Automated Identification of Security Discussions in Microservices Systems: Industrial Surveys and Experiments

Ali Rezaei-Nasab[a], Mojtaba Shahin[b,*], Peng Liang[a,*], Mohammad Ehsan Basiri[c], Seyed Ali Hoseyni Raviz[a],
Hourieh Khalajzadeh[b], Muhammad Waseem[a], Amineh Naseri[d]

[a]*School of Computer Science, Wuhan University, 430072 Wuhan, China*
[b]*Department of Software Systems and Cybersecurity, Faculty of IT, Monash University, 3800 Melbourne, Australia*
[c]*Department of Computer Engineering, Shahrekord University, 64165478 Shahrekord, Iran*
[d]*Department of Computer Engineering, Sirjan University of Technology, 7813733385 Sirjan, Iran*

Abstract

Lack of awareness and knowledge of microservices-specific security challenges and solutions often leads to ill-informed security decisions in microservices system development. We claim that identifying and leveraging security discussions scattered in existing microservices systems can partially close this gap. We define security discussion as "*a paragraph from developer discussions that includes design decisions, challenges, or solutions relating to security*". We first surveyed 67 practitioners and found that securing microservices systems is a unique challenge and that having access to security discussions is useful for making security decisions. The survey also confirms the usefulness of potential tools that can automatically identify such security discussions. We developed fifteen machine/deep learning models to automatically identify security discussions. We applied these models on a manually constructed dataset consisting of 4,813 security discussions and 12,464 non-security discussions. We found that all the models can effectively identify security discussions: an average precision of 84.86%, recall of 72.80%, F1-score of 77.89%, AUC of 83.75% and G-mean 82.77%. DeepM1, a deep learning model, performs the best, achieving above 84% in all metrics and significantly outperforms three baselines. Finally, the practitioners' feedback collected from a validation survey reveals that security discussions identified by DeepM1 have promising applications in practice.

Keywords: Microservices Architecture, Security, Machine Learning, Deep Learning, Automation

1. Introduction

The Microservices Architecture (MSA) style, as one of the latest trends in software design, aims to develop scalable software systems as a collection of small services (i.e., microservices) that can be deployed independently [1], [2]. Microservices are open to a diverse range of interpretations, and little consensus exists regarding their characteristics (e.g., the size of microservices) [3]. However, microservices can be generally characterized by providing a limited amount of functionality, communicating with each other via light-weight messaging approaches (e.g., HTTP APIs), being built around autonomous business capabilities, and being maintained and tested in high isolation [1], [2], [4]. Faster deployment, improved scalability, and greater autonomy are three main benefits of microservices systems (the systems that adopt the MSA style) [3].

Over the past few years, there has been ongoing research on different aspects of MSA. A significant body of literature reports migration techniques to move from a monolith to an MSA (e.g., [5], [6]) [7]. Others investigate testing and monitoring strategies for microservices systems (e.g., [8], [9]). Another line of research focuses on the required changes in organizational culture and structure to adopt the MSA style (e.g., [3], [10]).

However, security in MSA remains an open issue and has received insufficient research attention [1], [7], [11]. The security issues associated with the MSA style are numerous, which could not be found or are of much less interest in traditional monoliths and service-oriented architectures [12], [13], [14], [15]. They range from the need for establishing trust between individual microservices because a single compromised service may maliciously impact the entire system, to the increasing chance of attack surfaces as microservices can be developed and deployed by different technologies and tools (e.g., containers), to the greater difficulty to guarantee the security of tens or even hundreds of microservices running in operation. Furthermore, as the MSA style is a new paradigm and still evolving, there is a knowledge gap among practitioners and organizations on designing and implementing a secure microservices system [16], [17], [18], [19]. More significant unknowns exist around (new) technologies and tools (e.g., containers) that are used to develop and deploy microservices [20], [21], [22]. This is because a lot of confusion exists in the industry about the po-

---

*Corresponding authors
Email addresses: `rezaei.ali.nasab@gmail.com` (Ali Rezaei-Nasab), `mojtaba.shahin@monash.edu` (Mojtaba Shahin), `liangp@whu.edu.cn` (Peng Liang), `basiri@sku.ac.ir` (Mohammad Ehsan Basiri), `s.ali.hoseyni@gmail.com` (Seyed Ali Hoseyni Raviz), `hourieh.khalajzadeh@monash.edu` (Hourieh Khalajzadeh), `m.waseem@whu.edu.cn` (Muhammad Waseem), `naseri.amine@sirjantech.ac.ir` (Amineh Naseri)

tential security capabilities of those technologies and tools and the best practices to configure them for security purposes and address their security vulnerabilities [23], [24], [25].

Lack of awareness and knowledge of security in microservices systems often leads to ill-informed security decisions in microservices system development. This can become a real issue in the software industry because a growing number of organizations have employed or plan to use this architectural style to modernize their software solutions or develop new applications [7]. Our position to (partially) address this gap is to automatically identify and leverage security discussions from existing microservices systems. We define **security discussion** as "*a paragraph from developer discussions (conversations) that includes design decisions, challenges, or solutions relating to security*". It is argued that accessing and learning past design discussions and decisions (e.g., security decisions) can help make better design decisions (e.g., [17], [18], [26], [27]). However, we believe that security discussions collected from non-microservices systems may not be useful and practical to increase awareness and knowledge of security in microservices systems among microservices practitioners. To prove our claim, we first need to ensure that securing microservices systems is a unique challenge. This shows that security discussions that occur between microservices developers are different from other types of software systems. We then need to confirm if such security discussions will be useful in securing microservices systems (e.g., raising practitioners' awareness of microservices-specific security solutions).

Such security discussions can be found in the issue tracking systems of commercial and open-source microservices systems. Stack Overflow posts can be another source to identify security discussions on microservices systems. Research shows that such artifacts include a wide range of information, such as user stories, requirements, design decisions, design solutions, design rationale, bug reports, etc (e.g., [28], [27], [29], [30]). Hence, issue tracking systems and Stack Overflow posts are ideal places for microservices practitioners to share and communicate the knowledge about security issues, practices, design decisions, and justify their security decisions [31], [30]. However, manually identifying security discussions is challenging for practitioners because it is error-prone, takes significant time for practitioners, and requires substantial domain expertise [32], [33]. Hence, there is a need for an automated approach to identify security contents in such data sources.

In this paper, we first identified ten open-source projects from GitHub designed based on the MSA style. We then conducted an online survey completed by 67 microservices practitioners to solicit their perspectives on security in microservices systems. The survey participants confirmed that securing microservices systems is a unique challenge for them. They also found that previous security discussions collected from existing microservices systems are useful for making security decisions. Based on the insights from the survey, we developed twelve Machine Learning (ML) and three Deep Learning (DL) models to automatically identify security discussions from microservices developer discussions (GitHub issues and Stack Overflow posts). To evaluate the performance of the ML and DL models

(henceforth learning models), we manually built and labeled a dataset of 17,277 paragraphs. The 17,277 paragraphs, including 4,813 security discussions and 12,464 non-security discussions, were collected from two data sources: 1,692 issue discussions from five open-source projects that adopt the MSA style, and 498 Stack Overflow posts with "microservices" and "security" tags. The experimental results show that all learning models are promising, achieving, on average, a precision of 84.86%, recall of 72.80%, F1-score of 77.89%, AUC (Area Under the Receiver Operating Characteristic Curve) of 83.75%, and G-mean 82.77%. Of the 15 learning models, DeepM1, a DL model, performs the best, achieving 86.73% precision, 84.25% recall, 85.47% F1-score, 89.63% AUC, and 89.07% G-mean. DeepM1 also outperforms three state-of-the-art baselines with improvements ranging from 1.018x to 3.756x in all metrics.

We conducted another survey (validation survey) to show the usefulness and actionability of the results produced by DeepM1 in practice. Generally, 68%-78% of the validation survey respondents perceived that the security discussions detected by DeepM1 could have seven practical applications in the design and development of secure microservices systems. The notable applications are: the detected security discussions can help them make informed security decisions in the future or refine the existing sub-optimum security decisions and provide hints/clues to locate critical issues (e.g., security bugs, security mistakes, etc.) faster in microservices systems.

The key contributions of this paper are that: **(1)** We gain a better understanding of security concerns in microservices systems. **(2)** This is the first work that develops ML and DL models to discriminate security discussions from non-security discussions in developer discussions of microservices systems. **(3)** We construct a dataset consisting of 4,813 microservices security discussions and 12,464 non-security discussions. **(4)** We freely release the implementation of the models and the datasets used in this paper online [34].

Paper Organization: In Section 2, we provide a motivating scenario. Section 3 describes the process of locating MSA-based projects from GitHub and building a security discussions dataset. Section 4 describes our methodology, including survey and experiments. Section 5 reports the findings. Section 6 reflects on the findings, and Section 7 reports the threats to validity. We examine the related work in Section 8. Section 9 concludes the paper and outlines some future works.

2. Motivating Scenario

Bob is working for a company that develops a wide range of software systems in different domains. A few months ago, Bob was involved as a senior software engineer in developing an e-Shop web application (called the eShopOnContainers project [1]) in the company, which was designed based on the MSA style. Recently, the company won a tender to migrate a large-scale monolithic system to the MSA style (we call it
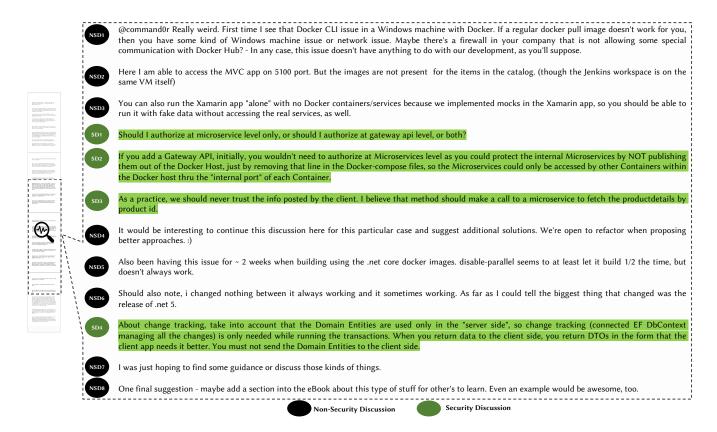
---

[1] `https://bit.ly/3uRN2qu`

NSD1 @command0r Really weird. First time I see that Docker CLI issue in a Windows machine with Docker. If a regular docker pull image doesn't work for you, then you have some kind of Windows machine issue or network issue. Maybe there's a firewall in your company that is not allowing some special communication with Docker Hub? - In any case, this issue doesn't have anything to do with our development, as you'll suppose.

NSD2 Here I am able to access the MVC app on 5100 port. But the images are not present for the items in the catalog. (though the Jenkins workspace is on the same VM itself)

NSD3 You can also run the Xamarin app "alone" with no Docker containers/services because we implemented mocks in the Xamarin app, so you should be able to run it with fake data without accessing the real services, as well.

SD1 Should I authorize at microservice level only, or should I authorize at gateway api level, or both?

SD2 If you add a Gateway API, initially, you wouldn't need to authorize at Microservices level as you could protect the internal Microservices by NOT publishing them out of the Docker Host, just by removing that line in the Docker-compose files, so the Microservices could only be accessed by other Containers within the Docker host thru the "internal port" of each Container.

SD3 As a practice, we should never trust the info posted by the client. I believe that method should make a call to a microservice to fetch the productdetails by product id.

NSD4 It would be interesting to continue this discussion here for this particular case and suggest additional solutions. We're open to refactor when proposing better approaches. :)

NSD5 Also been having this issue for ~ 2 weeks when building using the .net core docker images. disable-parallel seems to at least let it build 1/2 the time, but doesn't always work.

NSD6 Should also note, i changed nothing between it always working and it sometimes working. As far as I could tell the biggest thing that changed was the release of .net 5.

SD4 About change tracking, take into account that the Domain Entities are used only in the "server side", so change tracking (connected EF DbContext managing all the changes) is only needed while running the transactions. When you return data to the client side, you return DTOs in the form that the client app needs it better. You must not send the Domain Entities to the client side.

NSD7 I was just hoping to find some guidance or discuss those kinds of things.

NSD8 One final suggestion - maybe add a section into the eBook about this type of stuff for other's to learn. Even an example would be awesome, too.

● Non-Security Discussion   ● Security Discussion

Figure 1: An excerpt of developer discussions in the eShopOnContainers project

Project C). Given that Bob was involved in the eShopOnContainers project, the company has assigned Bob as a team leader of Project C. Bob's observations from the eShopOnContainers project are:

1. Conversations between developers of the eShopOnContainers project captured in the issue tracking system include various security information. An excerpt of developer discussions in the eShopOnContainers project is shown in Figure 1.

2. Some of the security challenges and concerns in the MSA style are unique, which have not been faced before by developers of the eShopOnContainers project. Therefore, the strategies adopted to address those challenges were new.

3. Although some security issues faced by developers of the eShopOnContainers project were not new and had common security solutions, the developers repeatedly had those security issues and mistakes [35]. The developers frequently looked at their previous discussions to recognize which ones were about security and understand how those security issues and mistakes were solved.

Inspired by the idea of the Open Web Application Security Project (OWASP)[2] and his observations from the eShopOnContainers project, Bob thinks that the security of Project B can be improved if its developers' awareness is increased about the most critical security issues in microservices systems, their impacts, and their corresponding solutions [35]. Further to this, such security information can be imported to (security) knowledge management tools used in the company [26]. In such a scenario, knowing where are the security information is a direct prerequisite for organizing it to a structured or company-specific format [36].

In the light of the above scenario, Bob asks a member of Project C, Alex, a newcomer to the company, to collect security discussions from developer discussions in the eShopOnContainers project. Given Alex has limited experience in (security in) microservices, he may face the following challenges in this task. (1) He needs the domain knowledge to distinguish developer security discussions from non-security discussions in microservices systems [37]. Some security discussions (e.g., SD4 in Figure 1) do not have (common) security-related terms. Some non-security discussions (e.g., NSD1 in Figure 1) include common security-related terms. (2) Doing this task for Alex is both time-consuming and tedious. For example, a developer discussion [3], which includes 293 comments, may take 10 hours for Alex to deeply read the entire of that developer discussion [27].

Hence, a tool that can automatically and accurately identify security discussions, including critical security issues in microservices systems, their impacts, and corresponding solutions, can help develop secure microservices systems. The out-
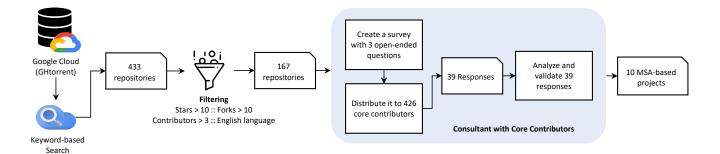

Figure 2: Process of locating 10 open-source projects that adopt the MSA style

put of such a tool has educational benefits for microservices practitioners and raise their awareness of the different aspects of security in microservices systems.

3. A Security Discussions Dataset

Our goal is to (partially) bridge the knowledge gap among practitioners in securing microservices systems by automatically identifying security discussions from previous microservices developer discussions. To this end, we need to create a dataset of security discussions. As discussed in the Introduction section, security discussions can be found in the issue tracking systems of commercial and open-source systems and Stack Overflow posts. Security discussions in commercial software systems usually are not accessible to the public. Hence, we decided to build our dataset based on security discussions in open-source microservices systems on GitHub and Stack Overflow. As there is no reliable information indicating which open-source systems on GitHub are designed based on the MSA style, we first had to identify such systems on GitHub. Section 3.1 describes the process of finding microservices systems on GitHub. Section 3.2 reports how a security discussions dataset is created from developer discussions in five microservices systems on GitHub and Stack Overflow posts.

3.1. *Phase I: Locate Microservices Systems in GitHub*

The process of finding microservices systems is composed of two steps: keyword-based search and consulting with core contributors.

3.1.1. *Keyword-based Search*

Figure 2 shows the process of locating MSA-based projects. As shown in Figure 2[4], we first executed a search string on the GHTorrent dump hosted on Google Cloud version 1/4/2018 [38]. Specifically, we searched the following search string on the title and description of open-source projects. The search process resulted in 2,690 repositories. Previous research showed that the number of forks and stars could be a good indicator of popular repositories [39]. We then used these factors to obtain the repositories that had such properties. We chose the repositories whose fork and star were more than 10, resulting in 433 repositories.

---

[4]Some of the icons are from `https://bit.ly/3dta9iM`

"microservice" OR "micro service" OR "micro-service"
OR
"Microservice" OR "Micro service" OR "Micro-service"

3.1.2. *Consult with Core Contributors*

Having "microservices" term in the title or description of a project does not guarantee that the project is designed based on the MSA style. Hence, we needed to make sure if those 433 projects follow the MSA style. We only focused on the repositories with more than three contributors and with the English language. This decreased the number of repositories from 433 to 167. Afterward, we tried to communicate with the core contributors of the 167 projects and directly asked them whether or not these projects follow the MSA style. We define the core contributors of a project as the top three contributors who have the most commits in the project, and their email addresses are publicly available. We supposed that this type of contributor has a broad view of a project and is more likely to know its design and structure. Then, we checked their GitHub profiles or searched their names to find their publicly available emails. We finally got the emails of 426 core contributors from the 167 projects, and we emailed and asked them to answer the following questions:

1. *Can you please confirm if our interpretation is correct that this project (URL of the project) is designed by following the MSA style? Note that if the projects (e.g., frameworks or tools) support the development of microservices systems but are not designed based on the MSA style, please clarify it.*

2. *If our interpretation is correct, what features or characteristics of the architecture of this open-source project show that the MSA style has been used?*

3. *Optional question: Do you know any other open-source projects that are designed by following the MSA style?*

Out of the 426 core contributors, 39 replied to our emails (response rate: 9%). Table 1 shows two core contributors' responses. It should be noted that while some respondents answered "**Yes**" to question (**1**) and claimed that the asked project follows the MSA style, they did not provide a detailed answer or any answer to question (**2**). To avoid any misinterpretation and possible risks, we did not consider this type of projects as a microservices project. The analysis of core contributors'
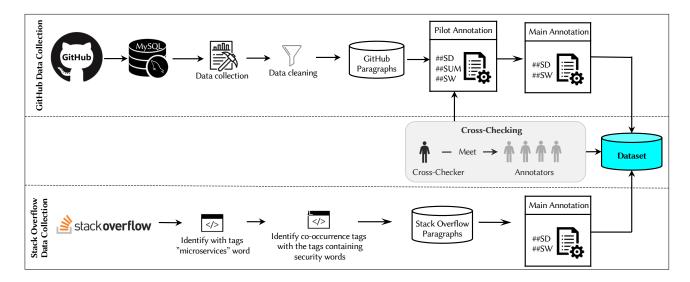
Figure 3: Process of building a Security Discussions Dataset

responses revealed that 10 open-source projects had been designed by following the MSA style. The basic information about these projects is shown in Table 2.

3.2. *Phase II: Construct a Security Discussions Dataset*

This section describes the construction of a dataset consisting of 17,277 labeled paragraphs from two sources: developer discussions of five open-source microservices systems on GitHub and Stack Overflow posts. This dataset is later in Section 4.2 used for training and testing learning models. Figure 3 shows the process of constructing this dataset.

3.2.1. *GitHub Data Collection*

Prepare Data. We chose five projects in Table 2, including *goa*, *eShopOnContainers*, *microservices−demo*, *light−4j*, and *deep-framework*, as the reference points for building a security discussions dataset. These five projects were selected because they are significantly larger than other projects. Hence, it is highly likely that their contributors had more discussions in issue tracking systems. The decisions and discussions relating to (the design of) a software system can be usually captured in issue tracking systems [27]. Hence, we leveraged discussions on issue tracking systems. Both open and closed issues were used to increase the chance of finding security discussions. We randomly selected 1,692 issues out of 5,724 issues (i.e., confidence level: 95% and margin of error: 2% [40]) extracted from the five mentioned projects using the GitHub v3 API on 09/01/2020. Issues may contain different types of information, such as code snippets [41]. Consequently, code snippets were removed from issues.

We borrowed the idea from Viviani et al. [27] and chose paragraph as the unit of analysis. This can also help avoid possible difficulties (e.g., identifying useful information from lengthy issues) for practitioners [42]. We divided 1,692 issue discussions into 12,393 paragraphs.

Pilot Study. Before annotating the paragraphs extracted from the five mentioned projects, we conducted a pilot study to understand if security discussions can be found in paragraphs. If so, what are the characteristics of security discussions? We randomly selected 575 paragraphs from 200 issues obtained from the 10 projects listed in Table 2. Then 575 paragraphs were divided between four annotators (i.e., four authors). The annotators were given a coding schema to annotate the paragraphs assigned to them. Another author was responsible for cross-checking the annotated paragraphs. Each paragraph was annotated using the following pattern: If the annotator specified a paragraph as a "**security discussion**", the annotator had to:

- *Write "1", indicating it is a security discussion [**SD**]*

- *Write a summary about the paragraph (one sentence) [**SUM**]*

- *Write the security-related words [**SW**]*

If the given paragraph **did not** include a "security discussion", the annotators were asked to:

- *Write "0", indicating it is a non-security discussion [**SD**]*

At the end of the pilot annotation, we found 152 out of the 575 paragraphs as security discussions, which gave us confidence that investigating paragraphs is the right way of identifying security information.

Main Study. In the main study, we annotated 12,393 paragraphs in *goa*, *eShopOnContainers*, *microservices-demo*, *deep-framework*, and *light-4j* projects. Our pilot annotation experience taught us that the **SUM** item was not useful as it took a significant time of the annotators and had no positive effect on identifying security discussions. Therefore, we decided not to extract this item in the main study. Three annotators (three authors) annotated 12,393 paragraphs.

5

Table 1: Core contributors' responses from eShopOnContainers and deep-framework

	Answers
eShopOnContainers project	**Answer to Question (1):** That's right, that project is based on microservices architecture. In fact, [there is] a whole book about microservices architecture with Microsoft .NET where eShopOnContainers is the reference application. Related free Microsoft eBook: https://bit.ly/3aF04Od. Other Microsoft books/guidance: https://bit.ly/2Ku89KH.
	Answer to Question (2): (a) Microservice's autonomy, including an autonomous database per microservice. (b) Use of API Gateways. (c) Use of Container Orchestrator when deploying to production (Kubernetes). (d) Asynchronous communication between updates/transactional operations between microservices to achieve eventual consistency between their data. You can see many more points in the Book, since it is related to this reference application.
	Answer to Question (3): This one by Microsoft as well (I was not involved in this one, though): https://bit.ly/2Vw62wo
deep-framework project	**Answer to Question (1):** Yes, deep-framework is designed using microservices architectural patterns. To be clear, this is an abstracted set of javascript / nodejs libraries that simplify development and provisioning of cloud resources (AWS to be specific).
	Answer to Question (2): Componentization via Services, Products not Projects, Decentralized Governance, Decentralized Data Management, Infrastructure Automation and more. We have been using Micro Frontends since 2015, way before Cam Jackson published his article from 2019.
	Answer to Question (3): Yeah, we can proudly point to another repository that uses microservices architectural patterns called AWS Landing Zone as Terraform Module: https://bit.ly/3iVXh7Q

Table 2: List of identified microservices systems on GitHub. Number of Issues (❗); Number of Releases (🏷); Number of Contributors (👥); Number of Stars (☆); Number of Forks (ᛦ); Active Date (📅); Line of Codes (</>); Languages (🖹)

Project Name (URL)	❗	🏷	👥	☆	ᛦ	📅	</>	🖹
goa (https://bit.ly/2Vz0GjV)	2442	31	73	4.1k	450	2014-now	82,193	GO
eShopOnContainers (https://bit.ly/3eMUFYy)	1200	15	98	16.2k	6.8k	2016-now	136,963	C#, Javascript, HTML
microservices-demo (https://bit.ly/3cKLxln)	805	13	43	2.6k	2.6k	2016-now	18,828	Shell, Python, HCL, Ruby
scalecube-services (https://bit.ly/3bBBWgq)	689	96	18	477	72	2015-now	11,764	Java
moleculer (https://bit.ly/3bytGxK)	649	87	65	4k	383	2017-now	93,242	Javascript
deep-framework (https://bit.ly/3cO6o79)	640	22	9	532	75	2015-now	920,806	HTML, Javascript, CSS
light-4j (https://bit.ly/2Y3eEwe)	637	104	25	3k	496	2016-now	50,099	Java, Objective-J
apollo (https://bit.ly/3axm4KM)	298	39	37	1.6k	215	2015-now	1,828	Java
spring-petclinic-microservices (https://bit.ly/2S3EXON)	142	4	24	747	894	2016-now	14,506	Java, HTML, Javascript
microservice_workshop (https://bit.ly/2Y2pNgw)	13	67	4	68	59	2014-now	163,237	Java, C#, Ruby

Table 3 shows examples of security (i.e., issue 647[5] and issue 14[6]) and non-security (i.e., issue 8[7]) discussions from GitHub, along with their annotations. The data collected from GitHub includes 1,602 security discussions and 10,791 non-security discussions.

3.2.2. *Stack Overflow Data Collection*

The number of security discussions collected from GitHub is relatively small compared to non-security discussions. This motivated us to explore Stack Overflow posts to identify more security discussions related to microservices systems. Moreover, the content, language usage, and sentence structure of Stack Overflow posts may differ from developer discussions captured by issue tracking systems in GitHub.

Prepare Data. We extracted the posts with the "microservice" tag, which resulted in 5,655 posts. Being made up of numerous paragraphs, annotating such a large number of posts could take a great deal of time. Hence, we opted to choose a limited number of these posts, particularly those that increase the chance of identifying security discussions. To this end, we further filtered the 5,655 posts and only selected the posts that contained at least one of the following tags: "*secur*", "*auth*", "*safe*", "*permiss*", "*credential*", "*access*", "*identity*", or "*jwt*". This process resulted in 498 posts. Each post includes a question and a list of answers to the question [43]. Stack Overflow provides mechanisms for users to append comments to both questions and answers. This enables them to have further discussions on the posted questions and answers. Hence, we included all 498 posts' questions, their questions' comments, answers, and answers' comments in our data analysis. In this way, we were exposed to valuable in-

---

[5] https://bit.ly/3fPyBiC
[6] https://bit.ly/3pk7Vt7
[7] https://bit.ly/3cg2KoW

Table 3: Six sample paragraphs with their annotations from our dataset including GitHub data and Stack Overflow data.

	Sample Paragraphs	Annotations
GitHub Data	"Agree with @mvelosop - Even when the microservices are internal and you could just set the authorization at the API Gateway level, it is a good security practice to set an authorization boundary on each microservice. But it is up to you how you want to balance security vs. simplicity". [Taken from issue 647[5]]	**Security Discussion?** Yes **Security Words:** authorization; security; Gateway
	"For OAuth2 Authorization Server, if client application render the SPA to collect user credentials, we need to make sure our Authorization Code service can handle CORS gracefully". [Taken from issue 14[6]]	**Security Discussion?** Yes **Security Words:** OAuth2; authorization; credential
	"Until now, I am always thinking to implement websocket, rpc and graphql in the same code based but it look like they are not compatible at all. At this moment, I am seriously thinking to split the current light-java to spin off light-java-rest and then in the future to implement light-java-rpc with websocket support and light-java-graphql. All of them are based on some common libraries within light-java. Nothing has been done yet and I want to hear from you guys". [Taken from issue 8[7]]	**Security Discussion?** No
Stack Overflow Data	"My biggest problem is for authentication (for now). After reading a LOT a documentation, It seems that the best solution is to use OpenID Connect to authenticate an user to retrieve a JWT that can by passed with the request to the microservices". [Taken from post 39134238]	**Security Discussion?** Yes **Security Words:** authentication; OpenID; JWT
	"If you would like to secure multiple services (deployed to the different app servers) with a common security provider, you should use Single-Sign-On approach". [Taken from post 58655233]	**Security Discussion?** Yes **Security Words:** security; Single-Sign-On
	"You have done well identifying three bounded contexts, one for each domain and implemented in three microservices (MS). You are conforming to the best practices regarding DDD". [Taken from post 44936115]	**Security Discussion?** No

formation regarding security discussions [43]. The 498 posts contain 498 questions, 547 question's comments, 470 answers, 520 answer's comments. We collected 4,884 paragraphs in the 498 posts using HTML tag <p>.

Main Study. Three authors annotated the 4,884 paragraphs collected from Stack Overflow posts. One of them was also involved in annotating the data collected from GitHub (Section 3.2.1). These 4,884 paragraphs were annotated using the same approach discussed in Section 3.2.1. The data collected from Stack Overflow includes 3,211 security discussions (paragraphs) and 1,673 non-security discussions (paragraphs). Post 39134238 and post 58655233 in Table 3 are two examples of the annotated security discussions from Stack Overflow. While post 39134238 is a Stack Overflow question, post 58655233 is a paragraph from the posted comments. Also, post 44936115 is a non-security discussion from the posted answers.

3.2.3. *Dataset Reliability*

Our dataset includes 17,277 labeled paragraphs. We randomly selected 581 paragraphs out of the 17,277 paragraphs (i.e., confidence level: 95% and margin of error: 4% [40]). We asked the second author (i.e., cross-checker), who was not involved in the annotation process described in Sections 3.2.1 and 3.2.2, to annotate the 581 paragraphs. We calculated Cohen's Kappa Coefficient to determine if there was an agreement between the 581 annotated paragraphs by the cross-checker and these 581 paragraphs annotated by other annotators. This resulted in a value of 0.82 for Cohen's Kappa Coefficient, which is commonly perceived as "almost perfect agreement" [44]. The cross-checker held several meetings with the annotators to identify disagreements and solve them. The security-related words collected during the annotation process helped the cross-checker and the annotators to reach an agreement more straightforward. It is mainly because the security-related words enabled both sides to understand the reason behind selecting a given paragraph as a security discussion. We make available the 581 paragraphs mentioned above of our dataset to enhance the reliability of our findings [34].

4. Research Design

The objective of this research is to help reduce the knowledge gap of practitioners in securing microservices systems. To this end, we formalized three research questions (RQs):

> ***RQ1.*** *How do practitioners perceive security in microservices systems?*

Rationale. The goal of **RQ1** is to collect software practitioners' perceptions of security in microservices systems. It also aims to realize if security discussions collected from past microservices systems can be useful for making security decisions during new microservices system development. **RQ1** is answered by a survey (Section 4.1). We refer to it as "preliminary survey".

> ***RQ2.*** *Can we effectively identify security discussions automatically in developer discussions of microservices systems?*

Rationale. Past and documented experiences of securing microservices systems are scarce (if any). Hence, security decisions in the context of microservices systems are often no well-informed. We believe that security knowledge (e.g., technical security advice) scattered in developer discussions can help bridge this gap [45]. As discussed in the Introduction section, manually identifying such security knowledge (e.g., security discussions) from developer discussions is not a straightforward task for practitioners. **RQ2** aims to use and experiment with different ML and DL models (henceforth learning models) to distinguish security discussions from non-security discussions automatically (Section 4.2).

> **RQ3**. *Do practitioners find the automated identification of security discussions in microservices systems useful in practice? If so, how?*

Rationale. The automated identification of security discussions would be helpful in practice only if the results produced by the ML/DL models (**RQ2**) are perceived useful by microservices practitioners. **RQ3** focuses on the results produced by the best-performing learning model and employs a survey (we call it "validation survey") to assess whether practitioners perceive those results useful in practice. If so, the validation survey further investigates how practitioners can use the produced results in practice (Section 5.3).

4.1. Preliminary Survey (RQ1)

We used an online survey to solicit practitioners' perspectives on security in microservices systems (**RQ1**). Here we demonstrate the process of conducting the survey.

Protocol. Following the guidelines suggested by [46], we designed a short survey[8] to collect software practitioners' perspectives on security in microservices systems. Our survey was anonymous to motivate more practitioners to participate and encourage them to provide honest answers [47]. The survey was hosted on Google Forms. In total, the survey had nine questions, including three demographic questions (e.g., "how many years have you been involved in microservices system development?"), five Likert scale questions, and one open-ended question. We made all questions except the open-ended question mandatory. At the preamble of the survey, we defined "security discussion" to avoid misinterpretations among the participants. The participants were asked to show how strongly they agree or disagree with five statements regarding security in microservices systems. The statements were designed based on the literature [23], [16], [12], [13] and rated on a five-point Likert scale ("strongly agree = 5" to "strongly disagree = 1"). The statements are:

- Statement 1. *"The microservices architecture (MSA) style brings unique security challenges"*.

- Statement 2: *"It is more challenging to address security in microservices systems compared to traditional service-oriented systems or monolithic systems"*.

- Statement 3. *"There is a knowledge gap among software practitioners on how to secure microservices systems"*.

- Statement 4. *"Software practitioners will make better security decisions while developing microservices systems if they are acquainted with the security discussions collected from the past microservices systems"*.

- Statement 5. *"A tool that can automatically collect security discussions would be useful"*.

The open-ended question allowed the participants to share any general comments about security concerns in microservices systems.

Participants. In Section 3.1.2, we found 167 open-source projects that had the potential to be designed based on the MSA style. Although we finally found that not all of these 167 projects followed the MSA style, many of these projects developed frameworks, tools, or libraries to support the development of microservices systems. We collected the publicly available email addresses of 868 software practitioners who contributed to these 167 open-source projects. We emailed all of the 868 software practitioners and invited them to fill out the survey. We got 67 responses (response rate: 7.7%).

Data Analysis. We employed descriptive statistics to analyze the demographic and Likert scale questions. We used open coding and constant comparison methods in grounded theory to analyze the qualitative responses to the open-ended question [48].

4.2. Experiments (RQ2)

To automatically identify security discussions, we developed a wide range of ML and DL based models. We applied these models to the dataset described in Section 3.

4.2.1. Pre-processing

We applied four pre-processing steps to remove possible noises from our dataset and polish the dataset.

Step 1. Removing useless characters and irrelevant words. Developers usually use informal language when they share their opinions in issue tracking systems or answer Stack Overflow questions [49]. For example, they may utilize emoji icons to indicate their feeling about a decision or a discussion. Links may also be added to developer discussions to provide, for example, background information [50]. Emoji icons, links, emails, and useless characters (e.g., '/', '*') in developer discussions do not provide credible information for Natural Language Processing (NLP) techniques. Even they may decrease the performance of classifiers. We used regular expressions in Python to remove all of them from our dataset.

Step 2. Converting short form of words to their full forms. Some developers prefer to use abbreviations (e.g., "wouldn't", "we'll", and "can't") when communicating their thoughts [41]. We identified such words and converted them to their full forms.

Step 3. Removing stop words. Removing stop words could positively affect the performance of learning classifiers, such as reducing noises and increasing recall value [51]. Therefore, we removed stop words from our dataset using Neural Language Toolkit (NLTK) [52].

Step 4. Stemming process. Stemming is a widely used normalization technique in NLP. Stemming removes the derivational affixes of a word and converts the word to its common base form (e.g., reducing "going" to "go") [53], [54].

Research has shown that simple stemming techniques (e.g., the Porter stemming technique) can improve recall without significantly degrading precision [55]. We used the Porter stemming technique to convert words in the dataset to their base forms.

4.2.2. *Machine Learning Models*

We used several ML algorithms, which are widely used in software engineering research [27], [56], [57], [58], to predict security discussions. More specifically, we used (1) Random Forest (RF) [59], (2) Decision Tree (DT) [60], (3) Support Vector Machine based on Linear Kernel (SVM-LR) [61], [62], and (4) Extreme Gradient Boosting (XGBoost) [63]. We used three text feature extraction techniques to extract features from paragraphs in our dataset, which can be used as input for ML algorithms. We briefly introduce them:

- **BoW**. BoW (Bag of Word) aims to count the occurrence frequency of a unique word in a document [64], [65]. This technique does not consider other language aspects of the sentence, such as grammar and word order.

- **TF-IDF**. In contrast to BoW, TF-IDF (Term Frequency-Inverse Document Frequency) is a more complicated feature engineering method that estimates each word's importance in a corpus of documents [66]. A word's weight is obtained by considering two factors: the number of appearing the word is in the document (TF) and the inverse document frequency (IDF) of the word across the corpus.

- **GloVe**. Word embedding techniques represent each word in a corpus (e.g., a set of text documents) with a word vector [67], [68]. Research has shown that word embedding techniques can enhance the performance of ML classifiers [69]. GloVe (global vectors for word representation), as one of the most-used embedding techniques, maps the words to a meaningful space and creates a semantic similarity between them [70]. GloVe uses a pre-trained word vector embedding composed of 300-dimensional vectors with 6B tokens and 400K vocabularies [71].

4.2.3. *Deep Learning Models*

We developed three DL models (DeepM1, DeepM2, and DeepM3) to automatically classify security discussions from developer discussions of microservices systems. Figure 4 shows an overview of DeepM1, DeepM2, and DeepM3. Below, we describe these DL models.

DeepM1. DeepM1 includes five blocks, in which three CNN (Convolutional Neural Networks) layers and two Bi-LSTM (Bi-directional Long Short-Term Memory) networks play significant roles (i.e., key building blocks) [72], [73], [74], [75]. While a wide range of deep network types can be used to develop a DL model, we opted to use CNN and Bi-LSTM in DeepM1 as they tend to be very popular in the text processing tasks [76], [77]. The popularity of CNN stems from its ability to learn local patterns in the textual data. LSTM can better deal with long-term dependencies in the input data than its competitors, such as recurrent neural networks (RNNs) [78].

❶ **Input and Word Embedding block.** In the first place, we assign a unique integer to each unique word in our dataset. The unique numbers are assigned to the unique words so that a smaller number shows that the word is more frequent in our dataset. Then, a corpus of the unique words and their assigned unique numbers is created.

Then, for each paragraph in our dataset, its words will be replaced by the unique numbers produced in the previous step [79]. This process converts each paragraph to a sequence of numbers. The size of each sequence is equal to its corresponding paragraph length. However, the lengths of all produced sequences need to be equal. We use the zero-padding technique [80], [81] to equalize the lengths of all the sequences (i.e., the length of all sequences is set to 100). In the first layer of DeepM1, we choose a word embedding layer [80], [81] and set the embedding dimension to 100. The embedding layer maps the word indexes of the corpus to their dense vector representations [79], [82]. To reduce overfitting, a GaussianNoise is applied immediately after the word embedding [80], [81].

❷ **Convolutional block.** This block performs feature extraction and includes three convolution layers and three pooling layers. The first layer of this block takes the embedding vectors as input and converts them to a feature map. The feature map captures all the relevant information about the embedding vectors. DeepM1 uses three convolutional networks with the kernel size and feature map of 7-gram and 10, 5-gram and 20, and 3-gram and 30. Hence, to increase the performance of DeepM1, each of these convolutional networks is configured based on the Same-Padding [83] and ReLU (Rectified Linear Unit) [84] functions. This block also employs max-pooling to merge the output of the convolutional networks [85]. To this end, a max-pooling operation is applied with a factor of 2 after each convolutional network to identify and select the most important features [86]. In the last step, a 0.25 dropout is added to handle the overfitting [87].

❸ **Bi-LSTM block.** This block takes the feature map produced in the previous block and transforms it into a compressed representation [88]. Recent studies [89], [77], [85] have shown that the DL models based on Bi-LSTM and CNN could obtain promising performance in the text classification tasks. Bi-LSTM is considered as an appropriate option when the sequence is long enough [90]. Bi-LSTM cells contain two LSTM layers in opposite directions (i.e., forward and backward), which can adequately deal with the sequential modeling problem [77]. Hence, we utilize two layers of Bi-LSTM with a dimensionality factor of 10.
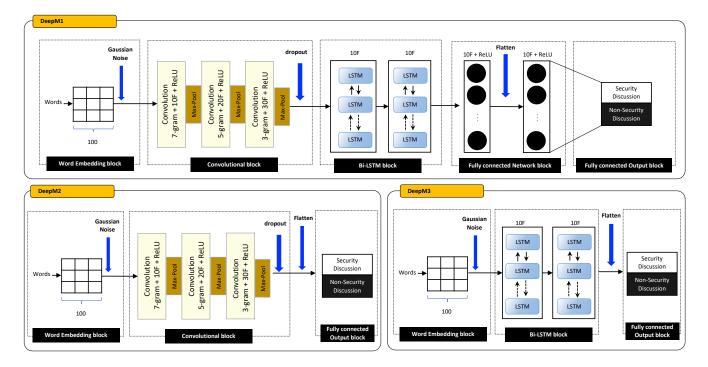
Figure 4: An overview of DeepM1, DeepM2, and DeepM3

❹ **Fully Connected Network block.** We also use two dense layers (i.e., fully-connected networks), in which each of them has 10 neurons. A flatten is placed between these two dense layers to merge 10 neurons into another column matrix [80]. The ReLU function is embedded in fully connected networks.

❺ **Fully Connected Output block.** The previous block's output is received by a fully connected output layer, which classifies a paragraph into a security discussion or non-security discussion.

DeepM2. This model uses three blocks of DeepM1: input and word embedding block, convolutional block, and fully connected output block. It should be noted that a flatten is applied between the last CNN layer and the fully connected output layer.

DeepM3. This model utilizes blocks 1, 3, and 5 of DeepM1 (i.e., input and word embedding block, Bi-LSTM block, and fully connected output block). After the second Bi-LSTM layer, a flatten is applied.

The rationale for choosing convolution layers in DeepM1 and DeepM2 is to extract different textual features. Specifically, using three convolution layers in DeepM1 and DeepM2, we can extract n-grams features. LSTM layers, on the other hand, are used in DeepM1 and DeepM3 to consider context information in the text. To investigate the effect of using these two types of layers, we examined them independently in DeepM2 and DeepM3 and jointly in DeepM1. Note that we optimize all deep learning models using Adam algorithm [91] with a learning rate of 1e-3 and a decay rate of 1e-6. Each DL model's batch size is set to 64, and its network is trained for 100 epochs with early stopping.

4.2.4. *Performance Evaluation*

To appraise the ML/DL models, we need a confusion matrix containing four factors: False Positive (FP), True Positive (TP), True Negative (TN), False Negative (FN). FP indicates the number of non-security discussions classified as security discussions TP signifies the number of security discussions that are accurately classified, and TN represents the number of non-security discussions that are precisely classified. FN represents the number of classes of actual security discussion paragraphs that are classified as non-security discussion paragraphs. Precision, recall, and F1-score are common metrics to compare the outputs of learning models (e.g., [32], [56], [58], [36]). Precision holds the percentage of identified security discussions that are actually security discussions [92] (i.e., $Precision = \frac{TP}{TP+FP}$). Recall displays the ratio between the number of accurately detected security discussions and the total number of security discussions (i.e., $Recall = \frac{TP}{TP+FN}$). F1-score is computed by the combination of the recall metric and precision metric [93] (i.e., $F1 - score = \frac{2*Precision*Recall}{Precision+Recall}$).

Other studies have recommended or used AUC [94], [95] as an evaluation metric to measure the performance of classifiers [27], [57], [96], [97]. In our context, the AUC metric calculates the likelihood that a prediction model will rank a randomly selected security discussion (i.e., TP) higher than a randomly chosen non-security discussion (i.e., FP) [27]. G-mean is another metric that can be used to measure the quality of binary classifications [98], [99]. Specifically, G-mean metric is highlighted as the best metric for imbalanced classes that consist of both positives and negatives, especially if classification errors being considered [100] (i.e., $G - mean = \frac{\sqrt{TP*TN}}{\sqrt{(TP+FN)*(TN+FP)}}$). All metrics range from 0 to 1. It is obvious that 1 represents the best

performance, and 0 value shows the worst performance prediction model.

We utilize 10-fold cross-validation technique for the performance evaluation of ML/DL models. The whole dataset (i.e., 17,277 paragraphs collected from both GitHub and Stack Overflow (Section 3.2)) is divided randomly into ten separate folds. The divided data is evaluated 10 times. Each time, 10% of data (i.e., one fold) is selected for performance evaluation, and the rest (i.e., 90%) is used to train the system. In this study, ML/DL models are set to a fixed seed for reproducibility [81], [101].

4.2.5. Experiment Setup and Implementation

All collected data were stored in a MySQL database by using a Python script. We implemented all experiments using scikit-learn[9] and Keras[10] libraries in Python. We executed the experiments on Google Colab with an Intel Xeon CPU and 13GB of RAM.

4.3. Validation Survey (RQ3)

We conducted an online survey[11] to seek and analyze the perceptions of practitioners about the usefulness of the results produced by the learning models (RQ2). We detail the execution of the survey in the following.

Protocol. We designed an anonymous survey hosted on the Qualtrics platform with 11 questions. In the servery preamble, we described the problem statement, our proposed approach, and the survey's objective. In the problem statement, we explained why security decisions in microservices systems are often not well informed. Then, we described that we developed an approach that can automatically identify security discussions with high accuracy from a large number of issue discussions in microservices systems to (partially) address this problem. We also defined "security discussion" to avoid misinterpretations. The participants were shown the distinguished security discussions from non-security discussions using our best learning model (i.e., DeepM1) in issue 303 from the Moleculer project and in issue 803 from the Goa project.

Out of the 11 survey questions, two questions were optional, and the rest was compulsory. The survey included three demographic questions, in which one of them was optional (i.e., which country do you currently work in?), seven Likert scale questions, and one open-ended question. The optional open-ended question was used to seek the participants' suggestions to improve our approach (i.e., what improvements to our approach would further help develop secure microservices systems?). We asked the participants to rate the level of their agreement or disagreement ("strongly agree = 5" to "strongly disagree = 1") with the following seven statements. We consulted the (microservices) security-related literature (e.g., [31], [13], [30], [15]) to design these seven statements.

- Statement 1. "*The approach is useful because security discussions identified by the approach convey meaningful and important security information*".

- Statement 2. "*The approach is useful because security discussions identified by the approach can be used to make informed security decisions in the future or refine the existing sub-optimum security decisions*".

- Statement 3. "*The approach is useful because I, as a practitioner, can find useful materials in a reasonable time slot from security discussions identified by the approach*".

- Statement 4. "*The approach is useful because security discussions identified by the approach can help us identify security-critical issues (e.g., security bugs, security mistakes, etc.) faster in our systems than if we do it manually*".

- Statement 5. "*The approach is useful because security discussions identified by the approach may contain incoming security-sensitive bug reports, with which they can be readily identified and more effectively prioritized and resolved*".

- Statement 6. "*The approach is useful because security discussions identified by the approach can provide cues/hints to trace backward and forward to security-critical artifacts (e.g., codes, services) and features*".

- Statement 7. "*The approach is useful because the security discussions identified by the approach could be beneficial for those who have little security experience in microservices or recently joined a microservices project as they can quickly access and learn security solutions and avoid common security mistakes*".

Participants. We recruited the participants in three ways. First, we invited all the 868 practitioners who were also asked to fill out the survey designed for RQ1. Then, we advertised our survey via social networks, such as LinkedIn (e.g., microservices groups on LinkedIn). Finally, we approached the relevant microservices practitioners on LinkedIn. We thoroughly analyzed their profiles and invited them directly via email or by sending a LinkedIn message. We received 51 responses finally. Due to the complexity of the recruitment process, we were not able to calculate the response rate.

Data Analysis. We used descriptive statistics to analyze the responses to the demographic and Likert scale questions. The responses to the open-ended question were analyzed using open coding and constant comparison [48].

5. Findings

This section first reports the results of the preliminary survey to answer **RQ1** (Section 5.1). Then, we present the findings of the experiments conducted to answer **RQ2** (Section 5.2). Finally, we report the findings of the validation survey to respond to **RQ3** (Section 5.3).
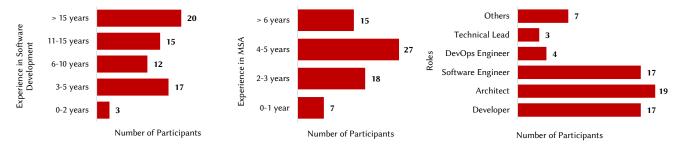
Figure 5: The preliminary survey: participants demographic data (n=67)
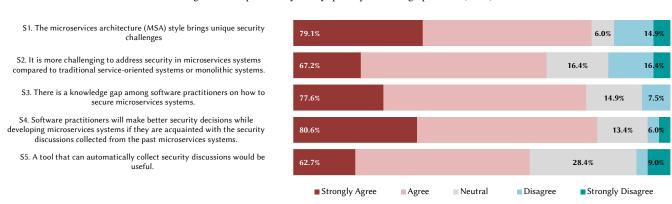

Figure 6: The preliminary survey: responses to the statements regarding security in microservices systems (n=67)

5.1. *Practitioners' Perspective on Security in Microservices Systems (RQ1)*

Figure 5 represents an overview of the participants' demographics. More than half of the participants (52.3%, 35 out of 67) had developed software for at least 11 years. Of 67 participants, only 3 had less than two years of software development experience. The majority of the participants (62.7%) were involved in microservices system development for more than 4 years. The dominant roles who participated in our survey were architects (19), followed by developers (17), software engineers (17), DevOps engineers (4), and technical leads (3).

Through 5 Likert scale questions, we sought the opinions of the respondents about security in microservices systems. As shown in Figure 6, the majority of them (79.1%) (strongly) agreed that the MSA style brings a unique challenge (statement S1). A participant pointed out the challenging nature of establishing security in microservices systems as follows: "*When we architect microservices, security is among the top things we nail down, i.e., will the microservice get exposed to the public, will the microservice perform authentication and authorization or does it trust all requests coming in*" (Developer).

On the other hand, only 10 (14.9%) participants (strongly) disagreed with this statement, and the rest (4 respondents, 6%) took a neutral position. When we asked "*it is more challenging to address security in microservices systems compared to traditional service-oriented systems or monolithic systems*", the participants rated this statement (S2) *strongly agree* (17.9%), *agree* (49.3%), *neutral* (16.4%), *disagree* (11.9%), and *strongly disagree* (4.5%). Our survey results reveal the knowledge gap amongst practitioners in securing microservices systems as

52 out of 67 respondents (77.6%) answered statement S3 as *strongly agree* or *agree*. No one rated statement S3 as *strongly disagree*. An architect believed that "*helping small teams develop secure microservices is important*". Another survey respondent shared his concern about the knowledge gap among practitioners in securing microservices deployed in a container: "*most people who implement microservices will deploy their services inside a container, and I heard/read that some of them assume that using container will by default is secure, this is misleading*" (Software Engineer). A software engineer elaborated on this and said, "*I think it would be a good idea to collect how people secure their microservices inside a container, did they use the public image? Did they inspect/scan image before using it, and so on*".

The respondents were asked to indicate to what extent they agree or disagree that the security discussions collected from the previous microservices systems can help practitioners make better security decisions while developing such systems (statement S4). The majority of the participants (80.6%) strongly agreed or agreed with this statement (S4). Less than 10% of the respondents rated statement S4 as *strongly disagree* or *disagree*. Our participants also confirmed the usefulness of the potential automated tools that can collect and detect security discussions from developer discussions (i.e., 16.4% strongly agree and 46.3% agreed with statement S5). While only 9% of the respondents (strongly) disagreed that such tools can be useful, statement S5 received the highest neutral responses (19 out of 67, 28.4%). One participant pointed out: "*Not sure how a tool can help in making a system secure. I believe Guidelines on Security can be a better alternative*" (Developer).

12

Table 4: Precision, Recall, F1-Score, AUC and G-mean for ML and DL models. The best results of each metric are grayed.

	Models	Precision	Recall	F1-score	AUC	G-mean
	BoW + RF	92.03	77.77	84.30	87.58	87.03
	BoW + DT	83.12	80.20	81.63	86.95	86.69
	BoW + XGBoost	93.43	71.20	80.82	84.63	83.56
	BoW + SVM-LR	86.92	83.08	84.96	89.12	88.92
	TF-IDF + RF	91.93	78.54	84.71	87.94	87.43
Machine learning	TF-IDF + DT	82.91	81.45	82.17	87.48	87.27
	TF-IDF + XGBoost	93.29	71.30	80.83	84.66	83.60
	TF-IDF + SVM-LR	91.65	78.47	84.55	87.85	87.35
	GloVe + RF	89.18	49.64	63.77	73.65	69.63
	GloVe + DT	52.90	55.72	54.28	68.27	67.11
	GloVe + XGBoost	82.46	59.48	69.11	77.30	75.22
	GloVe + SVM-LR	82.19	59.44	68.99	77.23	75.16
	DeepM1	86.73	84.25	85.47	89.63	89.07
Deep Learning	DeepM2	80.16	79.11	79.64	85.77	85.51
	DeepM3	84.01	82.36	83.18	88.15	87.96
Average		84.86	72.80	77.89	83.75	82.77

RQ1 Summary: *Although the preliminary survey participants perceive securing microservices systems as a unique challenge, they find collecting and leveraging developer security discussions scattered in existing microservices systems useful for making security decisions. Furthermore, a tool that automatically collects such security discussions is deemed useful by the participants.*

5.2. *Learning to Identify Security Discussions (RQ2)*

To answer RQ2, we evaluated 15 ML/DL models (See Section 4.2). ML models include Random Forest, Decision Tree, XGBoost, and SVM-LR algorithms based on three text feature selection techniques: BoW, TF-IDF, and GloVe. Three DL models (DeepM1, DeepM2, and DeepM3) use the word embedding technique. It should be noted that the training process of DL models is supposed to stop if there are no changes observed in the best accuracy in the last 10 epochs (i.e., early stopping). Table 4 shows the performance evaluation of ML and DL models using 10-fold cross-validation technique over 17,277 paragraphs. In Table 4, the grayed values show the best results for each metric. As shown in Table 4, ML and DL models achieve, on average, a precision of 84.86%, recall of 72.80%, F-score of 77.89%, AUC of 83.75%, and G-mean of 82.77%. Among all ML and DL models, DeepM1, as a DL model, performs the best in all metrics except in the precision metric. XGBoost with BoW achieved the best performance in the precision metric (precision: 93.43%). DeepM1, DeepM3, Decision Tree with TF-IDF and BoW, and SVM-LR with BoW achieved a performance of more than 80% in all metrics. The worst results were obtained when GloVe was used as the feature selection technique.

5.2.1. *Generalizability of Learning Models*

Motivation. All learning models are expected to be effective when performing on the new data that they were not trained on, particularly on the data that come from a range of domains (i.e., unseen data) [102], [103], [104]. This ability is referred to as generalization. We aim to measure how well the fifteen learning models detect security discussions in two independent datasets created from different contexts (i.e., out-of-sample accuracy [27], [56]).

Approach. First, we created two datasets. The first dataset (i.e., GitHubGen dataset) consists of 500 paragraphs from 2 projects (i.e., *scalecube-services and moleculer* projects) listed in Table 2. 250 paragraphs were randomly selected from each project. The second dataset (i.e., StackExGen dataset) is composed of 77 paragraphs (i.e., 77 paragraphs with HTML tag <p> from 7 posts) taken from Security Stack Exchange posts with the "microservices" tag. The next step included inviting two annotators to determine the presence of security discussions in these 577 paragraphs. Two of the authors acted as an annotator. An author was assigned to annotate 500 paragraphs of GitHubGen dataset, and another author annotated 77 paragraphs of StackExGen dataset. After finishing the annotation process, the second author randomly checked 185 of the annotated paragraphs (i.e., confidence level: 90% and margin of error: 5%), and any conflicts and disagreements among the annotators were resolved through internal discussions. It was found that GitHubGen dataset includes 142 security discussions and 358 non-security discussions, and StackExGen dataset consists of 63 security points and 14 non-security discussions. In the last step, we evaluated the previously trained models listed in Table 4 on GitHubGen dataset, StackExGen dataset, and their combination (i.e., GitHubGen + StackExGen dataset).

Result. Table 5 shows that the performance of Decision Tree with TF-IDF, DeepM1, and DeepM2 are not significantly

Table 5: Performance of fifteen learning models on two unseen datasets. *GitHubGen dataset*: 500 paragraphs from 2 microservices systems; *StackExGen dataset*: 77 paragraphs from 7 posts of Security Stack Exchange.

Unseen Datasets	Models	Precision	Recall	F1-score	AUC	G-mean
	BoW+RF	96.34	55.63	70.54	77.39	74.27
	BoW+DT	89.52	66.19	76.11	81.56	80.10
	BoW+XGBoost	100	46.47	63.46	73.23	68.17
	BoW+SVM-LR	80.39	57.74	67.21	76.07	73.83
	TF-IDF+RF	96.20	53.52	68.78	76.34	72.85
	TF-IDF+DT	85.59	71.12	77.69	83.18	82.30
	TF-IDF+XGBoost	98.44	44.36	61.17	72.04	66.51
GitHubGen	TF-IDF+SVM-LR	95.29	57.04	71.37	77.96	75.10
	GloVe+RF	84.62	15.49	26.19	57.18	39.14
	GloVe+DT	36.59	31.69	33.96	54.95	49.78
	GloVe+XGBoost	65.62	29.57	40.78	61.71	52.68
	GloVe+SVM-LR	61.64	31.69	41.86	61.93	54.04
	DeepM1	86.71	78.16	82.22	86.71	86.28
	DeepM2	82.30	75.35	78.67	84.46	83.97
	DeepM3	85.88	51.40	64.31	74.02	70.48
Average		78.90	51.03	61.62	73.25	68.63
	BoW+RF	96.08	77.77	85.96	81.74	81.64
	BoW+DT	98.08	80.95	88.70	86.90	86.70
	BoW+XGBoost	97.87	73.01	83.64	82.93	82.34
	BoW+SVM-LR	98.11	82.53	89.66	87.69	87.54
	TF-IDF+RF	98.00	77.77	86.73	85.31	84.98
	TF-IDF+DT	96.15	79.36	86.96	82.53	82.47
	TF-IDF+XGBoost	97.87	73.01	83.64	82.93	82.34
StackExGen	TF-IDF+SVM-LR	98.08	80.95	88.70	86.90	86.70
	GloVe+RF	93.94	49.20	64.58	67.46	64.94
	GloVe+DT	93.62	69.84	80.00	74.20	74.07
	GloVe+XGBoost	95.00	60.31	73.79	73.01	71.90
	GloVe+SVM-LR	90.48	60.31	72.38	65.87	65.63
	DeepM1	98.18	85.71	91.51	89.28	89.21
	DeepM2	96.55	88.88	92.56	87.30	87.28
	DeepM3	98.14	84.12	90.59	88.49	88.38
Average		96.41	74.91	83.96	81.50	81.07
	BoW+RF	97.58	59.02	73.56	79.10	76.51
	BoW+DT	91.03	69.26	78.67	82.75	81.64
	BoW+XGBoost	99.12	54.63	70.44	77.18	73.81
	BoW+SVM-LR	86.45	65.36	74.44	79.86	78.53
	TF-IDF+RF	95.56	62.92	75.88	80.65	78.68
	TF-IDF+DT	89.09	71.70	79.46	83.43	82.60
GitHubGen	TF-IDF+XGBoost	98.20	53.17	68.99	76.31	72.72
+	TF-IDF+SVM-LR	96.35	64.39	77.19	81.52	79.70
StackExGen	GloVe+RF	91.53	26.34	40.91	62.49	50.97
	GloVe+DT	51.72	43.90	47.49	60.66	58.30
	GloVe+XGBoost	76.92	39.02	51.78	66.28	60.42
	GloVe+SVM-LR	72.17	40.48	51.88	65.94	60.83
	DeepM1	90.16	80.48	85.05	87.82	87.51
	DeepM2	86.70	79.51	82.95	86.39	86.12
	DeepM3	90.64	61.46	73.25	78.98	77.01
Average		87.55	58.11	68.80	76.62	73.69

Table 6: Two examples of developer discussions labeled by DeepM1

Security Discussion Samples	How DeepM1 predicts?
"@DeividasJackus I agree that microsystems made of components should be isolated and internal calls should not leak outside its boudaries. The issue is that you are thinking of a microsystems as a single node for the sake of optimization and visibility. I think that @icebob already solved the former by calling local service first, and the latter should be resolved with a location independent solution." [Taken from issue 124[12]]	Correctly
"A general solution I've used in the past is Queue TTL policy. It has its own downsides but it felt a little safer to me." [Taken from issue 341[13]]	Incorrectly

degraded across GitHubGen dataset, StackExGen dataset, and GitHubGen + StackExGen dataset. All metrics' values obtained for StackExGen dataset are higher than those for GitHubGen dataset. Particularly, XGBoost With BoW achieves a value of 100% in the precision metric on GitHubGen dataset. As shown in Table 5, F1-score values obtained from four machine learning algorithms (i.e., RF, DT, XGBoost, and SVM-LR) with GloVe have the worst performance in all datasets. This may stem from the fact that GloVe uses a pre-trained dataset for feature selection.

Figure 7 shows the results of applying DeepM1 on the combination of GitHubGen and StackExGen. DeepM1 could correctly label 519 of 577 paragraphs. Of 58 mislabeled paragraphs, 40 had to be recognized as security discussions, but they were wrongly labeled as a non-security discussion. Table 6 shows a paragraph from issue 124[12] of the *moleculer* project that was correctly predicted by DeepM1 as a security discussion. It also displays a paragraph from issue 341[13] labeled by a human expert as a security discussion, which DeepM1 incorrectly classified it as a non-security discussion.
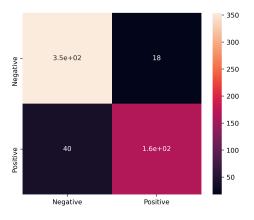
Figure 7: Confusion Matrix of DeepM1 for combination of GitHubGen dataset and StackExGen dataset

---

[12]https://bit.ly/3pgxnjg
[13]https://bit.ly/3igLEuL

5.2.2. Comparison with Baselines

Motivation. To the best of our knowledge, no study develops learning models to identify security discussions from microservices practitioner discussions. Despite this, we compared our best-performing model (DeepM1) with three baselines. Similar to [105], [106], [107], we developed a keyword-based search as the first baseline. The result of the keyword-based search also helps us understand if the identification of security discussions in microservices systems is a learning problem or not [105]. Two studies [98] and [31] are the closest works to our study. The study [98] developed PUMiner to identify security posts in Stack Overflow, and the study [31] introduced SecureReqNet to distinguish security issues from non-security issues in the GitHub repository. We used these two approaches as two other baselines.

Approach. Here we describe how the three baselines are implemented.

Keyword-based Search (Baseline1). As we described in Section 3.2, the annotators were asked to identify security-related words in security discussions. We used these security-related words to build our keyword-based search. Some of these security-related words appeared in security discussions as abbreviations, and others were in full forms. For example, we had "single sign on", "sso", and "single-sign-on" in the collected security discussions. In the first step, we converted such security-related words to one format (i.e., abbreviation). This resulted in 165 unique security keywords. We then applied the stemming process (see Step 4 in Section 4.2.1) on paragraphs in unseen datasets. Like [106], [107], our keyword-based search calculates the probability of a stemmed paragraph to include one or more of the 165 unique security keywords. To be more precise, suppose P is a stemmed paragraph, and SL is a list including all 165 unique security keywords. We define N_{SL} as the number of tokens in P that emerge in SL and N_P as the number of tokens in P. The keyword-based search measures the probability that P is classified as a security discussion with the following formula: $Pr_{(P,SW)} = \frac{N_{SW}}{N_P}$. It should be noted that there might be cases where some paragraphs are mistakenly labeled as security discussion because just one security-related word is included in P. To avoid this, the keyword-based search labels P as a security discussion if $Pr_{(P, SW)}$ is higher than 0.05 (i.e., security words compose at least 5% of a paragraph) [106], [107].

PUMiner (Baseline2) and SecureReqNet (Baseline3). To

Table 7: Comparison of our best-performing model (DeepM1) with three baselines on unseen datasets. Note that PUMiner[+] and SecureReqNet[+] are a modified version of PUMnier and SecureReqNet, respectively, in which their data pre-processing is replaced with our data pre-processing.

Models	Metrics	GitHubGen	Improv.	StackExGen	Improv.	GitHubGen + StackExGen	Improv.
DeepM1	Precision	86.71	—	98.18	—	90.16	—
	Recall	78.16	—	85.71	—	80.48	—
	F1-score	82.22	—	91.51	—	85.05	—
	AUC	86.71	—	89.28	—	87.82	—
	G-mean	86.28	—	89.21	—	87.51	—
Keyword-based Search	Precision	23.08	3.756 x	82.93	1.183 x	36.41	2.476 x
	Recall	23.24	3.363 x	53.97	1.588 x	32.68	2.462 x
	F1-score	23.16	3.550 x	65.38	1.399 x	34.45	2.468 x
	AUC	46.26	1.874 x	51.98	1.717 x	50.62	1.734 x
	G-mean	40.12	2.150 x	51.95	1.717 x	47.33	1.848 x
PUMiner	Precision	26.01	3.333 x	89.13	1.101 x	32.58	2.767 x
	Recall	72.53	1.077 x	65.10	1.316 x	70.24	1.145 x
	F1-score	38.29	2.147 x	75.23	1.216 x	44.51	1.910 x
	AUC	45.35	1.912 x	64.68	1.380 x	45.07	1.948 x
	G-mean	36.29	2.377 x	64.68	1.379 x	37.38	2.341 x
SecureReqNet	Precision	65.81	1.317 x	93.88	1.045 x	74.10	1.216 x
	Recall	54.22	1.441 x	73.02	1.173 x	60.00	1.341 x
	F1-score	59.46	1.382 x	82.14	1.114 x	66.31	1.282 x
	AUC	71.53	1.212 x	75.79	1.177 x	74.22	1.183 x
	G-mean	69.40	1.243 x	75.74	1.177 x	72.84	1.201 x
PUMiner[+]	Precision	44.94	1.929 x	82.81	1.185 x	54.96	1.640 x
	Recall	56.34	1.387 x	84.12	1.018 x	64.88	1.240 x
	F1-score	50.00	1.644 x	83.46	1.096 x	59.51	1.429 x
	AUC	64.48	1.344 x	52.77	1.691 x	67.79	1.295 x
	G-mean	63.96	1.348 x	42.46	2.101 x	67.73	1.292 x
SecureReqNet[+]	Precision	64.41	1.364 x	94.23	1.041 x	73.53	1.226 x
	Recall	53.52	1.460 x	77.78	1.101 x	60.98	1.319 x
	F1-score	58.46	1.406 x	85.22	1.073 x	66.67	1.275 x
	AUC	70.89	1.223 x	78.17	1.142 x	74.44	1.179 x
	G-mean	68.73	1.255 x	78.17	1.141 x	73.21	1.195 x

make the comparison fair, we replicated PUMiner and SecureReqNet in two distinct ways. First, we replicated them with their own data pre-processing and trained them with our dataset using 10-fold cross-validation. Second, we replaced PUMiner's and SecureReqNet's data pre-processing with our data pre-processing described in Section 4.2.1 and call them PUMiner[+] and SecureReqNet[+] respectively. We trained PUMiner[+] and SecureReqNet[+] with our dataset using 10-fold cross-validation. We evaluated the performance of all baselines (the keyword-based search and the trained PUMiner, SecureReqNet, PUMiner[+], SecureReqNet[+]) on unseen datasets (GitHubGen, StackExGen, and their combination) described in Section 5.2.1.

Result. Table 7 shows that DeepM1 outperforms all baselines on all unseen datasets in all metrics, with improvements ranging from 1.018x to 3.756x. SecureReqNet[+] has only comparable performance with DeepM1 on StackExGen dataset in the recall metric (i.e., recall value achieved by SecureReqNet[+] is 84.12% versus 85.71% recall value obtained by DeepM1). Except for this, the performance of all baselines is significantly lower than that of DeepM1 in all metrics on unseen datasets. Also, as shown in Table 7, PUMiner[+] and SecureReqNet[+] have a better performance than when they use their own pre-

processing (i.e., PUMiner and SecureReqNet).

RQ2 Summary. *The results show that the performances of both ML and DL models in identifying security discussions are promising: **precision** ($\overline{84.86\%}$), **recall** ($\overline{72.80\%}$), **F1-score** ($\overline{77.89\%}$), **AUC** ($\overline{83.75\%}$), and **G-mean** ($\overline{82.77}$). However, DeepM1, as a deep learning model, is the best performing learning model. Additionally, DeepM1's performance of detecting security discussions on unseen datasets (generalizability) is not significantly degraded. Finally, the evaluation demonstrates that DeepM1 outperforms the state-of-the-art baselines with improvements ranging from 1.018x to 3.756x in all metrics.*

5.3. Usefulness of Automated Identification of Security Discussions (RQ3)

As presented in Section 4.3, we conducted a survey (i.e., validation survey) to solicit the perceptions of practitioners on the usefulness of the security discussions detected by the best-performing learning model (i.e., DeepM1).
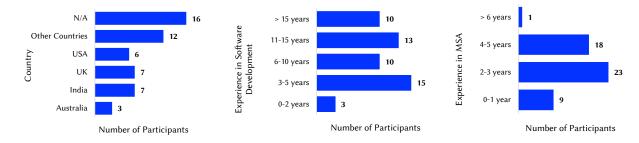
Figure 8: The validation survey: participants demographic data (n=51)
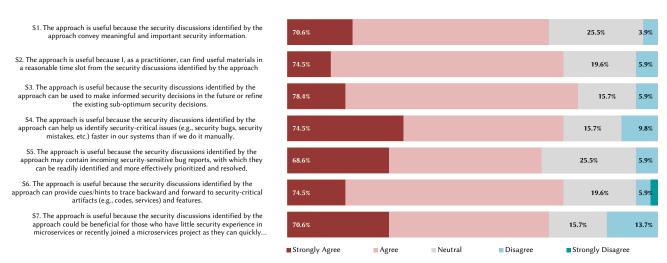

Figure 9: The validation survey: responses to the statements regarding the usefulness of detected security discussions using the best learning model (n=51)

Figure 8 shows the demographic information of 51 practitioners who completed the validation survey. 16 participants did not reveal which country they were working in. The rest 35 respondents came from 16 countries, in which India and UK, each with seven participants, were dominant. Almost 65% (33) of the respondents indicated that they had more than six years of software development experience in the industry. 15 participants had 3-5 years of experience, and three respondents developed software for less than two years. Of the 51 respondents, nine had less than one year of experience in microservices system development, 23 had 2-3 years experience, 18 had 4-5 years experience, and one had more than six years experience.

Figure 9 shows the feedback of the 51 practitioners on the usefulness of the results (e.g., distinguished security discussions from non-security discussions) generated by the best-performing learning model (i.e., DeepM1). We observed that almost 71% of the respondents (strongly) agreed that the security discussions identified by DeepM1 convey meaningful and important security information (Statement S1). 25.5% of the participants were neutral about Statement S1, and only 3.9% disagreed. They further indicated that identifying useful materials and information from the detected security discussions can be done in a reasonable time slot (Statement S2). 74.5% rated this statement as *strongly agree* or *agree*.

Statements S3 to S7 investigate how the detected security discussions can be helpful. Approximately 79% of the respondents answered Statement S3 as *strongly agree* or *agree*, indi-

cating that security discussions help them make informed security decisions in the future or refine the existing sub-optimum security decisions. Another benefit that the security discussions extracted from developer discussions can bring is to help practitioners find security-critical issues (e.g., security bugs, security mistakes) in their systems faster than if they do it manually (Statement S4). 74.5% strongly agreed or agreed with Statement S4, and 9.8% disagreed with this statement. In close to 70% of the responses to Statement S5, the respondents felt that the identified security discussions could help trace backward and forward to security-critical artifacts (e.g., codes, services) and features. Another benefit of security discussion is that security-sensitive bug reports can be readily identified and more effectively prioritized and resolved. It is because security discussions may contain incoming security-sensitive bug reports (74.5% strongly agreed or agreed with Statement S6, 19.6% were neutral, and 5.9% strongly disagreed or disagreed with this statement). Regarding Statement S7, 70.6% believed that the information in the identified security discussions could help newcomers and less experienced team members quickly access and learn security solutions and avoid common security mistakes.

RQ3 Summary: *The validation survey respondents confirm that the security discussions detected by the best-performing learning model (i.e., DeepM1) have promise for application into securing microservices systems, particularly in helping practitioners make informed security decisions in the future or refine the existing sub-optimum security decisions.*

6. Discussion

This section discusses the implications for practitioners and researchers based on our key findings and analysis of the responses to an open-ended question: "*What improvements to our approach would further help develop secure microservices systems?*".

Better support is still needed. The challenging nature of security in the MSA style (confirmed by the preliminary survey) poses difficulties for all microservices practitioners in general, and novice and non-security microservices developers in particular, to comprehend security in microservices systems and effectively adopt and implement best security practices. The validation survey confirms that microservices practitioners can identify and learn succinct security knowledge with reasonable time and effort from the results (security discussions) produced by the learning models. This matter becomes much more critical when we know that the security discussions investigated in this study include security solutions and principles. This could be more beneficial for those who have little security experience or recently joined a microservices project. They can quickly be aware of the up-to-date security solutions and best practices to secure microservices systems. However, our participants emphasized that microservices practitioners still need more support to design and implement secure microservices systems. New tools can be developed, or our models can be extended to automatically link security discussions in developer discussions to relevant learning documents and materials (e.g., blogs, code snippets). Further to this, for a detected security issue in a given microservices system, new tools should be able to come up with possible security solutions with relevant references.

Prioritize security discussions. Our learning models can only distinguish security discussions from non-security discussions. Our validation survey reveals that security discussions detected from developer discussions can help developers to identify security-critical services or security-critical issues (e.g., security bugs) in microservices systems. The importance of identifying such security-critical services and issues in the MSA style is increasing as the entire system may go down due to a single compromised microservice [12], [13] . Despite this, the participants commented that security discussions could be labeled with different priority levels, such as critical, important, or low-impact. Determining the types and severity of security discussions can help a software team assign expert members to address and audit more security-critical issues in a faster time [31]. Hence, there is a need to develop tools that label security discussions based on their types and severity.

Design context and security discussions. In this study, we chose to label and identify security discussions at the level of the paragraph. Viviani et al. [27] also identified design points ("*a piece of a discussion relating to a decision about a software system's design*") at the level of the paragraph in developer discussions. This decision was made to enable microservices practitioners to capture, identify, and share meaningful and succinct security knowledge faster and more accurately without going through an entire developer discussion to find and understand a security issue, decision, or solution. Despite this, developers may need more design context information to thoroughly understand some security discussions detected by our learning models and effectively apply security solutions discussed in the detected security discussions in practice. The participants mentioned the need to develop tools that can extract the relevant design context information for each detected security discussion and make them available as extra and optional resources (e.g., through external links) for microservices practitioners. Such design context information can even be extracted from non-security discussions.

7. Threats to Validity

This section lists possible threats to validity related to our research method and findings and the adopted mitigating strategies to these threats. We classify the threats into three types: internal, construct, and external [108].

7.1. Internal Validity

Surveys. The selection of microservices practitioners was a challenge for our surveys. In the preliminary survey, the participants were either the contributors to open-source microservices systems or the contributors to open-source projects that develop tools and frameworks for microservices systems. We also invited microservices practitioners working in the industry for the validation survey after analyzing their LinkedIn profiles. We are confident that most participants had enough experience and expertise in the MSA style. Specific roles may produce bias in the survey results. While software engineers, developers, and architects were the dominant roles in our surveys, other roles such as DevOps engineers, team leads, testers, etc., participated in our survey as well.

Experiments. The annotation process may have introduced two threats. The first one is the annotators' subjective bias. We implemented two strategies to minimize this threat. (1) We carried out a pilot study to reach a common understanding of the characteristics of security and non-security discussions. This step also helped us develop a robust coding schema for the annotation process. (2) We employed several annotators in the pilot annotation process and the main annotation process to reduce the potential personal bias (i.e., investigator triangulation [109]). The second threat that might have happened is that the annotators erroneously labeled the paragraphs. The measure taken to minimize this threat was that the second author, who was not involved in the annotation process, cross-checked 581 paragraphs out of 17,277 paragraphs (i.e., confidence level:

18

95% and margin of error: 4%). We also obtained the value of 0.82 for Cohen's Kappa Coefficient over the 581 cross-checked paragraphs, indicating an agreement between the cross-checker and the annotators. The second author held several meetings with the annotators to discuss the annotations and resolve any disagreements. Furthermore, the annotators had extensive experience in coding, design, and microservices systems. They ranged from a software engineer to a professor. With these efforts, we believe that the dataset is credible with minimum mislabeled paragraphs.

Given our unit of analysis is the paragraph, as we discussed in Section 6, our learning models may identify security discussions that lack enough design context information. We accept that practitioners may not understand completely some security discussions detected by our learning models. However, we believe that most security discussions (e.g., SD1 to SD4 in Figure 1) identified by our learning models include useful and necessary security information.

7.2. Construct Validity

Surveys. Practitioners may have different interpretations and understandings of "security discussion" in microservices systems. Our strategy to moderate this threat was defining "security discussion" at the beginning of both the preliminary survey and validation survey. We also provided some examples of security discussions from GitHub and Stack Overflow. It should be noted that none of the participants showed disagreement with our definition regarding security discussions. We only used one research method (the validation survey) to evaluate the usefulness of our approach (i.e., RQ3). Our validation survey may not have shown precisely and comprehensively all benefits and possible deficiencies (e.g., creating additional effort for practitioners) of our approach in the daily work of microservices practitioners. Other research methods, such as controlled experiments and in-depth qualitative studies, should be conducted to assess different aspects of our approach.

Experiments. The selection of metrics and ML/DL models can be another source of threat. We used a wide variety of metrics to show various aspects of the performance of the learning models. Among these four metrics, recall, precision, and F1-score metrics are widely used and highly recommended in the software engineering literature (e.g., [27], [33], [36], [96], [110]). As our dataset was imbalance, we employed the G-mean metric to show the effectiveness of ML/DL models with one number [111]. This work used fifteen classification models, including ML and DL models, among many possible learning models. Peters et al. [99] argued that it is not possible to use all classifiers in a single case study. It was also found that out of the 22 studied ML classifiers, the top 17 classifiers do not significantly differ in terms of performance [96]. Three ML classifiers used in our study are from the 17 top-ranked classifiers. We also built three DL models using Bi-LSTM and CNN. The DL models based on Bi-LSTM and CNN have shown desired results [73]. Thus, we acknowledge that our experiment results may differ if we use other ML/DL models.

Surveys. The number of responses gathered from the preliminary survey and validation survey can limit their findings. We only received 67 responses for the preliminary survey and 51 responses for the validation survey. Further, the majority of the participants who completed the preliminary survey were contributors to open-source projects. Hence, most observations in the preliminary survey are exclusive to open-source microservices systems. In the validation survey, we tried to reduce this threat by recruiting practitioners who worked in the software industry and contributors to open-source projects. However, we acknowledge that the results obtained from the surveys may not be generalized to all microservices practitioners, types of projects (e.g., closed source software), and companies.

Experiments. The dataset used in this work is composed of 17,277 paragraphs (i.e., 4,813 security discussions and 12,464 non-security discussions) extracted from two sources: issue discussions collected from five relatively large-scale open-source microservices systems and 498 Stack Overflow posts. We also applied all developed learning models to 500 unseen paragraphs from two open-source microservices systems and 77 unseen paragraphs from seven Security Stack Exchange posts with the "microservices" tag (Section 5.2.1). Hence, we admit that these open-source projects and Stack Overflow posts are not representative of all projects hosted on GitHub and Stack Overflow posts. Another limitation is that we only focused on issue discussions in the selected projects. Our dataset could be more comprehensive by considering and analyzing the other elements of the selected open-source projects, such as their source codes, comments, commits, and design documents. Finally, we acknowledge that our findings may not be generalizable, particularly to projects from different domains, closed source projects, and projects hosted on other platforms such as GitLab and BitBucket. We have made our datasets and the implementation of ML and DL models publicly available [34] to enable other researchers and practitioners to use, replicate, validate, or extend our study.

8. Related Work

To the best of our knowledge, there is no study that uses learning models to identify security information in developer discussions (such as GitHub issues and Stack Overflow posts) of microservices systems. We summarize the studies that discuss security support in microservices systems (Section 8.1), and discuss the studies that use ML- and DL-based models to recognize security contents in textual software artifacts (Section 8.2). In Section 8.3, we discuss the differences of our study with existing studies.

8.1. Security in Microservices Systems

Pereira et al. [18] conducted a systematic mapping of security mechanisms in MSA, revealing that most studies focus on "unauthorized access", "sensitive data exposure", and "compromising individual microservices". They also found that prevention and auditing based solutions are widely used to secure

microservices. In another review study, Waseem et al. [11] observed a lack of concrete solutions to achieve and address monitoring, security, and performance in microservices systems in DevOps. Yu et al. [14] surveyed and classified 19 security issues related to microservices communication into four categories: containers (e.g., kernel exploit, malicious process), data (e.g., data intercept, secret leak), permission (e.g., identify spoofing), and network (SDN issues, DOS attack). Based on identified issues, they proposed a solution to address security issues in microservices-based fog applications. Ghofrani and Lübke [16] surveyed 25 microservices experts to evaluate current practices and challenges in the MSA style. One of the main conclusions of their survey is that practitioners prioritize security, response time, and performance over other quality attributes (e.g., resilience) in microservices systems.

Sun et al. [112] built a monitoring and policy enforcement infrastructure using API primitive FlowTap. Their solution provides security-as-a-service to detect and block internal and external threats over the network for microservices-based cloud systems. Pahl and Donini [12] introduced a certificate-based (X.509) method for securing IoT microservices. The method helps the distributed nodes verify the security properties locally and change certificate properties across the distributed IoT nodes. A multilateral (e.g., confidentiality, accountability, and integrity) security framework was proposed in [113] to assess the security design and architecture quality for docker-based applications (e.g., microservices). The security framework provides multi-layered security by considering the concepts of TCP/IP, OSI stack, and cloud stack model. Ahmadvand and Ibrahim [114] suggested a method that considers security and scalability as the first-class citizens when decomposing monolithic systems into microservices. Their approach uses an asset-threat-risk triangle to elicit and capture security requirements and decisions.

The increasing importance of security in microservices systems and the lack of concrete security solutions and guidelines for such systems inspired us to explore and extract the security knowledge scattered in practitioner discussions.

8.2. *Learning Models for Recognizing Security Contents*

Yang et al. [115] identified 30 security topics (e.g., SQL injection, Timing attack) on Stack Overflow posts using Latent Dirichlet Allocation (LDA) tuned by the Genetic Algorithm (GA). The 30 topics are grouped into "web security", "mobile security", "system security", "cryptography", and "software security". Le et al. [98] introduced PUminer to identify security posts from Stack Overflow and Security Stack Exchange posts using a small number of labeled security posts when a considerable amount of posts are unlabeled. Based on the experiments on more than 17.3 million posts, PUMiner showed more than 0.85% F1-score and G-mean in recognizing security posts from Stack Overflow and Security Stack Exchange.

Palancio et al. [31] used several variations of SecureReqNet's CNN architecture to distinguish security issues from non-security ones in GitHub and GitLab. The SecureReqNet approach yielded, on average, an accuracy of 81.29% on open-source issues and 69.77% on the requirements of commercial

software systems. Han et al. [116] developed a DL approach to determine the difficulty level of a software vulnerability based on the vulnerability description. The approach's assessment showed its effectiveness in short-text vulnerability descriptions.

Pletea et al. [117] constructed a dataset of security-related terms to carry out sentimental analysis of security discussions on GitHub. Marrison et al. [118] found that each project or domain has unique security keywords, and using project-specific security keywords for predicting security issues of the same project leads to better performance compared to using general security keywords or security keywords from other projects. Tao et al. [49] introduced a Security-Related Review Miner (SRR-miner) that uses a keyword-based technique to extract security-related review sentences on 17 mobile applications in Google Play.

A set of studies have developed ML-based approaches to discriminate security requirements from other types of requirements (e.g., [32], [33], [36], [119]). In [32], the focus was on identifying security controls, an important task in assessing security requirements, and it was found that applying cost-sensitive learning (CSL) and SMOTE (Synthetic Minority Over-sampling TEchnique) methods to imbalanced classes would improve the performance of classifiers.

Others have focused on identifying security bug reports (SBRs). Peters and colleagues [99] proposed a framework to decrease the chance of mislabeling SBRs. The proposed framework achieved this goal by leveraging filtering and ranking techniques. In another study, Jiang et al. [120] utilized *learning to rank* and *word embedding* techniques for this purpose. While the previous works use the entire bug reports, Pereira et al. [121] predicted SBRs only based on the title of SBRs and in the presence of noisy labels (0.98% AUC).

8.3. *Differences with Existing Studies*

While the techniques mentioned above yield acceptable performance, our focus, context, and findings are different from the existing techniques:

(1) They are not specifically designed for MSA. It is argued that apart from standard security terms (e.g., password), which may be found in the security texts of any projects or domains, each project or domain has its own unique security-related terms [118], [119]. Our preliminary survey findings also reveal that security concerns in microservices systems are unique compared to other types of systems, such as monoliths. This denotes that security discussions that happen among microservices practitioners are unique.

(2) Our work is different from the studies (i.e., [32], [33], [36], [119]) focusing on distinguishing security requirements from non-security requirements. Requirements specifications mostly focus on functional requirements and lack the security design decisions, design rationale, the security issues faced by developers, and security solutions employed to address security challenges.

(3) The techniques proposed in [99], [120], [121] to discriminate security bug reports from non-security bug reports are also different from our work. An issue in an issue tracking system

20

might include various information, such as a task, new feature, or bug [28]. Hence, bug reports can only capture, *if any*, a small part of developer discussions and decisions [111]. Moreover, as a bug report is usually written when a bug happens, it does not document all security issues developers face and the security solution options they consider.

(4) One key design decision that we made when developing our dataset and learning models was to focus on paragraphs. In contrast to this key decision, Le et al. [98] tried to distinguish security contents from non-security ones at the granularity of an Stack Overflow post (i.e., each post includes a question, a list of answers, and the appended comments [43]). The studies [31], [118] do the same task at the granularity of an issue (i.e., with all comments on the issue) in issue tracking systems. We made that decision because Viviani et al. [27] found that a design decision can be presented and identified in a paragraph. Moreover, Xu et al. [42] revealed that a paragraph could summarize useful information of a lengthy answer post in Stack Overflow.

GitHub issues and Stack Overflow posts, particularly the lengthy ones, may discuss different topics (e.g., security, performance) [27]. It may also be difficult for practitioners to identify useful information from lengthy posts and issues [42]. Imagine post number 2283937[14] and issue number 107 [3], which have 25872 and 14233 words, respectively, and discuss different topics. Suppose the English silent reading rate is 238 words per minute [122]. If post number 2283937 is correctly labeled as security, a developer will still spend one hour and fifty minutes to read this post and probably spend more time to find which part(s) of the post discusses security. Hence, we argue that the security discussions detected by our learning models, compared to the ones identified by the models in [98] and [31], include succinct security knowledge. This enables practitioners to find the required security information from such security discussions in a more reasonable time and with less effort.

(5) Among the learning models mentioned above aiming to discriminate between security contents and non-security contents, PUMiner [98] and SecureReqNet [31] are the closest to our work. The comparison conducted in Section 5.2.2 shows that our best-performing learning model (DeepM1) significantly outperformed two configurations of PUMiner by 42.91-91.08% and two configurations of SecureReqNet by 27.56-28.26% in F1-score when they were applied on an unseen dataset including both GitHub and Security Stack Exchange data. Further to this, the performance of PUMiner and SecureReqNet on GitHubGen unseen dataset (GitHub data) was lower than once they were applied on StackExGen unseen dataset (Security Stack Exchange data) and the combination of GitHubGen and StackExGen dataset. Several factors may explain why DeepM1 achieves significantly higher performance. (1) PUMiner and SecureReqNet need an entire Stack Overflow post or GitHub issue to differentiate security contents from non-security contents. At the same time, DeepM1 needs smaller chunks of information (paragraphs) from the GitHub issue or Stack Overflow post for this purpose. (2) As discussed earlier

and confirmed by the preliminary survey, the nature of security discussions in microservices systems is different from security discussions in other systems. (3) PUMiner needs an ultra-large-scale data input for training to show its performance. (4) DeepM1 uses a combination of CNN and LSTM networks to consider both different textual features and context information in developer discussions.

9. Conclusion and Future Work

There is a knowledge gap among practitioners and organizations in designing and implementing secure microservices systems. This study tried to (partially) bridge this gap by automatically identifying security discussions from previous microservices developer discussions. We first conducted a preliminary survey with 67 microservices practitioners to understand their perspectives on security in microservices systems. The findings of the preliminary survey showed that securing microservices systems is a unique challenge for software practitioners. The survey respondents also confirmed the usefulness of (automatically) identifying and then leveraging developer security discussions scattered in existing microservices systems in making security decisions.

Hence, we have developed fifteen machine/deep learning models to identify security discussions from developer discussions (GitHub issues and Stack Overflow posts) of microservice systems. Our results show that DeepM1, a deep learning model, performs the best. On average, the developed machine/deep learning models could achieve 84.86% precision, 72.80% recall, 77.89% F1-score, 83.75% AUC, and 82.77% G-mean. We further solicited microservices practitioners' perception of the usefulness of the results produced by DeepM1 through another survey (i.e., validation survey). The respondents to the validation survey acknowledged that the security discussions detected by DeepM1 could have promising applications in practice. Notably, the security discussions can help practitioners make informed security decisions in the future or refine the existing sub-optimum security decisions and identify security-critical issues (e.g., security bugs, security mistakes, etc.) faster in microservices systems.

We plan to apply our models to a range of commercial microservices systems in the future. We also intend to qualitatively analyze security discussions to build an evidence-based knowledge about best security practices for microservices systems. Further to this, we aim to extend our models to identify and classify the types of security decisions in developer discussions.

---

[14]https://bit.ly/369XnFF

21

References

[1] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, L. Safina, Microservices: yesterday, today, and tomorrow, in: Present and Ulterior Software Engineering, Springer, 2017, pp. 195–216.

[2] M. Fowler, J. Lewis, Microservices a definition of this new architectural term (2014).
URL https://bit.ly/3zk5xXr

[3] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, S. Tilkov, Microservices: The journey so far and challenges ahead, IEEE Software 35 (3) (2018) 24–35.

[4] S. Newman, Building Microservices: Designing Fine-Grained Systems, O'Reilly Media, Inc., 2015.

[5] A. Balalaie, A. Heydarnoori, P. Jamshidi, Microservices architecture enables devops: Migration to a cloud-native architecture, IEEE Software 33 (3) (2016) 42–52.

[6] D. Taibi, V. Lenarduzzi, C. Pahl, Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation, IEEE Cloud Computing 4 (5) (2017) 22–32.

[7] P. Di Francesco, P. Lago, I. Malavolta, Architecting with microservices: A systematic mapping study, Journal of Systems and Software 150 (2019) 77–97.

[8] V. Heorhiadi, S. Rajagopalan, H. Jamjoom, M. K. Reiter, V. Sekar, Gremlin: Systematic resilience testing of microservices, in: Proceedings of the 36th IEEE International Conference on Distributed Computing Systems (ICDCS), IEEE, 2016, pp. 57–66.

[9] R. Heinrich, A. van Hoorn, H. Knoche, F. Li, L. E. Lwakatare, C. Pahl, S. Schulte, J. Wettinger, Performance engineering for microservices: research challenges and directions, in: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion, ACM, 2017, pp. 223–226.

[10] S. Haselböck, R. Weinreich, G. Buchgeher, An expert interview study on areas of microservice design, in: Proccedings of the 11th IEEE Conference on Service-Oriented Computing and Applications (SOCA), IEEE, 2018, pp. 137–144.

[11] M. Waseem, P. Liang, M. Shahin, A systematic mapping study on microservices architecture in devops, Journal of Systems and Software 170 (2020) 110798.

[12] M.-O. Pahl, L. Donini, Securing IoT microservices with certificates, in: Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS), IEEE, 2018, pp. 1–5.

[13] T. Yarygina, A. H. Bagge, Overcoming security challenges in microservice architectures, in: Proceedings of the 12th IEEE Symposium on Service-Oriented System Engineering (SOSE), IEEE, 2018, pp. 11–20.

[14] D. Yu, Y. Jin, Y. Zhang, X. Zheng, A survey on security issues in services communication of microservices-enabled fog applications, Concurrency and Computation: Practice and Experience 31 (22) (2019) e4436.

[15] C. Esposito, A. Castiglione, K.-K. R. Choo, Challenges in delivering software in the cloud as microservices, IEEE Cloud Computing 3 (5) (2016) 10–14.

[16] J. Ghofrani, D. Lübke, Challenges of microservices architecture: A survey on the state of the practice, in: Proceedings of the 10th Central European Workshop on Services and their Composition (ZEUS), CEUR-WS.org, 2018, pp. 1–8.

[17] O. Zimmermann, Microservices tenets, Computer Science - Research and Development 32 (3-4) (2017) 301–310.

[18] A. Pereira-Vale, G. Márquez, H. Astudillo, E. B. Fernandez, Security mechanisms used in microservices-based systems: a systematic mapping, in: Proceedings of the 45th Latin American Computing Conference (CLEI), IEEE, 2019, pp. 1–10.

[19] I. Nadareishvili, R. Mitra, M. McLarty, M. Amundsen, Microservice Architecture: Aligning Principles, Practices, and Culture, O'Reilly Media, Inc., 2016.

[20] McAfee, Navigating a cloudy sky: Practical guidance and the state of cloud security (2018).
URL https://bit.ly/3zeaTDj

[21] J. A. Scott, A Practical Guide to Microservices and Containers, MapR Data Technologies, 2018.

[22] H. Zhang, S. Li, Z. Jia, C. Zhong, C. Zhang, Microservice architecture in reality: An industrial inquiry, in: Proceedings of the IEEE International Conference on Software Architecture (ICSA), IEEE, 2019, pp. 51–60.

[23] J. Bogner, J. Fritzsch, S. Wagner, A. Zimmermann, Microservices in industry: insights into technologies, characteristics, and software quality, in: Proceedings of the IEEE International Conference on Software Architecture Companion (ICSA-C), IEEE, 2019, pp. 187–195.

[24] H. Knoche, W. Hasselbring, Drivers and barriers for microservice adoption-a survey among professionals in germany, Enterprise Modelling and Information Systems Architectures (EMISAJ)-International Journal of Conceptual Modeling 14 (1) (2019) 1–35.

[25] S. Baškarada, V. Nguyen, A. Koronios, Architecting microservices: practical opportunities and challenges, Journal of Computer Information Systems 60 (5) (2020) 428–436.

[26] R. Capilla, A. Jansen, A. Tang, P. Avgeriou, M. A. Babar, 10 years of software architecture knowledge management: Practice and future, Journal of Systems and Software 116 (2016) 191–205.

[27] G. Viviani, M. Famelis, X. Xia, C. Janik-Jones, G. C. Murphy, Locating latent design information in developer discussions: A study on pull requests, IEEE Transactions on Software Engineering (2019). doi:10.1109/TSE.2019.2924006.

[28] M. Bhat, K. Shumaiev, A. Biesdorf, U. Hohenstein, F. Matthes, Automatic extraction of design decisions from issue management systems: a machine learning based approach, in: Proceedings of the 11th European Conference on Software Architecture (ECSA), Springer, 2017, pp. 138–154.

[29] J. Tsay, L. Dabbish, J. Herbsleb, Let's talk about it: evaluating contributions through discussion in github, in: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE), Association for Computing Machinery, 2014, pp. 144–154.

[30] T. Lopez, T. Tun, A. Bandara, L. Mark, B. Nuseibeh, H. Sharp, An anatomy of security conversations in stack overflow, in: Proceedins of the 41st IEEE/ACM International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS), IEEE, 2019, pp. 31–40.

[31] D. N. Palacio, D. McCrystal, K. Moran, C. Bernal-Cárdenas, D. Poshyvanyk, C. Shenefiel, Learning to identify security-related issues using convolutional neural networks, in: Proceedings of the 35th IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2019, pp. 140–144.

[32] S. Bettaieb, S. Y. Shin, M. Sabetzadeh, L. Briand, G. Nou, M. Garceau, Decision support for security-control identification using machine learning, in: Proceedings of the 25th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ), Springer, 2019, pp. 3–20.

[33] T. Li, Z. Chen, An ontology-based learning approach for automatically classifying security requirements, Journal of Systems and Software 165 (2020) 110566.

[34] A. Rezaei Nasab, M. Shahin, P. Liang, M. E. Basiri, S. A. Hoseyni, H. Khalajzadeh, M. Waseem, A. Naseri, dataset, script, and data reliability (2020).
URL https://bit.ly/3kuG7Ca

[35] M. Tahaei, K. Vaniea, K. Beznosov, M. K. Wolters, Security notifications in static analysis tools: Developers' attitudes, comprehension, and ability to act on them, in: Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, 2021, pp. 1–17.

[36] S. Abualhaija, C. Arora, M. Sabetzadeh, L. C. Briand, M. Traynor, Automated demarcation of requirements in textual specifications: a machine learning-based approach, Empirical Software Engineering 25 (6) (2020) 5454–5497.

[37] D. Taibi, V. Lenarduzzi, C. Pahl, A. Janes, Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages, in: Proceedings of the XP2017 Scientific Workshops, 2017, pp. 1–5.

[38] G. Gousios, D. Spinellis, Mining software engineering data from github, in: Proceedings of the 39th IEEE/ACM International Conference on Software Engineering Companion (ICSE-C), IEEE, 2017, pp. 501–502.

[39] S. D. Joshi, S. Chimalakonda, Rapidrelease-a dataset of projects and issues on github with rapid releases, in: Proceedings of the 16th IEEE/ACM International Conference on Mining Software Repositories (MSR), IEEE, 2019, pp. 587–591.

[40] P. Kadam, S. Bhalerao, Sample size calculation, International Journal of Ayurveda Research 1 (1) (2010) 55.

[41] X. Li, P. Liang, Z. Li, Automatic identification of decisions from the

hibernate developer mailing list, in: Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering (EASE), ACM, 2020, pp. 51–60.

[42] B. Xu, Z. Xing, X. Xia, D. Lo, Answerbot: automated generation of answer summary to developers' technical questions, in: Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2017, pp. 706–716.

[43] H. Zhang, S. Wang, T.-H. Chen, A. E. Hassan, Reading answers on stack overflow: Not enough!, IEEE Transactions on Software Engineering (2019). doi:10.1109/TSE.2019.2954319.

[44] M. L. McHugh, Interrater reliability: the kappa statistic, Biochemia Medica 22 (3) (2012) 276–282.

[45] T. Lopez, H. Sharp, T. Tun, A. Bandara, M. Levine, B. Nuseibeh, Talking about security with professional developers, in: Proceedings of the IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER&IP), IEEE, 2019, pp. 34–40.

[46] F. Shull, J. Singer, D. I. Sjøberg, Guide to Advanced Empirical Software Engineering, Springer, 2007.

[47] B. Johnson, T. Zimmermann, C. Bird, The effect of work environments on productivity and satisfaction of software engineers, IEEE Transactions on Software Engineering (2019). doi:10.1109/TSE.2019.2903053.

[48] B. G. Glaser, A. L. Strauss, E. Strutzel, The discovery of grounded theory; strategies for qualitative research, Nursing Research 17 (4) (1968) 364.

[49] C. Tao, H. Guo, Z. Huang, Identifying security issues for mobile applications based on user review summarization, Information and Software Technology 122 (2020) 106290.

[50] H. Hata, C. Treude, R. G. Kula, T. Ishio, 9.6 million links in source code comments: purpose, evolution, and decay, in: Proceedings of the 41st IEEE/ACM International Conference on Software Engineering (ICSE), IEEE, 2019, pp. 1211–1221.

[51] C. Silva, B. Ribeiro, The importance of stop word removal on recall values in text categorization, in: Proceedings of the International Joint Conference on Neural Networks (IJCNN), Vol. 3, IEEE, 2003, pp. 1661–1666.

[52] S. Bird, E. Klein, E. Loper, Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit, O'Reilly Media, Inc., 2009.

[53] M. F. Porter, An algorithm for suffix stripping., Program: Electronic Library and Information Systems 14 (3) (1980) 130–137.

[54] C. D. Manning, H. Schütze, P. Raghavan, Introduction to Information Retrieval, Cambridge University Press, 2008.

[55] W. Kraaij, R. Pohlmann, Viewing stemming as recall enhancement, in: Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), ACM, 1996, pp. 40–48.

[56] M. Aniche, E. Maziero, R. Durelli, V. Durelli, The effectiveness of supervised machine learning algorithms in predicting software refactoring, arXiv preprint arXiv:2001.03338 (2020).

[57] L. Bao, X. Xia, D. Lo, G. C. Murphy, A large scale study of long-time contributor prediction for github projects, IEEE Transactions on Software Engineering (2019). doi:10.1109/TSE.2019.2918536.

[58] Z. Yu, F. M. Fahid, H. Tu, T. Menzies, Identifying self-admitted technical debts with jitterbug: A two-step approach, IEEE Transactions on Software Engineering (2020). doi:10.1109/TSE.2020.3031401.

[59] L. Breiman, Random forests, Machine Learning 45 (1) (2001) 5–32.

[60] J. R. Quinlan, Induction of decision trees, Machine Learning 1 (1) (1986) 81–106.

[61] C.-W. Hsu, C.-C. Chang, C.-J. Lin, et al., A practical guide to support vector classification, Tech. rep., National Taiwan University (2003). URL https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

[62] C. Cortes, V. Vapnik, Support-vector networks, Machine Learning 20 (3) (1995) 273–297.

[63] T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), Association for Computing Machinery, 2016, pp. 785–794.

[64] A. McCallum, K. Nigam, et al., A comparison of event models for naive bayes text classification, in: Proceedings of the AAAI Workshop on Learning for Text Categorization (TextCat), Vol. 752, Citeseer, 1998, pp. 41–48.

[65] T. Joachims, Text categorization with support vector machines: Learning with many relevant features, in: Proceedings of the 10th European Conference on Machine Learning (ECML), Springer, 1998, pp. 137–142.

[66] J. Ramos, Using tf-idf to determine word relevance in document queries, in: Proceedings of the 1st Instructional Conference on Machine Learning (iCML), Citeseer, 2003, pp. 133–142.

[67] J. Turian, L. Ratinov, Y. Bengio, Word representations: a simple and general method for semi-supervised learning, in: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL), ACL, 2010, pp. 384–394.

[68] A. Ollagnier, H. Williams, Classification and event identification using word embedding, in: Proceedings of the Conference and Labs of the Evaluation Forum (CLEF), CEUR-WS.org, 2019, pp. 1–9.

[69] S. Ghannay, B. Favre, Y. Esteve, N. Camelin, Word embedding evaluation and combination, in: Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC), ELRA, 2016, pp. 300–305.

[70] J. Pennington, R. Socher, C. D. Manning, Glove: Global vectors for word representation, in: Proceedings of the 19th Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1532–1543.

[71] J. Pennington, R. Socher, C. D. Manning, GloVe: Global vectors for word representation (2014). URL https://stanford.io/3rkY7QF

[72] N. Kalchbrenner, E. Grefenstette, P. Blunsom, A convolutional neural network for modelling sentences (2014) 655–665.

[73] R. Zhao, R. Yan, J. Wang, K. Mao, Learning to monitor machine health with convolutional bi-directional lstm networks, Sensors 17 (2) (2017) 273.

[74] M. Schuster, K. K. Paliwal, Bidirectional recurrent neural networks, IEEE Transactions on Signal Processing 45 (11) (1997) 2673–2681.

[75] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.

[76] L. Zhang, S. Wang, B. Liu, Deep learning for sentiment analysis: A survey, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 8 (4) (2018) e1253.

[77] M. E. Basiri, S. Nemati, M. Abdar, E. Cambria, U. R. Acharrya, ABCDM: An attention-based bidirectional CNN-RNN deep model for sentiment analysis, Future Generation Computer Systems 115 (2021) 279–294.

[78] M. Song, H. Park, K.-s. Shin, Attention-based long short-term memory network using sentiment lexicon embedding for aspect-level sentiment analysis in korean, Information Processing & Management 56 (3) (2019) 637–653.

[79] M. E. Basiri, R. S. Chegeni, A. N. Karimvand, S. Nemati, Bidirectional lstm deep model for online doctor reviews polarity detection, in: Proceedings of the 6th International Conference on Web Research (ICWR), IEEE, 2020, pp. 100–105.

[80] A. Gulli, S. Pal, Deep Learning with Keras, Packt Publishing Ltd, 2017.

[81] J. Brownlee, Machine Learning Mastery, Pty. Ltd, 2014.

[82] D. López-Sánchez, J. R. Herrero, A. G. Arrieta, J. M. Corchado, Hybridizing metric learning and case-based reasoning for adaptable clickbait detection, Applied Intelligence 48 (9) (2018) 2967–2982.

[83] A. Wiranata, S. A. Wibowo, R. Patmasari, R. Rahmania, R. Mayasari, Investigation of padding schemes for faster r-cnn on vehicle detection, in: Proceedings of the 4th International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC), IEEE, 2018, pp. 208–212.

[84] H. Ide, T. Kurita, Improvement of learning for CNN with ReLU activation by sparse regularization, in: Proceedings of the International Joint Conference on Neural Networks (IJCNN), IEEE, 2017, pp. 2684–2691.

[85] M. E. Basiri, M. Abdar, M. A. Cifci, S. Nemati, U. R. Acharya, A novel method for sentiment classification of drug reviews using fusion of deep and machine learning techniques, Knowledge-Based Systems (2020) 105949.

[86] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu,

P. Kuksa, Natural language processing (almost) from scratch, Journal of Machine Learning Research 12 (2011) 2493–2537.

[87] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, Journal of Machine Learning Research 15 (1) (2014) 1929–1958.

[88] J. Nabi, Machine learning - word embedding & sentiment classification using keras (2018).
URL https://bit.ly/33o1Oop

[89] P. Zhou, Z. Qi, S. Zheng, J. Xu, H. Bao, B. Xu, Text classification improved by integrating bidirectional lstm with two-dimensional max pooling, arXiv preprint arXiv:1611.06639 (2016).

[90] G. Rao, W. Huang, Z. Feng, Q. Cong, Lstm with sentence representations for document-level sentiment classification, Neurocomputing 308 (2018) 49–57.

[91] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).

[92] R. Abdalkareem, S. Mujahid, E. Shihab, A machine learning approach to improve the detection of CI skip commits, IEEE Transactions on Software Engineering (2020). doi:10.1109/TSE.2020.2967380.

[93] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[94] A. P. Bradley, The use of the area under the roc curve in the evaluation of machine learning algorithms, Pattern Recognition 30 (7) (1997) 1145–1159.

[95] T. Fawcett, An introduction to ROC analysis, Pattern Recognition Letters 27 (8) (2006) 861–874.

[96] S. Lessmann, B. Baesens, C. Mues, S. Pietsch, Benchmarking classification models for software defect prediction: A proposed framework and novel findings, IEEE Transactions on Software Engineering 34 (4) (2008) 485–496.

[97] C. Tantithamthavorn, A. E. Hassan, An experience report on defect modelling in practice: Pitfalls and challenges, in: Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), Association for Computing Machinery, 2018, pp. 286–295.

[98] T. H. M. Le, D. Hin, R. Croft, M. A. Babar, Puminer: Mining security posts from developer question and answer websites with pu learning, in: Proceedings of the 17th IEEE/ACM International Conference on Mining Software Repositories (MSR), Association for Computing Machinery, 2020. doi:10.1145/3379597.3387443.

[99] F. Peters, T. Tun, Y. Yu, B. Nuseibeh, Text filtering and ranking for security bug report prediction, IEEE Transactions on Software Engineering 45 (6) (2017) 615–631.

[100] A. Luque, A. Carrasco, A. Martín, A. de las Heras, The impact of class imbalance in classification performance metrics based on the binary confusion matrix, Pattern Recognition 91 (2019) 216–231.

[101] S. Zimmerman, U. Kruschwitz, C. Fox, Improving hate speech detection with deep learning ensembles, in: Proceedings of the 11th International Conference on Language Resources and Evaluation (LREC), ACL, 2018.
URL https://bit.ly/3wR3JDp

[102] O. Bousquet, S. Boucheron, G. Lugosi, Introduction to statistical learning theory, in: Summer School on Machine Learning, Springer, 2003, pp. 169–207.

[103] Quora, What is generalization in machine learning? (2016).
URL https://bit.ly/3Bn5pIf

[104] C. M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.

[105] E. A. AlOmar, W. Aljedaani, M. Tamjeed, W. Catzin, M. W. Mkaouer, Y. N. El-Glaly, Finding the needle in a haystack: On the automatic identification of accessibility user reviews, in: Proceedings of the 2021 ACM CHI Conference on Human Factors in Computing Systems (CHI), Association for Computing Machinery, 2021.

[106] H. O. Obie, W. Hussain, X. Xia, J. Grundy, L. Li, B. Turhan, J. Whittle, M. Shahin, A first look at human values-violation in app reviews, in: Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS), IEEE, 2021.

[107] A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Vis-

aggio, G. Canfora, H. C. Gall, What would users change in my app? summarizing app reviews for recommending software changes, in: Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE), ACM, 2016, pp. 499–510.

[108] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering, Springer Science & Business Media, 2012.

[109] N. Carter, D. Bryant-Lukosius, A. DiCenso, J. Blythe, A. J. Neville, The use of triangulation in qualitative research, Oncology Nursing Forum 41 (5) (2014) 545–547.

[110] D. Romano, M. Pinzger, Using source code metrics to predict change-prone java interfaces, in: Proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM), IEEE, 2011, pp. 303–312.

[111] K. Goseva-Popstojanova, J. Tyo, Identification of security related bug reports via text mining using supervised and unsupervised classification, in: Proceedings of the 18th IEEE International Conference on Software Quality, Reliability and Security (QRS), IEEE, 2018, pp. 344–355.

[112] Y. Sun, S. Nanda, T. Jaeger, Security-as-a-service for microservices-based cloud applications, in: Proceedings of the 7th IEEE International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, 2015, pp. 50–57.

[113] A. Manu, J. K. Patel, S. Akhtar, V. Agrawal, K. B. S. Murthy, Docker container security via heuristics-based multilateral security-conceptual and pragmatic study, in: Proceedings of the International Conference on Circuit, Power and Computing Technologies (ICCPCT), IEEE, 2016, pp. 1–14.

[114] M. Ahmadvand, A. Ibrahim, Requirements reconciliation for scalable and secure microservice (de) composition, in: Proceedings of the 24th IEEE International Requirements Engineering Conference Workshops (REW), IEEE, 2016, pp. 68–73.

[115] X. Yang, D. Lo, X. Xia, Z. Wan, J. Sun, What security questions do developers ask? a large-scale study of stack overflow posts, Journal of Computer Science and Technology 31 (5) (2016) 910–924.

[116] Z. Han, X. Li, Z. Xing, H. Liu, Z. Feng, Learning to predict severity of software vulnerability using only vulnerability description, in: Proceedings of the 33rd IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2017, pp. 125–136.

[117] D. Pletea, B. Vasilescu, A. Serebrenik, Security and emotion: sentiment analysis of security discussions on github, in: Proceedings of the 11th Working Conference on Mining Software Repositories (MSR), Association for Computing Machinery, 2014, pp. 348–351.

[118] P. Morrison, T. D. Oyetoyan, L. Williams, Identifying security issues in software development: are keywords enough?, in: Proceedings of the 40th International Conference on Software Engineering Companion (ICSE-C), Association for Computing Machinery, 2018, pp. 426–427.

[119] E. Knauss, S. Houmb, K. Schneider, S. Islam, J. Jürjens, Supporting requirements engineers in recognising security issues, in: Proceedings of the 17th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ), Springer, 2011, pp. 4–18.

[120] Y. Jiang, P. Lu, X. Su, T. Wang, Ltrwes: A new framework for security bug report detection, Information and Software Technology 124 (2020) 106314.

[121] M. Pereira, A. Kumar, S. Cristiansen, Identifying security bug reports based solely on report titles and noisy data, in: Proceedings of the IEEE International Conference on Smart Computing (SMARTCOMP), IEEE, 2019, pp. 39–44.

[122] M. Brysbaert, How many words do we read per minute? a review and meta-analysis of reading rate, Journal of Memory and Language 109 (2019) 104047.