JOURNAL OF BASIC SCIENCE AND ENGINEERING

UNSUPERVISED MACHINE LEARNING FOR FEEDBACK LOOP PROCESSING IN COGNITIVE DEVOPS SETTINGS

Hemanth Swamy
Senior Software Engineer
Motorola Solutions
hemanth.swamy1@motorolasolutions.com

Abstract—**Agile development and quick adaptation to new needs are two of the most pressing issues facing software applications and systems today. The need for separate deployments, such as in DevOps, arises from this. A system's quality, particularly when it is continually built, is greatly affected by continuous monitoring and feedback creation, which are essential components of DevOps because of the short release cycles and flexibility they include. To achieve this goal, we offer a system for feedback that integrates data from operations and development. This system can identify patterns, spot unusual behavior, and feed that information back into development, providing a more thorough investigation into production anomalies. In order to achieve this goal, we describe the dataset using two unsupervised machine learning approaches, k-means clustering and archetypal analysis. Based on the findings, we classify new data points as normal or abnormal. An application is being built inside a big industrial organization that provides real-time data for testing and evaluation purposes. This application is designed to support the feedback loop, which consists of continuous development, planning, deployment, and monitoring.**

Keywords—Feedback loop; Cognitive DevOps; Unsupervised Machine learning; Software quality; Improved Fuzzy C Means

I. Introduction

Research into non-invasive Brain-Computer Interfaces (BCI) has shown that error-related potentials (ErrPs) are beneficial for control in the last several decades. However, these brain correlates concerning merely the discrete sense of mistakes continue to be an issue when trying to continually correct for the incorrect action of an end result (such a robot arm) in a BCI [1]. The capacity of the DevOps phenomenon to provide continuous value delivery is a major factor in its rising popularity. Similar to any other software process shift or set of technical practices, DevOps isn't without its share of difficulties [2]. Social settings, cognitive mediations, brain processes, and behavior are all layers of organisms that are involved in culture's complex feedback loops. Some brain pathways are shaped by culturally dependent social processes, according to recent neuroscience research. Potentially new understandings of mental diseases could emerge from research on the impact of cultural setting on brain processes. Novel approaches from the neurosciences provide fresh perspectives on how to evaluate the cultural influences on psychological well-being and disease [3].

An integral aspect of the BCI loop, feedback tells the user if their cognitive efforts were fruitful. Theoretically, the feedback influences the user's capacity to learn how to operate the BCI. Facilitation of neuroplasticity is linked to the feedback of a BCI loop in the area of neurorehabilitation. It is not apparent whether a BCI loop allows for any facilitation, however, since techniques like motor visualization, tactile stimulation, and movement observation all have therapeutic effects on their own [4]. Security flaws are common in today's software systems because of the prevalence of commercial off-the-shelf (COTS) and legacy components. To keep up with the rapid speed of software delivery, verification techniques artefacts should be updated often, according to the contemporary approach of addressing ever-changing circumstances, DevOps [5]. Software development has entered a new phase of radical change with the fast adoption of DevOps as well as Continuous Integration/Continuous Deployment (CI/CD), which has improved release efficiency and simplified procedures. Integrating Dev and Ops into a single DevOps architecture allows for a software lifecycle that is more responsive and synergistic. Concurrent integration and continuous delivery (CI/CD) automation promises faster feedback loops along with more frequent feature releases, which speeds up the software release cycle [6]. An intriguing and quickly developing area of neuroscience is the development and refinement of closed-loop brain stimulation devices that do not need invasive procedures. Earlier suggestions for closing the feedback looping process in adaptive neurostimulation processes included using the patient's own rhythmic processes—like heart rate, respiration rate, as well as electroencephalogram (EEG) rhythms—to modulate the parameters of online automatic sensory stimulation [7]. From development to activities, including input from both, the quality and behavior of a system should be regularly monitored. The next step is to incorporate the comments into the development cycle again. Since the majority of developers are blissfully unaware of the myriad of things occurring in the manufacturing environment at any one time, the feedback loop is specifically designed to proactively identify harmful tendencies. An important part of DevOps managing projects is feedback loops. A longer feedback loop increases the risk of a subpar product or delays its delivery. It is critical for DevOps teams to comprehend the context of feedback loops as they work to bridge the gap between operations and development. Knowing what loops are currently in place and how to prevent accidental feedback loops are the first steps in optimizing feedback loops. In addition to concentrating on feedback loops based on humans, CI and CD techniques may substantially accelerate them. Injecting installations that caused undesirable behavior into the application allowed us to test the feedback system's anomaly detection methodology.

This work makes a double contribution: To start, we present a system that can use real-world data to analyze both the development and production environments for trends and novel forms of feedback by combining characteristics extracted from both. Furthermore, the data set is subjected to the suggested unsupervised machine learning methods, which not only prove that our notion can be implemented in reality but also provide a distinct picture of the data set.

The rest of the article is organized like this: Section II provides a synopsis of the relevant literature, while parts III and IV elaborate on the central idea and its practical application.

Section V then presents the results of our system assessment, and Section VI concludes the whole thing.

II. Related Work

Quick feedback loops and frequent software changes sent to production are key tenets of the DevOps methodology. Particularly in terms of performance, this is at odds alongside software quality assurance efforts. For example, in order to acquire findings that are statistically significant, performance assessment operations, like load testing [9], take a long period. To learn more about the approaches taken to performance in enterprise-level DevOps environments, we ran an industrial survey. The application of model-based methodologies, the level of detail of the performance data acquired, the frequency of running performance assessments, and the tools employed were of great interest to us. Performance evaluation in DevOps is not widely used due to the complexity involved in engineering methodologies and technologies, according to the study results. These respondents hail from a diverse range of industries. Our findings suggest that effectiveness analysis tools should be intuitive and straightforward to include into the DevOps pipeline if they want to gain traction among practitioners. Microservices is an emerging style of architecture that we attempted to use to solve the architectural issues. The system is now more resistant to design erosion, more easily modifiable, and easier to deploy. Concurrently, I saw additional difficulties related to testing, technological variety, shifting contracts among services, and a growth in the number of services overall. I go over several real-world ways to deal with these emerging problems, explain where Microservices aren't the best option, and point out where we need further study [10]. Factors impacting its execution are examined empirically in this work. It details the results of a comprehensive exploration case study that looked at how a product development company in New Zealand used DevOps. The research included in-depth interviews with six seasoned software engineers who were asked to track and reflect on the adoption of DevOps methods and concepts over time. The case study found that by implementing DevOps techniques, the deployment frequency increased from 30 releases per month to 120 releases per month on average, and there was a significant improvement in the level of natural communication and cooperation between the IT teams in development and operations. Our research shows that in order to reap the advantages of DevOps, certain technology enablers must be in place, such as an automated pipeline and cross-functional organizational frameworks [11]. Companies are using agile and lean software building practices to improve software quality and speed up the software development process. A combination of the words "development" and "operations," the name "DevOps" serves as an umbrella word to encompass their endeavors. We detail the methods that businesses use to adopt DevOps and the results that they get in this article. To begin, we provide a brief overview of the findings from a literature analysis that we conducted to find out what other scholars have said on DevOps. The findings of an exploratory research based on interviews with six firms spanning different sectors and ranging in size are subsequently detailed. Our research showed that when implementing

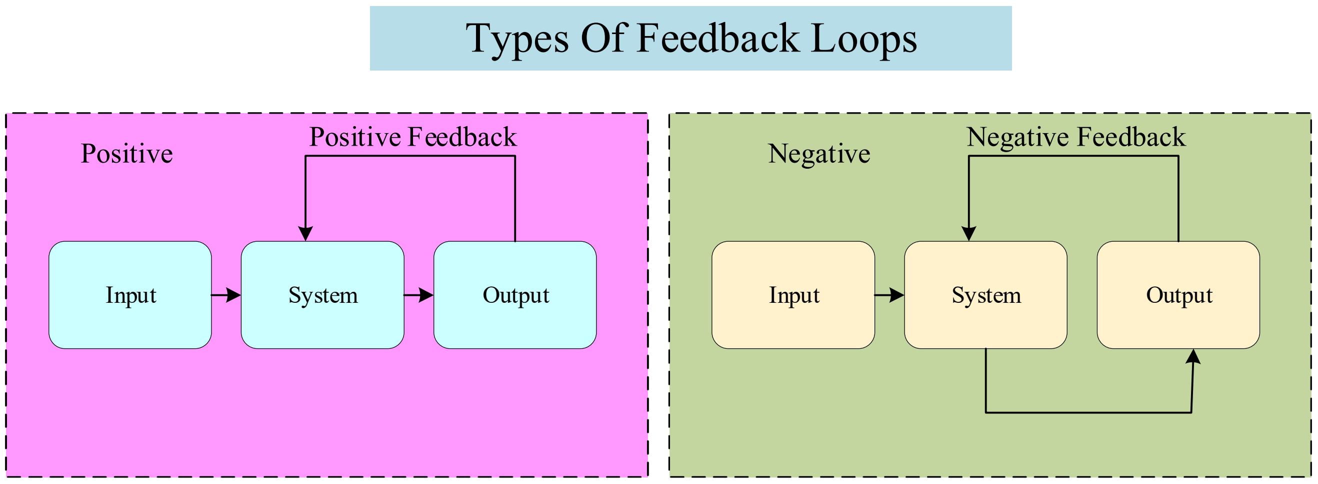
DevOps, all firms had favorable experiences with very few difficulties [12]. The goal of the software engineering practices that make up DevOps is to reduce the time it takes to implement changes to software design. Using infrastructure-as-code, or creating a blueprint with deployment specs prepared for cloud orchestration, is one of these strategies. In this abstract, we will take a quick look at the many abstractions and elements that go into creating and maintaining that blueprint, with a focus on the OASIS "Topology as well as Orchestration Specification for Cloud Applications" (TOSCA) standard that has been adopted by 60+ major industrial players across the globe [13]. The DevOps methodology is being used by software teams of every size to implement the microservice model and speed up application delivery. The usage of microservices is growing at a fast pace, according to industry trends. Nevertheless, there are difficulties associated with microservices. Autonomic concepts and solutions are required for large-scale deployment, which increases complexity and the likelihood of failure [14]. Researchers and practitioners in software engineering are increasingly interested in DevOps and continuous techniques. However, there is a lot of ambiguity, interchangeability, and uneven usage of the terminology. Our analysis of the published literature on continuous practices that DevOps reveals that the words are often used interchangeably, which hinders our capacity to evaluate these practices, their consequences, and the interactions between them. We argue that this lack of clarity and understanding is detrimental to the community as a whole. We provide recommendations to writers to assist them in reducing ambiguity in their writings based on this examination of assertions made by often referenced community sources, and our own study and practice with these ideas. Furthermore, we provide definitions that aim to represent the common understanding of the words while also separating them from each other [15].

III. Feedback Loop Processing in DevOps

In DevOps, feedback loops are essential for enhancing the quality of software development. Additionally, they guarantee the timely delivery of findings from development, testing, and deployment. The developers create the software and hand it over to the operations team, who work together as the DevOps team—a feedback loop in action. In order to identify bugs that the growth team has to address, the software is tested by the operations team. The feedback loop begins when the developers make the modifications and send them to the functional team. In the realm of DevOps, feedback loops are very essential. They guarantee high-quality and timely software deployment by offering several perks.

Types of DevOps Feedback Loops

Your DevOps process's feedback loop selection should be based on the loops' intended use. Feedback loops come in two varieties:



Reinforcing feedback loop (positive)

A positive feedback loop whose output increases the strength of the input is called a reinforcing or amplified feedback loop. Because the feedback loop alters only in one way, the total magnitude grows. It is the goal of the loop to improve the process. As an example, the operations team may easily deploy the code to reality after the development team develops high-quality software.

- **Balancing Feedback Loop (negative)**
  In a balanced feedback loop, the input is decreased by the output, creating a negative feedback loop. The teams in development and operations collaborate on this to restore system balance by slowing down the process. As an example, software is not released to production after development if the production team finds a few issues. The developer makes the necessary revisions to the procedure, resolves the errors, and commits the code once again. To increase the quality of software development and delivery, concentrate on tightening or shutting the feedback loop, regardless of the kind of loop.

An integral part of the DevOps methodology is the feedback loop. Through a series of well-defined steps, they guarantee that initiatives are simplified in a unified manner. Teams work together and communicate at each step to make sure there aren't any problems. We regularly review features once they are implemented based on real-world input from consumers. Permitting items to undergo several revisions in order to enhance their use and profitability. Users will also enjoy reduced downtime thanks to the pipeline's automated change pushes and ongoing monitoring.

Customer feedback

If you want to know how good and valuable your online goods are, look no further than customer reviews. Assumptions and hypotheses may be validated and the target audience's wants, preferences, as well as pain points can be better understood. Many methods exist for gathering consumer opinions, including questionnaires, in-person meetings, online reviews and ratings, data analysis, and social media. Iterating on features, prioritizing the backlog, and improving the user experience may all be aided by customer input.

User feedback

User feedback delves deeper into the specifics of how online goods are used and interpreted by the end users, offering a more targeted kind of consumer feedback. In order to assess the online products' usability, accessibility, and functioning, and to find any mistakes, bugs, or other problems that impact user happiness, it is helpful to collect feedback from users. User research, comments from users tools, user testing, and monitoring user behavior are some of the ways that user input may be gathered. Site speed, security, and optimization may all benefit from user input.

Code feedback

Both the code itself and code quality methods and tools may provide valuable input, which is referred to as code feedback. Code that follows the highest standards and best practices in web development, is consistent, legible, and maintainable is more likely to get positive comments. Code reviews, analysis, formatting, linting, and documentation are just a few of the ways that code feedback may be gathered. Refactoring, decreasing technical debt, and avoiding code smells are all possible outcomes of code feedback.

Test feedback

The feedback that is derived from testing procedures, testing instruments, test findings, and test reports is known as test feedback. In order to ensure that the code is up to par in terms of functionality, requirements, and quality standards, it is helpful to have test feedback. Several forms of testing—unit, integration, functional, performance, and security—are available for gathering test input. Code debugging, test coverage enhancement, and test automation may all benefit from test input.

Deployment feedback

Feedback on a deployment may be derived from a variety of sources, including the tools and pipelines used for the deployment, the results of the deployment, and the metrics measured by them. You can track and quantify how code changes affect online products and environments using deployment feedback. Several metrics, including deployment time, success rate, and failure rate, may be used to gather deployment feedback. Reducing deployment risks, increasing deployment value, and streamlining the deployment process are all possible with the aid of deployment feedback.

IV. Proposed WOrk

A. System Model

In order to find commonalities, abnormalities, produce feedback, and feed that input back into development, we built an input system that integrates data from operations with development. The fundamental idea behind our method is shown in Figure 1. The data that is now accessible is first retrieved from a constantly evolving application and stored in the database of the feedback system. To begin, several feature selection strategies are used to identify the important properties that will be used in subsequent analysis.

To further represent the current data, iterative applications of machine learning techniques (described in the step "Clustering") are next made. Data points are checked for suspicious activity and added to the feedback database with a behavior tag as soon as fresh data is generated. In a web app, all the data that is accessible is extracted and displayed. In addition, the ticket system generates an alert token anytime anything out of the ordinary happens, which gives the developer valuable insight into where the mistake may be coming from.

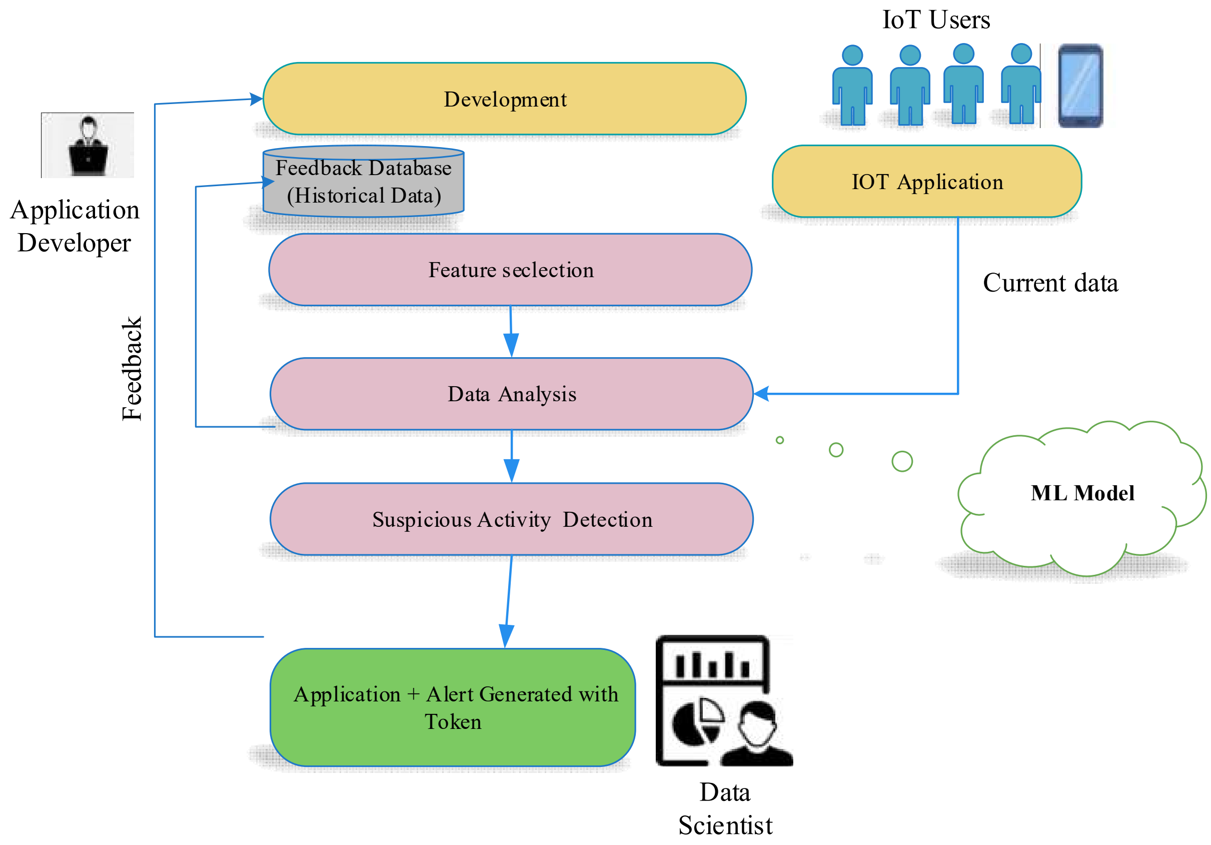


Fig. 1. Overview and concept of the feedback system**B. Data Collection & Feature Selection**

An internet of things (IoT) app built for the express purpose of proving DevOps techniques inside a big industrial organization provides the data used by our prototype. A number of edge devices collect and transmit sensor data from various places; a dashboard with its backend help to display this data. A Raspberry Pi and a temperature sensor are the components of each edge device, which measures the ambient temperature. The temperature and other system data are sent to the backend as well as saved in a database every five seconds. Processing unit utilisation (%), memory utilisation (%), networks transfer rate (B/s), as well as disk I/O (%) make up the four areas of system data. Each subproject, which may only be allocated to one part of the system, experiences an automated deployment process and is automatically activated throughout the system's continuous development. In order to match every system data category to its associated development data, the entries are separated into deployment segments.

We add up each segment's highest, lowest, mean, and difference in between the two. There is a constant flow of fresh details about the application's development as it is a continual process. Thus, by examining the repository commits, we are able to trace the code modifications for every deployment. In particular, the Git repository administrator made accessible the following features: the kind of commit, the quantity of changes per deployment, the amount of files added or deleted per commit, and the amount of lines added or deleted each commit. It is possible to directly map the two kinds of data in order to determine which code modifications resulted in which system data, as they may be separated according to deployments. The application's tools and database (such as the ongoing deployment tool and the Git repository manager) automatically and repeatedly extract the data needed by the feedback system, map it into the appropriate format, and store it in the database of the feedback system. The characteristics discussed in this section may be unique to this application, but the idea is generalizable to any application with measurable characteristics.

C. Improved Fuzzy C Means Clustering for Data Analysis

New understandings may be derived for the programmers from the data analysis. Therefore, a web app tells you how the app is doing right now, when it was last deployed, and anything that would indicate suspicious activity. The details page for each deployment gives you more information including the time stamp, the name of the project, the values of system data, the kind of deployment, the amount of commits, the amount of files added and deleted, and the amount of lines added and deleted.

The cluster is built using an approach called Upgraded Fuzzy C-Means Cluster (improved-FCM). When creating the cluster, this algorithm takes into account factors like past data. The majority of the time, the FCM method will increase latency by picking nodes at random to construct clusters. Our proposed Improved-FCM chooses the initial cluster based on the density of the Internet of Things setting at a given location, resulting in reduced latency. Here, Internet of Things (application features are clustered, producing data for processing. Applying the improved FCM clustering technique improves the outcomes of both the initial center decision and the target function. To do this, it improves performance by comparing random options. The membership function used for the Improved FCM is defined as follows:

$$M(p,q) = \sum_{i=1}^{n} \sum_{j=1}^{c} p_{ij}^{\sigma} S_{ij} \qquad (13)$$

Where "d" denotes the parameters and "M" denotes the group's function (d, l_d, r, E_r)For the clustered Internet of Things sensors, the membership function incorporates all membership levels of the IoT sensors.

$$\sum_{j=1}^{c} p_{ij} = 1 \; i = 1,2, \ldots.. n \qquad (14)$$

For clustering, the regional density that covers the area dictates which point in space is chosen as the center. Determining the average density between the sensors should be your first step. This density may be described in the following way,

$$Ad = \frac{n}{\pi}\left(\frac{d_{max}}{2}\right)^{-2} \qquad (15)$$

A definition of the local frequency calculation is given by the following statement:

$$l_d = \frac{n}{\pi\left(\frac{r'+r}{2}\right)^2 - \pi\left(\frac{r'}{2}\right)^2} \qquad (16)$$

The priori parameters used to determine the starting center are denoted by n, and r is the minimal value of the radius in this context. Using this method, we can prevent the random selection. Center selection based on calculation of the local densities reduces the number of technique iterations. Finally, here is a description of how we compute the membership degree using the area and distance densities,

$$M(p, q, l_d) = \sum_{j=1}^{c} \sum_{i=1}^{n} p_{ij}^{\sigma}\left(\frac{d_{ij}}{l_{dij}}\right)^2 \qquad (17)$$

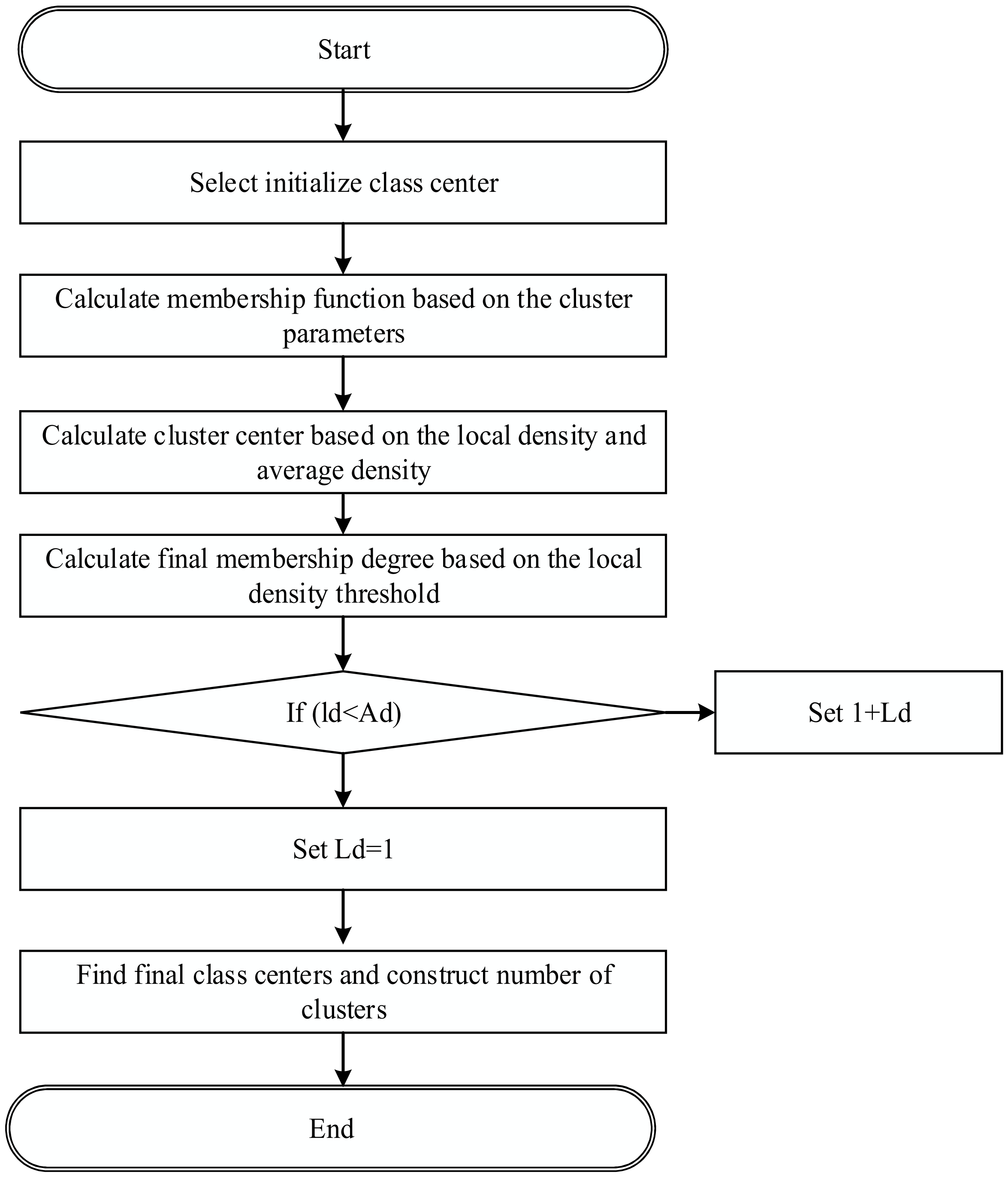

Start
Select initialize class center
Calculate membership function based on the cluster parameters
Calculate cluster center based on the local density and average density
Calculate final membership degree based on the local density threshold
If (ld<Ad)
Set 1+Ld
Set Ld=1
Find final class centers and construct number of clusters
End

Fig.3 Overall flowchart of local density based clusteringWhere, l_dThis shows the local density determined from the ith center of the kth sensor. The difficulty of collecting and classifying characteristics is significantly reduced by grouping the Internet of Things devices based on their proximity to one another and local density. If the density of the area is less than the average size, then the local density value should be set to 1. If the quantity of the local density is higher than the average density 1+l_d, which is defined as follows, then the number of the local density will correspond to the specified local density,

$$l_d = \{1, \quad l_{dij} \leq Ad \; 1 + l_{dij}, \; l_{dij} > Ad \tag{18}$$

Pseudocode for Improved FCM

```
1. Input: Devices $D = \{d1, d2..dn\}$
2. Output: Clusters $C = \{c1, \ldots, cn\}$
3. Begin
4. //Calculate initial cluster centers and local density
5. Initialize $D = \{d1, d2..dn\}$
6. for all devices do
7. Compute $d$ata points1
8. Compute $d$ata points2
9. Compute $d$ata points3
10. Compute $d$ata pointsn
11. Select minimum distance between $D$ for calculate $l_d$
12. if ($l_d$> $Ad$)
13. Set $i = i + 1$
14. else
15. Calculate the final membership degree and cluster center using eqn ()
16. end if
17. Compute local density threshold using eqn()
18. Construct cluster $\{c1, c2 \ldots. cn\}$
19. $C \leftarrow \{c1, c2 \ldots. cn\}$
20. end for
21. Return C
22. end
```

SERVICE DELIVERY PERFORMANCE CLUSTERS

Every group has its own unique way of measuring stability and throughput, which shows how well they're doing. Typically, this will produce four groups.

Performance level	**Lead time**	**Deployment frequency**	**Change failure rate**	**Mean time to resolve**
Elite	< 1 hour	Multiple times per day	0-15%	< 1 hour
High	1 day - 1 week	Weekly to monthly	16-30%	< 1 day
Medium	1-6 months	Monthly to biannually	16-30%	1 day - 1 week

Low	> 6 months	Fewer than once every 6 months	16-30%	> 6 months

They were compared to the clusters in terms of service delivery efficiency using performance and stability metrics in order to identify potential improvement areas.

New understandings may be derived for the programmers from the data analysis. Therefore, a web app tells you how the app is doing right now, when it was last deployed, and anything that would indicate suspicious activity. There is a detail view for every deployment that gives you more information like the time stamp, the name of the project, the values of the system data, the kind of deployment, the amount of commits, the amount of files added and deleted, and the amount of lines added and deleted. It also creates eight graphs that show the data representation using archetypes and clusters, and it shows where the current deployment is in relation to the previous deployments and how it stands out. Anomalies that are found are shown by highlighting the fields that are generating the anomaly. One probable reason for its detection might be the deployment's unusual behavior compared to the others, which can be easily understood in this fashion. Furthermore, should a mistake occur, a token will be immediately generated inside the token system. The purpose of this document is to assist the programmer in identifying where the program is crashing. Tokenizing important details on the issue and its potential causes is necessary for this to occur. The feedback system gathers information on the (sub-) project, the time and date, the error kind (such as a spike in CPU use), and the files that may have been updated, which might explain the behavior change. It then automatically generates the token after converting the data to a written and comprehensible format.

V. Results & Discussion

The system that is being examined is the backend system of a public transportation app that processes money and tickets. Built and managed via the cloud, this system makes use of DevOps practices and Microsoft products. An architecture based on microservices is being considered for the system. With Microsoft's data platform, Azure Monitor, we keep tabs on the health of every service. Azure Monitor is a platform that unifies data collection, processing, alerting, and visualization from various Azure services and applications. Metrics and logs are both accessible under this data platform. The numerical numbers that represent the observations of a certain system within a specified time stamp are called metrics. Numerical and textual data are used to represent logs, which reflect events that occurred at a certain instant in time. When anything out of the ordinary occurs in the monitored resources' data, alert rules may be built up using either metrics or logs. As can be seen in Table 2, the case firm has put in place basic criteria to identify 500 error requests, unexpected dependency call raises, and unsuccessful HTTP requests. Notifications are delivered to a specific Slack channel or by email when certain conditions are fulfilled, which triggers alarms. The operational data displayed in Table 2 is only a fraction of the total data accessible in Azure Monitor. However, for the sake of this study, we will be concentrating on the chosen logs and metrics. Metrics and logs pertaining to the data kinds used for establishing present alert rules and for debugging in the event of anomaly detection were selected from among all available observations of various system components. All twenty services have the alert rules stated in Table 2, and two platforms get messages on heightened alarms. The Slack channel gets notifications when there's an internal server error 500, and email gets notifications when there's an unexpected increase in dependency failures and unsuccessful requests. There have been many reports of difficulties from the development team about the management and response to alerts that have been triggered using this setup. In addition, a deluge of irrelevant warnings makes their development environment unpredictable, which makes ordinary work more difficult.

Table.2. Features for Feedback Loop Processing

Features for feedback information	Type of features	Alerts configured
System features	No of requests	(1). Server Error Message 500 (2). Failed HTTP Requests (3). High processing time (4). Flooded requests
	CPU Time	
	Errors HTTP	
	Server Errors	
	Response Time	
Log features	Requests	(1). Exception errors (2). Void Request
	Exceptions	
	Traces	
Application features	Server Exceptions	(1). Dependency failures (2). Service not matched for the given request
	Exceptions	
	Dependency features	
	Failed Requests	

By comparing our prototype to the pure unsupervised ML approach for finding outliers, which signify alerts, in multidimensional unlabeled data sets, we were able to verify our model and existing one, namely multivariate anomaly detection (MAD). For the implementation, we stuck with the PyOD Python toolkit and went with the DevOps application service provider approach.

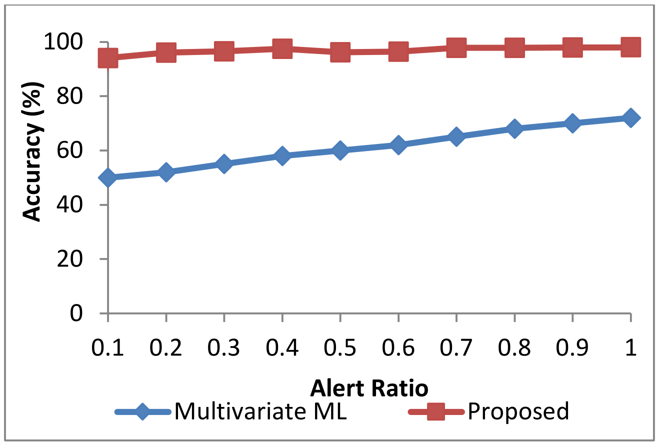


Fig.4. Accuracy Performance

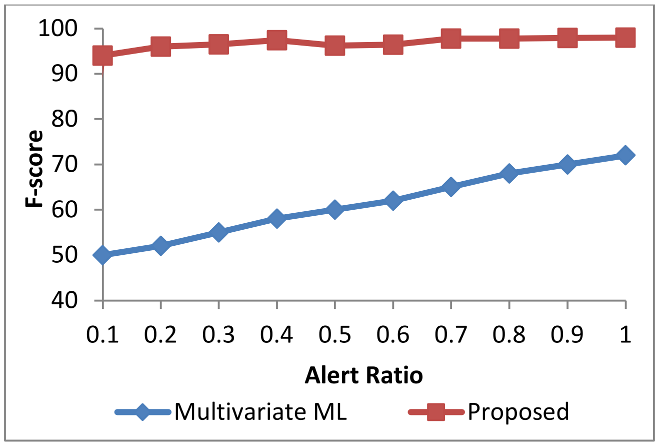


Fig.5. F-score Performance

Both models are in sync using DevOps and the Feedback Loop system of controls technologies, as shown in the results of the experiment (Fig.).

The F1-score, which is a harmonic mean of recall and precision that indicates a model's accuracy, is more than 0.9 for both the Devops model and the ML model. On the other hand, pure unsupervised ML may detect when there is a mismatch between the weight of some services with their metrics and the target classes.

In addition, we demonstrated that the systems' imbalanced operations information and system-specific traits, which are essential for detecting both minor and major failures, are better captured by the tailored unsupervised machine learning approach. Consequently, the feedback data collected has strengthened the link between development and operations and may now notify developers of any issues.

VI. Conclusion

The feedback system may provide fresh perspectives on an operational system by merging data from operations and development, taking development modifications into account. If an issue does arise, the system can identify it, locate its source inside the program, and provide developers with useful feedback. This method makes it possible for the DevOps feedback process to be enabled by facilitating a data cycle between the development environment and the production environment. Applying our approach to a real-world application development scenario allowed us to assess it. Any measurable information may be utilized as a characteristic in the feedback system, thereby expanding its applicability. Additionally, the algorithms are not restricted to a certain set of features, making the feedback system suitable for other applications as well. While there have been some improvements, there are still some gaps that might be filled in the future. Take the classification methods as an example; they need a substantial quantity of data. This means that thresholds must be employed until there is enough data to use the feedback system at the beginning of the design phase. Overhead due to data volume is another well-known issue with monitoring systems. Performance problems may arise during data transmission and database storage. It is possible to tailor the interval between measurements so that only essential data points are transmitted and saved in the database by examining various intervals. In addition, it might be helpful to automatically off monitoring points associated with characteristics that aren't utilized in the analysis since they weren't selected. Data quality is the ultimate determinant of feedback system quality and feedback substance. Developers incur extra expenses due to the fact that the gathering and storage of operations data is application-specific and must be performed by them. In order to reduce the inhibition level, more study might look at ways to help this process; for example, it could consider feedback optimization with all the necessary facts and information.

References

1. Medina, O., & Schumann, E. (2018). Getting up and Running: Set up Your Environment.
2. Fröschle, H. (2017). DevOps. HMD Praxis der Wirtschaftsinformatik, 54, 171-172.
3. Jabbari, R., Ali, N.B., Petersen, K., &Tanveer, B. (2018). Towards a benefits dependency network for DevOps based on a systematic literature review. Journal of Software: Evolution and Process, 30.
4. Luz, W.P., Pinto, G.H., &Bonifácio, R. (2018). Building a collaborative culture: a grounded theory of well succeeded devops adoption in practice. Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement.
5. Sánchez-Gordón, M., &Colomo-Palacios, R. (2018). Characterizing DevOps Culture: A Systematic Literature Review.
6. Glasgow, R.E., &Estabrooks, P. (2018). Pragmatic Applications of RE-AIM for Health Care Initiatives in Community and Clinical Settings. Preventing Chronic Disease, 15.

7. Perera, P., Silva, R.T., &Perera, I. (2017). Improve software quality through practicing DevOps. 2017 Seventeenth International Conference on Advances in ICT for Emerging Regions (ICTer), 1-6.
8. Bass, L.J. (2017). The Software Architect and DevOps. IEEE Software, 35, 8-10.

9. Bezemer, C., Eismann, S., Ferme, V., Grohmann, J., Heinrich, R., Jamshidi, P., Shang, W., Hoorn, A.V., Villavicencio, M., Walter, J., &Willnecker, F. (2018). How is Performance Addressed in DevOps? Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering.
10. Chen, L. (2018). Microservices: Architecting for Continuous Delivery and DevOps. 2018 IEEE International Conference on Software Architecture (ICSA), 39-397.
11. Senapathi, M., Buchan, J., & Osman, H. (2018). DevOps Capabilities, Practices, and Challenges: Insights from a Case Study. Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018.
12. Erich, F., Amrit, C., &Daneva, M. (2017). A qualitative study of DevOps usage in practice. Journal of Software: Evolution and Process, 29.
13. Artac, M., Borovsak, T., Nitto, E.D., Guerriero, M., &Tamburri, D.A. (2017). DevOps: Introducing Infrastructure-as-Code. 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), 497-498.
14. Pallis, G., Trihinas, D., Tryfonos, A., &Dikaiakos, M.D. (2018). DevOps as a Service: Pushing the Boundaries of Microservice Adoption. IEEE Internet Computing, 22, 65-71.
15. Ståhl, D., Mårtensson, T., & Bosch, J. (2017). Continuous practices and devops: beyond the buzz, what does it all mean? 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 440-448.