose of providing explanations of rules' chain behaviors [153].

With respect to the approaches that identify harmful ECA rules through information flow analysis [39, 138], we propose the use of NLP techniques to extract semantic and contextual information from ECA rules, which are used to classify them according to the type of damage they could cause when activated. Moreover, while the approaches [54, 80, 107, 152, 162] aim to identify possible interferences/interactions between ECA rules, which could damage the smart environment, or affect the safety of the user, the proposed approach focuses on the identification of individual ECA rules that are potentially dangerous for the security and privacy.

6.1.2 *Protecting TAPs from Privacy and Security Violations*

Xu *et al.* examined the possible leak of privacy information to which users may be exposed while using the main platforms for managing IoT devices and online services [163]. In particular, they analyzed how these platforms could reconstruct a user's behavior model through all the events for which s/he has defined a custom rule. To prevent this from happening, they introduced a process that filters and fuzzes the stored events. Bastys *et al.* demonstrated that IFTTT applets are vulnerable to attacks that could exfiltrate the private information of the users [17], and proposed two countermeasures. The former is based on an access control policy to prevent information flows from private sources to public sinks, the latter relies on a framework for monitoring the applet information flow over time, intending to identify what information the attackers might obtain from applet output and exploit them for future attacks.

Chiang *et al.* observed that TAP platforms are authorized to manipulate a lot of sensitive user information, and for this reason, they represent an attractive target for attackers [37]. To alleviate this problem, the authors proposed two platforms that can be integrated within a TAP to improve the privacy of users' data. The first aims to hide trigger information by sending fake information to the platforms, while the latter aims to preserve user privacy by masking the connection between the users and their data. Similarly, Chen *et al.* addressed the problem concerning the loss of users' sensitive data occurring when a TAP is compromised [35]. They proposed eTAP, an encrypted trigger-action platform capable of executing rules without accessing users' data in plaintext.

Fan *et al.* explored the possibility of attackers forcing rule executions with forged IoT devices or malicious events [60]. To face this issue, the authors proposed Ruledger, a ledger-based IoT platform that can be integrated within a TAP to guarantee the correct execution of rules.

IoTGUARD is a system that protects users from undesirable device states by monitoring trigger-action programs [33], blocking risky actions when integrity or confidentiality violations might happen. FlowFence is a framework that allows users to control their information flow once rules gained sensitive data access permissions [62]. SainT is a tool that identifies sensitive data flows by performing static analysis on information flow from sensitive sources to external sinks [31]. SOTERIA [32] and IotSan [116] are systems that apply model checking to control whether user security and safety properties are breached when using IoT platforms. IoT-Praetor is a system that exploits NLP techniques for extracting the interaction and communication behaviors of IoT devices from ECA rules, and comparing the obtained information to the actual behaviors detected at run-time [149].

The approach proposed in this thesis is complementary with respect to those that safeguard the security and privacy of the users by protecting the information processed by TAPs [35, 37, 163], because it evaluates the risks by only analyzing the ECA rule's information. Similarly, the approaches proposed in [17, 33, 60, 62] use rule process monitoring techniques to identify potential damages at run-time, while our proposal identifies them at the time the ECA rules are defined. Moreover, while the approaches [31, 32, 116] are functionally dependent on the structure of an IoT program's source code, our proposal evaluates the behavior of an ECA rule without the need to analyze the source code. Finally, while [149] exploits NLP techniques to produce an intermediate representation of an ECA rule, we exploit NLP techniques for analyzing the semantic structure of an ECA rule with the aim of identifying potential damages.

6.2 CONSTRUCTION OF CLASSIFICATION MODELS FOR IDENTIFYING HARMFUL ECA RULES

This section describes the steps we perform to construct a classifier of harmful ECA rules. In particular, Fig. 6.1 depicts the process yielding the definition of effective supervised models for properly discriminating ECA rules according to possible classes of risk. More specifically, the process consists of three main phases:

(a) *Labeling ECA Rules*: this phase aims to prepare labeled datasets for classification models. Before carrying out this procedure, it is required to define the possible classes of risk for ECA rules, and the corresponding labels. Successively, each ECA rule of the input dataset has to be annotated with a suitable label. Since the manual labeling process is time-consuming, and the number of rules in the datasets might be very high, we propose to use a semi-automatic labeling strategy. In particular, we partition the original dataset of ECA rules into two subsets: a random small set of rules, which is labeled manually, and the set of the remaining rules, which is automatically labeled using semi-supervised classification models. The latter exploits the set of manually labeled rules to acquire sufficient knowledge for providing the labels to the rules contained in the larger set. Finally, an ensemble approach is employed to establish, among the labels provided by the semi-supervised models, which should be assigned to each rule, yielding a final large dataset of rules labeled according to the classes of risk.

(b) *Training ECA Rules Classification Models*: this phase aims to train classification models by using the dataset of labeled ECA rules, which is called *training set*. The features considered for training the models correspond to the components of the ECA rules revealing the context of use and the meaning of the rule, i.e., the trigger, the action, and the description of the rule behavior. These components are usually provided in textual form, so NLP techniques can be used to extract semantic information that can be exploited by classification models to identify and discriminate among the classes of risk. It is worth noting that the training set is most likely imbalanced, in fact, most of the rules do not provide damage to users, while some classes of damage are more frequent than others. To deal with the classification errors caused by imbalanced datasets, we consider the application of a weighted loss function [86], which weights the classification errors according to the number of rules available for each class. The best settings for the weighted loss function can be inferred by employing the stratified k-fold cross-validation [53].

(c) *Testing ECA Rules Classification Models*: this phase aims to evaluate the performances of the classification models and, indirectly, the quality of the labels generated for the training set. This is performed by giving in input to the classification models a set of manually labeled ECA rules. The models' performances can be measured by using well-known metrics, such as *Precision*, *Recall*, *F1-score*, and *Accuracy*.

In the following sections, we describe how the considered process has been applied to a case study concerning the IFTTT platform.

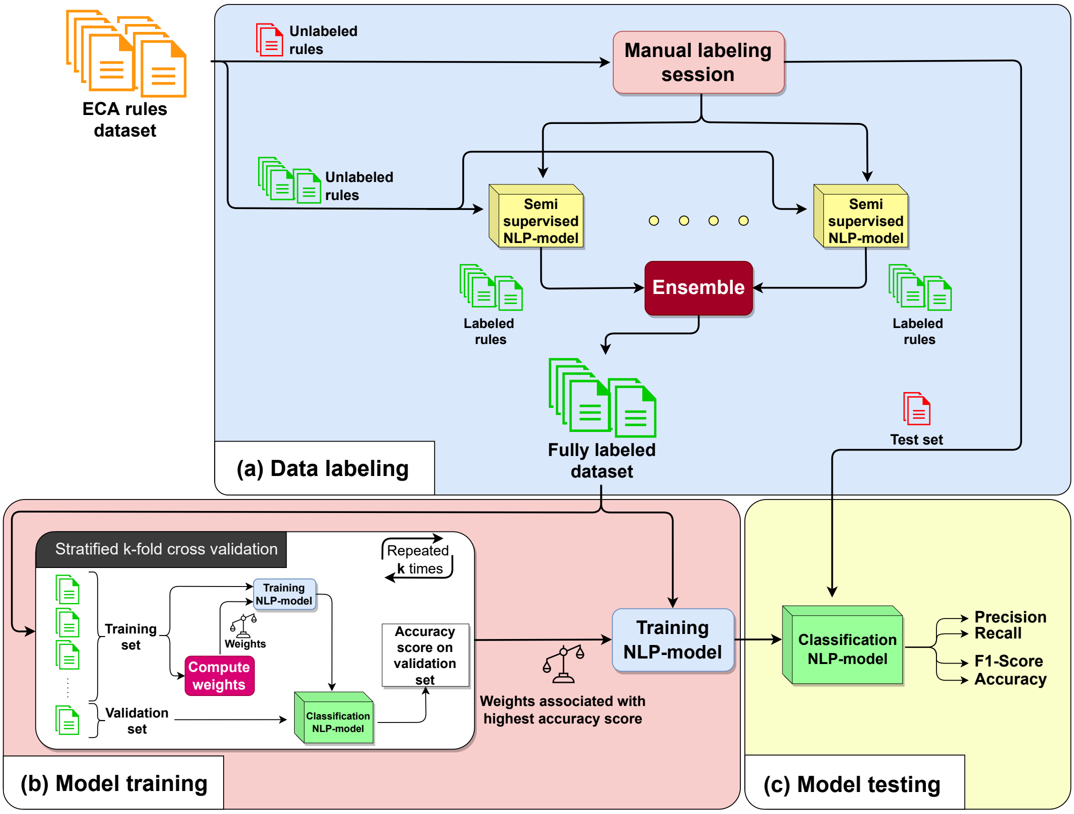


Figure 6.1: The proposed process for constructing and evaluating NLP-based models for detecting harmful ECA rules.

6.3 THE IFTTT DATASET

The dataset used for training and testing the models for classifying harmful IFTTT applets is the one published by Mi et al. [109] (see Section 4.1.3).

In our work, we only consider the following applet features derived from the terms originally assigned by Ur et al. (see Paragraph 4.1.2):

- Title: is a string containing the title of the applet;
- Desc: is a string containing a description of the applet's behavior;
- TriggerTitle: is a string representing the name of the trigger that "activates" the applet;
- TriggerChannelTitle: is a string representing the name of the trigger channel chosen by the user as a trigger for the applet;
- ActionTitle: is a string representing the name of the action to execute in response to the trigger;
- ActionChannelTitle: is a string representing the name of the action channel chosen as an action by the user.

It is worth noting that the first two features are continuous since they are defined by the applet's creator, whereas the remaining ones are discrete since they are automatically populated by IFTTT and can assume a finite number of values. As an example, the ECA rule shown in Fig. 7.4 could be specified by the following parameters:

- *Trigger channel*: `Netatmo Weather Station`
- *Trigger*: `Temperature rises above`
- *Action channel*: `Gewiss Smart Home IoT`
- *Action*: `Control your Shutter or Venetian`
- *Fields*: `Temperature threshold (25° in Fig. 7.4), Unit of measure (Celsius in Fig. 7.4)`

- *Title*: `Let some fresh air comes in your house when it gets too hot`
- *Description*: `If the temperature inside your house is above a certain threshold, automatically open the shutters.`

An accurate data cleaning process is performed on the initial dataset, which considerably reduces the number of applets. In particular, to obtain a uniform dataset in language, we use the LANGDETECT Python library to filter out the applets whose title and description are not written in English. Subsequently, we discard all applets without title or description or containing only numbers for these features. The resulting dataset consists of 116,825 applets, and is provided in the supplementary material [1]

6.4 DATA LABELING

As said above, the rule labeling process is performed by applying a combination of manual labeling and automatic labeling with semi-supervised models, and an ensemble strategy. This allows us to perform extensive labeling of the dataset minimizing the manual effort. In this section, we go through all the phases yielding a fully labeled dataset with respect to the considered classes of risk.

6.4.1 *Categorizing Applets based on Security and Privacy Risks*

We consider the work in [138] for categorizing the damages that could be inflicted by an applet on the user. An interesting result of the analysis performed in [138] is that not all applets need a third party to cause a risk. In fact, many applets are dangerous due to issues resulting from users' behaviors. For instance, at first sight, an applet having title "`Keep your Facebook and Twitter profile pictures in sync`" could not seem harmful. However, in a scenario where a user has a private Facebook profile, if s/he

[1] Supplementary material can be found online at `https://github.com/empathy-ws/Harmful-ECA-rules-classifiers`. The repository provides the source code and datasets used in the experimental sections of the thesis, as well as detailed notebooks showing the use of the provided models.

forgets that such an applet is active, unwanted photos will automatically be published to Twitter, where the audience might not be the same as the one on Facebook, causing possible embarrassment. In this case, an attacker does not need to intervene for providing damages, since is the user's behavior that generates harm to his/her own privacy. At the same time, other applets could violate users' privacy/security due to attackers exploiting the behavior of an applet to damage an IoT device or to disrupt online services.

In [138], by manually examining a set of applets, the potential damages they can cause are classified into four macro-categories:

1. *Innocuous:* causes no harm, i.e., an applet for which it is not possible to imagine a realistic harm. As an example, the applet: "`If I meet my daily step target, update a file with the statistics on my phone`" has no negative consequences because sensitive information is not shared with third parties, without causing embarrassing situations;

2. *Personal:* causes loss of sensitive data. This damage is self-inflicted since any damage is the result of the user behavior. As an example, the applet: "`If I take a new photo, then upload it on Instagram as a public photo`" could unintentionally leak sensitive information;

3. *Physical:* causes damage to physical health or goods. This damage is external as a third party can potentially inflict the damage. As an example, the applet: "`If the last family member leaves home, then turn off lights`" is dangerous since turning off the lights in a predictable way signals that the house is empty, making it easier for a thief to plan the right time for a theft;

4. *Cybersecurity:* causes interruption of an online service or distribution of malware. This damage is external too, as a third party can potentially inflict the damage. As an example, the applet: "`If there is a new email in your inbox with an attachment, then add the attached files to OneDrive`" could be used to spread malware to all devices synced with a OneDrive account. If a malicious at-

tachment is propagated to all synced devices, it increases the likelihood that the file will be opened by the user.

The analysis of the results highlighted that the most common damage is personal, i.e., the one caused by mistakes of users and not from the involvement of a third party. Concerning damages potentially inflicted by an attacker, cybersecurity damages are found to be more frequent than physical ones.

According to the considered macro-categories of risk, we use the following classes for applet labeling: class *0* corresponds to "Innocuous" applets, class *1* to the applets underlying a "Personal" damage, class 2 to the applets which could lead to a "Physical" damage, and class *3* to the applets exposing "Cybersecurity" damages. In the following, we provide details about the manual and automatic labeling processes.

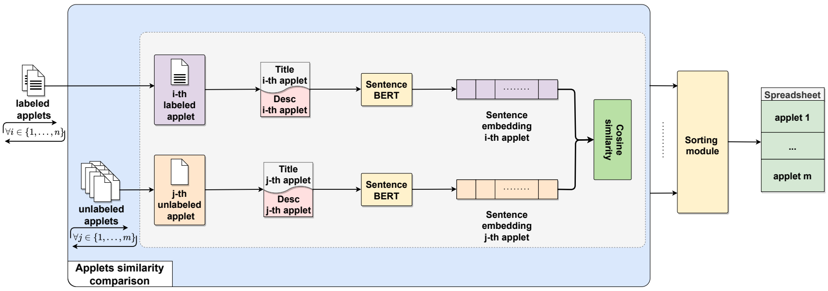


Figure 6.2: Similarity comparison process of IFTTT applets based on SentenceBERT.

6.4.2 *Manual Applet Labeling*

We apply the majority method for manually labeling the applets of the IFTTT dataset. In particular, the first and second authors are in charge of manually labeling the applets, while the third intervenes when there is no agreement. This phase leads to 492 class 0 applets, 296 class 1 applets, 105 class 2 applets, and 107 class 3 applets.

To balance the number of applets of each class, we define a process for selecting further applets to be labeled manually as shown in Fig. 6.2. For each labeled applet *a*, we build a spreadsheet containing all the unlabeled applets sorted in descending order based on their similarity with respect to *a*. The similarity is evaluated through a combination of vector semantics and similarity functions. The former is used to compute a vector representation of sentences, namely *sentence embeddings*, which takes into account their semantic meaning [87]. For computing the sentence embedding of each applet, we apply SentenceBERT [126] on the concatenation of the applet's title and description, whereas the cosine similarity function is used to compare the embeddings.

By manually reviewing each spreadsheet, we select and label a subset of applets having "similar" characteristics to those already labeled, but with at least a difference in the trigger, the action, and/or the involved channels. The resulting dataset of manually labeled applets consists of 503 class 3 applets, 502 class 2 applets, 501 class 1 applets, and 967 class 0 applets, for a total of 2,473 instances.

6.4.3 *Automatic Applet Labeling*

To increase the number of labeled data, we devise a process combining different semi-supervised learning models with an ensemble strategy, as shown in Fig. 6.1. This section illustrates the three semi-supervised learning techniques we use to automatically label additional applets, namely Self Learning, Label Propagation, and Generative Adversarial Learning, and the ensemble strategy we adopt to generate the final labeled dataset.

SELF LEARNING It consists in turning any supervised classifier into a semi-supervised method by iteratively labeling the unlabeled data, and adding these predictions to the set of labeled data until the classifier converges [165]. More specifically, we implement the following Self Learning process:

(a) Train a classifier C on the set of available labeled applets A;

(b) Use C to make predictions on the set of unlabeled applets U;

(c) Move from U to A the applets whose predictions satisfy a condition defined with a *confidence* parameter, which is specified a priori. These are called *"pseudo-labeled applets"* to distinguish them from the original labeled ones.

Steps (a)-(c) are repeated until one of the following convergence criteria is reached:

- The maximum number of iterations is reached;
- All predictions obtained on the applets of the set U do not satisfy the condition in (c);
- The set U is empty.

We set the maximum number of iterations to 5, the confidence parameter to 0.6, and use BERT as a classifier [51]. Moreover, we use the *softmax* function to compute the probabilities with which unlabeled applets are associated with each class. Thus, if the highest estimated probability is greater than 0.6 then the corresponding class is assigned to the applet, becoming a pseudo-labeled applet.

LABEL PROPAGATION It is an inference algorithm based on semi-supervised graphs [172]. The algorithm constructs a *similarity graph* that models the set of training data trying to propagate known labels across the edges of the graph, from labeled samples to samples for which the label is not available. Since the data are in textual form, we apply techniques for obtaining a vector representation of them. In particular, we use the sentence embeddings computed through SentenceBERT [126], which produces a 512-dimensional vector of the input sentences.

The graph construction process deals with converting the dataset X into a graph G, where X represents the input samples composed of the applets x_1, x_2, ..., x_n, and each applet x_i in turn is represented by a 512-dimensional vector. Each applet is assigned to a node of the graph, and a weight $w_{i,j}$ is assigned to each edge connecting the pair of nodes i and j. To identify the similarity between two nodes, a matrix of weights W is computed. To calculate the weights, it is necessary to employ a kernel. Among the possible applicable kernels, we use the *K-nearest neighbors*, which produces a fully connected graph represented in memory by a *sparse* matrix, and guarantees fast execution times.

The obtained weights are used to compute the probabilities of propagating a label from a labeled node to an unlabeled one. In particular, the probabilities are used to assign soft labels to each node that can be interpreted as distributions over labels. Labeled nodes have probability 1 of belonging to one of the four classes (since their classes are known), while unlabeled nodes get their class from "neighboring" nodes. During the execution, these distributions are altered causing the change of the labels assigned to unlabeled applets. The process iterates until convergence, i.e., the probabilities do not change and the labels associated with unlabeled applets are no longer changed, or a fixed number of iterations is reached. We set the maximum number of iterations to 1,000.

The `LabelPropagation` class of the SCIKIT-LEARN Python library is used to implement the algorithm.

GENERATIVE ADVERSARIAL LEARNING FOR ROBUST TEXT CLASSIFICATION WITH A BUNCH OF LABELED EXAMPLES

It is an extension of the BERT model within the Generative Adversarial Network (GAN) framework allowing for the implementation of an effective semi-supervised learning schema [48]. This model allows training BERT on a limited number of labeled samples, with respect to a larger number of unlabeled samples. The fine-tuning phase of BERT is extended by introducing a Discriminator-Generator setting, where:

- The generator G deals with the production of "fake" vector representations of sentences. In particular, G produces "fake" samples by taking as input a 100-dimensional noise vector based on a Gaussian distribution;

- The discriminator D is a BERT-based classifier that works on $k+1$ classes. In particular, D receives as input either a fake vector generated by G, or a vector from real data embedded by BERT. The final layer of D is a Softmax Output Layer producing a vector of size $k+1$, where k is the number of classes in the training set.

D has the role of classifying an instance as one of the k classes related to the task of interest (in this case $k = 4$) and must recognize the instances generated by G (to which it must associate the class $k + 1$). In other words, it must classify whether the input is a real instance or not; if it estimates the input as a real instance, it must predict which class the input belongs to. On the other hand, G must produce representations as similar as possible to the "real" instances. G is penalized when D correctly classifies an instance as fake.

We employed the PyTorch interface of the transformers Python library of Hugging Face for implementing the model. We set the parameters of the model as follows:

- Batch size: 32;
- Number of hidden layers for the Generator: 1;
- Number of hidden layers for the Discriminator: 1;
- Size of the noise vectors: 100;
- Learning rate Discriminator: $5e - 5$;
- Learning rate Generator: $5e - 5$;
- Epsilon: $1e - 8$.

Since D must also be trained to discriminate real sentences from fake ones, we introduce an additional class to those of the task of interest; in particular, the identifier "4" has been associated with the "fake" class.

ENSEMBLE Starting from the three datasets of labeled applets obtained with the semi-supervised learning approaches, we apply an ensemble strategy to get a single dataset. In particular, we use a majority-vote method across the three different semi-supervised models, assuming that if two models agree, the prediction would be more accurate. Thus, only the applets for which at least two semi-supervised techniques agree on their class labels are included in the final dataset with that class. This allows us to obtain more consistent labels for the evaluated applets.

With this strategy, we obtain a dataset containing 79,214 applets, where 56,236 belong to class 0, 16,344 to class 1, 3,433 to class 2, and 3,201 to class 3. Table 6.1 reports statistics about the service categories involved in the labeled dataset. For each service category, the table provides the percentage of involved services, and the percentage of applets whose triggers (actions, resp.) belong to a service within the category. We can observe that most of the services are for IoT devices, while triggers from social networking services are the most popular among the applets. On the other hand, almost half of the applets in the dataset have an action belonging to the "RSS feeds, online recommendation" category.

Table 6.1: Statistics about the services involved in the applets of the labeled dataset

Service category	% Services	% Triggers	% Actions
Smarthome devices (e.g., Light, thermostat)	44.79%	5.08%	4.33%
Online service and content providers	16.60%	11.72%	14.99%
Social networking, photo/video sharing	8.21%	35,78%	18.32%
Smartphones (e.g., battery, NFC)	5.56%	11.09%	2.62%
SMS, instant messaging, VoIP	5.55%	7.51%	2.29%
RSS feeds, online recommendation	4.11%	2.24%	44.34%
Smarthome hub (e.g., Samsung SmartThings)	2.90%	0.08%	0.03%
Personal data & schedule manager	2.90%	7.11%	0.64%
Wearables (e.g., smartwatch)	2.66%	0.37%	0.84%
Cloud storage (e.g., Google Drive)	2.17%	9.35%	1.01%
Time and location	1.60%	0.07%	6.49%
Email	1.45%	9.45%	4.01%
Other	1.50%	0.15%	0.09%

6.5 MODEL TRAINING

This section illustrates the models we implement for classifying the IFTTT applets, the techniques we adopt to solve the problem of imbalanced data in the final dataset, and the setup of the training phase. The models are implemented as Python modules.

We consider two types of classifiers. The first is based on artificial neural networks (ANNs), and treats discrete features as numerical values, by using the label encoder technique [154], and continuous features as textual values, by using a word embedding technique [70], namely *Global Vectors for Word Representation* (GloVe) [119]. The latter is based on BERT, a pre-training model of Natural Language Processing, which uses deep bidirectional transformers to train a language representation model through a large number of data [51]. In this model all features are treated as text.

The ANN-based and BERT-based models are trained considering three combinations of features:

combination 1: Title and Desc;

combination 2: TriggerTitle, ActionTitle, ActionChannelTitle, and TriggerChannelTitle;

combination 3: All features.

Before training the models, a pre-processing phase is carried out to remove noise from the data. In particular, we perform the operation of tokenization, normalization, noise removal, and lemmatization on the textual values. Successively, the encoding phase converts the dataset into a format valid for the classifiers.

6.5.1 *Classification by Artificial Neural Networks*

In the following, we illustrate the different architectures implemented for the ANN models. In particular, we consider a simple neural network (NN) when the features are discrete (combination 2), and the *Long Short-Term Memory* (LSTM) when continuous features are involved (combinations 1 and 3).

FIRST COMBINATION We train an LSTM model, named *LSTM-1c*, using the Title and Desc features. Their GloVe word embeddings are vector representations of textual data having a fixed length. Since the title and description of the applets have different lengths, an embedding with a longer length will be filled with zeros at the end (representing the so-called *padding*), if it is shorter, it will be truncated. To identify the most suitable embedding length, we analyze the distribution of the lengths of the dataset sentences obtained by concatenating Title and Desc. As shown in Fig. 6.3, most sentences are composed of 25 words or less, whereas the maximum length is 400. Using 25 as embedding length is unsuitable because many words may be lost. At the same time, setting the length to 400 leads to embedding with a lot of padding, which would not help the model to learn. Therefore, we set the length of the embeddings to 50, which corresponds to a good trade-off.

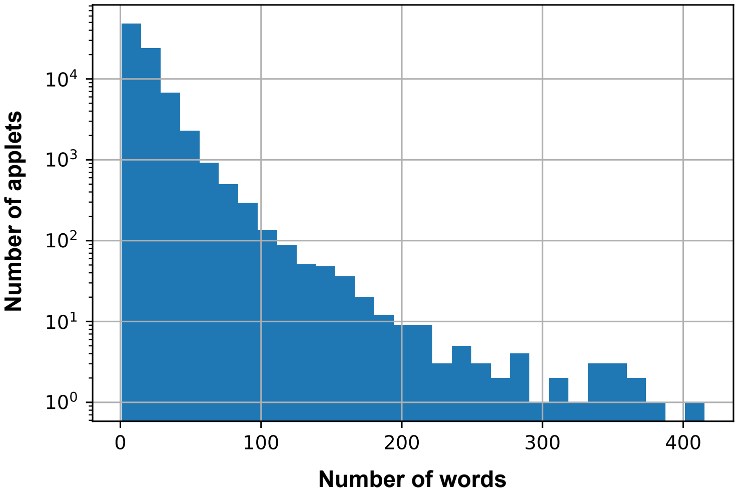


Figure 6.3: Length distribution (in logarithmic scale) of the sentences obtained concatenating the Title and Desc features.

Fig. 6.4 shows the architecture of the LSTM-1c model. Specifically, the *Input Layer* receives embeddings of 50 dimensions, then the *Embedding Layer* turns positive integers into dense vectors of fixed size. The latter requires the following parameters:

- The size of the dictionary (i.e., the number of distinct words in the input sentences);
- The size of the output vector, which is 50, as the dimension of the embeddings;
- The pre-trained embedding matrix specified using the *weights* attribute.

The *LSTM Layer* is composed of 70 neurons and uses the *tanh* (hyperbolic tangent) activation function, whose outputs are in the range [-1,1]. Subsequently, the *Dense Layer* is composed of 35 neurons and uses the *tanh* activation function. The values of

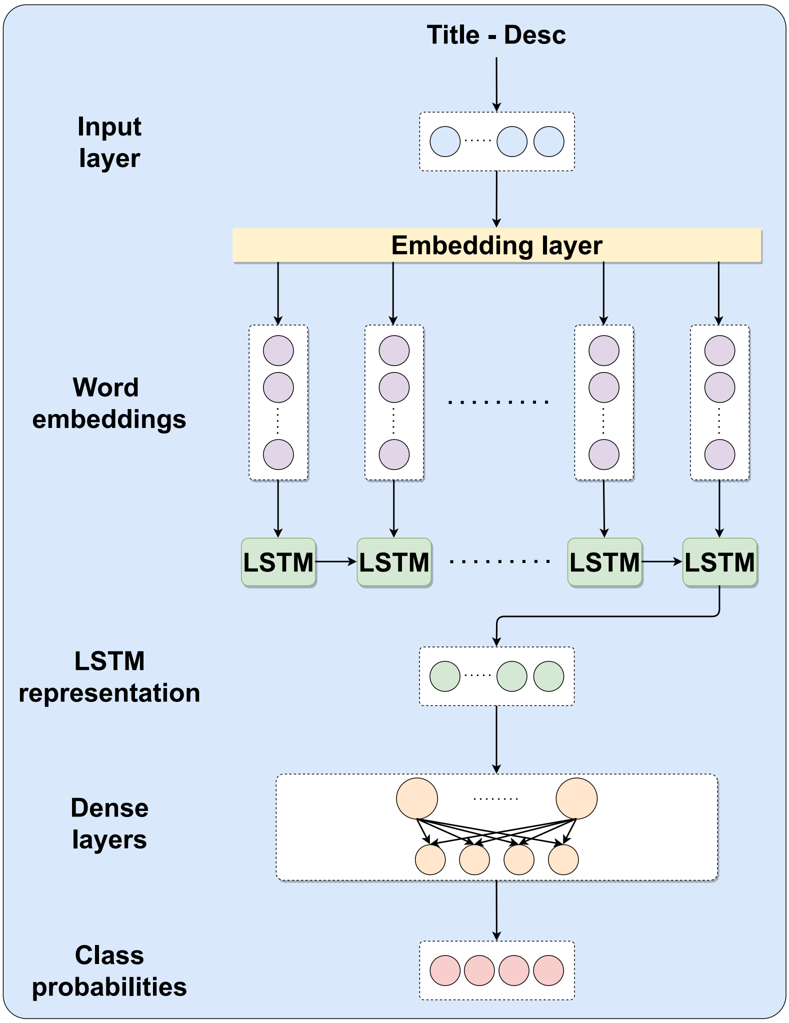


Figure 6.4: The architecture of the LSTM-1c model.

the two parameters are obtained through a fine-tuning process. The final Dense Layer acting as the *Output Layer* consists of 4 neurons and uses the *softmax* activation function. The number of neurons corresponds to the number of classes that the classifier has to predict. The LSTM-1c model is trained using 24 *epochs* and a *batch size* of 10.

SECOND COMBINATION We train a simple Neural Network model, named *NN-2c*, using the discrete features of the dataset. For these features, it is possible to apply the *label encoder technique* to convert them into numerical ones. This technique simply assigns a unique numerical value to each categorical value that a feature can assume. In this way, we obtain well-structured data, which can be used as input for densely connected neural networks.

Fig. 6.5 shows the architecture of the NN-2c model. It is composed of an *Input Layer* that accepts an input with size 4 (i.e., the values of discrete features) and three *Dense Layers*. Two of them consist of 50 and 20 neurons, and use the *tanh* activation function. As for LSTM-1c, the final Dense Layer acts as the *Output Layer* and consists of 4 neurons, with the softmax as an activation function. The NN-2c model is trained using 30 *epochs* and a *batch size* of 16.

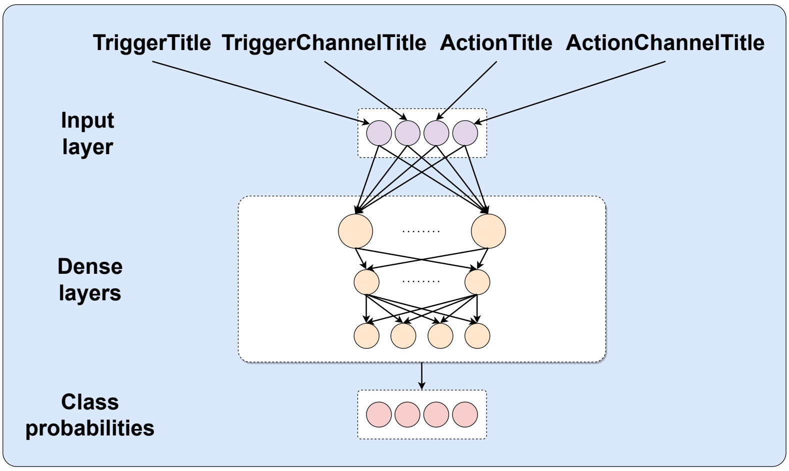


Figure 6.5: The architecture of the NN-2c model.

THIRD COMBINATION We train an LSTM model, named *LSTM-3c*, using all features of the dataset. The discrete features are converted from categorical to numeric, and combined with the continuous ones. To efficiently handle the different types of inputs, i.e., textual (i.e. Title and Desc) and numerical (i.e. TriggerTitle, ActionTitle, ActionChannelTitle and TriggerChannelTitle), we define two sub-models: the former receives textual input encoded with GloVe, while the latter receives numerical input encoded with the label encoder technique.

Fig. 6.6 shows the network architecture. The first sub-model is composed of an *Input Layer*, which receives embeddings of 50 dimensions, an *Embedding Layer*, and an *LSTM Layer* characterized by 70 neurons and using the *tanh* activation function. Also, the second sub-model is composed of three layers. In particular, an *Input Layer* receiving the four numerical values of the discrete features, and two *Dense Layers*, composed of 50 and 20 neurons respectively, on which the *tanh* activation function is used. The output of the LSTM Layer of the first sub-model is concatenated to the output of the second Dense Layer of the second sub-model, and used as input for another Dense Layer characterized by 10 neurons. Finally, the last Dense Layer acts as the *Output Layer* and is characterized by 4 neurons, corresponding to the classification classes. The LSTM-3c model is trained using 35 *epochs* and a *batch size* of 10.

6.5.2 *Classification by BERT*

In the following, we present the classifiers developed with BERT by considering the same three combinations of features, but all in a textual form. BERT is based on the concept of Transfer Learning, namely a Machine Learning technique in which a model exploits knowledge gained from a problem to improve its performances on a related one. Unlike directional models (e.g., the LSTM model), which read the textual input sequentially (from left to right or vice versa), BERT, as a contextual model, captures the relationships between words in a bidirectional manner. In this way, it can learn the context of a word based on everything around it.

We train BERT by freezing all the pre-trained layers, and a layer of untrained neurons is added to the top of the architecture. Thus, during the training phase, only the additional classification layer is trained on the dataset. The classifier has been implemented using the `BertForSequenceClassification` class of the TRANSFORMERS Python library, which corresponds to the BERT model with a single linear layer added at the top for classification.

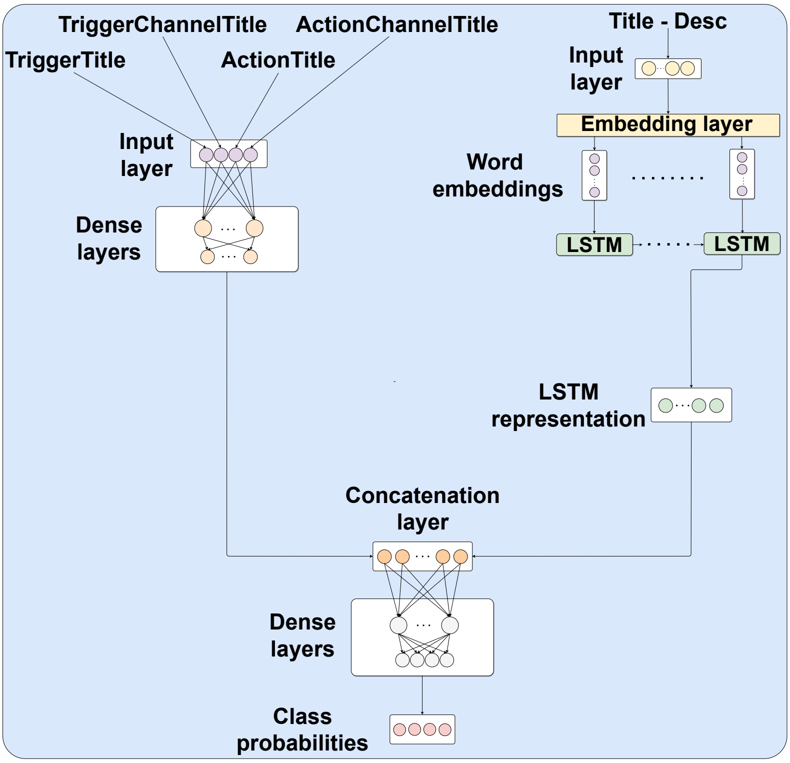


Figure 6.6: The architecture of the LSTM-3c model.

Among the different available pre-trained BERT models, we use the "bert-base-uncased", which represents the "base" version with 12 transformer blocks, 768 hidden units, 12 self-attention heads, and considers lowercase letters.

Fig. 6.7 shows the architecture of the implemented BERT-based model with all applet's features in input. In particular, the input is a sequence of tokens where a special classification token, denoted with *[CLS]*, is placed at the beginning. All tokens are first embedded and then processed in the following blocks. Each block applies self-attention [146] and provides the output to a feed-forward neural network. Finally, the representation corresponding to the token [CLS] (H) is given in input to the Dense Layer added at the top of the architecture, which is responsible for classifying the input applet.

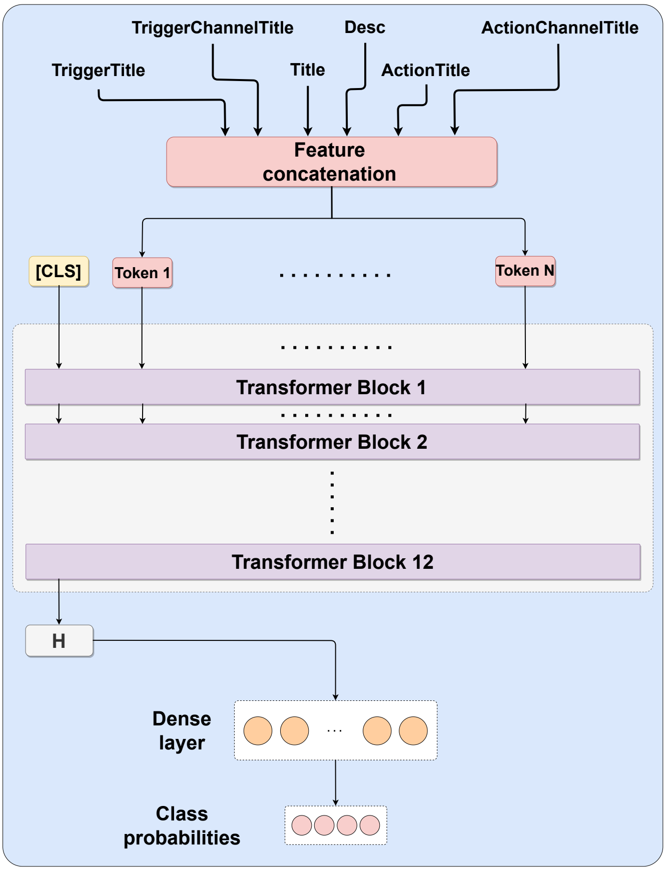


Figure 6.7: Architecture of the BERT-based model considering all applet's features (BERT-3c).

The models are trained by considering the following hyperparameters: 32 as batch size, $2e-5$ as learning rate, 2 epochs, and $1e-8$ as epsilon. This configuration is used with a maximum sentence length that changes based on the combination of considered features. In particular, for the first combination of features, we implement a model named *BERT-1c* with a maximum length of 50, as done in Section 6.5.1. For the second combination of features, we implement a model named *BERT-2c* with the maximum length of embeddings set to 20, which corresponds to the longer sentence (as shown in Fig. 6.8). With this choice, the semantic meaning of the longer sentences is better captured, but without excessively affecting the performances and the training quality of the model. Furthermore, it is also worth noting that the number of services provided by the IFTTT platform is constantly growing. Thus, the considered length allow us to correctly represent any new trigger/action characterized by many characters, that might be added in the future.

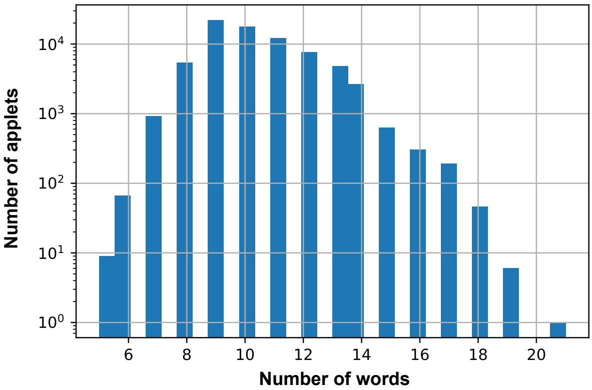


Figure 6.8: Length distribution (in logarithmic scale) of the sentences obtained concatenating the discrete features.

Concerning the combination that considers all the features, we implement a model, named *BERT-3c*, with the maximum length of embeddings set to 70, which allows us to fully consider the discrete features (since they are inserted at the beginning of each sentence and are certainly correct as provided by the IFTTT platform) and in addition, in whole or in part, the title and the description, which can have variable length and potentially a more or less distorted meaning.

6.5.3 *Training with Imbalanced Datasets*

The constructed dataset of IFTTT applets is "imbalanced" since the number of observations it contains is not the same for all classes. Imbalanced datasets are an important challenge in supervised classification [150]. In fact, a model trained on imbalanced data tends to classify the input instances with the class for which the most observations are available, namely the *majority class*. Different techniques can be used to alleviate the imbalanced dataset problem. In this work, we use "penalized" models, which try to penalize misclassifications related to minority classes more than those related to the majority class.

In general, the training objective of a model is the minimization of a loss function. Usually, every class in the loss function has the same *weight*, i.e., 1. Instead, in a penalized model these weights are altered for improving the accuracy of minority classes. In particular, it uses a *weighted loss function* that associates different weights to each class according to the number of class samples in the dataset. For the considered dataset, the minimum weight is associated with class 0, as it contains the largest number of samples.

We use the *categorical crossentropy* as a loss function for our multiclass classification task. To calculate the weight to be associated with each class, the `compute_class_weight` method of the SCIKIT-LEARN Python library is used. In particular, having indicated the string 'BALANCED' as a parameter, each weight is calculated as:

$$weight[C_i] = \frac{\#samples}{\#classes \cdot \#samples[C_i]}$$

where

- $\#samples$: represents the total number of instances considered for training;
- $\#classes$: represents the total number of considered classes;
- $\#samples[C_i]$: represents the total number of instances of class C_i considered for training.

The loss for a sample x of class C_i is calculated as:

$$loss(x, C_i) = weight[C_i] \cdot \Big(- x[C_i] + \log\big(\textstyle\sum_j \exp(x[j])\big)\Big)$$

Imbalanced datasets could also introduce biased estimations or overfitting in favor of the majority class when using k-fold cross validation to evaluate the performances of the models. To alleviate this problem, we use the *stratified k-fold cross validation* technique, which performs stratified rather than random sampling. Stratified sampling is a probabilistic sampling procedure whereby reference data are divided into subsets that are as homogeneous as possible to the variable whose value is to be estimated. It ensures that the folds of the data have a uniformly representative sampling of the target attribute.

We use the stratified *k*-fold cross validation for fine tuning hyperparameters. In particular, we employ it for the optimization of the weights to be adopted in the loss function previously introduced. We set *k* to 4 in this work.

6.6 experimental evaluation

We conduct a series of experiments to analyze the performances of the implemented models. This section illustrates the experimental setup, the adopted metrics, and the experimental results. The latter are compared with those obtained by a solution evaluating the risk of IFTTT applets through the information flow analysis [138].

6.6.1 *Evaluation Setup*

Each of the previously described models is trained on the set of 76,741 applets obtained by the ensemble approach, whereas the 2,473 manually labeled applets are used as a test set (named TS_2k). Moreover, since the training set is labeled through the application of semi-supervised techniques to the TS_2k applets, we make a further evaluation on 500 applets randomly chosen among the applets not yet labeled (named TS_500). In this way, we validate the quality of the proposed methodology, i.e., the performances of the classification models to properly discriminate applets' damage, and the quality of the labels provided by the ensemble approach.

Once the training and test sets are selected, as said above, we apply a weighted loss function to solve the imbalanced dataset problem, obtaining the weights through the stratified k-fold cross validation. As a new validation set is created at each iteration, it is not necessary to extract one from the training set. Therefore, once the weights are obtained, each model is trained on all 76,741 applets.

6.6.2 *Evaluation Metrics*

The performances of each classifier are evaluated by considering the following metrics:

- *Accuracy*: is the ratio between the number of instances correctly classified and the total number of considered instances;
- *Precision*(C_i): is the ratio between the number of instances of class C_i correctly classified and the total number of instances to which the classifier associates class C_i;
- *Recall*(C_i): is the ratio between the number of instances of class C_i correctly classified and the total number of instances of the test set labeled with class C_i;
- *F1-score*(C_i): is the harmonic mean of Precision and Recall;

where C_i is one of the four classes of damage. We also compute the average of the per-class metrics, weighted by the number of samples for each class in the test set (*WAvg*).

6.6.3 *Results and Discussion*

TS_2K EVALUATION Tables 6.2-6.5 report the values of Accuracy, Precision, Recall, and F1-score obtained by the different models on the dataset TS_2k. We can observe that the worst results are achieved by the models trained considering discrete features only (i.e., NN-2c and BERT-2c), whereas the best results are achieved by the BERT model trained on all features (BERT-3c), which obtained 88% for all the metrics.

LSTM-3c and BERT-1c are the models that obtained results closer to those of BERT-3c. However, it is worth noting that the LSTM-3c model achieves a Precision for class 0 (72%) lower than BERT-3c. This means that it classifies many applets that provide damage as Innocuous, and this is particularly dangerous for users. At the same time, this model obtains higher values for Recall of classes 0, 1, and 2, but a very low value for class 3 (22%). This means that it rarely classifies an applet with class 3, as also highlighted by the high Recall values of the other classes.

Table 6.2: Accuracy of the considered models on TS_2k

Model	Accuracy (%)
LSTM-1c	80
BERT-1c	86
NN-2c	31
BERT-2c	72
LSTM-3c	79
BERT-3c	**88**

Table 6.3: Precision of the considered models on TS_2k

	Precision (%)				
Model	class 0	class 1	class 2	class 3	WAvg
LSTM-1c	73	80	86	87	80
BERT-1c	**90**	80	85	88	86
NN-2c	47	26	31	25	35
BERT-2c	73	66	80	69	72
LSTM-3c	72	77	**91**	**93**	81
BERT-3c	89	**87**	88	87	**88**

Table 6.4: Recall of the considered models on TS_2k

	Recall (%)				
Model	class 0	class 1	class 2	class 3	WAvg
LSTM-1c	81	62	90	86	80
BERT-1c	77	82	98	**95**	86
NN-2c	24	32	48	28	31
BERT-2c	54	78	94	78	72
LSTM-3c	**90**	**93**	**99**	22	79
BERT-3c	86	79	97	91	**88**

Table 6.5: F1-score of the considered models on TS_2k

	F1-score (%)				
Model	class 0	class 1	class 2	class 3	WAvg
LSTM-1c	77	70	88	86	80
BERT-1c	83	81	91	**91**	86
NN-2c	32	28	37	27	31
BERT-2c	62	71	87	73	71
LSTM-3c	80	**84**	**95**	36	75
BERT-3c	**87**	83	92	89	**88**

In addition, since the lowest Precision values are obtained by LSTM-3c for classes 0 and 1 (72% and 77%, respectively), we infer that the model erroneously classifies most of class 3 applets with one of these two classes.

The BERT-1c model is characterized by performances very similar to those of BERT-3c, achieving slightly higher Recall (for classes 1, 2, and 3) and Precision (for classes 0 and 3) values, but lower values for Accuracy (86%) and weighted average results. Concerning the F1-score metric, the BERT-1c model shows a slightly higher value only for class 3, confirming that the BERT-3c model performs better on average. Since the BERT-1c model classifies the applets by only considering their title and description, the performances of the model strongly depend on the semantic consistency of the applets' title and description contained in the training set. This information is specified by the user when creating the applet, and it might happen that it is *inconsistent* with the applet's behavior. In these cases, the BERT-3c model can exploit the discrete features (trigger, action, and the corresponding channels) populated by the IFTTT platform.

By focusing on the results per class, we can observe that for some models the applets of classes 1 and 3 are the most difficult to identify. For class 1 applets, the reason is that they could be erroneously classified with class 2 or 3 due to their slight differences in the context of use. As an example, the applet "`Any new photo by me uploaded in a specific Google Drive folder, publish it on Twitter`" should be classified as class 1 because a user could share an embarrassing photo unintentionally, whereas the applet "`Any new photo uploaded by anyone in a specific Google Drive folder, publish it on Twitter`" should be classified as class 3, because the user's privacy may be compromised as it is not possible to know who will upload the photo that will appear on his/her Twitter profile. On the other hand, the applet "`New tweet by me with a specific hashtag, turn off lights`" could be used by a user to turn off lights with a goodnight tweet, but it might be triggered also in other situations causing the lights to go off unintentionally, and consequently, the applet should be classified as class 1. Instead, the applet "`New tweet by anyone in the area with a specific tag, turn on lights`" allows a user to know if there are people that published a tweet in the zone, but its behavior might be compromised by a third party causing damage to the lights, and consequently, the applet should be classified as class 2. The differences between these applets are hard to grasp, and justify the low performances of the models in the discrimination of class 1 applets with respect to other classes.

Regarding class 3 applets, as said above, they are difficult to classify by the LSTM-3c model and by the models using only the information provided by the IFTTT platform (the discrete features on trigger and action). This might be due to the fact that the dataset considered for this work does not contain information about the *Fields* of the applets. To understand this aspect, let us consider the following applet: "`Record your daily Fitbit activity in a Google Spreadsheet`". In this case, the IFTTT platform upon creating such an applet requires the user to specify the spreadsheet to be involved in the automation, which is a field of the applet. Depending on the selected spreadsheet, the applet may or may not compromise the user's privacy. This is because, if the user specifies a private spreadsheet, which only s/he can access, then such an applet would not pose any risk. Conversely, if the user specifies a shared spreadsheet, then his/her privacy may be compromised as private information may be leaked to other people.

TS_500 EVALUATION To verify if the considered models generalize well on new applets, we perform an additional evaluation by considering a new test set obtained through the random selection of 500 applets among those discarded by the ensemble approach (i.e., the applets that obtained conflicting predictions by the three semi-supervised techniques). The result of the manual classification on these new applets is the TS_500 dataset containing: 264 applets belonging to class 0, 161 to class 1, 28 to class 2, and 47 to class 3.

Table 6.6 reports the values of Accuracy and F1-score obtained by the classification models on TS_500. We can observe that all models perform worse than the evaluation on TS_2k, except for BERT-3c, which increases its Accuracy (F1-score, resp.) from 88% to 91% (92%, resp.). In the following, we provide a detailed analysis of the results achieved by BERT-3c. For this model, we also analyze whether semantically consistent titles and descriptions influence its performances. To this end, we manually verify the consistency of the applet's titles and descriptions with respect to the applet's behavior. This process leads to the identification of 75 applets whose titles and descriptions do not describe the goal of the automation. The dataset without these applets (named TS_425) consists of 209 class 0 applets, 147 of class 1, 27 of class 2, and 42 of class 3.

Table 6.6: Accuracy and F1-score achieved by the considered models on TS_500

Model	Accuracy (%)	WAvg F1-score (%)
LSTM-1c	60	57
BERT-1c	71	71
NN-2c	29	29
BERT-2c	63	61
LSTM-3c	75	73
BERT-3c	**91**	**92**

Table 6.9 reports the values of the metrics obtained by BERT-3c on TS_500 and TS_425. The results are better than those obtained on TS_2k in almost all classes, as also highlighted by F1-score. In fact, BERT-3c correctly discriminates class 1 applets, differently from what happened previously. However, the results for class 2 applets are considerably worsened.

Tables 6.7 and 6.8 show the confusion matrices obtained from the classification results on TS_500 and TS_425. We can observe that the model's performances are slightly better when only *consistent* applets are classified, confirming that this applet's property allows the model to better discriminate among classes. By analyzing the Recall, we can observe that BERT-3c correctly classifies all class 2 and class 3 applets, while it makes some mistakes on the other two classes. In fact, with this evaluation, we discover another ambiguity in discriminating between class 2 and class 0 applets. As an example, the applet with description "`Email a map of where I parked`" might be classified as class 2 because a user could share his/her car position unintentionally, and a third party might exploit this information to cause damage. However, as the ActionTitle of this applet is "`Send me an email`", it should be classified as class 0 because the information remains

Table 6.7: Confusion Matrix Obtained for TS_500

Target Class	0	1	2	3
0	232 97%	8 5%	22 42%	2 4%
1	6 3%	151 94%	1 2%	3 6%
2	0 0%	0 0%	28 54%	0 0%
3	0 0%	1 1%	1 2%	45 90%

Output Class

Table 6.8: Confusion Matrix Obtained for TS_425

Target Class	0	1	2	3
0	187 98%	2 1%	18 39%	2 4%
1	4 2%	139 99%	1 2%	3 6%
2	0 0%	0 0%	27 59%	0 0%
3	0 0%	0 0%	0 0%	42 89%

Output Class

private. Since the number of class 0 applets is much higher than those of class 2, it happens that these misclassifications for class 0 applets are relatively frequent with respect to class 2 applets, leading to a very low precision value for class 2.

6.6.4 *Comparative Evaluation*

To further assess the validity of the proposed approach, we compare the performances of the BERT-3c model with those of a baseline system implementing an information flow analysis similar to the one proposed in [138]. The latter exploits a secrecy lattice to identify possible violations caused by an IFTTT applet. The lattice of the baseline system introduces two levels of restriction, namely public and private, asserting that a secrecy violation occurs when the information flows from the private level toward the public one. Considering this type of approach, we devise a

Table 6.9: Performances of BERT-3c on TS_500 and its subset of "consistent" applets TS_425

	Accuracy (%)		Precision (%)		Recall (%)		F1-score (%)	
	TS_500	TS_425	TS_500	TS_425	TS_500	TS_425	TS_500	TS_425
Class 0			97	98	88	89	92	94
Class 1			94	99	94	95	94	97
Class 2			54	59	100	100	70	74
Class 3			90	89	96	100	93	94
WAvg	91	93	93	95	91	93	92	93

Table 6.10: A comparison of the results obtained by IFC and BinaryBERT-3c for TS_2k

	Accuracy (%)	Precision (%)		Recall (%)		F1-score (%)	
Methodology		class HL	class HF	class HL	class HF	class HL	class HF
IFC	46	41	70	87	20	56	32
BinaryBERT-3c	92	89	93	90	93	89	93

methodology following such a principle for classifying an applet through the analysis of its channels, namely *Information Flow Classifier* (IFC). The classification labels of IFC are:

- *Harmless*: indicates that the applet contains no elements that may lead to violations;
- *Harmful*: indicates that the applet contains elements that may lead to violations.

To implement the baseline system, we first extract all the channels of the IFTTT platform from the constructed dataset. Each channel belongs to one of the following categories:

- Smart Objects;
- Cloud Services;
- Social Networks.

We assign to each channel one of the following labels:

- *Private*: refers to channels that by default tend to privatize the information they manage, e.g., Google Drive by default allows users to privately store their files;

- *Public*: refers to channels that by default tend to publicly share the information they manage, e.g., Facebook by default allows other people to view the content that a user uploads on the platform.

The channels belonging to the "Smart Objects" or "Cloud Services" categories are categorized as operating in private contexts, while the "Social Networks" channels are categorized as operating in public contexts.

The process of applet labeling is performed as follows:

- If the trigger channel is labeled as "Private" and the action channel as "Public", then the applet is labeled as "Harmful";
- in all other cases the applet is labeled as "Harmless".

The reasoning behind such a choice is based on the assumption made in [138], where the authors state that *"it is safe to allow information to flow from a lower (public or trusted) label to a higher (private or untrusted) label, but not the other way around"*. Here, the terms higher and lower refer to the nodes of the secrecy lattice.

Since this methodology allows us to classify an applet as Harmful or not, it is necessary to modify the labels of our dataset in terms of binary classification. In particular, the previous applets of classes 1, 2, and 3 are modified with the new class *HF*, which corresponds to "Harmful", whereas the class 0 applets are labeled with the new class *HL*, corresponding to "Harmless". Thus, to compare the two classifiers, we train the BERT-3c model on the dataset with binary labels. We will refer to this model as BinaryBERT-3c.

For the evaluation phase, we use the test set TS_2k obtaining the results reported in Table 6.10. We can observe that IFC is not capable of accurately identifying Harmful applets as highlighted by the Recall value of class *HF* (20%) and the Precision value of class *HL* (41%). This is due to the prominent lack of contextual information available when IFC classifies an applet. In particular, a static approach that considers only the trigger and action channels provides a too high generalization of the applet's behavior. In fact, as highlighted by the literature, the classification of an applet with respect to different classes of damage can be strongly derived from the analysis of its semantic components [39, 130]. As an example, if we consider the classification with IFC of the applet introduced in Section 6.6.3, the trigger channel "Fitbit" and the action channel "Google Drive" would be labeled as *Private*, and the applet is classified as Harmless. Conversely, BinaryBERT-3c exploits NLP techniques that allow extracting the semantic meaning of the descriptions and understanding the applet's action context, which helps to disambiguate the rules that are difficult to classify by analyzing only the trigger and the action. In fact, as shown in Table 6.10, BinaryBERT-3c is capable of correctly classifying 92% of applets, as highlighted by the Accuracy value. In addition, high values for Precision, Recall, and F1-score in both classes provide a further hint about the ability of the model to classify them.

In conclusion, the evaluation of the results revealed that the BERT-3c classifier, which was fine-tuned considering all rule's features, achieved the highest Precision and Recall values, scoring an average of 88% for the TS_2k test set and 93% for the TS_425 test set. Furthermore, the BERT-3c classifier was compared to an information flow analysis-based approach on a binary classification task, which proved that our approach is the most suitable for classifying harmful ECA rules because it can extract the semantic and contextual meaning of the rules. This empirical evaluation has confirmed the capability of our classification model to highlight underlying security and privacy issues concealed within ECA rules. Therefore, this model could be promptly capable of supporting end-users, making them feel safer and at ease in using TAPs, contributing to providing a concrete response directed toward **RQ0**. It is worth mentioning, that a complete and concrete answer to the **RQ0** could only be achieved by means of a user study we are already planning to conduct, where we will assess the impact of such a solution on end-user perceptions.

7 EXPLAINING RISKS RELATED TO BEHAVIORAL RULES

The work presented in the previous chapter allows for the identification, with respect to 3 different risk classes, of vulnerabilities that can be introduced within smart environments following the activation of certain ECA rules.

However, the results produced by classification models can be difficult to understand without expert knowledge. Thus, it is fundamental to provide end-users with valid explanations describing the risks connected to a rule in a comprehensible manner, with the aim of enhancing the end-users trust. Such a task can be faced by employing the *Explainable Security* (XSec) paradigm, which is inspired by the *eXplainable Artificial Intelligence* (XAI) research field [74]. The latter deals with elucidating, in total transparency, the reason behind the outputs of a classification model.

In this chapter, we will present an approach for automatically generating natural language justifications, end-users understandable, related to Security and privacy harms caused by ECA rules.

7.1 LITERATURE REVIEW

In this section, we revise the literature concerning solutions for producing users' understandable explanations of AI-based systems. We first discuss popular agnostic approaches for explaining AI-model's results, then we review solutions tailored to providing explanations of security and privacy-related issues. Finally, we discuss works presenting explanations using natural language.

7.1.1 *Explainable Security*

The term Explainable Security (XSec) has been first presented as a novel paradigm, an extension of XAI, in [147]. It includes those security research proposals focused on bridging the gap between the *actual security* identified and the *perceived security*. In fact, with respect to explanations focused on acquiring trust, i.e., *explanation-for-trust*, which entails unveiling the black-box nature of models by explaining how a system works, the XSec research field focuses more on providing *explanation-for-confidence*, in order to make the user feeling confident in using the system, but without opening the black box [120].

A first approach for explaining security policy requirements has been proposed in [20]. The authors have designed a model for explaining why, due to a lack of permissions required for accessing data, queries might fail. Instead of only rejecting an unauthorized query, the explainable security model illustrates why the query failed and which permissions are required in order for the request to be granted.

In [116], the authors have proposed IoTSan, a model-checking-based approach for identifying inconsistencies within IoT Systems with respect to a set of safety properties. Downstream the model-checking process, the authors included a module called "Output Analyzer" which performs the task of analyzing and verifying the logs, classifying a violation either as a misconfiguration or behavior caused by a malicious app. The output of such an analysis is then presented to the user in the form of suggestions to either remove the app or change its configuration.

The approach presented in [162] is capable of detecting implicit rule interferences, a scenario where two or more rules are triggered simultaneously, causing contradictory effects for the environment. To allow the user to understand the nature of the interference problem and its cause, the authors generate explanations consisting of four parts, i.e., the scope of the actuators, the effect of the actuators on the environment, the list of functionalities it provides, and the type of interference produced. With a similar purpose, the authors in [153] proposed ProvThings, a platform-centric approach for performing an automated analysis of IoT apps and device APIs leading to the identification of faulty relationships between them. The authors aim at providing holistic behavioral explanations through a simplified front-end, specifically designed for the typical user, with not many computer skills.

In [132], the authors propose a real-time intrusion detection and mitigation system aimed at providing autonomous security in the IoT networks through flow analysis. The approach also aims at achieving a high degree of interpretability thanks to the employment of the Random Forest Machine Learning algorithm as a classifier.

Finally, in [151], the authors defined a SHAP-based framework that, alongside the execution of intrusion detection systems (IDs) mechanisms, combines local and global explanations to improve the interpretability of any IDs. The ultimate goal is both to improve understanding of the predictions made by the IDS and to build cyber users' trust in the IDSs themselves.

7.1.2 *Explainable AI with Natural Language Explanations*

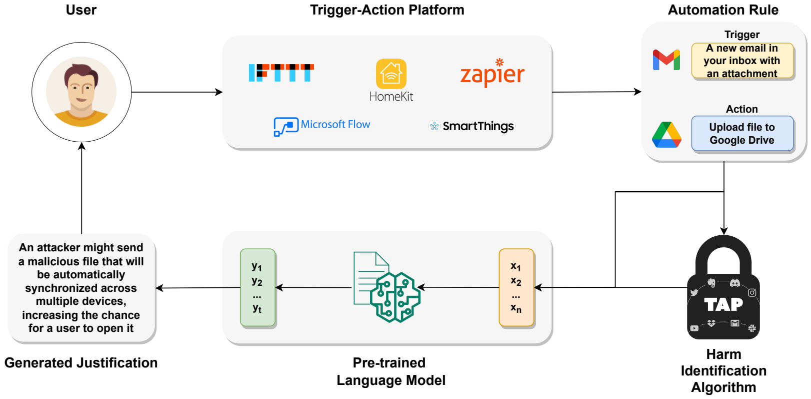
The generation of explanations using natural language has gained increasing interest in the field of research, particularly in the generation of personalized recommendations [45, 93, 114, 170]. In fact, solutions falling in this research topic try to enforce recommendation systems by supporting their output with interpretable justifications of the items. Providing justifications using natural language sentences is ideal for this task both for the considerable availability of textual data in form of reviews and for the advancement in natural language generation techniques [93].

In [170], the authors introduced an approach for providing human-like justification for a song recommendation system to improve users' awareness. With the purpose of training a language model to generate motivations related to the recommendation of a specific song, the authors generated a training set by crawling the comments associated with 1.2M songs posted on a music streaming platform, collecting a total of 80M samples with which they trained the language model. Then, by combining information related to the user's reasons, such as his or her mood or job position, with the details of the recommended song, the proposed solution is able to generate a justification for choosing it.

Musto et al. present a framework for building post hoc natural language justifications providing textual motivation to a recommendation algorithm [114]. The proposed approach stands as a black box solution that can generate justifications on any topic, starting from a recommendation and a set of revisions to generate post hoc justifications in natural language. To this end, the authors have developed three different implementations, progressively more complex, for solving the problem. The first uses sentiment analysis and natural language processing techniques to identify relevantly and distinguishing features highlighted in the reviews, combining them into a natural language justification of the suggested article. The second methodology builds on the first by including more advanced machine learning techniques, such as automatic feature extraction and text summarization. It uses these techniques to produce a distinctive summary that encapsulates the primary qualities of the item. Finally, the last approach addresses the issue of creating context-aware justifications by automatically learning a vocabulary for each context and utilizing that vocabulary to diversify the justifications based on the various contextual circumstances.

7.2 METHODOLOGY

In this section, we present the methodology for generating *post-hoc natural language justifications* that describe why an automation rule might cause specific harm. In particular, we define the problem and explain how to adapt pre-trained language models in producing contextual sentences that support the decisions of algorithms that identify harmful rules.



An attacker might send a malicious file that will be automatically synchronized across multiple devices, increasing the chance for a user to open it

Figure 7.1: Workflow for generating justifications concerning harmful rules

7.2.1 *Problem Formulation and System Overview*

In the natural language generation task considered, we assume that a user has defined an automation rule r using a TAP and that an algorithm has identified a harm h associated with r. The goal is to generate a justification j that justifies why h could arise from r. Here, language models maximize $P(j \mid r, h)$, i.e., the probability of producing a justification j given the information of a rule r and the harm h identified by the algorithm.

It is worth noting that our proposal is *algorithm-independent*. Indeed, the harm could be identified by an AI-based classifier [23], a system that performs information-flow analysis [118] or applies model checking [32].

The methodology requires a dataset $D = (R, H, J)$ consisting of (r, h, j) triplets employed to guide language models in constructing comprehensible justifications. Clearly, defining a justification dataset involves high human effort, so it may be challenging to have a large dataset for training an effective language model. However, thanks to the latest achievements in the field of AI, it is possible to obtain effective models even having a small amount of data. For example, pre-trained models make it possible to reuse the knowledge learned from a specific purpose to a problem in a related context. Many studies use pre-trained models on millions of data and specialize them in downstream tasks, achieving cutting-edge results [156]. Therefore, we suggest using pre-trained language models to generate high-quality justifications in the TAP domain, exploiting their knowledge gained from training on large amounts of raw textual data.

Figure 7.1 shows an example of the workflow for generating a justification concerning a harmful rule. Initially, information about the rule, i.e., triggers, actions, and services, is exploited by an algorithm to identify potential harm. Then, the rule information and classification result feed into a pre-trained language model that tries to connect them with the target justification: "*An attacker might send a malicious file that will be automatically synchronized across multiple devices, increasing the chance for a user to open it*".

In this chapter, we propose three strategies to implement pre-trained language models for justification generation in the TAP domain. First, we consider an approach based on a *Keywords to Sentences model* that requires a parser for extracting keywords from rules' information. Then, inspired by recent advancements in *prompt learning* [98], we introduce two strategies that leverage prompt-based finetuning. Here, instead of modifying the structure of pre-trained language models, we try to adapt them to our task by adding a prompt. In particular, the first strategy is based on *discrete prompt learning* as natural language words are added to every training and test example. Instead, the second strategy focuses on *continuous prompt learning* as we embed some rule information into continuous vectors by means of a customized version of the *Skip-Gram neural network model* [110, 111].

7.2.2 *Justifications based on Keywords to Sentences Model*

Keyword-based sentence generation can be viewed as a machine translation task [143]. Specifically, given the source sentence $x = \{x_1, ..., x_s\}$, its respective translation into the target language is determined by finding the best sentence $y = \{y_1, ..., y_t\}$. Assuming the application of a pre-trained language model that consists of pre-trained parameters Θ, we can formally write the goal as

$$y_{best} = \arg\max_{y} P(y \mid x, \Theta) \tag{7.1}$$

where the sentence y that maximizes the conditional probability $P(y \mid x, \Theta)$ is calculated.

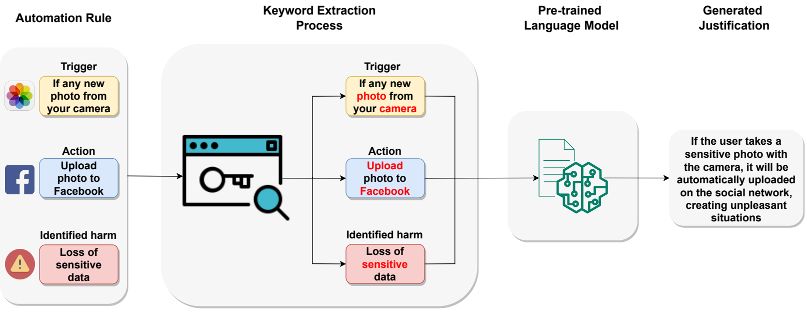

Automation Rule
Keyword Extraction Process
Pre-trained Language Model
Generated Justification
Trigger
If any new photo from your camera
Action
Upload photo to Facebook
Identified harm
Loss of sensitive data
Trigger
If any new photo from your camera
Action
Upload photo to Facebook
Identified harm
Loss of sensitive data
If the user takes a sensitive photo with the camera, it will be automatically uploaded on the social network, creating unpleasant situations

Figure 7.2: Workflow for generating justifications using keywords

In our context, we need to replace the source sentence x with r and h, i.e., the information of a rule and associated harm, respectively, and the target translated text y with j, i.e., the justification. Furthermore, we assume that the pre-trained language model does not receive a complete set of words to generate natural language justifications but a set of keywords, and learns how to provide the complementary information needed. If we denote K as a set of keywords in the TAP domain, we can express Eq. (7.1) as

$$j_{best} = \arg\max_{j} P(j \mid K, \Theta)$$

This equation is the same as Eq. (7.1) when a source text sentence is replaced with a set of keywords. Therefore, the task can be regarded as a model that translates keywords into text sentences.

As one would expect, such an approach requires a parser capable of inferring relevant words from sentences that describe rule triggers, actions, and associated harm. Fig. 7.2 depicts the process yielding the generation of a justification. Given a rule r and the potential harm h, we extract keywords from the rule triggers (TK_r), rule actions (AK_r), and rule harm (HK_h), respectively. Moreover, we take service names (e.g., Facebook and IoS Photos) used to define triggers (TS_r) and actions (AS_r) as additional keywords. To avoid redundant information in selected keywords, we consider two groups: the intersection of the five sets $I_{r,h} = TK_r \cap AK_r \cap HK_h \cap TS_r \cap AS_r$ and the remaining keywords $F_{r,h} = (TK_r \cup AK_r \cup HK_h \cup TS_r \cup AS_r)/I_{r,h}$. Thus, the final set is defined as:

$$K_{r,h} = [I_{r,h}, F_{r,h}]$$

During training, the model learns to correlate the combination of input keywords to the actual text. Instead, during inference, the model attempts to generate the output justification keeping the set of keywords in mind.

Ideally, the greater the number of keywords employed, the better the understanding of the model and the chances that it will generate a well-structured and coherent justification. This is because adding more keywords helps narrow the scope of knowledge and possibilities where these keywords occur in the training data distribution. However, if most of the keywords used represent noise, the model may consider irrelevant contexts and generate incorrect justifications. At the same time, it is plausible to think that if we provide only one keyword, the chance of the model generating a correct justification is quite low because the keyword might occur in several contexts in the training data, creating confusion.

7.2.3 *Justifications based on Prompt Learning*

Recently, prompt learning is becoming increasingly popular in training pre-trained language models because it enables promising results in scenarios with few or no labeled data [98]. The key idea is to push pre-trained language models in the right direction to solve an end-task by encoding training and test examples with a prompt. In this manner, we can exploit the prior knowledge exhibited in the model and manipulate its behavior to produce the desired output.

The first step is to apply a prompting function f to each example x of the dataset. Specifically, f adds some information to x and an unfilled slot *[Y]* that the model has to predict. The result is a prompt x' that feeds a pre-trained language model. Here, given pre-trained parameters Θ, the model maximizes $P(y \mid x', \Theta)$, i.e., the probability that an answer y should fill the slot *[Y]*.

Although its ease to use, this paradigm introduces a new challenge: finding the most appropriate prompt to solve the end-task. In particular, there are two different types of prompts: *discrete prompts* (a.k.a *hard prompts*) and *continuous prompts* (a.k.a *soft prompts*). The first type is to define a textual string so that the prompt consists of real words of the same type as those on which pre-trained language models were trained. This approach does not add new parameters and is known as *discrete prompt learning*. For example, we can recognize the sentiment of the sentence "*I like this product.*" by adding a prompt such as "*It is very [Y]*" and asking the pre-trained language model to predict a word to fill the blank. Finally, the model's prediction is mapped into a sentiment, e.g., positive or negative. However, since the goal is to find a prompt that enables pre-trained language models to perform a task, it is not necessary to limit the prompt to natural language interpretable by humans. In this regard, we can consider the definition of a soft prompt, which consists of a prompt composed of virtual tokens (e.g., represented by numeric ids) embedded in a continuous space [91, 94, 99]. This approach, known as *continuous prompt learning*, allows us not to use only natural language words in a prompt and not to parameterize it with pre-trained parameters. In fact, the prompt has additional learnable parameters that can be tuned on the training examples. Finally, there is also the opportunity to combine previous approaches by inserting tunable embeddings into a hard prompt. This approach is known as *hard-soft prompt hybrid tuning* [75, 98, 99].

As can be seen, we can design prompts for justification generation in the TAP domain. In this section, we describe our prompting approaches that leverage both discrete and continuous prompt learning. In particular, we discuss a *span-infilling* prompt, a *question-answering* prompt, and a prompt designed by combining *hard and soft* prompts. The latter approach applies a non-substantial amendment to the input, replacing the pre-trained language model embeddings of some rule information with embeddings inferred using a customized version of the Skip-Gram neural network model.

7.2.3.1 *Discrete Prompt Learning*

Traditionally, given a pre-trained language model, its embedding layer e is employed to embed the input token sequence $x = \{x_1, ..., x_n\}$ into embeddings $\{e(x_1), ..., e(x_n)\}$. In a downstream task, x is conditioned by a specific context c, which influences the model's behavior in producing the target sentence j. Instead, in discrete prompt learning, we use a prompting function that arranges c, j, and a prompt p into a template T. For example, in our case, the additional natural language tokens represent the prompt, the rule information and associated harm constitute the context, and the justification is the target sentence. Therefore, assuming that p_i refers to the i^{th} prompt token in a template T, we consider $T = \{p_1, ..., p_m, x, j\}$ and map it into $\{e(p_1), ..., e(p_m), e(x), e(j)\}$.

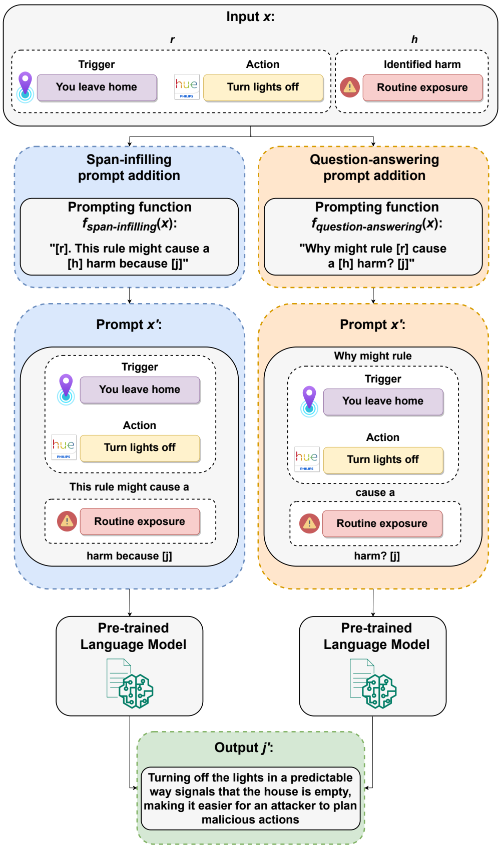

Input *x*:
r
h
Trigger
You leave home
Action
Turn lights off
Identified harm
Routine exposure
Span-infilling prompt addition
Prompting function $f_{span\text{-}infilling}(x)$:
"[r]. This rule might cause a [h] harm because [j]"
Question-answering prompt addition
Prompting function $f_{question\text{-}answering}(x)$:
"Why might rule [r] cause a [h] harm? [j]"
Prompt *x'*:
Trigger
You leave home
Action
Turn lights off
This rule might cause a
Routine exposure
harm because [j]
Prompt *x'*:
Why might rule
Trigger
You leave home
Action
Turn lights off
cause a
Routine exposure
harm? [j]
Pre-trained Language Model
Pre-trained Language Model
Output *j'*:
Turning off the lights in a predictable way signals that the house is empty, making it easier for an attacker to plan malicious actions

Figure 7.3: Workflow for generating justifications using hard prompts

In this section, we approach justification generation with prompt-based finetuning using hard prompts. The goal is to encode the rule information and the associated harm and apply manually-designed prompts described in discrete space (i.e., natural language sentences) to directly model pre-trained language models text probability. Specifically, we propose two different prompts where the input is placed entirely before the slot to be filled, named *prefix prompts* [98].

The most natural way to create prompts is to analyze the task under consideration and formulate an intuitive template. Therefore, we propose a prefix prompt with a format relatable to what a human would expect in a justification for an identified harm. In particular, it is a simple span consisting of a natural language tokens sequence placed after the encoded rule information:

```
[r]. This rule might cause a [h] harm because [j]
```

On the other hand, it has been proven that using the question-answering format can be very useful in transferring linguistic or other knowledge learned from one task to a related task [69]. In this regard, we propose a question-answering prompt on the assumption that: *it is possible to redefine a non-question-answering task into an equivalent question-answering form* [104]. Again, we deal with a prefix prompt since it is necessary to continue a string prefix, but the encoded rule information are placed in the middle of the prompt. Here, the prompting function receives as input the encoded rule r and harm h, and outputs

```
Why might rule [r] cause a [h] harm? [j]
```

Figure 7.3 shows an example of the workflow for generating a justification using the span-infilling/question-answering prompt. In detail, given the rule information r and the associated harm h, we first apply the prompting function to transform the input example x in x'. Then, x' is fed into a pre-trained language model, which has to determine an appropriate justification to replace *[j]*.

In both cases, prompts are fixed, and training and test examples share the same prompt. The hypothesis is that the additional natural language tokens could be beneficial because they force the model to compute similarities between the input text and the words in the human-written templates, suggesting the subset of the vocabulary that provides better information in generating justifications.

7.2.3.2 *Continuous Prompt Learning*

In the previous section, we introduced a way to generate justifications for harmful rules using hard prompts. However, upon studying the characteristics of the rules more carefully, we found that this approach has weaknesses. Specifically, using discrete tokens to represent service names could result in the loss of semantic information. In fact, services are fundamental components in the TAP domain, as they allow us to determine a rule's context of execution. For example, if a rule uses Philips Hue as a service for trigger or action, it is clear that the rule acts on a physical device. Conversely, if it uses Facebook, we can easily deduce that software operations will be performed.

Pre-trained models map words in contextualized representations by leveraging the attention mechanism [146]. The latter assigns a different amount of weight or "attention" to the elements in a sequence, producing representations that encode some contextual information. Clearly, this solution is much more efficient than a static word embeddings approach because it can generate different representations for words with which different meanings can be associated. For example, the word "mouse" may refer to a small rodent or a pointing device if the words "bluetooth" or "wireless" are close to it. Here, a static approach assigns the same vector to both situations even though the word mouse has a different meaning. However, in the TAP domain could be challenging to identify the functionality of a service by analyzing its context, as there may be words that obscure it. For example, whether the word "post", recalling a social media post, falls within the context of the Philips Hue service, it may be considered in the definition of the service's embedding, adversely affecting the inference of the rule's context of execution. In fact, we require that Philips Hue is represented by using words such as "light" and "bulb" since it enables effective management of smart bulbs. Therefore, if we consider services as discrete tokens, we may have hoped for efficient mapping, but this is in no way guaranteed since language models are not trained to encode their similarity.

The key idea is to have embeddings for services such that if two services expose the same functionality (e.g., camera or light services), they must have similar embeddings. In this way, in the inference phase, we can guide the model in deducing the rule's context of execution as the services with which a rule is defined might be similar to others, enabling a more effective selection of tokens to use in constructing the justification related to the identified harm. In this regard, we can use soft prompts in which services are mapped as continuous word embeddings rather than discrete tokens. Thus, unlike the previous solution, we introduce additional prompt parameters, i.e., a set of embeddings for services made available by a TAP, which are inserted into the embedded input and optimized during the model training.

Training soft prompts requires addressing two challenges [99]: 1) they cannot be initialized randomly and optimized by stochastic gradient descent because sub-optimal results might be obtained [2], and 2) their values should be dependent on each other. In our context, we handle the dependence among services according to the functionalities they expose and derive their embeddings by leveraging the architecture of the Skip-Gram model.

Skip-Gram is part of the *Word2vec* [110] family consisting of neural network model architectures that can learn word embeddings of a vocabulary, ensuring that similar words have similar representations. The key idea is a word can be represented according to its context, i.e., the words preceding and following it in documents. Thus, words with similar contexts should have close meanings.

From a general perspective, the model is trained to solve a subtask by using word embeddings as input. During training, the model tries to improve the subtask accuracy by performing several iterations and backpropagation steps. Meanwhile, the embeddings are modeled and become ever more accurate. Finally, model predictions are discarded, and only optimized embeddings are considered.

The subtask of the Skip-Gram model is to predict the context words of a given target word. To this end, we need to model the Skip-Gram architecture as a deep learning classification model capable of predicting the context words. As is well known, classification model training needs a suitable dataset. In particular, we need to build (*target*, *context words*) pairs since we have multiple words within the context of a word. They can be easily collected by analyzing the available documents. In fact, given a context window, i.e., the number of words we want to consider that precede and follow the target word, the pairs can be automatically built by looking at each unique word as a target word and examining its context. In addition, we can simplify the task by breaking down each (*target*, *context words*) pair into several (*target*, *context word*) pairs in order to have only one word at a time in a context. However, as is well known, we need a labeled dataset to train a classification model. Specifically, we need to train the model to discriminate if a word is contextual concerning a target word or not. In this regard, we feed the Skip-Gram model with pairs (x, y) where x is a (*target*, *context word*) pair and y is the label. The latter assumes the value 1 if the context word occurs near the target word in the text, indicating a contextually relevant pair named *positive input sample*. On the other hand, contextually irrelevant pairs named *negative input samples* (to which is assigned the label 0) are built by considering (*target*, *random*) pairs where target is the word of interest and random is just a randomly selected word from the vocabulary.

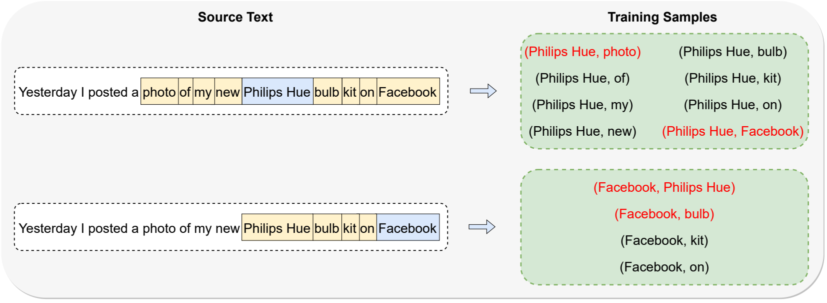


Figure 7.4: Example of erroneous positive input samples (highlighted in red) generated through Skip-Gram

In our context, the vocabulary contains the service names, and the goal is to get their embeddings. As already anticipated, we want to model the similarities of services according to their functionalities. Therefore, we cannot build a labeled dataset performing previously described steps. In fact, if a service occurs in the context of (and thus it is near in the text to) a target service, they may not necessarily expose a similar functionality, yielding incorrect samples. Fig. 7.4 shows cases of mistaken generation. In particular, using a context window size of 4 and the source text "*Yesterday I posted a photo of my new Philips Hue bulb kit on Facebook*", we get the words "photo", "of", "my", "new", "bulb", "kit", "on" and "Facebook" as the context of Philips Hue. For example, the pairs (*Philips Hue*, *Facebook*) and (*Philips Hue*, *photo*) are labeled with the value 1, even though Facebook is a social network and Philips Hue is a solution for organizing, controlling, and customizing lights. Likewise, using Facebook as the target service, we get the words "Philips Hue", "bulb", "kit", and "on" in its context, generating the pair (*Facebook*, *bulb*) as a positive input sample even though Facebook is not in the business of managing bulbs.

Again, the key idea is a service can be represented according to its context, but we need to slightly change the definition of "*context*". In particular, we consider a service contextual with respect to a target service if it exposes a similar functionality. Thus, services with similar contexts should have close embeddings. Precisely in order to avoid erroneous training samples, we consider only pairs characterized by services. Indeed, as shown, it is challenging to automatically infer correct and relevant words from documents (e.g., post for Facebook and bulb for Philips Hue). Likewise, we cannot manually define such words because the number of services might be very high, which would be time-consuming. Also, it is conceivable that different platforms expose different services, requiring a custom set of words for each platform. To alleviate these problems, we propose a novel automated solution to define increasingly reliable positive and negative input samples according to a threshold.

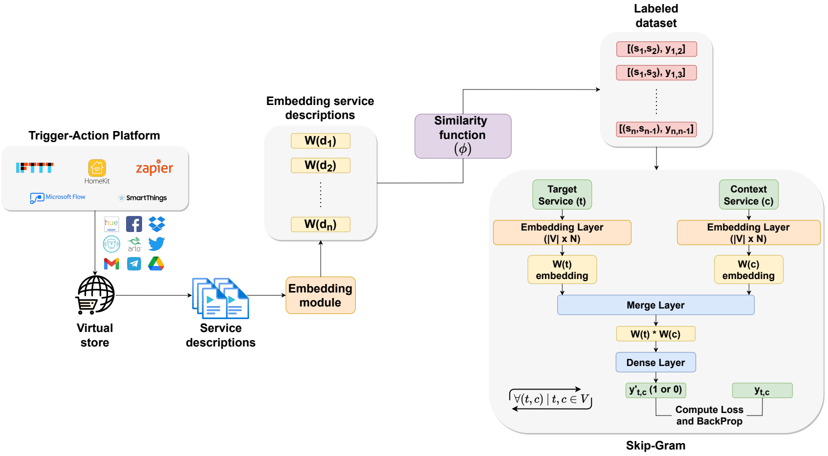


Figure 7.5: Workflow for computing the embedding of services

As shown in Figure 7.5, the architecture involves four components: a virtual store, an embedding module, a similarity function paired with a threshold, and the Skip-Gram mechanism. The concept of the methodology is that services with similar functionality should have similar descriptions on the virtual stores that enable their download (e.g., Google Play Store and App Store). Specifically, given the service set exhibited by a TAP, we fix a virtual store and crawl the service descriptions d_1, d_2, ..., d_n. The latter are embedded with an embedding module, yielding vector representations $W(d_1)$, $W(d_2)$, ..., $W(d_n)$ that summarize the semantic information of services' functioning. Finally, given a threshold $\phi \in [-1,1]$, we compare each embedding pair ($W(d_i)$, $W(d_j)$) with a similarity function and assign the label 1 to the service pair (s_i, s_j) if the similarity between their embedding descriptions exceeds ϕ, 0 otherwise. In this way, the greater the fixed threshold, the more reliable the generated pairs are. However, as one might expect, a high threshold does not always lead to good results because services with similar functionality may have slightly different descriptions, resulting in false negative input samples. In addition, there is no need to set a context window size because the value of ϕ already affects the services' context size. In fact, the smaller it is, the greater the number of pairs labeled with 1, and so the number of services that fall within the context of a target service, and vice versa. Therefore, it is necessary to fine-tune ϕ depending on the case and available information.

In the last step, we need to build the deep learning architecture for the Skip-Gram model. Once again, the subtask consists of predicting the context words (services) of a target word (service) while modeling the vector representation of the latter. In particular, we want to learn an N-dimensional word embedding for $|V|$ services, where V is the vocabulary. It is worth noting that N cannot be fixed arbitrarily but is dictated by the constraints of the pre-trained language models. On the other hand, V contains services of a specific TAP. The model's input is pair of input words consisting of one input target service and one context service. These pairs go by an embedding layer (initialized with random weights) having size $(|V| \times N)$ that produces a dense word embedding $(1 \times N)$ for both services. Then, a merge layer computes the dot product of the two embeddings and sends the result to a dense layer. The latter predicts either a 1 or 0, depending on whether or not the service pair is contextually relevant. We want to maximize the likelihood of the positive input samples, such as (Facebook, Twitter), and minimize the likelihood of the negative input samples, like (Facebook, Philips

Hue). To this end, we leverage the sigmoid function intending to maximize

$$P(y = 1 \mid t, c, \Theta) = \sigma(v_c^T v_t) = \frac{1}{1 + e^{(-v_c^T v_t)}}$$

where Θ is the model parameter, t is a target service, c is a context service, and $(t, c) \in P$ is a positive input sample. Meanwhile, we need to maximize

$$P(y = 0 \mid t, c, \Theta) = 1 - P(y = 1 \mid t, c, \Theta) = \frac{1}{1 + e^{(v_c^T v_t)}}$$

when a negative input sample $(t, c) \in N$ is considered. Formally,

$$\Theta = \arg\max_{\Theta} \prod_{(t,c) \in P} P(y = 1 \mid t, c, \Theta) \cdot \prod_{(t,c) \in N} P(y = 0 \mid t, c, \Theta)$$

Finally, we compare the prediction y' with the actual label y, compute the loss by leveraging the mean squared error, and backpropagate the errors to adjust the weights in the embedding layer. This process is repeated cyclically on all (target service, context service) pairs for multiple epochs, yielding a set of service embeddings $S \in \mathbb{R}^{|V| x N}$. In this way, the model can learn the contextually relevant service pairs and generate similar embeddings for services with similar functionalities. Moreover, in the inference phase, we only need the output embeddings and can discard the model.

Although it is advisable to map services with additional embeddings inferred through Skip-Gram, we cannot use them as the only inputs in the prompt due to the fact that rules with the same services as trigger and action might have different behaviors. For example, let us consider the following two rules: "`New tweet by you, turn off lights`" and "`New tweet by anyone in the area, turn on lights`". Both rules can be defined using *Twitter* as a trigger service and *Philips Hue* as an action service. However, as is easy to guess, they have different behaviors, exposing a user to different harms. In fact, the former might be triggered unintentionally by the user causing the lights to go off. Instead, the second might be compromised by an attacker to damage lights. Therefore, we still need to use rule information and the related harm in the prompt to better understand the behavior of a rule. In this regard, we insert the Skip-Gram representations within the hard prompts presented in the previous section, yielding hard-soft prompts that allow us to grasp both the rule's context of execution and the rule behavior and generate a technical justification consistent with the harm involved.

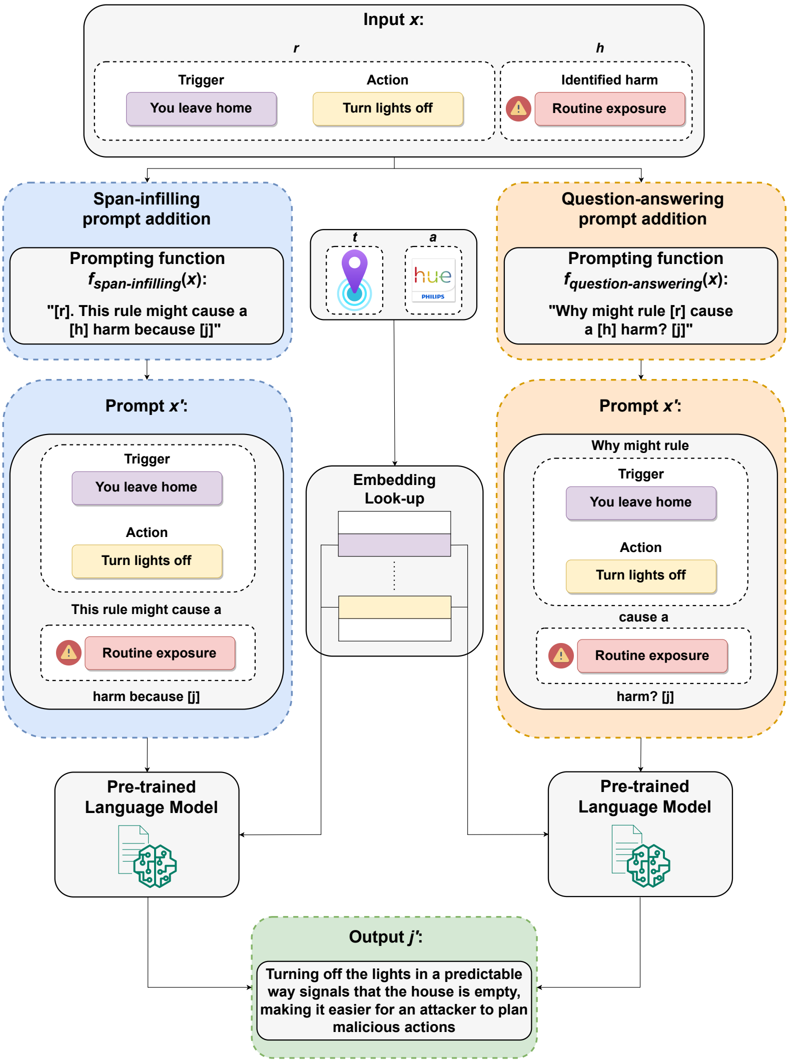

Input *x*:
r
h
Trigger
You leave home
Action
Turn lights off
Identified harm
Routine exposure
Span-infilling prompt addition
Prompting function $f_{span\text{-}infilling}(x)$:
"[r]. This rule might cause a [h] harm because [j]"
t
a
hue
PHILIPS
Question-answering prompt addition
Prompting function $f_{question\text{-}answering}(x)$:
"Why might rule [r] cause a [h] harm? [j]"
Prompt *x'*:
Trigger
You leave home
Action
Turn lights off
This rule might cause a
Routine exposure
harm because [j]
Embedding Look-up
Prompt *x'*:
Why might rule
Trigger
You leave home
Action
Turn lights off
cause a
Routine exposure
harm? [j]
Pre-trained Language Model
Pre-trained Language Model
Output *j'*:
Turning off the lights in a predictable way signals that the house is empty, making it easier for an attacker to plan malicious actions

Figure 7.6: Workflow for generating justifications using hard-soft prompts

Conceptually, assuming that p_i refers to the i^{th} prompt token, t is the rule trigger service, a is the rule action service, x represents the remaining rule information and the identified harm, and j relates to the target justification, the template can be represented as $T = \{p_1, ..., p_m, t, a, x, j\}$. As shown in Figure 7.6, we regard t and a as pseudo tokens and map T into $\{e(p_1), ..., e(p_m), s(t), s(a), e(x), e(j)\}$, where $s(t)$ and $s(a)$ denote the Skip-Gram representations retrieved from S.

7.2.3.3 *Training Strategies*

As opposed to the traditional approach, where we only consider the parameters of a pre-trained model, in prompt learning-based tasks, we have to consider also prompts as additional parameters. Hence, the choice of what parameters to update is an essential part of the design process, as it may affect the performance of a model. In this section, we summarize the main strategies for tuning pre-trained language models, highlighting those that fit best for our proposals.

In particular, existing training strategies differ in the parameters to be considered for the update [98]. For example, *Promptless Fine-Tuning* refers to the strategy that performs pre-training and fine-tuning without the involvement of prompts, so we do not consider it. Instead, *Tuning-free Prompting* enables the generation of texts by leveraging frozen pre-trained language models based on a prompt. It does not require parameter updates and is usually applied in zero-shot settings. Thus, we omit it as our proposals include the training on a justification dataset. *Fixed-prompt LM Tuning* tunes the parameters of pre-trained language models and usually applies a discrete template to training and test examples. Accordingly, it can be used with our discrete prompt learning-based solution. In fact, as presented in Section 7.2.3.1, both in the span infilling and question-answering prompts, we consider the input text, i.e., rule information, identified harm, and prompt tokens, as discrete tokens to be employed for feeding the pre-trained language models. Finally, *Fixed-LM Prompt Tuning* and *Prompt+LM Fine-tuning* strategies include prompt parameters updating, and thus they can be used with our continuous prompt learning-based solution. Specifically, the former updates only the prompt parameters while freezing those of the pre-trained language models, whereas the second updates both the prompt parameters and all or some of the parameters of the pre-trained language models. In the proposed solution, we initially encode prompts, i.e., rule services, through a revised version of Skip-Gram. Successively, the soft prompts are further optimized during the training of the pre-trained language model.

7.3 EXPERIMENTAL EVALUATION

In this section, we illustrate the experiments performed to evaluate the performance of the proposed solutions, including the user studies conducted to collect and examine a justification dataset in the TAP domain. In particular, we consider both human-based and automatic metrics. The former allows for analyzing the quality of the dataset, whereas the latter enable us to determine the readability of model-generated justifications and compare them against the gold human-written justifications. Finally, we detail how pre-trained language models are customized for the generation task.

7.3.1 *Evaluation Metrics*

Here, we discuss two automatic metrics commonly used in natural language generation tasks that have proven to have a high correlation with human ratings, namely the *Bilingual Evaluation Understudy with Representations from Transformers* (BLEURT) and the *Automatic NT Translation Metric* (METEOR) [38, 58]. For last, we consider the *Coleman-Liau index* readability test to further investigate the understandability of the generated justifications.

7.3.1.1 *Automatic Evaluation*

This section describes the automatic metrics selected in this study. In particular, we regard a word-overlap metric, i.e., METEOR [14], an embedding-based metric, i.e., BLEURT [134], and a grammar-based metric, i.e., the Coleman-Liau index [40]. The first two metrics compare a generated text (candidate) with a *"gold standard"* (reference), i.e., a natural language text annotated by humans as a correct solution for a given task. In general, the higher the score is, the more similar to the reference the output is. On the other hand, the Coleman-Liau index does not consider references but focuses on a fundamental property of texts, readability.

- *METEOR:* It relies on the harmonic mean of unigram precision and recall. More specifically, it tries to find the largest subset of words that can form an alignment between the reference and a candidate. To this end, it computes word-to-word matches and considers Porter stemming and synonyms from the WordNet dictionary when there is no exact match. Finally, the result is given to a function that penalizes candidates containing correct words but in the wrong order.
- *BLEURT:* It is a learned evaluation metric based on *Bidirectional Encoder Representations from Transformers* (BERT) [51] capable of capturing non-trivial semantic similarities between sentences. In particular, it is trained on the WMT Metrics Shared Task dataset[1] and uses the contextual word representations of BERT. Furthermore, in order to avoid the domain drift problem, it also takes into account additional training data generated by applying random perturbations to Wikipedia sentences.

We use METEOR and BLEURT as they try to identify whether two sentences convey the same meaning even though they are phrased differently. In fact, in the TAP domain, the harm associated with a rule might be expressed in different ways, resulting in a low score when a generated justification is correct but syntactically different with respect to the reference. For example, a well-known metric such as BLUE, which relies solely on lexical match analysis, unfairly rewards candidates who resemble the references but do not capture their meaning and penalizes other paraphrases. Instead, METEOR considers important additional features such as stemming and synonym matching. Moreover, BLUE only considers precision, while researchers have shown that using both precision and recall leads to a higher correlation with human judgments [14]. Likewise, an evaluation of sentence

[1] http://www.statmt.org/wmt20/metrics-task.html

meanings using embeddings, such as that performed by BLEURT, could capture some relevant characteristics of justifications because words are dynamically represented by considering the words around them.

Another important aspect we analyze is the complexity of the generated texts. In fact, as the major beneficiaries of TAPs are users with little technical knowledge, either in the areas of security and privacy [152] or in the context of IoT technology [171], justifications should describe scenarios in which harm may occur but not provide too many low-level technical details that might confuse a user. For this reason, we evaluate the readability of justifications, i.e., the difficulty with which a reader understands a text.

- *Coleman-Liau Index:* It is a readability assessment test designed to approximate the U.S. grade level needed to understand a text (from straightforward texts with an index below 1 to academically challenging texts with an index above 16). In particular, the test is standardized for English texts and uses sentences and letters as variables. As an example, a score of 8 corresponds to the 8th grade in the U.S. educational system.

7.3.2 *Dataset Collection*

We evaluated the proposed methodologies using rules created through the IFTTT platform, which has been extensively studied in the literature due to its popularity [39, 138, 145]. In the following, we describe the process of creating the justification dataset for our case study.

7.3.2.1 *IFTTT Dataset*

The dataset on which we performed our study was initially collected in 2017 by Mi *et al.* through a scraping process [109]. The dataset comprised over 320,000 rules (also known as applets) gathered over a six-month period, with each applet consisting of (*i*) a title (*Title*), (*ii*) a description of its behavior (*Desc*), (*iii*) the trigger component divided into its trigger (*TriggerTitle*) and relative channel (*TriggerChannelTitle*), and (*iv*) the action component divided into its action (*ActionTitle*) and relative channel (*ActionChannelTitle*). To train and evaluate our language models, we considered a subset of 525 applets randomly selected that were classified as harmful by our harm identification algorithm proposed in [23]. In particular, the algorithm can classify an applet according to three classes of harm, namely *Personal harm*, identifying the applets that may result in the loss or compromise of sensitive data, which is solely due to the user's behavior, *Physical harm*, which refers to all the applets that may cause physical harm or damage to goods, and the harm is external, i.e., inflicted by third parties, and *Cybersecurity harm*, encompassing applets that may disrupt online services or distribute malware, and the harm is external as well. The subset of applets comprised 183 applets for the Personal harm class, 153 applets for the Physical harm class, and 189 applets for the Cybersecurity harm class.

7.3.2.2 *Gathering justifications*

The 525 applets underwent annotation by 13 independent annotators, who provided a justification as a real-life threat scenario for each of the harmful rules assigned to them. The procedure followed to collect these justifications has been organized as follows:

- A group of 10 annotators, comprised of both master's and bachelor's degree computer science students, provided justifications for 400 applets randomly selected. We assessed the grammatical correctness of the justifications, manually correcting any misspellings or missing punctuation.
- We annotated the remaining 125 applets.

7.3.3 *Implementation details*

In the following, we provide the implementation details of the proposed Continuous prompt learning-based strategy and the two baseline models, namely the discrete prompt learning-based strategy, and the Keyword to Sentences model. All of them rely on pre-trained language models based on Transformer [146]. The dataset of applets was split into three sets: a training set with 435 applets, a validation set with 30 applets, and a test set with 60 applets.

7.3.3.1 *Keyword to Sentences strategy*

The objective is to generate justifications through the use of a pre-trained language model that is fed with keywords obtained from applet information and the associated harms. To this end, we adopted the method outlined in [72], which introduced a keyword extraction technique leveraging DistilRoBERTa embeddings [131]. The first step involves creating a set of keyword candidates by selecting all n-grams from texts. For the purpose of this study, we considered n values between 1 and 2 in order to capture short and concise keywords, and we extracted keyword candidates from three sources: the applet behavior description, the applet's title, and a text in the format "IF *TriggerTitle* THEN *ActionTitle*", which is a reference to the traditional rule format on the IFTTT platform. After generating a list of n-grams, we selected the keyword candidates that accurately reflect the semantics of the text by comparing their embeddings using cosine similarity. The candidates were sorted based on their similarity scores and the top-m candidates were selected. Once the list of n-grams that fall within the specified range is produced for each text, we filter the keywords that accurately capture the semantics of the text. Here, the intuition is that the embeddings of the best keywords live close to the embedding of the main text into the same latent embedding space. Therefore, keyword extraction is reduced to applying a similarity function between the main text embedding and candidate keywords embeddings. Concerning this matter, we use the embeddings of the DistilRoBERTa model to represent texts and compare them by applying the cosine similarity. The final set of keywords is obtained by combining the keywords extracted from D, TI, and SD, as well as the identified harm and the channel names, and removing duplicates, as shown in the following: Finally, we can sort candidates based on the similarity scores and select the top n candidates closest to a text. We combine the keywords extracted from D, TI, and SD, consider the identified harm and the channel names as additional keywords, and then remove duplicates, yielding the final set of keywords

$$\{k_{D_1}, k_{D_2}, ..., k_{D_M}, k_{TI_1}, k_{TI_2}, ..., k_{TI_P}, k_{SD_1}, k_{SD_2}, ..., k_{SD_Q}, H, T_C, A_C\}$$

The final set of keywords is obtained by concatenating the keywords extracted from D, TI, and SD, as well as the identified harm and the channel names, and removing duplicates.

Ideally, the greater the number of keywords employed for training the better the model understands the task and generates well-structured and coherent justifications. This is because adding more keywords helps narrow the scope of knowledge and possibilities where these keywords occur in the training data distribution. However, if most of the keywords represent noise, the model may consider irrelevant contexts and generate incorrect justifications. For this reason, we conducted several experiments using $m = \{4, 5, 6, 7, 8\}$ as the number of keywords extracted from each text.

Transformer models have proven to be effective in tasks such as machine translation and summarization by incorporating both encoder and decoder components [124, 146, 166]. We, therefore, chose the *Text-To-Text Transfer Transformer* (T5) model [123] as the backbone for our keyword-based justification generation task. T5 trains by masking words in sentences with a special token and learns to predict them. This technique, known as *corrupting span denoising* objective, enables the model to produce a sequence as an output. We used the base version of T5, with 11 billion parameters trained on diverse internet text and fine-tuned for various NLP tasks such as text summarization and text-to-text generation.

To assist the model in determining the specific task it should perform, we added a task-specific text prefix to the original input that is fed to the model. In this case, we used the text *"justification generation:"* as the prefix. This allows the model to focus its attention on our target task, restricting its generation scope. Training data were obtained by combining keywords through the " | " symbol. Additionally, in accordance with the pre-trained T5 format, we included the </s> token and added it to both the input and output text. The training process utilized the *Fixed-prompt LM Tuning* strategy, treating the keywords as discrete tokens and using a discrete template for applets. We adopted the *cross-entropy* as a loss function, the *Adafactor* algorithm as an optimizer, a *learning rate* of $1e-2$, 30 *epochs*, and a *batch size* of 10. During generation, the keywords were concatenated using the "|" symbol, and the model produced the output justification through greedy decoding.

7.3.3.2 *Continuous prompt learning-based strategy*

As described in Section 7.2.3.2, the prompt strategy relies on pre-trained language models to extend a prefix prompt and generate a justification. To accomplish this, we adopted the GPT-2 model [122], which only incorporates the decoder component of the Transformer architecture and represents a given token through the *masked self-attention* mechanism. During training, the model's aim is to predict one word at each step, based on all previous words, and append it to the input sequence, producing a modified sequence that feeds the model in the next step. Thus, the left-to-right nature of GPT-2, combined with its ability to finish a sequence by iteratively predicting the next most likely word, makes it a suitable choice for the proposed span-infilling and question-answering hard prompts.

We used the small version of GPT-2, consisting of 12 transformer blocks, 117 million parameters, and an embedding size of 768.

We applied hard-soft prompts by augmenting applets with the span-infilling or question-answering prompt, considering the features *Title*, *Desc*, *TriggerTitle*, *ActionTitle*, and *Harm* as discrete tokens and treating the *TriggerChannelTitle* and *ActionChannelTitle* features (i.e., the channel used for the trigger and the action, respectively) as continuous tokens. The latter were computed by applying the Channel2Vec embedding strategy. We extracted the list of IFTTT channels to be mapped into embeddings from the dataset we proposed in [23]. It contains a variety of categories, ranging from online services and content providers to smart home devices, for a total of 435 channels. Channel descriptions were crawled on the Google Play Store app using the SELENIUM Python module. Sentence embeddings of descriptions were calculated by SentenceBERT [126], whereas we used the cosine similarity to compare the embeddings. To study the effect of having more or fewer channels in the channel context of a target channel, we conducted several experiments using ϕ = {0.7, 0.75, 0.8, 0.85, 0.9} as a threshold for constructing the labeled pairs (target channel, context channel). The size of embeddings in GPT-2 is 768, so we built 768-dimensional representations. We leveraged the TensorFlow Python library to build the deep learning architecture for the Skip-Gram model and trained it on the labeled pairs for 50 *epochs* using a *batch size* of 80. Once the Skip-Gram model was trained, we extracted channel embeddings from the embedding layer and trained GPT-2 using the *Prompt+LM Fine-tuning* strategy. Therefore, we initially encoded rules' channels through Channel2Vec and then fine-tuned soft prompts together with all GPT-2 parameters during training. We adopted the *negative log-likelihood* as a loss function, the *AdamW* algorithm as an optimizer, a *batch size* of 128, a *learning rate* of 1e-3, and stopped the process when the loss did not decrease for 25 times. It is important to mention that we placed the channel embeddings as the initial parameters in the input to influence the representation processing of all tokens associated with the features of an applet. Moreover, we did not freeze the GPT-2 parameters because it has been shown that the gap between prompt-tuning and fine-tuning disappears only when the model size climbs to 10 billion parameters [91]. Thus, we did not apply the *Fixed-LM Prompt Tuning* strategy.

Finally, we used the greedy decoding method and incorporated two special tokens, <*bos*> (begin-of-sequence token) and <*eos*> (end-of-sequence token), into the vocabulary. At each step, we selected the word with the highest probability as the prediction, and appended it to the sequence to create the new input sequence for the following step. This process continued until the <*eos*> token was produced or the loss did not decrease for 25 consecutive times.

7.3.3.3 *Discrete prompt learning-based strategy*

To generate justifications, this model considers the information of training and testing applets as discrete tokens and exploits a hard prompt. Thus, we employed the GPT-2 model and conducted training via the *Fixed-prompt LM Tuning* strategy. Specifically, for both span-infilling and question-answering prompts, we considered the input text, i.e., applet information, identified harm, and prompt tokens, as discrete tokens to be employed for feeding the pre-trained language model. We adopted the *negative log-likelihood* as a loss function, the *AdamW* algorithm as an optimizer, a *batch size* of 35, a *learning rate* of $1e-3$, greedy decoding, and stopped the process when the loss failed to decrease for 25 consecutive times.

7.3.4 *Results and analysis*

In this section, we will thoroughly assess the effectiveness of the justification generation methods outlined previously. This includes an examination of whether certain harms are more challenging to justify than others. Furthermore, we will perform a qualitative analysis of the justifications generated for three potentially harmful rules, in order to evaluate the impact of our continuous prompt learning-based strategy.

7.3.4.1 *Quantitative Analysis on Justifications*

Tables 7.1-7.3 report the values of BLEURT, METEOR, and Coleman-Liau index obtained by the different strategies on the test set. As previously stated, the first two metrics assess the Explainability of the generated justifications, measuring how well they align with the gold standards. Thus, high values are better. Conversely, the Coleman-Liau index evaluates the Readability of the generated justifications, determining their level of difficulty to read. In this case, low values are desirable as they indicate that the generated justifications are easily understandable by a wider audience. We apply the Coleman-Liau index to each generated justification and calculate several statistics (i.e., median, mean, maximum, and minimum) to assess the overall performance of the models, as well as to identify cases where the Readability of the justifications is particularly high or low.

From Tables 7.1 and 7.2, we can observe that the T5 model performs worse, in terms of Explainability, than the GPT-2 model trained with discrete prompt learning, when provided a list of

Table 7.1: Discrete Prompt Learning by T5 - Fixed-prompt LM Tuning

	Explainability		Readability			
n	**BLEURT**	**METEOR**	**Median**	**Mean**	**Max**	**Min**
4	23	55	**10.4**	**11.01**	**14.75**	7.03
5	24	56	12.69	12.15	16.71	7.2
6	**27**	**57**	12.47	12.19	17.23	8.88
7	25	56	11.81	11.94	16.94	**6.21**
8	24	55	11.41	11.57	15.55	6.51

Table 7.2: Discrete Prompt Learning by GPT-2 - Fixed-prompt LM Tuning

	Explainability		Readability			
Prompt	**BLEURT**	**METEOR**	**Median**	**Mean**	**Max**	**Min**
Span-infilling	33	60	10.51	10.92	15.44	**5.53**
Question-answering	**34**	**61**	**10.35**	**10.83**	**15.33**	7.72

keywords as input. This discrepancy can be attributed to the parser employed to extract the keywords. Specifically, compared to the other strategies, errors may occur that are not due to the model's poor capabilities in generating high-quality justifications but rather to the presence of irrelevant input keyword sets. Additionally, it is worth noting that the number of keywords extracted has a significant impact on the model's performance. In particular, when a low number of keywords is used, the model cannot fully understand the behavior of a rule, resulting in the inability to generate a correct justification. On the other hand, when the number of keywords considered is excessive, noise is introduced, which negatively affects the generation process. Concerning this matter, we achieve the best results when considering $n = 6$ (27% and 57% for BLEURT and METEOR, respectively), which allows for the appropriate balance between relevant keywords to be considered and those that represent noise to be discarded. However, the best results in terms of Readability are obtained with $n = 4$. Through manual analysis of the generated justifications, we notice that these values are dictated by the fact that the model generates meaningless justifications composed of a few repeating words. We can affirm that the Coleman-Liau index values

Table 7.3: Continuous Prompt Learning by GPT-2 - Prompt+LM Finetuning

		Explainability		Readability			
Prompt	ϕ	**BLEURT**	**METEOR**	**Median**	**Mean**	**Max**	**Min**
Span-infilling	0.7	36	61	10.51	11.23	16.31	8.71
	0.75	39	63	10.51	10.91	15.79	7.61
	0.8	**42**	**68**	**10.35**	**10.71**	**15.04**	**6.51**
	0.85	39	63	10.86	11.03	15.85	7.43
	0.9	36	62	10.71	11.77	16.44	8.26
Question-answering	0.7	38	64	10.51	11.32	16.31	8.65
	0.75	41	67	10.51	11.09	**15.04**	8.72
	0.8	39	64	10.51	10.87	15.33	7.55
	0.85	40	67	10.51	11.26	16.71	6.68
	0.9	36	63	10.51	11.40	16.94	8.89

become relevant only when the values of BLEURT and METEOR are relatively high. Thus, we do not consider Readability results in this case.

The GPT-2 model trained on training samples that are provided with a hard prompt achieves better results, for both span-infilling and question-answering prompts, in comparison to those obtained with the T5 model. In particular, there is a marked improvement in the quality of the generated justifications, highlighted by higher values of BLEURT and METEOR. This is because adding a prompt to the training samples enables the model to be steered in the right direction, as the prompt provides additional semantic knowledge that the model can exploit to more effectively understand the relationship between the behavior of a rule and the associated harm. In addition, the justifications exhibit a satisfactory level of Readability, as evidenced by the values of the median (10.51 and 10.35, corresponding to the 11th and 10th grades in the U.S. educational system, respectively) and the mean (10.92 and 10.83, corresponding to the 11th grade in the U.S. educational system). This means that, on average, the justifications are understandable by a user with a secondary school education grade. In both cases, two justifications are generated that correctly justify the corresponding harmful rule but are characterized by many words and cyclic periods (highlighted by Max values). At the same time, when analyzing the Min values, it is worth noting that the model is also capable of generating justifications that are extremely easy to read.

As the utilization of GPT-2 with hard prompts has yielded satisfactory results, we also employ it with our strategy based on continuous prompt learning, in which service names are mapped via the Skip-Gram model. In fact, as evidenced by manual analysis, mapping service names to discrete tokens often leads to the generation of justifications where the rule context of execution is totally distorted. For this reason, we combine hard and soft prompts and employ GPT-2 for the generation task. Table 7.3 shows the results obtained from training GPT-2 with the Prompt+LM Fine-Tuning training strategy. It is noticeable that the GPT-2 model consistently achieves superior results when compared to the utilization of discrete tokens for mapping service names. In particular, the best results are achieved when using the span-infilling prompt and a threshold ϕ of 0.8. In general, we can observe that ϕ deeply affects the performance of the model, as it establishes the service context size of a service when its embedding is modeled by Skip-Gram. In fact, when a low threshold is used, many services providing unrelated functionalities may be included within the service context of a target service, negatively affecting the definition of embeddings and, consequently, the identification of the rule context of execution. By increasing the threshold, we can discard more and more services and focus attention solely on those that are truly relevant. However, when using thresholds that are too high, the number of identifiable related services turns out to be very low, making the task of service functionalities similarity mapping to the Skip-Gram model challenging. Looking at Table 7.3, we can see that the worst results are obtained when 0.7 and 0.9 are used as thresholds, which reflects the aforementioned intuition. Furthermore, it is worth noting that the choice of threshold not only affects Explainability but also Readability. Indeed, when we employ a very low (0.7) or very high (0.9) threshold, satisfactory values can still be obtained for the BLEURT and METEOR metrics, but the generated justifications are more complex. This is due to the fact that when a low (high, resp.) threshold is used and thus a high (low, resp.) number of services is included in the service contexts, the model cannot promptly identify the rule's context of execution and needs to generate more words to obtain a correct justification. Instead, using more reasonable thresholds (e.g., 0.75, 0.8, and 0.85) allows for a balance between correctly and improperly included services in the service contexts, enabling more efficient service mapping and quicker identification of the rule context of execution, resulting in more terse justifications.

7.3.4.2 *Performances on Applets with Unknown Channels*

To further demonstrate the superiority of our continuous prompt learning-based strategy, we present and compare justifications generated by the GPT-2 model using both discrete and continuous prompt learning for three new harmful rules, each representing a different class of harm. In particular, we analyze the scenario where a rule is defined through services not present within the training set employed to train the model. Table 7.4 shows the characteristics, associated harm, and justifications generated by the corresponding models for each rule. Discrete - QA and Discrete - SI represents the GPT-2 model trained on samples augmented by the question-answering and span-infilling prompts, respectively. Conversely, Continuous - QA/0.75 and Continuous - SI/0.8 depict the GPT-2 model trained using one of the hard prompts and the Skip-Gram-based process with a specific threshold. We consider these configurations for the model as they provided the best results in terms of Explainability and Readability on the test set. Red sentences denote events that do not reflect the behavior of the corresponding rule, while green sentences indicate consistent descriptions. Blue words for the TriggerChannelTitle and ActionChannelTitle features highlight services not present within the training set.

We can observe that the models based on our continuous strategy accurately identify discriminating characteristics of rules and provide reasoning behind the associated harm. Even when faced with new services, models are able to exploit similarities with other services exhibiting same functionalities, effectively capturing the rule context of execution and thus generating a correct justification. Furthermore, it is worth noting that such models have the capability to generate both more general justifications, such as mentioning "user's device", as well as more specific ones. For example, the Continuous - SI/0.8 model can deduce that the second harmful rule, associated with the Physical class, could lead to unexpected changes in temperature if an attacker sends malicious emails. Similarly, this model can also infer that the third harmful rule, associated with the Cybersecurity class, could cause spam emails to be sent to the "user's mailbox" rather than simply stating "user's device" as the Continuous - QA/0.75 model does. This also highlights the slight superiority in terms of Explainability achieved by the Continuous - SI/0.8 model compared to the Continuous - QA/0.75 model.

Regarding models that map service names to discrete tokens (i.e., Discrete - QA and Discrete - SI), we can observe that they consistently fail to generate a completely correct justification, committing errors in either the event that triggers the rule or the final action that leads to the harm, or both. This echoes their poor abilities to disambiguate the rule context of execution. In reality, these models can only accurately capture a rule's behavior in a few cases. For example, the Discrete - QA model correctly detects that harm could be inflicted if the user posts on the social network in the first harmful rule. However, it incorrectly states that the potential harm refers to an attacker learning the user's location. Similarly, in the second harmful rule, both the Discrete - QA and Discrete - SI models recognize that an attacker could inflict harm by sending malicious emails to the user. However, they fail to infer the right harm to be reported to the user. Such findings highlight a significant issue with these models: they are unable to comprehend the context of execution when handling a service that they have not seen during the training phase. In fact, they can only generate a correct event when it is related to a service they are familiar with, such as Facebook and Gmail. Conversely, they provide inconsistent descriptions when encountering new services such as Yeelight, Nest Thermostat, and Ooma.

7.3.5 *Discussion*

The ultimate goal of this study led to the definition of models for generation natural language explanations with the aim of raising users' awareness regarding how harmful some rules can be if defined improperly. Our post-hoc approach for justification gener-

Table 7.4: Justifications generated for three harmful rules, one for each class of harm considered. Blue words highlight services not included in the training set. Green sentences indicate cases where the execution context of the trigger (action, resp.) is correctly identified, while red sentences show cases where a wrong event is generated.

Personal Harm	*TriggerChannelTitle*: **Facebook**, *ActionChannelTitle*: **Yeelight**, *TriggerTitle*: **New status message by you with hashtag**, *ActionTitle*: **Turn on lights**, *Title*: **Turn on lights with a message**, *Desc*: **If new status message with #light, turn on lights**
Continuous - QA/0.75	The state of the user's device could also be altered when the user does not want to if he or she posts on social networks with the specific hashtag.
Discrete - QA	An attacker could learn the user's location posted on the social network to plan malicious actions.
Continuous - SI/0.8	If the user forgets to have defined the rule and publishes a social network post with the chosen hashtag, the state of the user's device would unintentionally change.
Discrete - SI	An attacker could like many posts on the social network to allow unwanted notifications to be sent to the user's device.
Physical Harm	*TriggerChannelTitle*: **Gmail**, *ActionChannelTitle*: **Nest Thermostat**, *TriggerTitle*: **New email in inbox matches search**, *ActionTitle*: **Set temperature**, *Title*: **Change my nest temperature**, *Desc*: **Setup a feed that controls temperature and have an email turn it on!**
Continuous - QA/0.75	An attacker could send emails in such a way as to change the state of the user's device.
Discrete - QA	An attacker could send malicious emails so as to automatically send spam SMS or advertisements.
Continuous - SI/0.8	An attacker could send malicious emails so as to change the temperature.
Discrete - SI	An attacker could send emails with malicious links to allow them to be set as wallpaper on the user's device.
Cybersecurity Harm	*TriggerChannelTitle*: **Ooma**, *ActionChannelTitle*: **Gmail**, *TriggerTitle*: **Any phone call missed**, *ActionTitle*: **Send an email**, *Title*: **Email me when I miss a call**, *Desc*: **Send yourself an email if you miss a call**
Continuous - QA/0.75	An attacker could make multiple calls to allow spam emails to be sent to the user's device.
Discrete - QA	An attacker could send multiple SMS to the user causing the lights to flash extremely, damaging them.
Continuous - SI/0.8	An attacker could make repeated calls to allow spam emails to be sent to the user's mailbox.
Discrete - SI	An attacker could send an email to allow the publications of tasks on work organization platforms.

ation is independent of the algorithm used for harm classification and exploits a combination of hard and soft prompts derived from the components that characterize an automation rule. We found that the justifications generated using the proposed models are highly readable and are deemed to accurately describe the security risks. Furthermore, the CLI values demonstrated that the justifications produced by this model can be easily understood by users with a secondary school education. Although the evaluation using automatic metrics yielded more than satisfactory results, these are still merely indicative assessments. In fact, the metrics considered express results in terms of correlation with a human judgment, reasoning that although the possible range of values for BLEURT and METEOR varies between 0 and 1, at the sentence level a value of 0.403 is on average recognized in the literature as more than valid [14]. This recalibrates the results obtained from our experimental evaluations by enhancing the importance and validity of the scores. However, a final indication could be provided by studies carried out on users to assess not only the readability but also the plausibility of the motivations generated, with reference to both the rules and the highlighted risk.

In conclusion, net of the results obtained so far, the strategies adopted for generating natural language justifications have proven to be largely capable of highlighting real-life risk scenarios related to the activation of risky ECA rules. This could be an important added value for end users, who would be supported not only in identifying risks but also in explaining risk scenarios that emphasize their severity. This provides an important contribution to the **RQ0**.

Part III

CONCLUSION

8 CONCLUSION AND FUTURE WORK

This section presents the conclusion and the future direction of this thesis.

THESIS SUMMARY In this thesis, we formalized the following research question as the guiding thread for our contributions:

> **RQ0.** "Can we support end-users in dealing with security and privacy concerns within smart environments?"

In seeking for providing an answer, we presented solutions that are focused on end-users, with the goal of safeguarding both privacy and overall security in the increasingly popular Trigger-Action Platforms. In particular, a review of the state of the art highlighted the need for solutions that support end-users in achieving a more secure smart ecosystem that reflects their needs, Thus, in this thesis, we discussed the tasks and challenges associated with TAPs, as well as the security and privacy risks that can arise from TAPs and the rules that can be created through them.

Among these challenges, we first examined the primary research papers in the literature that analyze IFTTT, the most popular and used TAP. These studies delved into the platform's proposed creation paradigm and the types of rules created by end-users, highlighting several security and privacy risks caused by a lack of end-user oversight during rule design. As the most recent study on IFTTT was conducted in 2017, we conducted an empirical analysis to evaluate the platform's evolution over the past five years. To do so, we retrieved over 70,000 applets from the website through a custom web-scraping process and conducted a preliminary analysis that revealed a significant increase in both the number of rules and the platform's user base.

Among the findings that emerged during our empirical analysis was an important percentage of rules whose title and description, i.e., those fields to which the author is given full freedom by the IFTTT platform, did not match the actual behavior of the rule. To address this issue, we proposed a model using Google BERT to accurately determine the semantic consistency between the rule description and its actual behavior, taking into account the static parameters (trigger and action) that users must enter when creating the rule.

Next, we presented a solution that allows end users to set rules based on security events. To accomplish this, we integrated a custom-made IoT device called *Intrusion Defender* (ID) into a trigger-action platform. The ID is able to detect unusual network activity that may indicate a cybersecurity attack. The events captured by the ID are presented to the end user in a simplified way, making it easy for them to create a rule that responds to a specific event, such as a Denial-of-Service attack. A user study involving 20 participants showed that using embedded metaphors makes it possible for inexperienced users to set rules for their smart environment just as effectively as someone with more experience in cybersecurity.

Creating trigger-action rules through various TAPs can sometimes lead to the definition of behaviors that put both the security of the smart environment and the privacy of end users at risk. To address this issue, we proposed a model for identifying different types of harm associated with trigger-action rules. The model, which is based on BERT and trained on a dataset of over 80,000 rules defined through the IFTTT platform, is able to classify different types of risks by semantically analyzing the various components of a rule. The model can identify three different risk classes: personal harm, physical harm, and cybersecurity harm, with an average accuracy of 92%. Comparison with other solutions proposed in literature, such as information flow analysis and recurrent neural network architecture, revealed that our BERT-based model, which takes into account semantic properties, is better able to distinguish between different types of risks.

Finally, we have linked a model for identifying risks in trigger-action rules with an explainability component. This component generates a scenario, called "justifications", that shows how the rule's activation could lead to harm. We use prompt-based fine-tuning to generate these justifications, testing both hard and soft prompts. Soft prompts, which use virtual tokens embedded in a continuous space, produced more consistent results. This approach allows the model to understand the context in which the rule is used, as similar rules can be grouped together, leading to more accurate justifications for identified risks. We assessed the effectiveness of the suggested solutions by utilizing a widely used automatic metric in natural language generation tasks, known as BLEURT. This metric has been shown to have a strong correlation with human evaluations. Our evaluations revealed a relatively high BLEURT score, with an average score of 0.412 for the QA Prompt and 0.416 for the Span Prompt for the test set under consideration.

In conclusion, we can affirm that the three major contributions introduced in this thesis work could result relevant to the research question, providing valuable support to end-users in protecting their smart environment from security and privacy threats which could both derive from external malicious individuals or could be introduced by the end-users themselves without realizing.

LIMITATIONS As detailed several times throughout this paper, while the work conducted has provided important insights into solutions for supporting and protecting end users from security and privacy issues in IoT scenarios, the work lavished still has some limitations. Primarily, part of the user studies that could and should have been carried out was undermined by the outbreak of the COVID-19 pandemic, which prevented the involvement of end-users in controlled environments that could simulate the smart environments we went to work on by defining the approaches presented in this thesis. In addition, the approaches for identifying the risks associated with rule-making, and for generating real-life threat scenarios are based on specific models whose rationale is mainly related to the use of English. This represents a major limitation on the inclusivity front, in light of the fact that the user group that is the ultimate target of our solutions, i.e., people of all age groups and with widely varying knowledge, may be unlikely to be able to understand the English language.

FUTURE WORK In the future, further evaluation of the impact the proposed solutions have in supporting the end users will play a critical role. In particular, we would like to assess the capabilities of the discussed approaches in protecting the security of the smart environment and user privacy. Additionally, an area that has yet to receive adequate attention is access control mechanisms from the end user's perspective. Currently, there are no solutions that address this issue. Therefore, significant effort will need to be invested in developing automated or streamlined approaches for end users to set access rules for IoT devices and services in the smart environment.

BIBLIOGRAPHY

[1] Mohannad Alhanahnah, Clay Stevens, and Hamid Bagheri. “Scalable analysis of interaction threats in iot systems.” In: *Proceedings of the 29th ACM SIGSOFT international symposium on software testing and analysis*. 2020, pp. 272–285.

[2] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. “A convergence theory for deep learning via over-parameterization.” In: *Proceedings of International Conference on Machine Learning*. PMLR. 2019, pp. 242–252.

[3] Abdulmajeed Alqhatani and Heather Richter Lipford. ““There is nothing that I need to keep secret”: Sharing Practices and Concerns of Wearable Fitness Data.” In: *Proceedings of Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*. 2019, pp. 421–434.

[4] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. “Understanding the Mirai botnet.” In: *26th USENIX security symposium (USENIX Security 17)*. 2017, pp. 1093–1110.

[5] *Apiant Inc. 2020. We wired web.* URL: https://wewiredweb.com/..

[6] Carmelo Ardito, Paolo Buono, Giuseppe Desolda, and Maristella Matera. “From smart objects to smart experiences: An end-user development approach.” In: *International Journal of Human-Computer Studies* 114 (2018), pp. 51–68.

[7] Carmelo Ardito, Giuseppe Desolda, Rosa Lanzilotti, Alessio Malizia, and Maristella Matera. “Analysing trade-offs in frameworks for the design of smart environments.” In: *Behaviour & Information Technology* 39.1 (2020), pp. 47–71.

[8] Carmelo Ardito, Giuseppe Desolda, Rosa Lanzilotti, Alessio Malizia, Maristella Matera, Paolo Buono, and Antonio Piccinno. “User-defined semantics for the design of IoT systems enabling smart interactive experiences.” In: *Personal and Ubiquitous Computing* 24.6 (2020), pp. 781–796.

[9] *Atooma (2020). Atooma mobile app.* URL: https://www.atooma.com/.

[10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. “The internet of things: A survey.” In: *Computer networks* 54.15 (2010), pp. 2787–2805.

[11] Amos Azaria, Jayant Krishnamurthy, and Tom M Mitchell. “Instructable intelligent personal agent.” In: *Proceedings of Thirtieth AAAI conference on artificial intelligence*. 2016.

[12] K. Balachandran. *kardSort*. https://kardsort.com/. [Online; accessed 14-March-2021].

[13] Fabrizio Balducci, Paolo Buono, Giuseppe Desolda, Donato Impedovo, and Antonio Piccinno. “Improving smart interactive experiences in cultural heritage through pattern recognition techniques.” In: *Pattern Recognition Letters* 131 (2020), pp. 142–149.

[14] Satanjeev Banerjee and Alon Lavie. “METEOR: An automatic metric for MT evaluation with improved correlation with human judgments.” In: *Proceedings of the ACL Workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*. 2005, pp. 65–72.

[15] Aaron Bangor, P Kortum, and JA Miller. “The system usability scale (SUS): An empirical evaluation.” In: *International Journal of Human-Computer Interaction* 24.6 (2008), pp. 574–594.

[16] Barbara Rita Barricelli, Fabio Cassano, Daniela Fogli, and Antonio Piccinno. “End-user development, end-user programming and end-user software engineering: A systematic mapping study.” In: *Journal of Systems and Software* 149 (2019), pp. 101–137.

[17] Iulia Bastys, Musard Balliu, and Andrei Sabelfeld. "If this then what? Controlling flows in IoT apps." In: *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*. 2018, pp. 1102–1119.

[18] Chris Beckmann and Anind Dey. "Siteview: Tangibly programming active environments with predictive visualization." In: *adjunct Proceedings of UbiComp*. 2003, pp. 167–168.

[19] Andrea Bellucci, Paloma Díaz, Ignacio Aedo, and Alessio Malizia. "Prototyping device ecologies: physical to digital and viceversa." In: *Proceedings of the 8th international conference on tangible, embedded and embodied interaction*. 2014, pp. 373–376.

[20] Gabriel Bender, Lucja Kot, and Johannes Gehrke. "Explainable security for relational databases." In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 2014, pp. 1411–1422.

[21] Massimo Benerecetti, Fausto Giunchiglia, and Luciano Serafini. "Model checking multiagent systems." In: *Journal of Logic and Computation* 8.3 (1998), pp. 401–423.

[22] Will Brackenbury, Abhimanyu Deora, Jillian Ritchey, Jason Vallee, Weijia He, Guan Wang, Michael L Littman, and Blase Ur. "How users interpret bugs in trigger-action programming." In: *Proceedings of the 2019 CHI conference on human factors in computing systems*. 2019, pp. 1–12.

[23] Bernardo Breve, Gaetano Cimino, and Vincenzo Deufemia. "Identifying Security and Privacy Violation Rules in Trigger-Action IoT Platforms with NLP Models." In: *IEEE Internet of Things Journal* (2022).

[24] Bernardo Breve, Gaetano Cimino, and Vincenzo Deufemia. "Towards Explainable Security for ECA Rules." In: (2022).

[25] Bernardo Breve, Giuseppe Desolda, Vincenzo Deufemia, Francesco Greco, and Maristella Matera. "An end-user development approach to secure smart environments." In: *Proceedings of International Symposium on End-User Development*. Springer. 2021, pp. 36–52.

[26] Bernardo Breve and Vincenzo Deufemia. "Empowering End-Users in the Specification of Security Rules." In: *Proceedings of EMPATHY@ AVI*. 2020, pp. 53–56.

[27] John Brooke et al. "SUS-A quick and dirty usability scale." In: *Usability evaluation in industry* 189.194 (1996), pp. 4–7.

[28] Margaret Burnett, Todd Kulesza, Alannah Oleson, Shannon Ernst, Laura Beckwith, Jill Cao, William Jernigan, and Valentina Grigoreanu. "Toward Theory-Based End-User Software Engineering." In: *New Perspectives in End-User Development*. Springer, 2017, pp. 231–268.

[29] Danilo Caivano, Daniela Fogli, Rosa Lanzilotti, Antonio Piccinno, and Fabio Cassano. "Supporting end users to control their smart home: design implications from a literature review and an empirical investigation." In: *Journal of Systems and Software* 144 (2018), pp. 295–313.

[30] Roy R Cecil and Jorge Soares. "IBM Watson studio: a platform to transform data to intelligence." In: *Pharmaceutical Supply Chains-Medicines Shortages*. Springer, 2019, pp. 183–192.

[31] Z Berkay Celik, Leonardo Babun, Amit Kumar Sikder, Hidayet Aksu, Gang Tan, Patrick McDaniel, and A Selcuk Uluagac. "Sensitive Information Tracking in Commodity IoT." In: *Proc. 27th USENIX Security Symposium*. USENIX Association, 2018, pp. 1687–1704.

[32] Z Berkay Celik, Patrick McDaniel, and Gang Tan. "Soteria: Automated IoT Safety and Security Analysis." In: *Proc. USENIX Annual Technical Conference*. USENIX Association, 2018, pp. 147–158.

[33] Z Berkay Celik, Gang Tan, and Patrick D McDaniel. "IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT." In: *Proc. Annual Network and Distr. Syst. Sec. Symp.* 2019.

[34] Xuyang Chen, Xiaolu Zhang, Michael Elliot, Xiaoyin Wang, and Feng Wang. "Fix the leaking tap: A survey of Trigger-Action Programming (TAP) security issues, detection techniques and solutions." In: *Computers & Security* (2022), p. 102812.

[35] Yunang Chen, Amrita Roy Chowdhury, Ruizhe Wang, Andrei Sabelfeld, Rahul Chatterjee, and Earlence Fernandes. "Data Privacy in Trigger-Action Systems." In: *Proc. IEEE Symposium on Security and Privacy*. IEEE. 2021, pp. 501–518.

[36] Haotian Chi, Qiang Zeng, Xiaojiang Du, and Jiaping Yu. "Cross-app interference threats in smart homes: Categorization, detection and handling." In: *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE. 2020, pp. 411–423.

[37] Yu-Hsi Chiang, Hsu-Chun Hsiao, Chia-Mu Yu, and Tiffany Hyun-Jin Kim. "On the Privacy Risks of Compromised Trigger-Action Platforms." In: *Proc. 25th European Symposium on Research in Computer Security*. Guildford, United Kingdom: Springer-Verlag, 2020, 251–271. ISBN: 978-3-030-59012-3.

[38] Miruna Clinciu, Arash Eshghi, and Helen Hastie. "A study of automatic metrics for the evaluation of natural language explanations." In: *arXiv preprint arXiv:2103.08545* (2021).

[39] Camille Cobb, Milijana Surbatovich, Anna Kawakami, Mahmood Sharif, Lujo Bauer, Anupam Das, and Limin Jia. "How Risky Are Real Users' IFTTT Applets?" In: *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*. 2020, pp. 505–529.

[40] Meri Coleman and Ta Lin Liau. "A computer readability formula designed for machine scoring." In: *Journal of Applied Psychology* 60.2 (1975), p. 283.

[41] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. "A semantic web approach to simplifying trigger-action programming in the IoT." In: *Computer* 50.11 (2017), pp. 18–24.

[42] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. "A high-level semantic approach to end-user development in the Internet of Things." In: *Int. J. Hum. Comput.* 125 (2019), pp. 41–54.

[43] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. "HeyTAP: Bridging the Gaps Between Users' Needs and Technology in IF-THEN Rules via Conversation." In: *Proceedings of the International Conference on Advanced Visual Interfaces*. 2020, pp. 1–9.

[44] Miguel Coronado and Carlos A Iglesias. "Task automation services: automation for the masses." In: *IEEE Internet Computing* 20.1 (2015), pp. 52–58.

[45] Felipe Costa, Sixun Ouyang, Peter Dolog, and Aonghus Lawlor. "Automatic generation of natural language explanations." In: *Proceedings of the 23rd international conference on intelligent user interfaces companion*. 2018, pp. 1–2.

[46] Maria Francesca Costabile, Daniela Fogli, and Rosa Lanzilotti. "Supporting work practice through end-user development environments." In: *Journal of Organizational and End User Computing (JOEUC)* 18.4 (2006), pp. 43–65.

[47] Joëlle Coutaz and James L Crowley. "A first-person experience with end-user development for smart homes." In: *IEEE Pervasive Computing* 15.2 (2016), pp. 26–39.

[48] Danilo Croce, Giuseppe Castellucci, and Roberto Basili. "GAN-BERT: Generative Adversarial Learning for Robust Text Classification with a Bunch of Labeled Examples." In: *Proc. 58th Annual Meeting of the Association for Comp. Ling.* 2020, pp. 2114–2119.

[49] Edgar Dale and Jeanne S Chall. "The concept of readability." In: *Elementary English* 26.1 (1949), pp. 19–26.

[50] Giuseppe Desolda, Carmelo Ardito, and Maristella Matera. "Empowering end users to customize their smart environments: model, composition paradigms, and domain-specific tools." In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 24.2 (2017), pp. 1–52.

[51] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." In: *Proceedings of Conference of the North American Chapter of the ACL: Human Language Technologies, Volume 1*. 2019, pp. 4171–4186.

[52] Anind K Dey, Timothy Sohn, Sara Streng, and Justin Kodama. "iCAP: Interactive prototyping of context-aware applications." In: *Proceedings of International conference on pervasive computing*. Springer. 2006, pp. 254–271.

[53] NA Diamantidis, Dimitris Karlis, and Emmanouel A Giakoumakis. "Unsupervised stratification of cross-validation for accuracy estimation." In: *Artif. Intell.* 116.1-2 (2000), pp. 1–16.

[54] Wenbo Ding and Hongxin Hu. "On the safety of IoT device physical interaction control." In: *Proc. ACM Conf. on Comp. and Comm. Sec.* 2018, pp. 832–846.

[55] Paul Dourish, Rebecca E Grinter, Jessica Delgado De La Flor, and Melissa Joseph. "Security in the wild: user strategies for managing security as an everyday, practical problem." In: *Personal and Ubiquitous Computing* 8.6 (2004), pp. 391–401.

[56] William H DuBay. "The principles of readability." In: *Online Submission* (2004).

[57] Greg Eisenhauer, Matthew Wolf, Hasan Abbasi, and Karsten Schwan. "Event-based systems: Opportunities and challenges at exascale." In: *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*. 2009, pp. 1–10.

[58] Desmond Elliott and Frank Keller. "Comparing automatic evaluation measures for image description." In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 2014, pp. 452–457.

[59] Michael Fagan and Mohammad Maifi Hasan Khan. "Why do they do what they do?: A study of what motivates users to (not) follow computer security advice." In: *Proceedings of Twelfth symposium on usable privacy and security (SOUPS 2016)*. 2016, pp. 59–75.

[60] Jingwen Fan, Yi He, Bo Tang, Qi Li, and Ravi Sandhu. "Ruledger: Ensuring Execution Integrity in Trigger-Action IoT Platforms." In: *Proc. IEEE Conf. on Comp. Comm.* IEEE. 2021, pp. 1–10.

[61] S Federici, ML Mele, R Lanzilotti, G Desolda, M Bracalenti, G Gaudino, A Cocco, and M Amendola. "UTASSISTANT: a new semi-automatic usability evaluation tool for Italian public administrations." In: *Proceedings of the International Conference on Advanced Visual Interfaces-ECONA Workshop (AVI 2018)*. 2018, pp. 1–3.

[62] Earlence Fernandes, Justin Paupore, Amir Rahmati, Daniel Simionato, Mauro Conti, and Atul Prakash. "FlowFence: Practical Data Protection for Emerging IoT Application Frameworks." In: *Proc. 25th USENIX Security Symposium*. USENIX Association, 2016, pp. 531–548.

[63] Gerhard Fischer. "End-user development and meta-design: Foundations for cultures of participation." In: *International Symposium on End User Development*. Springer. 2009, pp. 3–14.

[64] Gerhard Fischer and Elisa Giaccardi. "Meta-design: A framework for the future of end-user development." In: *End user development*. Springer, 2006, pp. 427–457.

[65] Rudolf Flesch. "Flesch-Kincaid readability test." In: *Retrieved October* 26.3 (2007), p. 2007.

[66] Rudolph Flesch. "A new readability yardstick." In: *Journal of applied psychology* 32.3 (1948), p. 221.

[67] *GMBH, elastic.io.* URL: http://www.elastic.io/..

[68] William A Gale and Geoffrey Sampson. "Good-turing frequency estimation without tears." In: *Journal of quantitative linguistics* 2.3 (1995), pp. 217–237.

[69] Matt Gardner, Jonathan Berant, Hannaneh Hajishirzi, Alon Talmor, and Sewon Min. "Question Answering is a Format; When is it Useful?" In: *arXiv preprint arXiv:1909.11291* (2019).

[70] Sahar Ghannay, Benoit Favre, Yannick Esteve, and Nathalie Camelin. "Word embedding evaluation and combination." In: *Proc. Tenth Int. Conf. on Language Resources and Evaluation*. European Language Resources Association, 2016, pp. 300–305.

[71] Giuseppe Ghiani, Marco Manca, Fabio Paternò, and Carmen Santoro. "Personalization of context-dependent applications through trigger-action rules." In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 24.2 (2017), pp. 1–33.

[72] Maarten Grootendorst. "KeyBERT: Minimal keyword extraction with BERT." In: *Available:* `https://maartengr.github.io/KeyBERT/index.htm` (2020).

[73] Joshua B Gross and Mary Beth Rosson. "Looking for trouble: understanding end-user security management." In: *Proceedings of the 2007 Symposium on Computer Human interaction For the Management of information Technology*. 2007, 10–es.

[74] David Gunning, Mark Stefik, Jaesik Choi, Timothy Miller, Simone Stumpf, and Guang-Zhong Yang. "XAI—Explainable artificial intelligence." In: *Science Robotics* 4.37 (2019), eaay7120.

[75] Xu Han, Weilin Zhao, Ning Ding, Zhiyuan Liu, and Maosong Sun. "PTR: Prompt tuning with rules for text classification." In: *arXiv preprint arXiv:2105.11259* (2021).

[76] Sandra G Hart. "NASA-task load index (NASA-TLX); 20 years later." In: *Proceedings of the Human factors and ergonomics society annual meeting*. Vol. 50. 9. Sage publications Sage CA: Los Angeles, CA. 2006, pp. 904–908.

[77] Sandra G Hart and Lowell E Staveland. "Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research." In: *Advances in psychology*. Vol. 52. Elsevier, 1988, pp. 139–183.

[78] Weijia He, Maximilian Golla, Roshni Padhi, Jordan Ofek, Markus Dürmuth, Earlence Fernandes, and Blase Ur. "Rethinking Access Control and Authentication for the Home Internet of Things (IoT)." In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018, pp. 255–272.

[79] Amir Herzberg. "Why Johnny can't surf (safely)? Attacks and defenses for web users." In: *computers & security* 28.1-2 (2009), pp. 63–71.

[80] Kai-Hsiang Hsu, Yu-Hsi Chiang, and Hsu-Chun Hsiao. "SafeChain: Securing Trigger-Action Programming From Attack Chains." In: *IEEE Trans. Inf. Forensics Secur.* 14.10 (2019), pp. 2607–2622. DOI: 10.1109/TIFS.2019.2899758.

[81] Justin Huang and Maya Cakmak. "Supporting mental model accuracy in trigger-action programming." In: *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. 2015, pp. 215–225.

[82] Ting-Hao'Kenneth' Huang, Amos Azaria, Oscar J Romero, and Jeffrey P Bigham. "InstructableCrowd: Creating IF-THEN Rules for Smartphones via Conversations with the Crowd." In: *arXiv preprint arXiv:1909.05725* (2019).

[83] Iulia Ion, Rob Reeder, and Sunny Consolvo. ""No one Can Hack My Mind": Comparing Expert and Non-Expert Security Practices." In: *Proceedings of Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*. 2015, pp. 327–346.

[84] *JS_Foundation (2020). Node-RED.* URL: http://nodered.org/.

[85] Björn A Johnsson and Boris Magnusson. "Towards end-user development of graphical user interfaces for internet of things." In: *Future Gener. Comput. Syst.* 107 (2020), pp. 670–680.

[86] Mohammad Jafari Jozani, Éric Marchand, and Ahmad Parsian. "On estimation with weighted balanced-type loss function." In: *Statistics & Probability Letters* 76.8 (2006), pp. 773–780.

[87] Ryan Kiros, Yukun Zhu, Russ R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. "Skip-thought vectors." In: *Adv. Neural Inf. Process. Syst.* 28 (2015).

[88] Brandon Knieriem, Xiaolu Zhang, Philip Levine, Frank Breitinger, and Ibrahim Baggili. "An overview of the usage of default passwords." In: *Proceedings of International Conference on Digital Forensics and Cyber Crime*. Springer. 2017, pp. 195–203.

[89] Ajay Krishna, Michel Le Pallec, Radu Mateescu, and Gwen Salaün. "Design and deployment of expressive and correct Web of Things applications." In: *ACM Trans. Internet Technol.* 3.1 (2021), pp. 1–30.

[90] Vu Le, Sumit Gulwani, and Zhendong Su. "Smartsynth: Synthesizing smartphone automation scripts from natural language." In: *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*. 2013, pp. 193–206.

[91] Brian Lester, Rami Al-Rfou, and Noah Constant. "The power of scale for parameter-efficient prompt tuning." In: *arXiv preprint arXiv:2104.08691* (2021).

[92] James R Lewis and Jeff Sauro. "The factor structure of the system usability scale." In: *International conference on human centered design*. Springer. 2009, pp. 94–103.

[93] Lei Li, Yongfeng Zhang, and Li Chen. "Personalized prompt learning for explainable recommendation." In: *arXiv preprint arXiv:2202.07371* (2022).

[94] Xiang Lisa Li and Percy Liang. "Prefix-tuning: Optimizing continuous prompts for generation." In: *arXiv preprint arXiv:2101.00190* (2021).

[95] Henry Lieberman, Fabio Paternò, Markus Klann, and Volker Wulf. "End-user development: An emerging paradigm." In: *End user development*. Springer, 2006, pp. 1–8.

[96] Zhen Ling, Kaizheng Liu, Yiling Xu, Yier Jin, and Xinwen Fu. "An end-to-end view of IoT security and privacy." In: *Proceedings of IEEE Global Communications Conference GLOBECOM*. IEEE. 2017, pp. 1–7.

[97] Chang Liu, Xinyun Chen, Eui Chul Shin, Mingcheng Chen, and Dawn Song. "Latent attention for if-then program synthesis." In: *Advances in Neural Information Processing Systems* 29 (2016), pp. 4574–4582.

[98] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing." In: *arXiv preprint arXiv:2107.13586* (2021).

[99] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. "GPT understands, too." In: *arXiv preprint arXiv:2103.10385* (2021).

[100] Loop11. *Loop11 User Testing*. https://www.loop11.com/. [Online; accessed 27-January-2023].

[101] Yuan Luo, Long Cheng, Hongxin Hu, Guojun Peng, and Danfeng Yao. "Context-Rich Privacy Leakage Analysis Through Inferring Apps in Smart Home IoT." In: *IEEE Internet Things J.* 8.4 (2020), pp. 2736–2750.

[102] Bill Manaris. "Natural language processing: A human-computer interaction perspective." In: *Advances in Computers*. Vol. 47. Elsevier, 1998, pp. 1–66.

[103] Marco Manca, Fabio Paternò, Carmen Santoro, and Luca Corcella. "Supporting end-user debugging of trigger-action rules for IoT applications." In: *International Journal of Human-Computer Studies* 123 (2019), pp. 56–69.

[104] Ana Marasović, Iz Beltagy, Doug Downey, and Matthew E Peters. "Few-shot self-rationalization with natural language prompts." In: *arXiv preprint arXiv:2111.08284* (2021).

[105] Panos Markopoulos, Jeffrey Nichols, Fabio Paternò, and Volkmar Pipek. "End-user development for the internet of things." In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 24.2 (2017), pp. 1–3.

[106] G Harry Mc Laughlin. "SMOG grading-a new readability formula." In: *Journal of reading* 12.8 (1969), pp. 639–646.

[107] McKenna McCall, Faysal Hossain Shezan, Abhishek Bichhawat, Camille Cobb, Limin Jia, Yuan Tian, Cooper Grace, and Mitchell Yang. *SAFETAP: An Efficient Incremental Analyzer for Trigger-Action Programs*. Tech. rep. Carnegie Mellon University, 2021.

[108] Georgios Metaxas and Panos Markopoulos. "Natural contextual reasoning for end users." In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 24.2 (2017), pp. 1–36.

[109] Xianghang Mi, Feng Qian, Ying Zhang, and XiaoFeng Wang. "An empirical characterization of IFTTT: ecosystem, usage, and performance." In: *Proc. Internet Measurement Conference*. ACM, 2017, pp. 398–404.

[110] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word representations in vector space." In: *arXiv preprint arXiv:1301.3781* (2013).

[111] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. "Distributed representations of words and phrases and their compositionality." In: *Advances in neural information processing systems* 26 (2013).

[112] Andrés Molina-Markham, Prashant Shenoy, Kevin Fu, Emmanuel Cecchet, and David Irwin. "Private memoirs of a smart meter." In: *Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-efficiency in Building*. 2010, pp. 61–66.

[113] Leonardo de Moura, Bruno Dutertre, and Natarajan Shankar. "A tutorial on satisfiability modulo theories." In: *Proc. Int. Conf. on Computer Aided Verification*. Springer. 2007, pp. 20–36.

[114] Cataldo Musto, Marco de Gemmis, Pasquale Lops, and Giovanni Semeraro. "Generating post hoc review-based natural language justifications for recommender systems." In: *User Modeling and User-Adapted Interaction* 31.3 (2021), pp. 629–673.

[115] Nataliia Neshenko, Elias Bou-Harb, Jorge Crichigno, Georges Kaddoum, and Nasir Ghani. "Demystifying IoT security: An exhaustive survey on IoT vulnerabilities and a first empirical look on Internet-scale IoT exploitations." In: *IEEE Communications Surveys & Tutorials* 21.3 (2019), pp. 2702–2733.

[116] Dang Tu Nguyen, Chengyu Song, Zhiyun Qian, Srikanth V Krishnamurthy, Edward JM Colbert, and Patrick McDaniel. "IoTSan: Fortifying the safety of IoT systems." In: *Proc. 14th Int. Conf. on Emerging Networking EXperiments and Technologies*. 2018, pp. 191–203.

[117] J. Nielsen. *Card Sorting: How Many Users to Test*. https://www.nngroup.com/articles/card-sorting-how-many-users-to-test/. [Online; accessed 27-January-2023].

[118] Federica Paci, Davide Bianchin, Elisa Quintarelli, and Nicola Zannone. "IFTTT privacy checker." In: *International Workshop on Emerging Technologies for Authorization and Authentication*. Springer. 2020, pp. 90–107.

[119] Jeffrey Pennington, Richard Socher, and Christopher Manning. "GloVe: Global Vectors for Word Representation." In: *Proc. Conf. on Empirical Methods in Natural Language Processing*. ACL, 2014.

[120] Wolter Pieters. "Explanation and trust: what to tell the user in security and AI?" In: *Ethics and information technology* 13.1 (2011), pp. 53–64.

[121] Chris Quirk, Raymond Mooney, and Michel Galley. "Language to code: Learning semantic parsers for if-this-then-that recipes." In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2015, pp. 878–888.

[122] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. "Language models are unsupervised multitask learners." In: *OpenAI blog* 1.8 (2019), p. 9.

[123] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. "Exploring the limits of transfer learning with a unified text-to-text transformer." In: *J. Mach. Learn. Res.* 21.140 (2020), pp. 1–67.

[124] Alessandro Raganato and Jörg Tiedemann. "An analysis of encoder representations in transformer-based machine translation." In: *BlackboxNLP '18*. 2018.

[125] Readable. *readable app*. https://app.readable.com/text/. [Online; accessed 27-January-2023].

[126] Nils Reimers and Iryna Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks." In: *Proc. Conf. on Empirical Methods in Natural Language Processing and the 9th Int. Joint Conf. on Natural Language Processing*. ACL, 2019, pp. 3982–3992.

[127] Marcos Román-González, Juan-Carlos Pérez-González, and Carmen Jiménez-Fernández. "Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test." In: *Computers in human behavior* 72 (2017), pp. 678–691.

[128] Marcos Román-González, Juan-Carlos Pérez-González, Jesús Moreno-León, and Gregorio Robles. "Extending the nomological network of computational thinking with non-cognitive factors." In: *Computers in Human Behavior* 80 (2018), pp. 441–459.

[129] Douglas Rushkoff. *Program or be programmed: Ten commands for a digital age*. Or Books, 2010.

[130] Mahsa Saeidi, McKenzie Calvert, Audrey W. Au, Anita Sarma, and Rakesh B. Bobba. "If This Context Then That Concern: Exploring users' concerns with IFTTT applets." In: *Proc. on Privacy Enhancing Technologies* 2022 (2021), pp. 166 –186.

[131] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter." In: *arXiv:1910.01108* (2019). Retrieved from https://arxiv.org/abs/1910.01108.

[132] Alper Kaan Sarica and Pelin Angin. "Explainable security in SDN-based IoT networks." In: *Sensors* 20.24 (2020), p. 7326.

[133] Bruce Schneier. *Secrets and lies: digital security in a networked world*. John Wiley & Sons, 2015.

[134] Thibault Sellam, Dipanjan Das, and Ankur P Parikh. "BLEURT: Learning robust metrics for text generation." In: *arXiv preprint arXiv:2004.04696* (2020).

[135] Prasad Shinde, Gerd Szwillus, and Ing Reinhard Keil. "Application of Existing k-means Algorithms for the Evaluation of Card Sorting Experiments." PhD thesis. Paderborn University Paderborn, Germany, 2017.

[136] *Spacebrew*. URL: http://docs.spacebrew.cc/.

[137] Donna Spencer. *Card sorting: Designing usable categories*. Rosenfeld Media, 2009.

[138] 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*. 2017, pp. 1501–1510.

[139] Gerd Szwillus, Adrian Hülsmann, Yevgen Mexin, and Anastasia Wawilow. "Casolysis 2.0-Flexible Auswertung von Card Sorting Experimenten." In: *Mensch und Computer 2015–Usability Professionals* (2015).

[140] Daniel Tetteroo, Iris Soute, and Panos Markopoulos. "Five key challenges in end-user development for tangible and embodied interaction." In: *Proceedings of the 15th ACM on International conference on multimodal interaction*. 2013, pp. 247–254.

[141] Mike Thelwall. "The Heart and soul of the web? Sentiment strength detection in the social web with SentiStrength." In: *Cyberemotions*. Springer, 2017, pp. 119–134.

[142] 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: *26th USENIX Security Symposium (USENIX Security 17)*. 2017, pp. 361–378.

[143] Kiyotaka Uchimoto, Satoshi Sekine, and Hitoshi Isahara. "Text generation from keywords." In: *COLING 2002: The 19th International Conference on Computational Linguistics*. 2002.

[144] Blase Ur, Elyse McManus, Melwyn Pak Yong Ho, and Michael L Littman. "Practical trigger-action programming in the smart home." In: *Proceedings of the SIGCHI conference on human factors in computing systems*. 2014, pp. 803–812.

[145] Blase Ur, Melwyn Pak Yong Ho, Stephen Brawner, Jiyun Lee, Sarah Mennicken, Noah Picard, Diane Schulze, and Michael L Littman. "Trigger-action programming in the wild: An analysis of 200,000 ifttt recipes." In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. 2016, pp. 3227–3231.

[146] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." In: *Adv. Neural Inf. Process. Syst.* 30 (2017).

[147] Luca Vigano and Daniele Magazzeni. "Explainable security." In: *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE. 2020, pp. 293–300.

[148] Usman Wajid, Abdallah Namoun, and Nikolay Mehandjiev. "Alternative representations for end user composition of service-based systems." In: *Proceedings of International Symposium on End User Development*. Springer. 2011, pp. 53–66.

[149] Juan Wang, Shirong Hao, Ru Wen, Boxian Zhang, Liqiang Zhang, Hongxin Hu, and Rongxing Lu. "IoT-Praetor: Undesired Behaviors Detection for IoT Devices." In: *IEEE Internet Things J.* 8.2 (2021), pp. 927–940. DOI: 10.1109/JIOT.2020.3010023.

[150] Le Wang, Meng Han, Xiaojuan Li, Ni Zhang, and Haodong Cheng. "Review of classification methods on unbalanced data sets." In: *IEEE Access* 9 (2021), pp. 64606–64628.

[151] Maonan Wang, Kangfeng Zheng, Yanqing Yang, and Xiujuan Wang. "An explainable machine learning framework for intrusion detection systems." In: *IEEE Access* 8 (2020), pp. 73127–73141.

[152] Qi Wang, Pubali Datta, Wei Yang, Si Liu, Adam Bates, and Carl A Gunter. "Charting the attack surface of trigger-action IoT platforms." In: *Proc. Conf. on Comp. and Comm. Sec.* 2019, pp. 1439–1453.

[153] Qi Wang, Wajih Ul Hassan, Adam Bates, and Carl Gunter. "Fear and logging in the internet of things." In: *Proc. 25th Annual Network and Distributed System Security Symposium*. 2018.

[154] Ran Wang, Robert Ridley, Weiguang Qu, Xinyu Dai, et al. "A novel reasoning mechanism for multi-label text classification." In: *Information Processing & Management* 58.2 (2021), p. 102441.

[155] Mark Weiser. “The Computer for the 21 st Century.” In: *Scientific american* 265.3 (1991), pp. 94–105.

[156] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. “A survey of transfer learning.” In: *Journal of Big data* 3.1 (2016), pp. 1–40.

[157] Alma Whitten and J Doug Tygar. “Why Johnny Can’t Encrypt: A Usability Evaluation of PGP 5.0.” In: *USENIX security symposium*. Vol. 348. 1999, pp. 169–184.

[158] *WigWag Inc. (2020). WigWag Smart Home*. URL: http://www.wigwag.com/..

[159] Wikipedia. *Abstract syntax tree (AST)*. https://en.wikipedia.org/wiki/Abstract_syntax_tree. 2021. URL: https://en.wikipedia.org/wiki/Abstract_syntax_tree.

[160] Jeannette M Wing. “Computational thinking and thinking about computing.” In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 366.1881 (2008), pp. 3717–3725.

[161] Michael S Wogalter, Dave DeJoy, and Kenneth R Laughery. *Warnings and risk communication*. CRC Press, 1999.

[162] Ding Xiao, Qianyu Wang, Ming Cai, Zhaohui Zhu, and Weiming Zhao. “A3ID: an automatic and interpretable implicit interference detection method for smart home via knowledge graph.” In: *IEEE Internet of Things Journal* 7.3 (2019), pp. 2197–2211.

[163] Rixin Xu, Qiang Zeng, Liehuang Zhu, Haotian Chi, Xiaojiang Du, and Mohsen Guizani. “Privacy leakage in smart homes and its mitigation: IFTTT as a case study.” In: *IEEE Access* 7 (2019), pp. 63457–63471.

[164] Ziyu Yao, Xiujun Li, Jianfeng Gao, Brian Sadler, and Huan Sun. “Interactive semantic parsing for if-then recipes via hierarchical reinforcement learning.” In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2019, pp. 2547–2554.

[165] David Yarowsky. “Unsupervised word sense disambiguation rivaling supervised methods.” In: *Proc. 33rd Annual Meeting of the Association for Computational Linguistics*. 1995, pp. 189–196.

[166] Yongjian You, Weijia Jia, Tianyi Liu, and Wenmian Yang. "Improving abstractive document summarization with salient information modeling." In: *ACL '19*. 2019, pp. 2132–2141.

[167] *Zapier*. URL: https://zapier.com.

[168] Eric Zeng, Shrirang Mare, and Franziska Roesner. "End user security and privacy concerns with smart homes." In: *Proceedings of Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*. 2017, pp. 65–80.

[169] Zhi-Kai Zhang, Michael Cheng Yi Cho, Chia-Wei Wang, Chia-Wei Hsu, Chong-Kuan Chen, and Shiuhpyng Shieh. "IoT security: ongoing challenges and research opportunities." In: *2014 IEEE 7th international conference on service-oriented computing and applications*. IEEE. 2014, pp. 230–234.

[170] Guoshuai Zhao, Hao Fu, Ruihua Song, Tetsuya Sakai, Zhongxia Chen, Xing Xie, and Xueming Qian. "Personalized reason generation for explainable song recommendation." In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 10.4 (2019), pp. 1–21.

[171] Serena Zheng, Noah Apthorpe, Marshini Chetty, and Nick Feamster. "User perceptions of smart home IoT privacy." In: *Proc. ACM on Human-Computer Interaction* 2 (2018), pp. 1–20.

[172] Xiaojin Zhu and Zoubin Ghahramani. *Learning from labeled and unlabeled data with label propagation*. Tech. rep. CMU-CALD-02-107. Carnegie Mellon University, 2002.

[173] *Zipato (2020)*. URL: https://www.zipato.com/.

[174] IFTTT official website. *Documentation of IFTTT*. https://platform.ifttt.com/docsl. 2022. URL: https://platform.ifttt.com/docsl.

[175] IFTTT official website. *Plans of IFTTT*. https://ifttt.com/plans. 2022. URL: https://ifttt.com/plans.