Spectrum of Engineering Sciences
ISSN (e) 3007-3138 (p) 3007-312X
Volume 4, Issue 3, 2026

AUTOMATED IDENTIFICATION OF BUILD DISCUSSIONS ON MICROSERVICES SYSTEMS: AN EMPIRICAL STUDY

Ayesha Anjum[1], Muhammad Nasir[*2], Zakia Jalil[3]

[1,*2,3]*Faculty of Computing & Information Technology, International Islamic University, Islamabad, Pakistan*

[1]ayesha.msse467@iiu.edu.pk, [*2]m.nasir@iiu.edu.pk, [3]zakia.jalil@iiu.edu.pk

DOI: https://doi.org/10.5281/zenodo.18872314

Keywords
Microservices Architecture, Build Discussions, Machine Learning

Article History
Received: 06 January 2026
Accepted: 19 February 2026
Published: 05 March 2026

Corresponding Author: *
Muhammad Nasir

Abstract

In recent years, microservices architecture has gained widespread popularity over traditional systems, largely due to its flexible development cycle and enhanced scalability. In software development, software quality is a major concern. Issues within the software can significantly impact its overall quality. Microservices developers face several challenges in monitoring and managing issues such as failures, faults, and errors. These challenges often arise from a lack of evidence and understanding, hindering the effective implementation of quality practices in Microservices Architecture (MSA). The process of converting source code into an executable file is known as a build, while any problems that occur during this process are referred to as build issues. In the current literature, the methods available for identifying build issues in microservices are primarily qualitative or rely on manual research approaches. Methods such as thematic analysis (TA) and grounded theory (GT) can be challenging to manage due to their complexity and time-consuming nature. To address this gap, we identify build-related discussions within the existing microservices based systems. We define a build discussion as a developer conversation that addresses challenges and decisions related to the build process, typically presented in the form of paragraphs. This research focuses on identifying build issues, often stemming from poor management and dependencies, using machine learning (ML) techniques. We applied ML and deep learning (DL) models to a manually curated dataset consisting of project discussions and annotations. The results identified 11,663 non-build discussions and 1,997 build-related discussions. The ML models, evaluated using k-fold cross-validation, achieved the following performance metrics: Precision 83.60%, Recall 72.34%, F-score 77.79%, AUC 80.44%, and G-Means 68.55%. Among the three baseline models, DeepM1 performed the best. The validation survey further confirmed that build discussions identified through DeepM1 are beneficial in practice.

INTRODUCTION

Microservices systems have become more popular in recent years as compared to traditional systems because of their development cycle and scalability. Amazon, Netflix, and SoundCloud these well-known platforms, are based on micro-services [1], [2]. It provides frequent releases of products and system maintainability [3]. According to [2], the annual rise of CVE (Common Vulnerabilities and Exposure) reported financial losses to development companies because of software development issues. The use of micro-services is widely spreading, and one of the concerns that arise is the build issue. It can be an error or fault.

Independent execution of modules is possible in a microservice architecture, as a single application is divided into small services with their process and a lightweight mechanism like (HTTP, API). They are easy to maintain because of their size, if failure/fault arises in one of the services, the whole system will not break. The conversion of source code into an executable file (e.g., APK, War) is known as build. The issues that arise while execution is called build issues. According to [1] which was based on five open-source systems, 10.75% of issues in microservices belong to build issues, they subcategories these issues as missing artifacts or broken errors. The issues of build are errors that arise while source file loading, docker builds, and build Script. The problem related to broken and missing artifacts are broken files, property packages or files, and illegal symbols. The major cause of their arrival is due to programming errors (incorrect naming, type declaration), legacy version compatibility (old versions of the tool operating system which is not compatible), and invalid configuration and communication (incorrect request handling).

The concept of build discussion refers to a discussion or sharing of thoughts regarding a particular event in the form of a paragraph. In a build discussion practitioners/ individuals share their concepts, expertise, and abilities to solve issues that arise by evaluating potential difficulties or limitations. To improve the result, participants in a build discussion might provide alternate methods, solicit input, and work through problems. The discussion may focus on issues including project management, resource allocation, and timeframes. A build discussion's ultimate goal is to encourage collaboration and communication among participants to improve the entire building process and provide the desired outcomes.

Different organizations adopt modern architecture for the development of their application or software but due to lack of professional knowledge they fail to provide qualified systems. Research demonstrates artifacts present in discussions are related to solution user stories and decisions. We will be using an idea proposed by [10] to identify build discussion issues in existing microservices systems. They conducted a survey to choose only projects based on microservice systems. It is suggested that gathering and studying previous design discussions and judgments can help in the formulation of more rational choices. to increase practitioners' knowledge of integrated microservices systems.

In this study, based on the microservice architecture style we used open-source GitHub projects. We use 3 Deep Learning and 12 Machine Learning models [10] to identify GitHub project issues regarding build discussion via developer discussion. For the performance evaluation of models, we collect two datasets from GitHub and manually labeled them 13660 paragraphs which include 11,663 non-build discussions and 1,997 build discussions for the main study, and for the initial study, manually labeled them 500 paragraphs which include 328 non-build discussions and 172 build discussions.

The experimental outcomes demonstrate the potential of all learning models, with an average Recall, F-score, AUC, and G-Means (72.34%, 77.79%, 80.44%, 68.55%). Additionally, 1.018 to 3.756 range in all metrics, DeepM1 surpasses three advanced baselines. A validation survey was carried out to show the model DeepM1's practical use. We got 62%–68% of those who responded that said build discussions attained by DeepM1 would have 7 practical uses: they might be helpful, refined sub-optimal, speedier, effectively prioritized, trace key artifacts, and help people avoid frequent mistakes.

Background/ Motivation

In a microservices system, a developer needs domain knowledge to differentiate build discussions from non-build discussions. The problem arises when non-build discussion terms include in build discussion and vice versa. When building microservices systems, a tool that can exactly identify critical build issues, corresponding solutions, and their effects can be helpful. In the field of microservices, they will gain a better understanding of the various components that make up the microservices system as a result of the tool's output.

Problem Statement

In software systems, many build issues remain undetected due to unreasonable processes of development, insufficient experience in development, and less effective models. The presence of issues results in decreased quality which cause failure.

Research Gap

Organizations and practitioners have different levels of expertise when it comes to designing and deploying microservices systems since the MSA style is a novel paradigm that is presently being investigated. The use of (new) tools and technologies (like containers) in the development and deployment of microservices raises more important unresolved problems. This is a result of the industry's general lack of understanding about such technologies' and tools' potential capabilities and the ideal configurations for using them during building.

Research Questions

RQ 1: Can we effectively identify build issues in a microservices system using ML Models?

RQ 2: Do practitioners find it beneficial in practice to automatically identify build discussions in microservices systems?

Research Objectives

There is a knowledge gap that is currently among practitioners when it comes to building microservices-based systems. The aim is achieved with the help of the following objectives:

RO1: To identify build issues so that the developers can concentrate on the removal of it with optimized effort.

RO2: To identify effective ML techniques for detecting potential defective components for build issues in Microservices System as an early stage, it will be used to optimize the testing effort, utilization of code inspections, and effective detection

Research Contribution

1. Various concerns related to the build process in microservices systems are understood.
2. Developer discussions of microservices systems, which implement the ML/DL model to differentiate build discussion from the non-build discussion.
3. Construction of two datasets that consists of 1997 microservices build discussions and 11663 non-build discussions and 328 non-build discussions and 172 build discussions.
4. The proposed technique provides better accuracy and uses a machine-learning model.
5. Early and accurate detection of issues provides an efficient software model.

Literature Review

In this section, we discuss the studies which support built-in microservices. The studies use ML or DL-based models in a context related to recognizing build in textual software artifacts. In the build process, multiple activities depend on the programming language, development process, scripting language, and operating system. Mainly the issues arise from operating system compatibility with tools or software, legacy versions, dependencies, and missing features or artifacts. It's being observed that the developer mistakenly adds dependencies in the build script, which causes the build problem at the issue-triggering phase. While the deployment of the microservices system pipeline, practitioners should avoid old versions and unnecessary dependencies. [1]

In modern development and software maintenance, building systems are essential. The build failure in the software systems affects development activities with major cost increases. In a study [4], build issues are investigated with a monolithic systems perspective, they claim the effort to report build issues by the developer is non-trivial. To facilitate build failure, adopted fix pattern and fix symptoms. In the study [6] researcher studied fixed patterns that can be used to resolve compiler errors in the build process; the investigation was based on Java projects taken from open source. Specific failure type was discussed and many build failures remain unexplored.

In research [7] build issues with long duration were discussed concerning continuous integration. The longer the build time holds the performance of another task. The development time should be optimized. They observed by running multiple times failed commands increase the build duration. Build failures via search-based patch generation and code analysis a technique named Hire build was proposed [5] in which a history-oblivious fixing technique is being implemented which effectively fixes 18% of build failures successfully. In the research [8], they proposed an approach that repairs the break dependency-related issues Maven builds. For build failure reasons and a possible solution from the internet a tool that summarizes the name, Bart is proposed [9]. It reduces the fix-build break by understanding it. Table 1 provides a summary of the current relevant work.

Table 1 Summary of the present work

Reference, Year	Data Set	Method	Proposed Technique	Result / Findings
[23], 2023	3 open source repositories, GitHub, Reddit, Quora	Qualitative Analysis	IRA based on Transformer, BPE encoder, transfer learning for reorganizing discussion structures	Helps most comments find accurate parent comments and extract discussion structure
[30], 2022	GitHub	Mixed Methods, Survey with Developers	Analysis of discussions, duplication detection between discussions and issues	Errors, unexpected behavior, code reviews identified. Discussions play crucial role in development. Duplication exists between discussions and issues.
[1], 2021	5 OSS systems, goa, eShopOnContainers, light-4j, Molecular Microservices demo	Grounded Theory, Survey based study	Taxonomical classification of issues in OSS systems, root cause extraction, analysis of why issues arise in microservices	Leading issues are Technical Debt, Build, Security. Main categories 17, Sub categories 46, Total issue types 137. Literature gap identified in fixing build issues
[4], 2020	Stack Overflow, Ant, Gradle, Maven	Thematic Analysis	Summarized fix patterns, taxonomy of failure symptoms, created 50 categories, analysis of different failure types	Modifying build scripts reduces 67.96 percent of build issues. Microservices perspective investigation required
[7], 2019	GitHub Projects	Grounded Theory	Logistic mixed effect models using build duration factors, team size, project size, test density, build configuration	Failed builds take more time than passed builds. Build time depends on multiple factors. Proper CI configuration helps maintain successful builds
[5], 2019	GitHub, Gradle	Quantitative and Qualitative	Mining historical fixes, history driven	HireBuild based on historical fix information

Analysis	build fix approach	provides 2 times more reproducible build failures

Research Gap: Microservices adoption faces expertise gaps and challenges in utilizing new tools like containers, given limited industry familiarity with their potential and ideal application.

RESEARCH METHODOLOGY

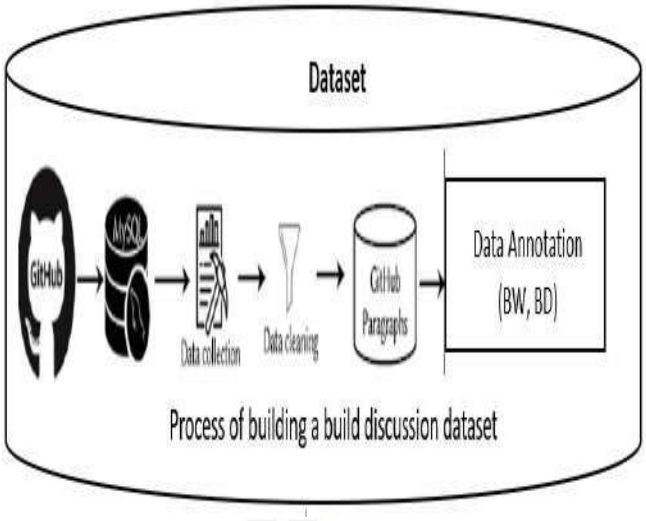

Figure 1 Research Process

This study has been done concerning a previous study related to security discussions about microservices systems. However, this study aims to assist in evaluating the impact of build discussion about microservices systems.[10] They have conducted a survey to extract micro-service projects so we already know these projects are based on microservices. In this chapter, the details of the research methodology are explained. To achieve the research objectives and to identify build discussion, experimentation is adopted. Validation of the approach is done by conducting controlled experimentation by using Python Language evaluated using GitHub projects datasets. Experimentation follows a systematic step-by-step method and the independent variable is being tested using ML/DL models to identify its effectiveness on the dependent variable accordingly.

Experiment Design:
This experiment aims to develop a machine learning-based method for automatically identifying build discussions within microservices development discussions. In the first phase, discussions will be collected and labeled as "build" or "non-build." Machine learning models will be trained and evaluated using these labeled discussions. In the second phase, a validation survey will gather feedback from participants about the model's effectiveness and usefulness. This includes participants' familiarity with microservices, their experiences with build discussions, and their perceptions of the model's accuracy. The study seeks to enhance the automation of identifying build discussions and validate the approach's practicality in real-world scenarios.

Dependent Variable: The dependent variable for our research identified build issues effectively.

Independent Variables: The independent variable for our research will be OSS Microservices.

Intervention:
Machine Learning/Deep Learning Algorithms

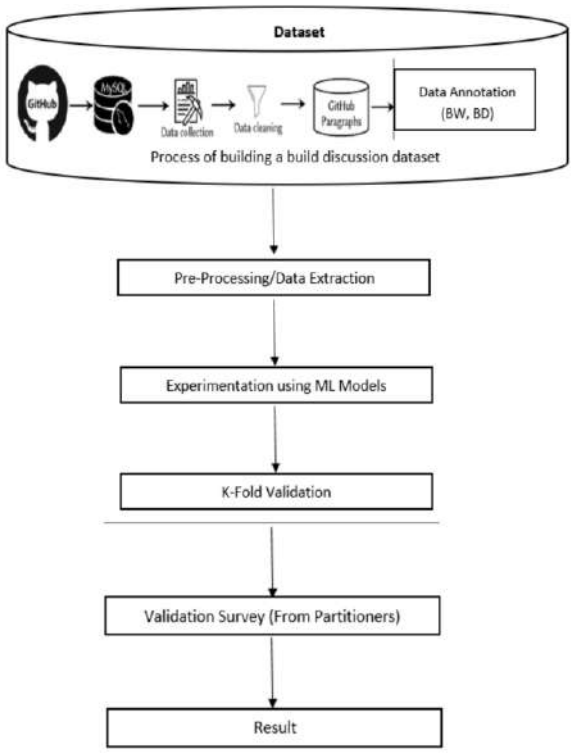

Figure 2 Dataset Construction of Build Discussion

GitHub Data:

Prepare Data:

Five projects shown in Table 2, have been selected for the dataset of the research which is as follows "goa, eShopOnContainers, microservices demo, light-4j, and, deep-farmwork". These projects were chosen because of their size as compared to other projects [10] Number of discussions in these projects was large. As an outcome, their contributor participates in more discussions in the issue tracking system. The problem-tracking system keeps track of a build's design. As a result, issue-tracking systems are used to support discussions. To increase the possibility of finding build conversations, both closed and open issues were used. 1825 issues are chosen at random from 5724 total [24]. Code snippets may be present in issues; as a result, code snippets are taken out of issues. The concept of using a paragraph as the unit of analysis was taken from [11] and 1825 issues have been broken into 13660 paragraphs to help minimize potential problems like trying to find meaningful information from lengthy issues.

In Table 3, we have mentioned the selected words related to build, which encapsulate key aspects of software development and the build process. They encompass components like building itself, containerization (Dockers), security (signed, token, authorization), modular design (module), networking (connect, multicast), and deployment (deploy). Additionally, terms like configuration, dependencies, and API highlight vital elements in creating functional software systems. This comprehensive collection reflects the diverse considerations essential for successful software development and deployment.

Table 2 List of Selected GitHub Projects

Project Name	URL	No. of Issues	No. of Releases	No. of Contributors	No. of Stars	No. of Forks	Active Date	Line of Codes	Languages
goa	https://bit.ly/2Vz0GjV	2442	31	73	4.1k	450	2014-now	82,193	GO
light-4j	https://bit.ly/2Y3eWe	637	104	25	3k	496	2016-now	50,099	Java, Objective-J
deep-framework	https://bit.ly/3cO6o79	640	22	9	532	75	2015-now	920,806	HTML, JavaScript, CSS
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

Table 3 Build Related Words

Sr No.	Build Related Words	Sr No.	Build Related Words
1	build	21	loading
2	dockers	22	authorize
3	building	23	service
4	ThreadFactoryBuilder	24	prefetch
5	builder	25	packet
6	clash	26	configuration
7	patch	27	packages
8	isolate	28	dependencies
9	logged	29	multicast
10	configured	30	amqp server
11	connect	31	connection
12	signed	32	plug-in
13	token	33	deploy
14	module	34	refactor
15	accessing	35	reconnecting
16	authorization	36	built
17	isolation	37	mergeOptions
18	API	38	merging
19	release	39	config
20	compile		

Initial Study: We performed initial research to see if any discussions could be detected in the paragraphs that were collected from the five projects listed before annotating them. And what

traits do those build discussions have? To discover, we chose 500 paragraphs at random from 200 issues [24] that came from 10 different projects [10]. We annotated the paragraph on a scheme described as if it build discussion [BD] write '1' and if it non-build discussion [BD] write '0'. 'BW' also creates terms that are relevant to the build. We were able to identify 172 of the 500 paragraphs after the first annotation as build discussion, It encouraged us to believe that analyzing paragraphs carefully is the best way to find build information. Figure 3 displays some examples of the concerns that are being annotated.

Major Study: For our main studies, we annotated 13660 paragraphs from the "goa,light-4j eShopOnContainers,microservices-demo, and deep-framework projects". Table 4 includes an example of a GitHub build discussion with annotations. The build discussion annotated includes 1997 and non-build discussion11663.

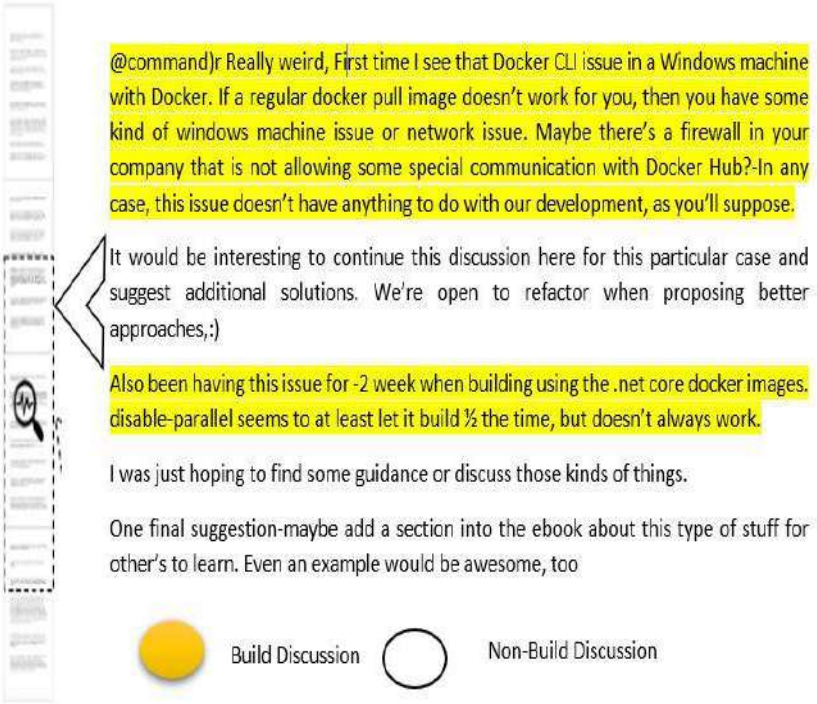

Figure 3 Extraction of Build Discussion Dataset

Table 4 Sample of Build Discussion in Micro-Services extract from GitHub project

Sr No.	Paragraph	Build Discussion	Build Words
1	The generated client should be a standalone API client package that is used by the cli to build the command line tool. [issue 228]	✓	API, build
2	Great thank you! could you please add a test that would have failed before but is fixed after this PR? thanks! [issue 1767]	✗	
3	By default, TCP transporter detects all network interfaces and send broadcasts on all interfaces and it works well. [issue 46]	✗	
4	I got the error when I build in the docker (golang image) [issue 1766]	✓	Docker, build

Experimentation (RQ1)
To identify build discussions, we implemented ML-based models already present in literature [10] to the dataset as described above.

Pre-processing
To polish the dataset and remove possible noises from the dataset, four pre-processing steps are applied.
Removal of Irrelevant Words and Characters:
Through issue-tracking systems or in their responses, developers commonly use casual language. For example, people could employ graphical emoji to convey their feelings about a choice or discussion. Links might also be used to provide historical context in developer discussions. Links, emails, and meaningless characters (such as "/" and "*") also include emoji symbols. In developer discussions, The absence of reliable information about Natural Language Processing (NLP) techniques could lead to negative impacts on classifier performance, potentially making it worse. Python uses regular expressions to eliminate them from the dataset as a result.
E.g.

```
doc1 = []

for i in range(len(doc)):

doc_lower = doc[i].lower()

isurl = re.sub(r"(https?|):\/?\S+", "", doc_lower)

doc1.append(isurl)
```

Changing Abbreviations:
Acronyms are preferred by some developers (such as "cause" aren't") while expressing their ideas. We picked out such terms and changed them to their full form Eg.

```
def decontracted(phrase):
contractions = {
```

```
"aren't": "are not",

"can't": "can not",

"can't've": "can not have",

"'cause": "because",

"could've": "could have",}
```

Eliminating Stop Words:
By reducing noise and increasing recall value, eliminating stop words can enhance the

performance of learning classifiers. Therefore, we used the Neural Language Toolkit (NLTK) to

exclude stop words from the dataset.

Eg.

```
from nltk.corpus import stopwords

stop_words = stopwords.words('english')

doc4 = []

for string in doc3:

string_split = string.split()

remove_stopword_data = [word for word in string_split if word not in stop_words] join_split_data = ' '.join(remove_stopword_data) doc4.append(join_split_data)
```

Stemming_Process:
It is used in the NLP technique for normalization. By eliminating a word's derivational affixes, stemming reduces it to its basic form. It is a straightforward stemming method that researchers have shown, may move recollection deprived of noticeably decreasing precision. I break down words in the dataset into their most basic forms using the Porter Stemming Technique. Eg.

```
from nltk.stem import PorterStemmer

stemmer = PorterStemmer()

doc5 = []

for i in range(len(doc4)):

temp1 = []

split_data_stem = doc4[i].split()

for j in range(len(split_data_stem)):

temp1.append(stemmer.stem(split_data_stem[j]))

doc5.append(temp1)

return doc5
```

ML Models
For predicting build discussion, we employ ML methods that are extensively utilized in software engineering research. We employed Decision Tree (DE), Random Forest (RF), Decision Tree (DE), Support Vector Machine (SVM-LR), and Extreme Gradient Boosting (XGBoost), 3 text feature extraction methods are utilized to identify features in our dataset's paragraphs, that may be used as data for machine learning algorithms. The following is a succinct description of these methods:

- **Bag of Words** is a method that seeks to determine how often a specific word appears in a text. This method ignores the sentence's grammatical structure, word choice, and other linguistic components.
- **Term Frequency-Inverse Document Frequency** is the importance of each word in a corpus of documents, BoW is a level-up direct feature engineering method. The weight of a word is determined by taking into account both its frequency of occurrence Term Frequency and its (IDF) across the corpus.
- **Global Vector:** Word vectors are used in word embedding techniques to represent each word in a corpus, such as a collection of texts.

According to research, using word embedding approaches can improve how well ML classifiers function. Its one of the well-liked embedding techniques [14], places the words in the right context and forges a semantic link between them.

DL Models

Three DL models are used DeepM1, DeepM2, and DeepM3 to separate developer discussions of microservices systems into build discussions and non-build discussions. The following is a

description of these 3 models:

Deep M1

In DeepM1 five blocks mainly consists of three CNN (Convolutional Neural Networks) layers and 2 Bi-LSTM (Bidirectional Long Short-Term Memory) networks. (i.e., essential components). CNN and Bi-LSTM are used in DeepM1 because text-processing applications are used although there are other options we used CNN due to its highly recognized feature. When addressing long-term dependencies in the input data, LSTM outperforms competitors like recurrent neural networks (RNNs). [12,13]. In Figure 4, we showed the DeepM1.

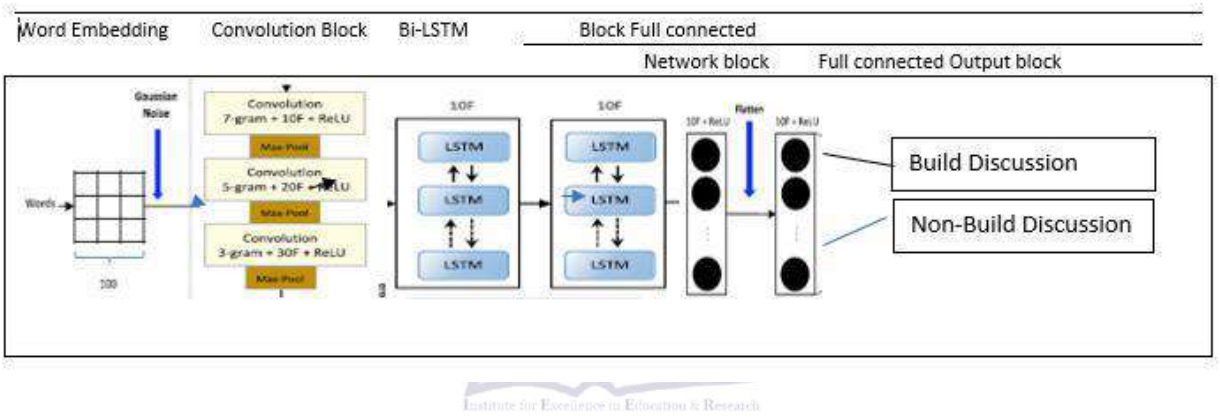

Figure 4 Overview of DeepM1

Input & Word Embedding

Each word has given a unique number in the dataset. A smaller number indicates that the term is more prevalent in our collection, and vice versa. The unique words are then collected into a corpus together with the unique numbers allocated to them. Then, the unique numbers generated in the preceding phase will be used to replace each paragraph's words in our dataset. Each text is changed using this procedure into a series of digits. Each sequence is the same length as the accompanying paragraph. But every sequence that is generated must have the same length. Using the zero-padding method, we equalize the lengths of each sequence. (That is, all sequences have a length of 100). In DeepM1's first layer, The embedding dimension is set to 100 for a word embedding layer. The word indexes from the corpus are transformed into dense vector representations by the embedding layer. Just after the term "embedding," a Gaussian Noise is used to reduce overfitting.

Convolutional Block

There are three layers of convolution layers and pooling layers in this block to extract features. The embedding vectors are input into the block's first layer, which transforms them into a feature map. The embedding vectors' pertinent information is all captured by the feature map. Three convolutional networks, with kernel sizes of 7, 10, and 5, as well as feature maps of 3, are used by DeepM1. Therefore, each of these convolutional networks is designed using the Same-Padding and ReLU (Rectified Linear Unit) function to improve the performance of DeepM1. Additionally, this block uses max-pooling to combine the convolutional network

output. To achieve this, a max-pooling operation with a factor of 2 is used after each convolutional network which identifies and selects the most crucial characteristics. To deal with the overfitting, a 0.25 dropout is introduced in the final phase.

Fully Connected Output Block.
A fully linked output layer receives the output from the preceding block and categorizes each paragraph as either a build discussion or a non-build discussion.

Fully Connected Network block.
Additionally, we employ two thick layers, each of which consists of 10 neurons (i.e., fully connected networks). To integrate 10 neurons into another column matrix, a flatten is positioned in between these two thick layers. Networks with full connection incorporate the ReLU feature. The "flatten" layer is situated between these two dense layers, and fully connected networks utilize the Rectified Linear Unit (ReLU) activation function due to its ability to introduce non-linearity and enable effective learning of complex patterns in data.

Bi-LSTM Block.
The feature map that was built in the block before is converted into a compressed representation in this block. Recent studies have demonstrated that DL models based on Bi-LSTM and CNN are capable of completing text categorization tasks. When the sequence is sufficiently long, bi-LSTM is seen to be a promising alternative. Because bi-LSTM cells have two LSTM layers traveling in opposing directions forward and backward they can tackle the sequential modeling problem. I use two Bi-LSTM layers with a 10-dimensionality factor as a result.

Deep M2
In this model, three DeepM1 blocks the convolutional block, the fully connected output block, and the input and word embedding block are employed. It is important to note that a flatten is used in the space between the final CNN layer and the fully connected output layer. We demonstrated an overview of the DeepM2 in Figure 5.

Deep M3
In this model, the Bi-LSTM block, the input and word embedding block, and the fully linked output block (also known as DeepM1 blocks 1, 3, and 5) are all employed. A flattening is applied after the second Bi-LSTM layer. In Figure 6, we show the DeepM3 in a generalized manner.

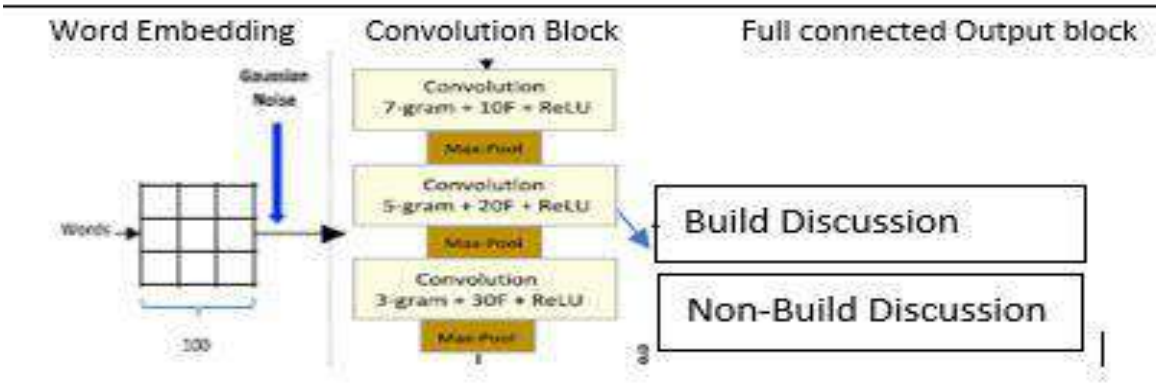

Figure 5 Overview of DeepM2

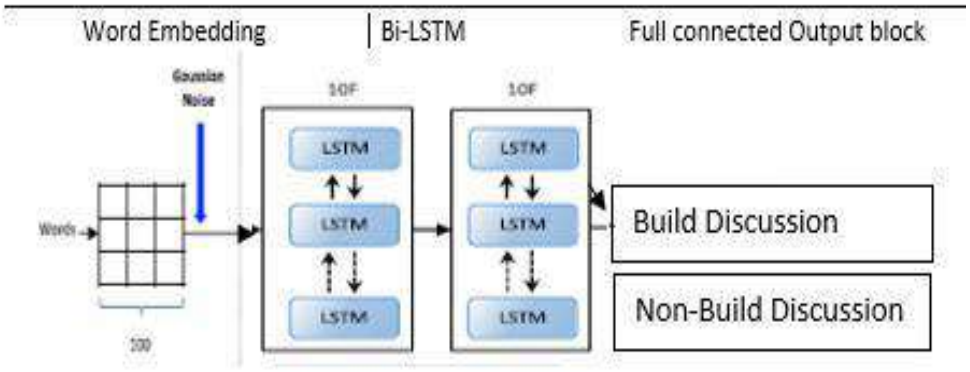

Figure 6 Overview of DeepM3

Convolution layers were chosen DeepM1 and DeepM2 to excerpt various textual properties. We can extract n-gram features specifically from DeepM1 and DeepM2 using three convolution layers. DeepM1 and DeepM3 use LSTM layers to take into account contextual information from the text.

Validation Survey (RQ2).
We conducted a survey [19] online to collect and examine practitioners' views on the importance of the results provided by learning models (RQ1). In the paragraphs that follow, we go into greater depth about the survey's methodology.

Protocol. We created a 9-question anonymous survey that is hosted on the Google form platform. We outlined the problem description, suggested strategy, and survey's goal in the survey. We outlined the reasons behind the lack of informed build decisions in microservices systems in the problem statement. Then, we developed a way to properly identify build discussion from an extensive number of issue discussions in microservices systems to address this problem. We also included "Build Discussion" to avoid misunderstandings. The participants were taught how to distinguish between build conversations and non-build discussions in issue 1108 from the Goa project using our best learning model, DeepM1.

Three demographic questions, one of which was optional (i.e., your country), and four Likert scale questions, on which three questions were generic and one open-ended question were all included in the eleven survey questions. The open-ended question was meant to get feedback from participants on how to strengthen our approach (i.e., what changes to our methodology will help the creation of microservices systems even more?).

- Question 1. "The technique is beneficial because the build discussions it identifies provide valuable and significant build information".
- Question 2. "The technique is beneficial because the build discussions can be used to inform new build discussions or improve current sub-optimal build discussions".
- Question 3. "The technique is useful because it allows practitioners like myself to identify relevant content from build discussions in a reasonable amount of time".
- Question 4. "The technique is useful because the build discussions it identifies enable us to locate build-related flaws in our systems more quickly than we could manually".
- Question 5. "The technique is useful because build discussions it identifies may include incoming build-sensitive bug reports, allowing for easier identification and more efficient prioritization and resolution".

- Question 6. "The technique is useful because build discussions it uncovers might act as cues or suggestions to track back and forth to features and codes that are crucial to building issues".
- Question 7. "The technique is useful because the build discussions it identifies could be advantageous for people who have little background in microservices build issues or who have just joined a microservices project. This would allow them to quickly access build solutions, learn about them, and avoid common build-related mistakes".

Participants. We used two methods to find the participants. First, we shared information about our poll on social media sites. we contacted the appropriate microservices professionals. We carefully reviewed their profiles before immediately inviting them via email. Finally, 19 answers were received. We were unable to determine the response rate because of how difficult the procedure was.

Result Analysis

The chapter assesses how well ML/DL models performed on the build discussion dataset. The results of the studies performed to address RQ1 are initially presented in this section. The validation survey results are then presented to address RQ2.

Tool and System Specification:

A Python script was used to save all of the gathered data in a MySQL database. Scikit-Learn and Keras both Python libraries were used to implement every experiment. We conducted the trials using Intel(R) Core(TM) i3-350M CPU 2.27GHz, 4.00 GB RAM.

Performance Evaluation:

In the confusion matrix, there are 4 factors: "False Positive (FP), True Positive (TP), True Negative (TN), and False Negative (FN)" to evaluate the ML/DL models. The number of non-build discussions that are categorized as build discussions is specified by FP. The number of accurately classified non-build discussions is denoted by TN, while the number of accurately classified build discussions is denoted by TP. FN is a measure of how many categories of real-build discussion paragraphs fall under the non-build discussion paragraph category. When comparing the results of learning models, standard measures include precision, recall, and F1-score. [25,26] The proportion of recognized build discussions that are genuinely built discussions is held by precision.[27] Recall shows the proportion of build discussions that were correctly detected to all build discussions. F1-score is calculated using a combination of precision and recall metrics. AUC has been suggested or used in several studies as an evaluation tool to gauge classifier performance. [11,28] In this case, the AUC measure determines whether a prediction model would score a randomly selected build discussion (i.e., TP) higher than a randomly selected non-build discussion (i.e., FP). Another metric for evaluating the effectiveness of binary classifications is the G-mean. It is underlined that the G-mean metric, especially when classification mistakes are taken into consideration, is the best measurement for classes that are unbalanced and comprise both positives and negatives. [29] All measurements range from 0 to 1. The highest performance is represented by 1, while the lowest performance is represented by a value of 0. We employ the 10-fold cross-validation method to rate the performance of ML/DL models. 13,660 paragraphs total, taken from GitHub, are split into 10 folds at random.

Ten times are evaluated from the divided data. Every time, 10% (or one-fold) of the data is chosen for performance evaluation, while the remaining 90% (or the remaining folds) are utilized to train the system. For reproducibility in this investigation, ML/DL models are set to a fixed seed.

Learning to Identify Build Discussions (RQ1)

To address RQ1, we examined 15 ML/DL models. The BoW, TF-IDF, and GloVe text feature selection techniques are used to construct some machine learning models, including Random Forest, Decision Tree, XGBoost, and SVM-LR. The word embedding approach is used

by the Deep Learning (DL) models DeepM1, DeepM2, and DeepM3. It should be underlined that if there are no changes in the best accuracy throughout the preceding 10 epochs, the training of DL models is designed to finish early. Table 5 illustrates the performance analysis of the ML and DL models across 13,660 paragraphs using the 10-fold cross-validation method. Table 4-1 shows that the average accuracy, recall, F-score, AUC, and G-mean for ML and DL models are 83.60%, 72.34%, and 68.55%, are calculated with the help of formula G-mean = √(Sensitivity * Specificity) respectively. Except for accuracy measurements, all data indicate that DeepM1 as a DL model performs best when combined with XGBoost and BoW to yield the best outcomes. The accuracy is expressed as 92.63%. The performance for Decision Tree with TF-IDF and BoW, DeepM1, DeepM3, and SVM-LR with BoW was all over 80%. The poorest results were produced when GloVe was used as the feature selection approach.

Table 5 Performance Evaluation for ML/DL Model

Sr No.	Models	Precision	Recall	F1-Score	AUC	G-Mean
1	BoW+RF	90.41	74.95	84.3	73.65	82.31
2	BoW+DT	81.37	80.2	81.63	85.23	80.78
3	BoW+XGBoost	92.63	71.2	80.82	81.76	81.21
4	BoW+SVM-LR	85.9	80.1	84.96	73.65	82.94
5	TF–IDF+RF	90.64	78.54	78.55	85.21	84.37
6	TF–IDF+DT	80.91	81.45	82.17	81.11	81.17
7	TF–IDF+XGBoost	91.02	71.3	80.83	84.66	80.55
8	TF–IDF+SVM-LR	89.22	78.47	83.41	87.85	83.67
9	GloVe+RF	84.26	49.64	63.77	73.65	64.67
10	GloVe+DT	51.66	54.88	54.28	61.22	53.24
11	GloVe+XGBoost	82.99	59.48	69.11	75.4	70.25
12	GloVe+SVM-LR	82.19	59.44	68.99	77.23	69.89
13	DeepM1	86.73	82.99	86.69	89.98	84.83
14	DeepM2	80.16	80.16	84.3	87.94	80.16
15	DeepM3	84.01	82.36	83.18	88.15	83.18
Average		83.60	72.34	77.79	80.44	68.55

Generalizability of Learning Models

Motivation. All learning models are predicted to perform well when applied to new data that they were not trained on, in particular, data that originate from a range of domains (i.e., unseen data). This ability is known as generalization. We want to evaluate the effectiveness of the fifteen learning models in identifying build debate (i.e., out-of-sample accuracy) in two distinct datasets collected from varied situations.

Approach. First, we created two datasets. the projects moleculer and scalecube-services. Each of them has 500 paragraphs. 250 paragraphs from each project were randomly selected. The annotator went over these 500 paragraphs looking for build discussions. When the annotation procedure was finished, 172 annotated paragraphs were evaluated at random (confidence level: 90%, margin of error: 5%). The GitHubGen dataset was found to contain 172 build talks and 328 non-build discussions. On GitHubGen, we evaluated the previously trained models that are given in Table 4-2.

Table 6 Performance of ML/DL Models on GitHubGen Dataset

Sr No.	Models	Precision	Recall	F1- score	Auc
1	BoW+RF	96.34	57.18	70.54	77.39
2	BoW+DT	89.52	54.95	76.11	81.56
3	BoW+XGBoost	98.86	61.71	63.46	73.23
4	BoW+SVM-LR	76.54	61.93	98.44	76.07
5	TF–IDF+RF	96.2	53.52	95.29	76.34
6	TF–IDF+DT	85.59	71.12	84.62	83.18
7	TF–IDF+XGBoost	98.44	42.31	36.59	72.04
8	TF–IDF+SVM-LR	95.29	57.04	65.62	77.96
9	GloVe+RF	84.62	15.49	55.63	74.27
10	GloVe+DT	36.59	31.69	64.15	80.1
11	GloVe+XGBoost	65.62	29.57	46.47	68.17
12	GloVe+SVM-LR	61.64	31.69	41.86	67.21
13	DeepM1	86.71	78.16	82.22	86.71
14	DeepM2	82.3	75.35	78.67	84.46
15	DeepM3	85.88	51.4	64.31	74.02
Average		82.67	51.54	68.26	76.84

Result. Table 6 demonstrates that the performance of the Decision Tree with TF-IDF, DeepM1, and DeepM2 is not noticeably degraded. throughout the GitHubGen dataset, XGBoost With BoW specifically achieves a precision metric value of 98.86% on the GitHubGen dataset. The performance of four machine learning methods (RF, DT, XGBoost, and SVM-LR) using GloVe has the lowest results across all datasets, as shown in Table 4-2. This might be caused by the fact that GloVe chooses its features based on a pre-trained dataset.

Comparison with Baselines

Motivation. As far as we know, there hasn't been any research that creates learning models for separating build discussions from microservices practitioner discussions. Despite this, three baseline models with a top model (DeepM1). Comparable to [15, 16]. As a starting point, we created a keyword-based search. We can

determine whether or not the detection of build discussions in microservices systems is a learning problem based on the results of the keyword-based search. The two studies that come closest to our investigation are [17] and Palacios et al. [18] PUMiner was created by the study [17] to identify security posts in Stack Overflow, and SecureReqNet by the study [18] to distinguish security issues from non-security issues in the GitHub repository. These two methods served as two additional baselines for us. **Approach.** Here, we outline the three baseline implementations.

Keyword-based Search (Baseline1).
The annotators were tasked with locating build-related words in build discussions. These build-related words served as the foundation for our keyword-based search. Some of these build-related terms appeared as abbreviations in build discussions, and others were in full forms. For Instance, In the compiled build discussions, we saw terms like "application programming interface," and "API," First, we transformed these build-related words to a single format (i.e., abbreviation). This produced 39 distinctive build keywords. Then, using previously undisclosed datasets, we used the stemming procedure on paragraphs. Our keyword-based search determines the likelihood that a stemmed sentence will contain one or more of the 39 distinct build keywords. To be more specific, imagine that P is a stemmed paragraph and that SL is a list that contains each of the 39 distinct build keywords. We mentioned NSL as the number of tokens in P which emerge in SL and NP as the number of tokens in P. Because only one build-related term is included in P, it should be noted that some sentences may be wrongly categorized as build discussion. To prevent this, if Pr(P, SW) is greater than 0.05 (i.e., build words make up at least 5% of a paragraph), the keyword-based search classifies P as a build discussion.

PUMiner (Baseline2) and SecureReqNet (Baseline3).
To achieve a fair comparison, we duplicated PUMiner and SecureReqNet using two alternative approaches. We replicated them using their data preparation and trained them with our dataset using 10-fold cross-validation. Second, we applied our data pre-processing in place of PUMiner's and SecureReqNet's, giving them the names PUMiner+ and SecureReqNet+, respectively. We used our dataset to train PUMiner+ and SecureReqNet+ using 10-fold cross-validation. We evaluated the performance of all baselines, including the trained PUMiner, SecureReqNet, PUMiner+, and SecureReqNet+, on the dataset GitHubGen.

Result. As demonstrated in Table 7, DeepM1 outperforms all baselines across all unknown datasets in all metrics, with gains ranging from 1.018 to 3.756. With this exception, DeepM1 consistently performs better than all baselines across the board on unknown datasets. Furthermore, PUMiner+ and SecureReqNet+ perform better than when they employ their respective internal pre-processing (PUMiner and SecureReqNet), as seen in Table 7.

Table 7 Baseline Metrics Result

Models	Metrics	GitHub	Improve	Models	Metrics	GitHub	Improve
DeepM1	Precision	84.01		**Keyword**	Precision	23.35	3.765x
				Based Search			
	Recall	82.36			Recall	22.98	3.363x
	F1-Score	83.18			F1-Score	23.64	3.550x
	AUC	86.15			AUC	44.62	1.077x
	G-Mean	73.65			G-Mean	39.62	2.147x
PUMiner	Precision	26.01	3.333x	**PUMiner+**	Precision	43.87	1.912x
	Recall	73.55	1.874x		Recall	56.34	2.377x
	F1-Score	38.29	2.150x		F1-Score	50	1.317x
	AUC	45.35	1.929x		AUC	64.48	1.441x
	G-Mean	36.29	1.387x		G-Mean	63.96	1.382x
SecureReq Net	Precision	65.81	1.644x	**SecureReqNet +**	Precision	65.14	1.212x
	Recall	54.22	1.344x		Recall	53.52	1.243x
	F1-Score	59.46	1.348x		F1-Score	61.48	1.406x
	AUC	71.53	1.364x		AUC	70.89	1.223x
	G-Mean	69.4	1.4060x		G-Mean	68.73	1.255x

Usefulness of Identification of Build Discussions (RQ2)

We performed a validation survey to find out how practitioners felt about the value of the build discussion found by the top-performing learning model (DeepM1). The demographic data of the 19 practitioners who responded to the validation survey is shown in Figure 7 5 people remained quiet about where they were working in. The remaining 14 respondents were from 5 different nations, with the majority being Pakistan each with 7 respondents. 2 of the respondents, claimed to have more than 6 years of experience in the field of software development. 9 respondents had experience in developing software for less than two years, while 8 participants had experience of three to five years.

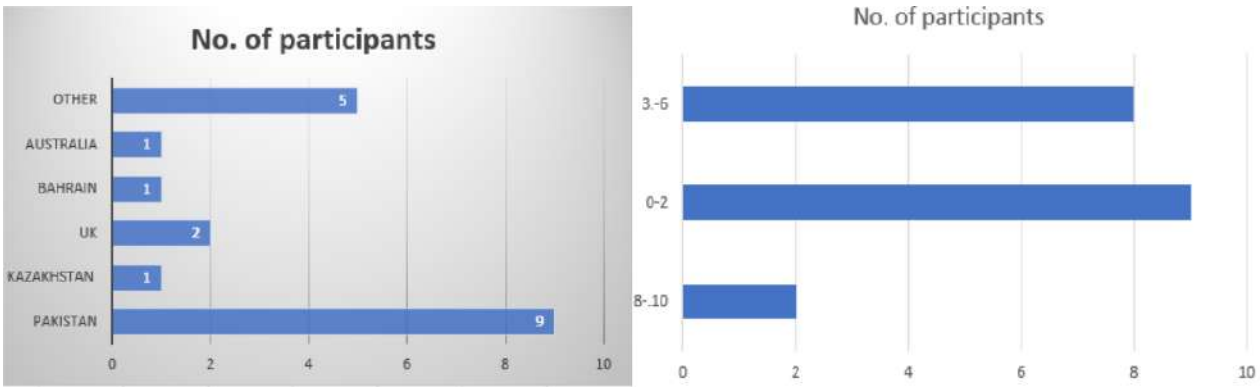

Figure 7 Participant Demographic Representation

Figure 8, 9, 10 display the practitioners' opinions on the DeepM1 learning model, which produced the greatest results in terms of separating build discussions from non-build discussions. Almost

79% of the respondents (strongly) agreed, as we found (Q1), that the build discussions detected by DeepM1 provide valuable and significant build information. no participants disagreed with Q1, they also stated that it is possible to extract relevant materials and information from the discovered build discussion on time (Q2). The rating for this statement is 68.4% as "strongly agree or agree". Q3 through Q7 look into the potential benefits of the observed build discussions. Q3 was replied to by almost 66.2% of the respondents, demonstrating that build discussion aid in the making of informed build decisions. In the future or refine the existing sub-optimum build decisions. Another advantage that the build discussions extracted from developer discussions can bring is to help practitioners find build-critical issues (e.g., build mistakes) in their systems more quickly than if they do it manually (Q4). Q4 was endorsed by 67% of respondents agreed and strongly agreed, while 12% disagreed. Nearly 79% of those who responded to Q5 felt that the identified build discussions could assist trace backward and forward to build artifacts. Another advantage of build discussion is that build-sensitive bug reports can be more easily detected, given higher priority, and fixed. Because incoming build-sensitive may be contained by build discussions. (84.2% with Q6, 4.8% were neutral). Regarding Q7, 78.9% of respondents agreed that the knowledge in the identified build discussions might assist newcomers and less seasoned team members in finding and learning about build solutions quickly, avoiding common build mistakes.

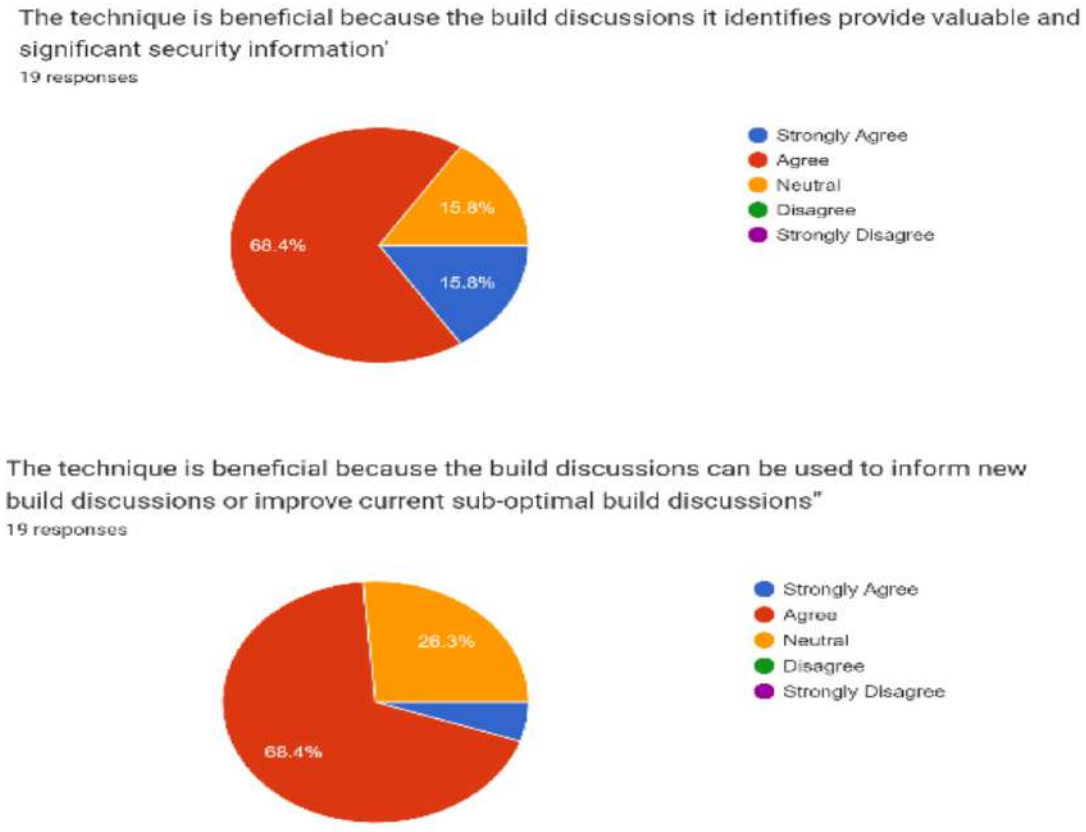

Figure 8 Validation Survey Response (Q1, Q2)

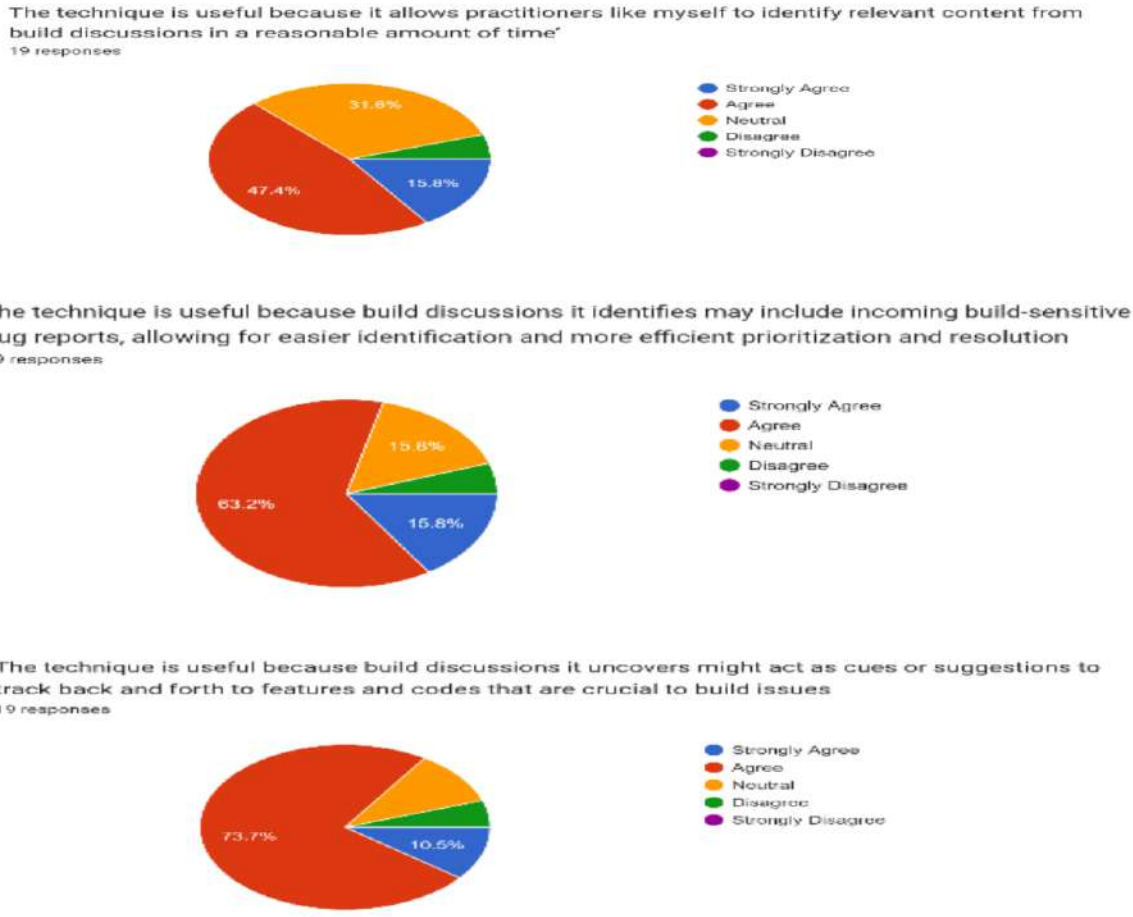

Figure 9 Validation Survey (Q3, Q4,Q5)

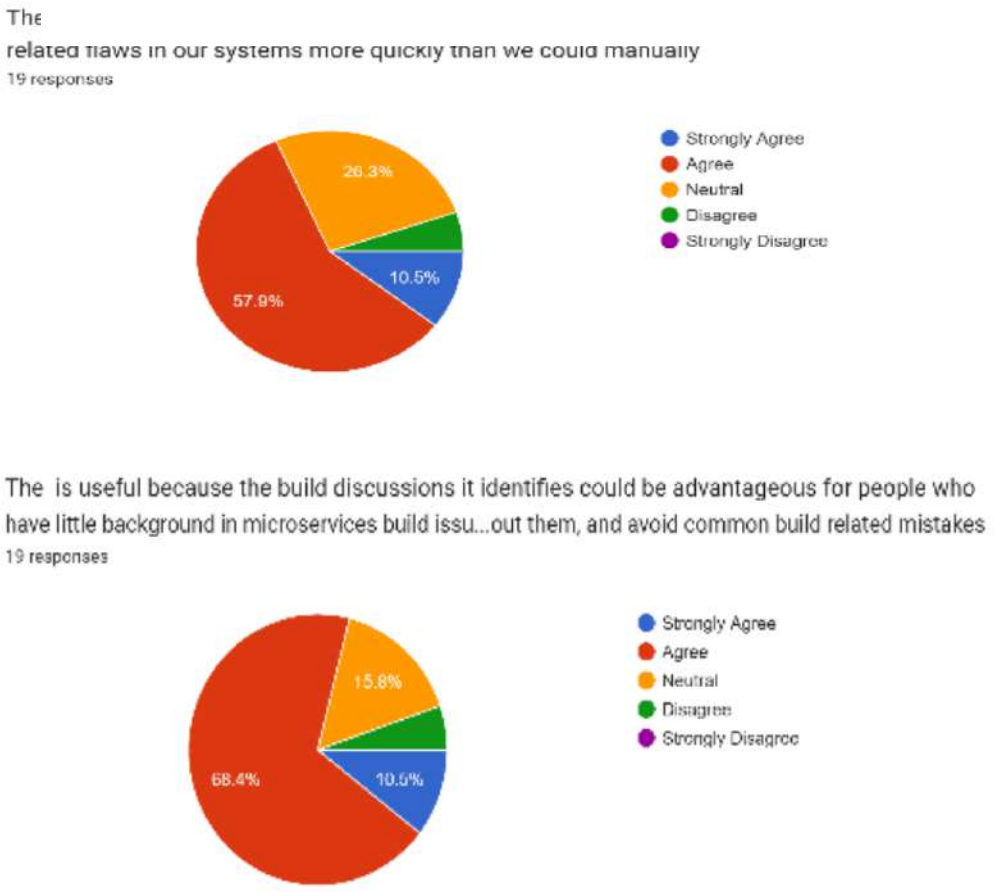

Figure 10 Validation Survey (Q6, Q7)

RQ2 Summary: According to the validation survey respondents, build discussions discovered by the top-performing learning model (DeepM1) show promising results in building microservices systems, particularly in assisting practitioners in making informed build decisions going forward or improving the current sub-optimal build decisions.

Findings and Discussions

This study aims to fill the knowledge gap that exists among practitioners regarding the building of microservices systems. We established two research questions to do this.

RQ1: Can we effectively identify build issues in a microservices system using ML Models? The results demonstrate that both ML and DL models perform well in terms of recall, F-score, AUC, and G-Means (72.34%, 77.79%, 80.44%, and 68.55%) when it comes to finding build discussions. Additionally, DeepM1's generalizability, or capacity to discover and build discussions on new datasets, is not noticeably affected. Finally, the analysis shows that DeepM1 outperforms baselines, with improvements in every metric ranging from 1.018 to 3.756x.

RQ2: Do practitioners find it beneficial in practice to automatically identify build discussions in microservices systems?

The validation survey results confirm that the build discussions selected by the top-performing learning model (i.e., DeepM1) have the potential for use in building microservices systems, particularly in aiding practitioners in making informed build decisions going forward or

improving the current sub-optimal build decisions. Automatically identifying build discussions in microservices systems is immensely beneficial to practitioners. It saves time by swiftly sifting through discussions, allowing professionals to focus on critical tasks. Early detection of build-related challenges ensures timely resolution, enhancing software quality. Resource allocation becomes efficient as experts can target discussions requiring their expertise. Consistent classification mitigates subjectivity and maintains accuracy. This automation aids in orienting new team members, aligns seamlessly with agile and DevOps practices, and empowers data-driven decision-making based on historical trends. Collaboration is streamlined, team scalability is supported, and the overall project's success is elevated through this technology-driven approach.

Discussion:

In this section, we will be discussing suggestions for researchers plus practitioners constructed on the significant conclusions and evaluation of the replies to an open-ended inquiry: "What improvements to our approach would further help develop secure microservices systems?" Better support is always needed. All microservices practitioners, non-build microservices developers, in particular, face challenges due to the build's challenging nature in the MSA style. These challenges include the need to properly understand build in microservices systems, adopt optimal build measures, and put those measures into practice.

Microservices practitioners can recognize and acquire concise data, according to a survey. Build information from the results (build discussion) provided by the learning models with reasonable effort and time. When we realize that the build discussions examined in this study involve build solutions and principles, the situation becomes even more critical. This could be more advantageous for people who are new to microservices projects or have limited build expertise. They may rapidly get knowledgeable about the most recent build solutions and recommended precautions for microservices systems. However, our participants highlighted that to design and execute safe microservices systems, practitioners still require additional assistance.

To automatically integrate build discussions in developer discussions to key learning documents and resource material (such as blogs and code snippets), new tools may be created or these models can be expanded. Additionally, new tools should be able to suggest potential build fixes with appropriate references for a build issue that is identified in a particular microservices system. Make build discussion a priority. Only build discussions and non-build discussions may be distinguished using these learning models. According to the results of our validation survey, developers can discover build-critical services or build-critical problems in microservices systems by using build discussions that are extracted from developer discussions. Given that a single compromised microservice has the potential to bring down the entire system, it is becoming more crucial than ever to identify such build-critical services and issues in the MSA. Despite this, the participants noted that different priority levels, such as crucial, important, or low-impact, might be assigned to distinct construction discussions. A software team may more quickly assign experts to solve more build-critical issues by knowing the types and severity of build discussions. [18]. Therefore, it is necessary to create tools that categorize discussions according to their nature and seriousness. Create the context and construct the discussion. In this study, we decided to classify and categorize build discussions at the paragraph level. [11] in developer discussion at the paragraph level. This choice was taken to make it possible for microservices practitioners to quickly and precisely document, identify, and communicate valuable and concise build information without having to go through a lengthy developer discussion to find and comprehend a build issue, choice, or solution. Even so, developers may require more design context details to fully comprehend some build discussions picked up by our learning models and to successfully

implement the build solutions suggested in the picked-up build discussions.

Threats to Validity

To implement mitigation techniques to these concerns, we shall identify possible risks to the research method's validity in this part. We categorize them into three groups: construct, internal, and external.

Internal Validity

In experiments, the annotation process was divided into two threads. To remove personal bias, we conducted pilot research to come to a comprehension of the qualities of build and non-build discussions. The incorrect labeling of the paragraphs poses another possibility that may have occurred. The cross-check was the action done to reduce this risk. The annotators should have a lot of expertise in microservices systems, design, and coding like included professors and software engineers. Due to these efforts, we think that the dataset is reliable and contains a few paragraphs that are incorrectly categorized. Given that a paragraph is our unit of analysis, as we said in our learning models, we could be able to identify build discussions that are lacking essential design context details. We understand that some build discussions picked up by learning models may not be completely understood by practitioners. Nevertheless, we think that the majority of build discussions (such as BD1 through BD4 in Fig. 3-3) recognized by learning models include important and helpful build information.

Construct Validity

Build discussion" in microservices systems may be observed and understood differently by practitioners. "Build discussion" was defined at the beginning of the validation survey as part of our approach to reducing this issue. Additionally, we included GitHub examples of build discussions. It should be emphasized that no one disagreed with our concept of build discussion during the discussion. To assess the applicability of our approach (i.e., RQ2), we only employed one research technique (the validation survey). Our validation survey may not have accurately and completely demonstrated all advantages and potential drawbacks (such as adding to practitioners' loads) of our method in their day-to-day work as microservices practitioners. To evaluate different aspects of our strategy, further research techniques. It is recommended to employ methods like controlled tests and in-depth qualitative research.

Another potential risk is the choice of metrics and ML/DL models. To demonstrate various elements of the learning models' performance, we employed a wide range of indicators. Of these four metrics, recall accuracy, and F1-score metrics are the most popular and strongly advised in the literature on software engineering. (e.g., [11,20,21]). G-Mean is used for the effectiveness of our imbalanced dataset.

External Validity

Their conclusions may be limited by the number of replies from the validation survey. The validation survey only receives 19 responses. We made an effort to lessen the risk of experience in the validation survey by enlisting experts who worked in the software sector and open-source project contributors. However, we understand that not all microservices practitioners, project types (such as closed-source software), or organizations may be affected by the survey results.

The dataset utilized in this study is made up of 13,660 paragraphs, including 11,663 non-build discussions and 1997 build discussions, that were extracted from source GitHub and gathered from five open-source microservices systems of a substantial scale. Additionally, we used 500 unseen paragraphs from two open-source microservices systems to apply to learning models. Therefore, we acknowledge that not all projects hosted on GitHub are comparable to these open-source projects. Another drawback is that we only paid attention to the topic discussion in the projects that were chosen.

Conclusions

This study tries to overcome the gap, to ensure while implementing and designing organizations

and partitioners securely build microservice systems. We identify build discussions from previous developer discussions. We construct a data set by using GitHub projects of microservices and test our dataset on 15 DL/ML models. The Result shows that Model DeepM1 performs better than other models. On average we get 83.60%, "recall of 72.34%, F-score of 77.79%, AUC of 80.44%, and G-mean of 68.55%". Out of three baselines, DeepM1 performs best with an average of 86.24, and in the end, we conduct a validation survey for the usefulness of the model from practitioners. According to responders to the validation survey, the build discussion acquired by DeepM1 may have seven practical uses: they are relevant, refined sub-optimal, helpful, quicker, effectively prioritized, trace key artifacts, and help prevent frequent mistakes.

Future Work

In the future, we can focus on enhancing the performance of machine learning models by combining advanced NLP techniques and investigating various feature representations. We can develop real-time monitoring tools to provide prompt intervention and decision assistance in microservices by detecting and analyzing build discussions as they happen. The use of automated summarization and knowledge extraction techniques may be used to extract insightful knowledge and useful data from specific build discussions.

REFERENCES

M Waseem, P Liang, M Shahin, An Ahmad "On the Nature of Issues in Five Open Source Microservices Systems: An Empirical Study" In Evaluation and Assessment in Software Engineering (EASE 2021), June 21–23, 2021, Trondheim, Norway. ACM, New York, NY, USA, 10 pages.

Gupta, A., Suri, B., Kumar, V., & Jain, P. (2021). Extracting rules for vulnerabilities detection with static metrics using machine learning. International Journal of System Assurance Engineering and Management, 12, 65-76.

P. Di Francesco, P. Lago, and I. Malavolta, "Migrating towards microservice architectures: An industrial survey", 2018 IEEE International Conference on Software Architecture (ICSA), pp. 29-2909, April 2018.

Y. Lou, Z. Chen, Y. Cao, D. Hao, and L. Zhang. 2020. Understanding build issue resolution in practice: symptoms and fix patterns. In Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering (ESEC/FSE). ACM, 617–628.

Yiling Lou, Junjie Chen, Lingming Zhang, Dan Hao, and Lu Zhang. 2019. "History-driven build failure fixing: how far are we?" In Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2019, Beijing, China, July 15-19, 2019, Dongmei Zhang and Anders Møller (Eds.). ACM, 43–54.

Chen Zhang, Bihuan Chen, Linlin Chen, Xin Peng, and Wenyun Zhao. 2019. "A large-scale empirical study of compiler errors in continuous integration". In Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019, Marlon Dumas, Dietmar Pfahl, Sven Apel, and Alessandra Russo (Eds.). ACM, 176–187.Taher Ahmed Ghaleb, Daniel Alencar da Costa, and Ying Zou. 2019. "An empirical study of the long duration of continuous integration builds. Empirical Software Engineering" 24, 4 (2019), 2102–2139.

Christian Macho, Shane McIntosh, and Martin Pinzger. 2018. Automatically repairing dependency-related build breakage. In 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 106–117

Carmine Vassallo, Sebastian Proksch, Timothy Zemp, and Harald C. Gall. 2018. Un-break My Build: Assisting Developers with Build Repair Hints. In Proceedings of the 26th International Conference on Program Comprehension. 41–51.

AR Nasab, M Shahin, P Liang, ME Basiri, M Waseem, M.E., Raviz 2021."Automated identification of security discussions in microservices systems: Industrial surveys and experiments"- Journal of Systems and Software Elsevier 181, p.111046.

Viviani, G., Famelis, M., Xia, X., Janik-Jones, C., Murphy, G.C., 2019. Locating latent design information in developer discussions: A study on pull requests. IEEE Trans. Softw. Eng.

Song, M., Park, H., Shin, K.-s., 2019. Attention-based long short-term memory network using sentiment lexicon embedding for aspect-level sentiment analysis in Korean. Inf. Process. Manage. 56 (3), 637–653.

Basiri, M.E., Nemati, S., Abdar, M., Cambria, E., Acharrya, U.R., 2021. ABCDM: An attention-based bidirectional CNN-RNN deep model for sentiment analysis. Future Gener. Comput. Syst. 115, 279–294.

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

Obie, H.O., Hussain, W., Xia, X., Grundy, J., Li, L., Turhan, B., Whittle, J., Shahin, M.,2021. 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.

AlOmar, E.A., Aljedaani, W., Tamjeed, M., Catzin, W., Mkaouer, M.W., El- Glaly, Y.N., 2021. 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.

Le, T.H.M., Hin, D., Croft, R., Babar, M.A., 2020. 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,

Palacio, D.N., McCrystal, D., Moran, K., Bernal-Cárdenas, C., Poshyvanyk, D., Shenefiel, C., 2019. 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, pp.140–144.

The usefulness of Detection Tool for Build Discussion in Microservices (Mini-Survey) https://docs.google.com/forms/d/e/1FAIpQLSfxbGZgTsMpBuyPPZyNfaVvjYATZhkvfTWtbUIkb6WVD2D3OA/viewform?usp=sf_link

Li, X., Liang, P., Li, Z., 2020. 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, pp. 51–60.

Abualhaija, S., Arora, C., Sabetzadeh, M., Briand, L.C., Traynor, M., 2020. Automated demarcation of requirements in textual specifications: a machine learning-based approach. Empir. Softw. Eng. 25 (6), 5454–5497

Savchenko, D. I., Radchenko, G. I., & Taipale, O. (2015, May). Microservices validation: Mjolnirr platform case study. In 2015 38th International Convention on Information and communication technology, electronics and Microelectronics (MIPRO) (pp. 235-240). IEEE.

Bai, S., Liu, L., Meng, C., & Liu, H. (2023). Automating discussion structure re-organization for GitHub issues. Expert Systems with Applications, 120024.

Kadam, P., Bhalerao, S., 2010. Sample size calculation. Int. J. Ayurveda Res. 1 (1), 55.

Aniche, M., Maziero, E., Durelli, R., Durelli, V., 2020. The effectiveness of supervised machine learning algorithms in predicting software refactoring.

Abualhaija, S., Arora, C., Sabetzadeh, M., Briand, L.C., Traynor, M., 2020. Automated demarcation of requirements in textual specifications: a machine learning-based approach. Empir. Softw. Eng. 25 (6), 5454–5497.

Abdalkareem, R., Mujahid, S., Shihab, E., 2020. A machine learning approach to improve the detection of CI skip commits. IEEE Trans. Softw. Eng.

Bao, L., Xia, X., Lo, D., Murphy, G.C., 2019. A large-scale study of long-time contributor prediction for GitHub projects. IEEE Trans. Softw. Eng.

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

Hata, H., Novielli, N., Baltes, S., Kula, R. G., & Treude, C. (2022). GitHub Discussions: An exploratory study of early adoption. *Empirical Software Engineering*, *27*, 1-32.