Robust Procedural Learning for Anomaly Detection and Observability in 5G RAN

Tobias Sundqvist, *Member, IEEE*, Monowar Bhuyan, *Member, IEEE*, and Erik Elmroth

***Abstract*—Most existing large distributed systems have poor observability and cannot use the full potential of machine learning-based behavior analysis. The system logs, which contain the primary source of information, are unstructured and lack the context needed to track procedures and learn the system's behavior. This work presents a new trace guideline that enables a component- and procedure-based split of the system logs for the future 5G Radio Access Network (RAN). As the system can be broken into smaller pieces, models can more accurately learn the system's behavior and use the context to improve anomaly detection and observability. The evaluation result is astonishing; where previously state-of-the-art methods struggle to learn the behavior, a fast, dictionary-based algorithm can detect all anomalies and keep false positives close to zero. Troubleshooters can also more quickly identify anomalies and gain useful insights into the component interaction in RAN.**

***Index Terms*—Observability, trace guidelines, anomaly detection, radio access network, 5G.**

I. Introduction

MACHINE learning (ML) can aid developers in understanding the behavior of large distributed systems like the 5G Radio Access Network (RAN). To reach the full potential of ML-based behavior analysis, it is essential to instrument the software with a larger scope than the developers are used to. System logs are the primary source of information when troubleshooting a system. They can reveal the system's functionality and guide troubleshooters in the search for anomalies and root causes. Unfortunately, system logs are often unstructured and may vary in a large system, making it difficult to distinguish between normal and abnormal behavior. ML has been seen by many as a solution that could suggest where developers should add logs in the code [1], how to mine logs [2], and also learn the behavior [3]. An underlying problem in many large distributed systems is that data quality is often too poor to allow ML to learn the complex behaviors of the system [4]. As hundreds or thousands of developers develop and maintain small parts of a system, there is no consensus on instrumenting the software from an ML perspective. Unifying the instrumentation and enabling procedure tracking through the whole system is also necessary. Lately, there have been attempts to define guidelines on instrumenting software, interpreting, and using system logs for anomaly-detection purposes [5]. The guidelines aid in extracting helpful information from system logs, but it is still difficult to grasp how parts are affected and interact when the system solves different tasks. The OpenTracing standard[1] with OpenTelemetry[2] provides a guideline for how to instrument a system to track the microservice interaction. Previous research based on these guidelines has, e.g., illustrated how bottlenecks and abnormal latencies could be found [6], [7]. However, experience in large industries, such as RAN, shows that it is difficult to implement distributed tracing in large systems [4], [8]. Instead, we propose a procedure-based software instrumentation guideline that enables novel methods to learn the RAN behavior from system logs events. Our proposed method enhances the observability of RAN and identifies anomalies quicker and more accurately than current state-of-the-art methods.

The main contributions of this work are:

- Defines trace guidelines based on a system's functional description. The guidelines facilitate understanding how objects are interconnected and enable tracking of procedures through an extensive distributed system, such as the 5G RAN.
- Explains and visualizes how observability improves when instrumenting the system with the new guidelines.
- Introduces a procedure-based anomaly detection method that can take advantage of the extra context the new guidelines provide. The method learns the component interaction in RAN together with the behavior of each component for systemized procedures. The increased granularity improves anomaly detection accuracy, surpassing previous state-of-the-art methods, and enhances the system's observability.
- Evaluates anomaly detection and observability improvement when updating an advanced 5G testbed with the new trace guidelines.

Organization: The rest of this paper has the following organization: Section II reviews the related research, and Section III provides a profound description of the background problem. Section IV explains how to implement the new trace guidelines and how anomaly detection and observability can be improved using them. Section V presents the

Manuscript received 19 April 2023; revised 28 August 2023; accepted 24 September 2023. Date of publication 2 October 2023; date of current version 15 April 2024. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP), funded by the Knut and Alice Wallenberg Foundation. The associate editor coordinating the review of this article and approving it for publication was D. Puthal. *(Corresponding author: Tobias Sundqvist.)*

The authors are with the Department of Computing Science, Umeå University, 901 87 Umeå, Sweden (e-mail: sundqtob@cs.umu.se; monowar@cs.umu.se; elmroth@cs.umu.se).

Digital Object Identifier 10.1109/TNSM.2023.3321401

[1] https://opentracing.io/specification/

[2] https://opentelemetry.io

experimental results, which are discussed with the limitations in Section VI. Finally, Section VII concludes the paper and mentions interesting future topics.

II. Related Work

This work provides new methods to improve anomaly detection and enhance the observability of RAN using a new software trace guideline. This section explores the limitations of previous research in these three areas. The related techniques are also further explored in Section IV.

A. Trace Guidelines

Mining system logs to extract useful information and dependencies between events is an extensive research field; a survey of some of this research can be read in [2], [9]. Rather than proposing yet another approach to mining unstructured logs, we define trace guidelines that, if followed, enable enhanced observability and add a valuable context for the machine learning methods. The guidelines can also aid in reducing the noise that is typically introduced when collecting and pre-processing the data [10].

A recent industry survey [4] on microservice tracing and analysis concludes a great need for intelligent trace analysis. Still, the main hurdles are the lack of trace-data annotation and poor data quality. An attempt to structure system logs and make them more suitable for machine learning methods is presented in [5]. The applications in a leading telecommunications company are then instrumented to produce a text format that machine learning methods can easily interpret. The downside is that the new format makes it more difficult for humans to interpret the logs and understand the system. Some attempts exist to use the OpenTracing standard with OpenTelemetry to gain more insight from the distributed systems with the span and trace context [6], [7], [11]. Still, we have not found any research investigating how the standard would improve log sequence analysis methods and observability.

Knowing what, how, and where to instrument the code is today a large research area of which an extensive survey is available at [12]. Most debug information is added during the early development of features to verify that the implementation works as intended. Extra debug information may be added later when the feature is tested in a larger perspective, before releasing the software, or when troubleshooting the system. The trace guidelines proposed in this work aim to improve the perception of the system as all parts work together at the end of the development cycle.

B. Anomaly Detection in System Logs

The system logs contain debug information that reflects the system's behavior. An anomaly is, by definition, something that deviates from normal or expected behavior. The anomalies can be detected by manually analyzing the logs, but the vast amount of data in RAN makes it difficult for RAN troubleshooters to find the deviations. Machine learning methods can learn from features in system logs to help end users identify these anomalies [12], [13], [14], [15]. A challenge for these methods is the poor quality of the input data. Without the structure and context later discussed in Section II-A, the methods need to learn from noisy data that lack context, which results in poor performance. Adding the missing procedure context using the trace guidelines can improve anomaly detection and increase the system's observability. An added context can also provide opportunities to find new types of anomalies [6], [7].

This work is limited to identifying anomalies from a supervised approach where the training data is labeled as normal. This approach suits continuous integration environments where the software functionality is continuously tested. The models can then learn the behavior from passed test cases and identify the anomalies in the failing test cases. Previous state-of-the-art methods such as LogRobust [10], DeepLog [16], or BoostLog [17] can learn the behavior from such continuous integrations environments, but cannot differ between the many systemized scenarios in RAN. Each model must then learn the increasing number of configurations and features in RAN, resulting in larger and more complex models. The proposed methods in this work utilize the added procedure information in the system logs to automatically group similar behavior and detect deviations more quickly and accurately than previous methods.

C. Observability

Understanding how system parts interact is essential, and much has been done to improve observability. One approach is to mine system logs from distributed systems to find dependencies between system parts [18]. Another method is to enhance or monitor the protocol between the applications to track message paths. This is the case in, e.g., tools like X-Trace [19] or CloudSeer [20]. CloudSeer also instruments the applications to extract information as messages are sent between the distributed parts. Another example of this approach is presented in [21], where the authors instrument an event sourcing system to track request paths through a system. Many of the 5G RAN protocols cannot be extended to include the extra information proposed by previous research. This work suggests an alternative solution where the system interaction can automatically be identified through procedural learning.

Many industrial microservice applications have implemented distributed tracing to improve the system's observability [22]. The tracing information provides useful information that can help end users understand the microservice interaction, but it is problematic to integrate into large systems [4], [8]. Many of the protocols in RAN cannot be altered to add the extra information that distributed tracing requires. Instead, this work explores how observability can be enhanced using the information in system logs. While distributed tracing only provides information about the microservice interaction, the system logs also contain valuable behavior information that helps developers troubleshoot the system.

A special *observability-score* is defined in Section IV-D. The score measures how fast troubleshooters understand the ongoing procedures and identify the anomalies. Other researchers, e.g., [23], have used this term to monitor security attacks, and the score then reflects the number of features

detected. In [24] and [25], the score represents the tracking of a mobile robot, respectively, an object. Since observability is a broad area, opinions differ regarding how observability should be measured. To the best of our knowledge, this is the first time that observability is measured from a procedure- and anomaly-detection perspective where people, not algorithms, try to interpret the inner states of the system.

III. Background

This section explains the challenges of identifying log anomalies in large distributed systems. It also defines the terminology further used in this work.

A. Terminology

To more easily understand the challenges and methods in this work, some terms need to be defined:

- *Component* - A part of a system, e.g., a microservice, part of a microservice, container, or a systemized area that performs a specific task.
- *Procedure* - A task handled by the system, usually defined in the functional description of the system.
- *Sub-procedure* - Several system components can work together to fulfill a procedure, for example, in a distributed system with several microservices or containers. We define a sub-procedure as the work performed by a single component.
- *System log* - The log that contains valuable debug information. The debug information is typically added during the design or system test to aid in troubleshooting.
- *Event* - The software instrumentation stores important information as one or several lines in the system log; we call these lines an event.

B. Problem Definition

To understand the need for a new trace guideline, we first describe the development of large systems. When planning new features, the functional design specification (FDS) specifies the requirements and a high-level implementation description. The FDS contains flow charts, screenshots, sequence diagrams, or use cases and is the primary source of information for developers that implement the feature. The feature can affect several system parts, and many teams and developers are usually involved in the implementation. Tests at various system levels ensure that individual parts and the whole system work. The software instrumentation then provides valuable information in the system log or the system's standard output.

Fault identification is usually fast in small systems where the developers are experts on the whole system. Nevertheless, locating faults in large distributed systems can be much more troublesome, where troubleshooters only know small parts of the system. Each developer decides on the debug information to store, and the instrumentation can differ significantly between components. To track procedures that affect several components, troubleshooters need to be experts on all parts or consult other experts to correctly interpret the events and relate them to the systemized feature in the FDS. The enormous variation in instrumentation is troublesome for the machine learning society, and it is often difficult to extract accurate information [5]. Developers also tend to change the instrumentation during development, and previous studies have shown that 20-45% of logging statements change during the lifetime of a system [10], [26]. This work highlights the importance of unifying the instrumentation throughout the whole system and makes the developers aware that a slight change in the instrumentation may affect machine learning models that learn the behavior from system logs.

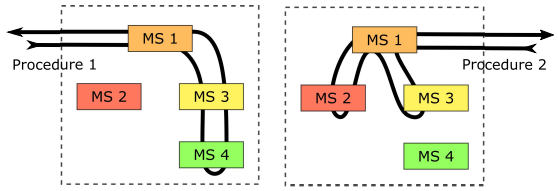

Fig. 1. A simplified example of how four microservices are affected by two different procedures.

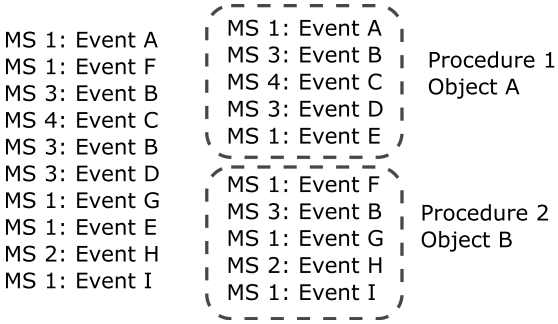

Fig. 2. Simplified log example from two concurrent procedures.

Another problem is the lack of context in system logs that allows linking between events and a specific procedure. This reduces the system's observability and increases the noise for machine learning algorithms. To illustrate the problem, we examine one simplified example where a distributed system consisting of four microservices (MS) handles two different procedures, see Fig. 1.

The arrows in the figure illustrate the triggering order of the MSs. The procedures may occur in parallel with each other and affect various objects. The simplified example only stores one event each time an MS is affected by a procedure, see Fig. 2.

From the stored event list to the left, it may be possible to determine that several simultaneous procedures are ongoing. Still, it is difficult to grasp which events belong to a particular procedure without the procedure context. Another problem is that one procedure may trigger different events depending on the object affected and the past procedures. In such cases, we also need the object context to realize how the procedures have affected the object.

From a machine learning perspective, it is essential to distinguish between objects in a system log, e.g., utilized in [16], [17], [27]. Separating objects also help troubleshooters follow and compare the chain of events, but it would be even more efficient if the ongoing procedure were known. Without the procedure information, troubleshooters need much experience to relate each event to a specific procedure. The

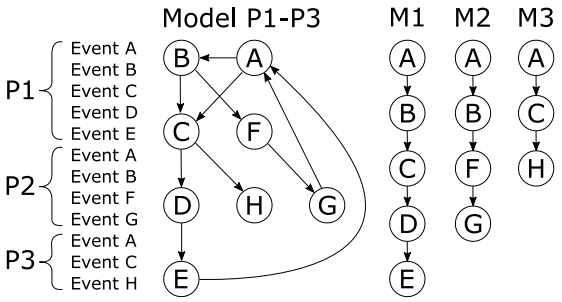

Fig. 3. Sequences of events are easier to learn when the procedure context is known.

problem is that the troubleshooter's expertise is often limited to a small part of an extensive system. It can also be challenging to distinguish between normal and abnormal behavior when the procedure affects several system parts. To further illustrate the problem when handling many procedures, we examine the sequence of events in Fig. 3.

A machine learning method unaware of the procedure context must learn the complex behavior seen as Model P1-P3. This work explains how an added procedure context allows us to train separate models for each procedure and component.

When procedures affect several components in a system, as illustrated in Fig. 2, it is also essential to know the relation between objects and components. Previous research has attempted to utilize object IDs, and specific log messages to create hierarchical relations between the objects [28], [29]. Instead of using statistical methods to guess which objects and components belong, we propose using extra instrumentation to tie parts together.

IV. System Design

To cope with the problems described in Section III-B, this work starts by defining a new trace guideline in Section IV-A. The new guidelines add procedural information to existing debug information and highlight the importance of structured logs. Section IV-B then explains how this procedural information can be utilized by a new procedural-based method to improve the speed and accuracy of anomaly detection. Section IV-C further describes how observability can be enhanced using the new procedural-based methods to visualize the system behavior and highlight where anomalies occur. Finally, the evaluation process is described in Section V-A.

A. Trace Guidelines

There are various levels of maturity when it comes to system debugability. With debugability, we mean the possibility to observe a system by analyzing the system logs. The maturity of the system can be divided into the following levels:

0) There are no events stored at all; not possible to use the system logs to debug the system.
1) Some events are stored, but no level of instrumentation is used.
2) It is possible to distinguish between DEBUG, INFO, WARNING, and ERROR.

```
User equipment 5, state: ENABLED, num bearers 1
State: ENABLED, num bearers 1, user equipment 5
```

Fig. 4. Example of unstructured events.

3) Various levels of instrumentation are used for DEBUG traces, which makes it easier to monitor specific parts of an extensive system.
4) The events are structured to make it easier to extract meaningful information.
5) The events are given a context that allows procedure tracking and understanding how parts work together to fulfill procedures.

All lower levels are fulfilled for each level, and as levels increase, the system's observability is also improved. Most large systems never reach level 4, where machine learning methods can efficiently extract data from system logs. This work demonstrates the advantage of reaching level 5 using the new trace guidelines.

The simplified log in Fig. 4 illustrates the downside of only reaching level 3 in debugability.

Both events in the figure indicate the same thing, and a human expert, who automatically filters out keywords such as state, bearers, and user equipment, can easily interpret them. To use this information in machine learning algorithms, experts must create clever parsing rules adopted for the particular system version. As debug information is changed or added during development, there is a constant need to update the parser, and this is not a very good long-term solution. A more suitable approach for machine learning is to separate the information into an **identifying** section, and a **parameter** section [5]. The purpose of the identifying section is to locate the affected part of the system, which may be, depending on the system, e.g., server, file, function, component, class, or instance of an object. The parameter part should contain an informative message about the event or provide important key parameters from a debugging perspective. In the example in Fig. 4, the identifying and parameter part would be *user equipment 5* and *state: ENABLED, num bearers 1*, respectively. Knowing the connection between events and procedures is vital to further enhance the system's observability. Machine learning methods can automatically identify different procedures and train a model for each procedure by adding a procedure context. To summarize, we propose that developers use the following debug information when instrumenting the software:

- *Time* - Time the event occurred, also referred to as a timestamp. It enables sorting events and measuring the time between procedures and events.
- *Identity* - This part usually contains a hierarchy of objects used to locate the lowest-level object. It could, for example, be a server, component, or object that handles a request.
- *Parameter* - Message that provides insight into the system. For instance, state updates, variables, signals received, or finished and starting services.
- *Procedure* - Contains two fields: identity and procedure name. An identity is a sequential number used

[12:07:37.653] Identity: { node=1, service=CU_UE, id=2}, Parameter: { ContextSetup:Pending }, Procedure: {id = 43, name = setup_ue }

Fig. 5. Example of an event written with new trace guidelines.

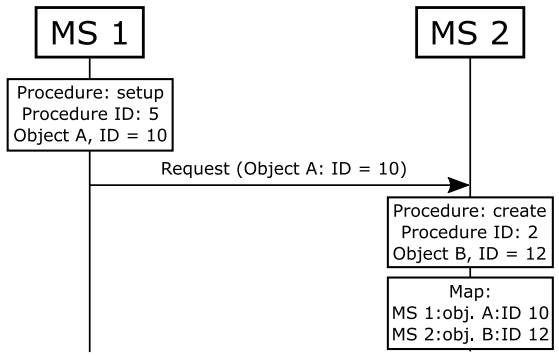

Fig. 6. Mapping between Object A in microservice 1 and object B in microservice 2.

to connect events to a certain procedure. The procedure name relates the events to procedures described in the FDS. Several events in one procedure can use the procedure name field. In these cases, the combination of procedure names creates a unique name for that procedure. One event could state *Setup*, another event: *Alternative 2*, and the combined procedure name will then be *Setup:Alternative 2*.

Fig. 5 illustrates an example of an event written with the new guidelines. The event starts with a timestamp and continues with the Identity, Parameter, and Procedure. The events may also contain information such as function, line of code, or application name. This kind of information can be helpful in quickly identifying the location of the events in the software, but it is not investigated further in this work.

In systems where several components handle a procedure, it is crucial to provide information that enables tracking of the procedure throughout the whole system. One easy solution would be to extend the protocols between the components to include the procedure ID. Unfortunately, this is impossible in systems like RAN, where the many standardized protocols disallow this information. The protocols only contain the identities of the affected objects but it is up to each vendor to systemize the procedures and decide how the system parts should interact to fulfill the procedures. To overcome this problem where the interfaces are restricted, this work proposes an alternative solution to track the procedures through the system. Consider the two microservices in Fig. 6.

Microservices 1 and 2 are part of the same procedure, but the signaling protocol does not allow sending the procedure ID between the microservices. Instead, two unique procedure IDs are set in the two microservices. The procedure information would in the example state that there is a sub-procedure called *setup* with procedure ID 5 affecting object A with ID 10 and another sub-procedure called *create* with procedure ID 2 for object B with ID 12.

A special mapping technique is introduced to connect objects and procedures that enables tracking of the procedure interaction through the system. Consider the example in Fig. 6

[14:11:23.420] Mapping: { object1=A, object2=B, id1=10, id2=12 }

Fig. 7. Example of a mapping event that connects two objects.

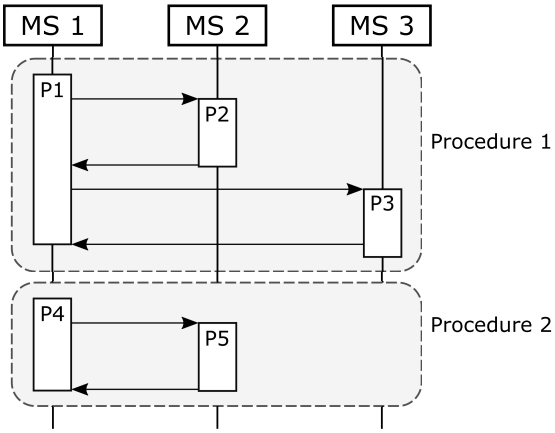

Fig. 8. Grouping of procedures that belong together, based on the signals sent.

where a sub-procedure in MS1 triggers a sub-procedure in MS2. The objects A and B in Fig. 6 can, for example, be the same cell object or an user equipment requesting an update of a specific tunnel or radio bearer. To map the sub-procedures to each other and enable tracking procedures through the system, a special event is triggered as the components interact:

- *Mapping* - A special event connecting objects from two components. The mapping should contain the identity and names of the two objects and enable the pairing of objects and components. The object IDs are typically sent between components in RAN, and the mapping information can be stored when certain requests or indication signals are received.

While the procedural information connects a procedure ID to an object ID, the mapping information connects two objects. An example of the mapping trace can be seen in Fig. 7.

The mapping information states that object A with ID 10 is connected to object B with ID 12. Combining the mapping and procedure information implies that sub-procedures *setup* for MS1 and *create* for MS2 can be part of the same procedure.

The sub-procedures should then be grouped into procedures. Consider the example in Fig. 8 where sub-procedures P1, P2, and P3 are part of procedure 1, and sub-procedures P4 and P5 are part of procedure 2.

There is always a starting point for the systemized procedures; in Fig. 8, it occurs at the beginning of P1 and P4. These sub-procedures are defined as *Trigger-procedures* and can work as a procedure divider. An easy solution to mark them is to add a unique string tag to the procedure name information. All sub-procedures mapped to a *Trigger-procedure* belong to the same procedure. Section IV-B1 further explains the grouping procedure.

B. Anomaly Detection

In this work, an anomaly is defined as a sequence of events that differ from normal behavior. The methods learned from system log behavior and the detectable deviations are categorized as follows:

- *Order of events* - For some features, the events must occur in a well-defined order; otherwise, the procedure will fail.
- *Missing or extra event* - Events not seen before or new execution paths may cause additional events that generally do not belong to a procedure. Functionality may also fail, which leads to the shortening of procedures and missing sub-procedures.
- *Latency* - A software update, heavy load, or conflicts in the system may delay part of a procedure. Measuring the time between the events in a procedure can detect these delays, and the total procedure time can be calculated using the time stamps.
- *Parameter check* - The usual range of parameters in a procedure can be part of the events and utilized to detect deviations.

As most previous research has developed algorithms to detect anomalies in the first two categories, the scope of this work is limited to covering these two.

In the last ten years, it has become popular to use recurrent neural networks, such as Long Short-Term Memory (LSTM) [30] or Gated Recurrent Unit (GRU) [31], to detect anomalies in system logs. In 2017, DeepLog [16] was a state-of-the-art example of how system logs could be divided into a sequence of events for various objects, and an LSTM method learned the order of sequential events. Our work later improved the anomaly detection accuracy using an AdaBoost ensemble of LSTM models [17]. The attention layer [32] was introduced in 2017 and has helped to enhance anomaly detection in system logs further [10], [28], [33], [34], [35], [36], [37].

The rest of this section describes our new procedural learning algorithm together with the baseline methods that are part of the evaluation.

1) Procedural Learning: The architecture of our new procedure-based method is presented in Fig. 9. The system is a 5G RAN testbed that stores system logs for control and user plane scenarios. The traditional method for analyzing logs is to create rules or use machine learning to parse the logs and retrieve helpful information. In systems that handle many objects simultaneously, it is sometimes possible to separate events that belong to the same object [16], [17]. Machine learning (ML) methods can then learn the order of an object's events. A problem with this approach is that the rules and parsers will lose some information [10], and the ML methods need to learn all events that belong to one object. The methods' accuracy will likely deteriorate as the system grows with more features.

Our trace guidelines enable the automatic extraction of important information without parsers or complex rules. It can also help distinguish between procedures and automatically learn the behavior for each procedure and object type.

We define a new dictionary-based method to demonstrate how a less complex method can benefit from the procedure and component context. We create two dictionaries where the first can detect anomalies within sub-procedures, and the second handle anomalies that arise due to new, missing, or extra sub-procedures. The dictionaries are called *sequence-dictionary* and *procedure-dictionary*, respectively. The methods learn the occurrences of events within sub-procedures and which sub-procedures occur for a procedure.

The main procedure for training the *sequence-dictionary* follows the four steps:

Step 1: The sub-procedures are grouped based on the procedure ID and process name.

Step 2: All procedure name elements for one sub-procedure are concatenated with the process name to form a procedure key (*process-name$procedure-names*).

Step 3: Create an event vector for each sub-procedure where the number of occurrences of each event is stored.

Step 4: Concatenates the event vector into a string and adds it as a dictionary element within the procedure.

The sub-procedure can then be grouped into procedures using the following steps:

Step 1: Sort all sub-procedures found in step 1 of the training phase for *sequence-dictionary*, with an increasing timestamp. Examine all procedure names in the list and repeat steps 2-3 for each name.

Step 2: Check the procedure name to identify if the sub-procedure is a *Trigger-procedures*. If so, create an empty vector with the size of all unique sub-procedures in step 1. This vector contains the number of occurrences of each sub-procedure within the procedure. If one vector contains a sub-procedure already identified (see step 3), then store the previous vector as a concatenated string together with the *Trigger-procedures* name in the *procedure-dictionary*.

Step 3: If the sub-procedure is not a *Trigger-procedures*, increase the sub-procedure's count in the current procedure vector.

Step 4: Finally, check the mapping information to ensure that all sub-procedures belong together. If more than one sub-procedure is present in the procedure vector, remove those sub-procedures that cannot be mapped to the other sub-procedure.

The sub-procedures are first identified in the data in the same steps as for the *sequence-dictionary* to detect anomalies. Each sub-procedure is then examined according to the steps:

Step 1: Create the *process-name$procedure-names* key and retrieve all event vectors from the *sequence-dictionary*. If the *process-name$procedure-names* is not present in the *sequence-dictionary*, mark the sub-procedure as abnormal, and skip the rest of the steps.

Step 2: Create the event vector and compare it with the event vectors from step 2.

Step 3: Mark the sub-procedure as abnormal if no event vectors are matched.

Step 4: If the event vector is abnormal, calculate the Euclidean distance between all event vectors and select the one that has the smallest distance. Present the most similar sub-procedure to the troubleshooters; learn more about this in Section IV-C.

The second part utilizes the *procedure-dictionary* to detect procedure anomalies. The procedures are first detected using the steps to create the *procedure-dictionary*. The identified procedure is then compared with the procedures in the *procedure-dictionary* using the steps:

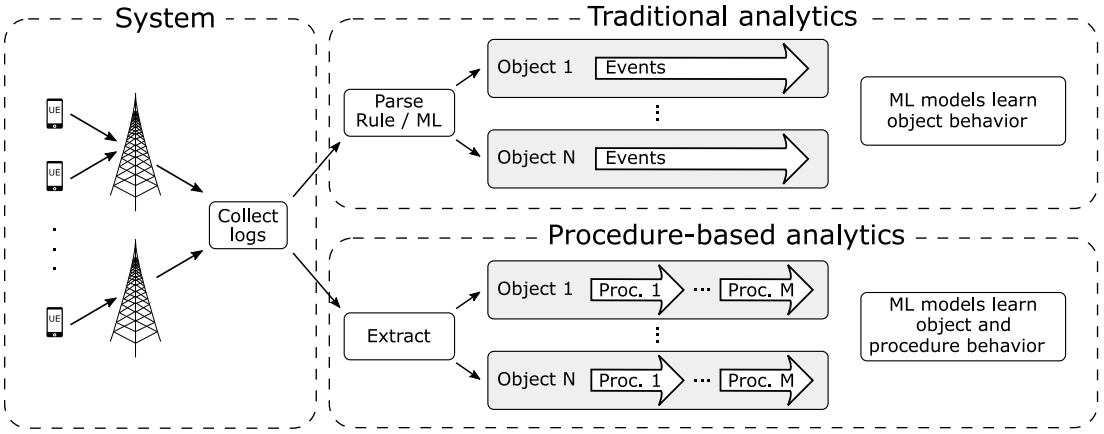

Fig. 9. The new trace guidelines enable automatic extraction of events for certain procedures.

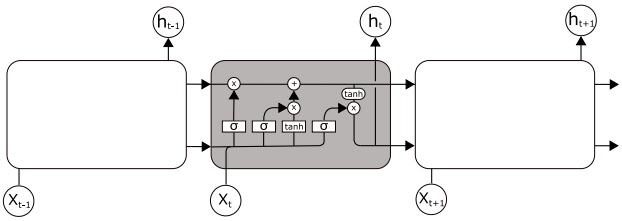

Fig. 10. LSTM architecture, X is the input, and h is the output for each prediction.

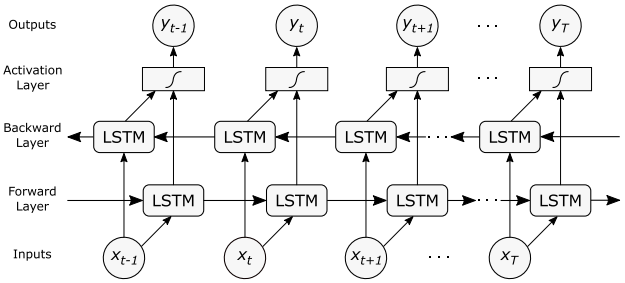

Fig. 11. Bi-LSTM Attention architecture, x is the input and y the output at a given time t.

Step 1: Check if the *Trigger-procedures* name is found in the *procedure-dictionary*. If not, then mark the procedure as an anomaly.
Step 2: If *Trigger-procedures* is found, compare the sub-procedure occurrences with the vectors stored in the *procedure-dictionary*.
Step 3: Mark the procedure as abnormal if no identical vector is found.
Step 4: If any anomalies are detected, use the Euclidean distance to find the most similar procedure vector and present the finding for the troubleshooter.

2) Bi-LSTM-Attention: As discussed, several state-of-the-art anomaly detection methods use recurrent neural network (RNN) variants in combination with an attention layer. We choose to implement one of the most common methods called LogRobust [10] and explain how the Bi-LSTM and attention are utilized to detect anomalies in the log.

The LSTM has internal memory and can remember inputs over a long period. Instead of having just one neural network in each module, like the RNN, the LSTM utilizes several interacting neural networks. In this way, the LSTM can choose the input, delete information that is not important and control the output of the data, see Fig. 10.

LSTM uses a sequence of previous events to predict the next event. If the whole sequence of events are $S = [s_1, s_2, \ldots, s_N]$, then the subsequences $S_i = [s_{m-1}, s_{m-2}, \ldots, s_i]$ are used as input, where $i \in [m, N]$, m is the subsequence length. For example, if a sequence of events is (1, 2, 3, 4, 5) and $m = 4$, then the subsequences are (1, 2, 3, 4) and (2, 3, 4, 5). Using an output equal to the number of unique events, the LSTM becomes a multi-classifier. The $m - 1$ serves as input for each subsequence, and the last value is one-hot encoded and utilized as the output label. Once trained, predictions with a probability greater than a probability threshold P_T are selected as output. If the real event is not among the predicted events, it is classified as an anomaly. P_T is also tuned to achieve the best F1-Score for each method.

To further enhance the accuracy of the LSTM model, the hidden neurons in the LSTM layer are split into two parts, where each part handles the forward and backward pass separately. In this way, it is possible to catch sufficient information from the log sequences from both directions. Such a model is often referred to as a Bi-LSTM model; see Fig. 11.

The activation layer then uses the flow of information through the backward and forward layers to improve the predicted output. A more detailed description of how the LSTM learns and remembers data can be read in [30].

The attention layer has recently been used in several variants to improve the anomaly detection accuracy of RNNs [10], [28], [38]. The embedding catches the relations between the events, and the attention layer allows the LSTM methods to gain insight from the whole sequence of events. The attention layer builds a correlation among the events in the LSTM output vectors ($[h_1, h_2, \ldots, h_T]$), where T is the sequence length. If H is a matrix of all the output vectors, the attention mechanism will produce an attention weight α and a hidden weighted representation r.

$$\alpha = softmax\left(w^T tanh(H)\right) \tag{1}$$

$$r = H\alpha^T \tag{2}$$

UE 1

Proc	Event	MS
1	A	MME
2	B	CU-CP
3	C	CU-UP
4	**D**	**CU-CP**
5	**E**	**CU-UP**
6	F	CU-CP
7	G	MME

UE 2

Time	Event	MS
1	A	MME
2	B	CU-CP
3	C	CU-UP
4	**E**	**CU-UP**
5	**D**	**CU-CP**
6	F	CU-CP
7	G	MME

Fig. 12. Comparison of part of a procedure for two user equipment (UE).

where $H \in \mathbb{R}^{d^w x T}$, d^w is the dimension of the event vectors, w and w^T is a trained parameter vector and its transpose. A softmax layer is applied to the attention output to create the final probability vector.

LogRobust uses a pre-trained dictionary to extract common English words from the events in the system log. The vectors of words are then transformed into a fixed-size vector representing each line of events. The logs fetched during the evaluation are more structured, and extracting the message information for each event is easy. We choose instead to transform each message to a unique number directly.

The number of anomalies the LSTM models find highly depends on the probability threshold P_T. The threshold needs to be tuned to find a suitable number of true and false positives. Each method is evaluated using the F1-score that considers false positives and true positive anomalies, but this might not always be the best choice in production environments. To further demonstrate the importance of keeping the false positives low, two Bi-LSTM-attention variants with different P_T are evaluated.

3) LSTM Microservice Split: When several MSs are involved with the same procedure, numerous parallel threads can occasionally change the order of events. Fig. 12 illustrates this variance.

The example is simplified to display time, event number, and MS name. The events originate from the same procedure but have a different order for events D-E. The reason for the mixed order is commonly threads that work in parallel where the response is received in a somewhat different order occasionally. Methods that learn the sequence order, such as the LSTM-based methods, need to learn both the order of *A-B-C-D-E* and *A-B-C-E-D*. In a system such as RAN, where many parallel threads handle thousands of components, learning the behavior from the individual components can be more advantageous than the combined behavior. To explore if anomaly detection can be improved using the microservice split, one LSTM model is trained for each microservice. A disadvantage of this approach is that the number of log events to learn from decreases. If there are too few events, it becomes impossible for the methods that learn from the sequence order to predict the preceding events.

4) BoostLog: In our previous work [17], we developed an AdaBoost ensemble of LSTM methods that could learn more complex behavior than individual LSTM methods and achieve a higher F1-score than previous state-of-the-art methods. AdaBoost has proven to detect many anomaly types in the advanced 5G testbed but requires more CPU time during training since several LSTM models are trained together. AdaBoost is included in the evaluation to compare the accuracy with the enhanced LSTM models that use Bi-LSTM and attention layers.

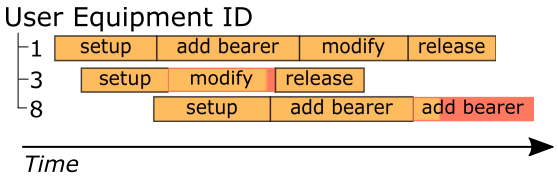

Fig. 13. Object view - Structure of objects affected by the procedures.

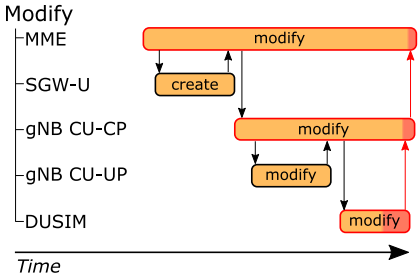

Fig. 14. Procedure view - Displays the sub-procedures within each procedure. It can also be enhanced to display the component interaction.

C. Observability

How well the internal states of a system can be understood from its external outputs is often referred to as observability. The external outputs, in this case, are metrics and logs. The metrics can provide insight into, e.g., memory usage, CPU load, or Key Performance Indicators (KPI), which measure special features of the system. They can be used to measure the system's performance but seldom provide an answer to why the system behaves in a certain way. Logs are instead the first place to look when something fails. One problem highlighted in Section III-B is that the events may differ significantly depending on the developer and component. Another problem is that developers often instrument the software from a component perspective and forget the larger scope where objects and procedures are important.

The new trace guidelines allow the separation of objects and procedures and make it possible to visualize how system parts work together. The anomalies are also presented using the steps in Section IV-B1 to enhance the observability further. The visualization is made from a top-down perspective where procedures affect components in the system. In this view, it is also possible to see in which procedure an anomaly occurs; see Fig. 13.

The example is from a 5G testbed, where the procedures and a list of user equipment's (UE) are visible in a timeline view. The red indicates where an anomaly occurs in the procedure, and troubleshooters can easily navigate deeper to see how the components are affected, see Fig. 14.

The procedure view presents the signaling between the components and how the sub-procedures contribute to the procedure. When anomalies occur, it is easy to identify failing sub-procedures and compare them with normal ones; see Fig. 15.

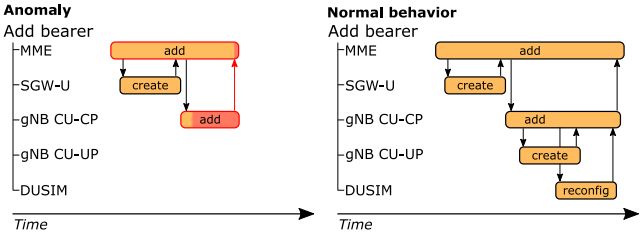

Fig. 15. Procedure view - Compare abnormal and normal procedures.

DUSIM - modify

Anomaly	Normal behavior	
Event A	Event A	Event A
Event B	Event B	Event B
Event C	Event C	Event D
Event F	Event D	Event C
Event G	Event E	Event E

Fig. 16. Sub-procedure view - Compares sequences of events for normal and abnormal behavior.

For each sub-procedure, it is possible to go deeper and compare logs regarding normal and abnormal behavior; see Fig. 16.

The example demonstrates how the sub-procedures can be compared to detect differences on the event level.

D. Evaluation Metrics

Each detected anomaly needs to be analyzed by developers, and it is crucial to keep the number of false positives (FP) low while identifying the true positives (TP). To consider both TP and FP, the methods are tuned to maximize the F1-score. Although, the precision, recall, FP, and TP values are also presented in the results. All methods analyze each UE sequence and count the number of abnormal sequences. The time to train each method is also evaluated.

An important aspect for RAN troubleshooters is understanding the system's behavior and quickly identifying when and where the anomalies occur. To measure the observability of the 5G testbed, five questions are defined:

- *Failing procedure* - What procedure fails during the test case? This shows that the troubleshooter understands what kind of scenario occurs when an anomaly occurs.
- *Historical procedures* - Previous procedures might change the object's state under test. Understanding past procedures may aid the troubleshooter in the root cause analysis.
- *Failing microservice* - When several MSs work together to complete a procedure, which MSs are part of the procedure, and which is most likely causing the anomaly?
- *The normal event* - What should have occurred instead of the anomaly? When the normal scenario is understood, the abnormal behavior can be compared to the normal.
- *Anomaly type* - How fast can the anomaly type be identified? Understanding the difference between normal and abnormal behavior can be crucial to fully understanding the problem.

As troubleshooters search for anomalies, they should try to answer each one of the questions as fast as possible. The time needed to answer the questions is then used to calculate an *observability-score* that ranges between zero and ten points. Ten points are given if the question is answered within one minute, nine points within two min, and so on, down to zero points when more than ten minutes are needed. Since the experience of the troubleshooter also determines how well the system can be understood, a mix of experienced and novice troubleshooters is part of the evaluation.

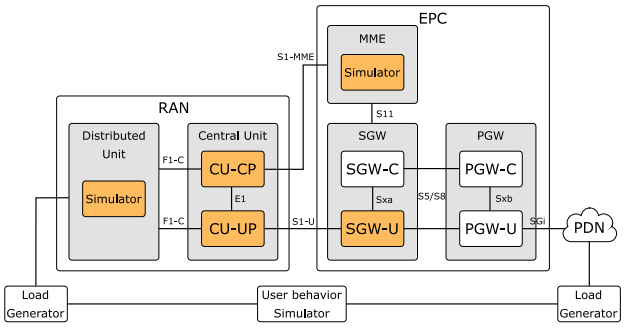

Fig. 17. Architecture overview of 5G testbed.

V. Evaluation

A. Testbed Setup and Data Acquisition

To evaluate if the procedure-based guidelines can improve anomaly detection and observability; the existing software instrumentation of an advanced 5G testbed was updated to follow the procedure-based guidelines. The testbed is developed by TietoEVRY[3] and is an extensive distributed system that tests hardware performance and various cloud concepts for the future 5G RAN. A simulated city generates traffic when residents make phone calls and move around the city. A user equipment and core network simulator use the simulated mobile usage to generate traffic for the base stations; see Fig. 17.

The scope of this work is limited to five main procedures and includes analysis of procedural information from the simulators, CU-CP, CU-UP, and SGW-U. All procedures affect a UE, and the identity of the UE is included in the new debug information. The debug-ability of the 5G testbed was increased from level 4 to 5 by updating 199 software instrumentations. The update and verification of the instrumentation took 40 hours. A highly intensive traffic test case ran in five minutes to acquire system logs with the updated instrumentation. The log contained an average of 863,153 events from 5,750 unique UEs, with 150 events per UE, see Table I. All sequences started with the **Setup** procedure and ended either with **Release** or **Inactivity**. **Add E-RAB** and **Modify** could occur any number of times in between. Fig. 18 presents an example of the procedure mapping. The data is publicly available at github.[4] The extra procedure information contained an average of 30 characters and increased the total log size by 13%. An extra evaluation was conducted to measure if the increased log size affected the memory or CPU usage, but no

[3]https://www.tietoevry.com
[4]https://github.com/Tobbis/qvistigt

TABLE I
PROCEDURES IN TEST SUITE

Name	Count
Setup	5750
Release	2945
Add E-RAB	2815
Modify E-RAB	2480
Inactivity	2532

```
[08:32:55.270099799] com_tieto_tnp_lte_ue:procedure: { packet_seq_num = 0, cpu_id = 0 },
        { vpid = 28825, vtid = 28825, process = "cnsim" },
        { cell_id = 0xFF, ue_id = 0x1, proc_id = 0x0, proc_name = "Context_setup", new_state = "InitialUeMessage" }
[08:32:55.270100962] com_tieto_tnp_lte_ue:map_procedure: { packet_seq_num = 0, cpu_id = 0 },
        { vpid = 28825, vtid = 28825, procname = "cnsim" },
        { id1 = 0x1, id2 = 0x1, id1_name = "CU_CP_ue_id", id2_name = "MME_ue_id" }
```

Fig. 18. Example of procedure and mapping event in the system log using the new trace guidelines.

statistical difference was found. Neither could we detect any deviation in the number of procedures handled due to the extra procedure information.

Based on our experience with the testbed and the vendor's real RAN, we have seen that failing scenarios can be detected in the system logs by missing, extra, or changes in the order of events. Delays or hangs of procedure can also be detected using the timestamp of each event [27], but the time aspect is not included in this work. Instead of adding new anomalies that might generate these changes in the system logs, the anomalies are introduced directly into the logs. The advantage of this approach is that the abnormal behavior can be controlled in greater detail, and many anomalies can be tested. The changes are randomly introduced among the UE's and cover most anomalies previously seen in similar systems. Eleven types of anomalies, defined in Table III, are introduced in the log 100 times each. We selected 70% of the data for training and introduced anomalies in the remaining data. In total, there were 4,025 UE sequences in training data, where 3,076 had a unique sequence of events. The test data contained 1,725 normal and 1,100 abnormal UE sequences.

B. Experimental Setup

The hardware specifications of the 5G testbed cannot be revealed due to trade secrets, but LTTng version 2.12 was used to instrument the system. All methods used Ubuntu 18.04.5 OS with one Intel(R) Xeon(R) CPU @ 2.20GHz processor and 13GB of memory when training and detecting anomalies. A 5-folded cross-validation ensures the stability of the results in all experiments.

C. Anomaly Detection

The three baseline LSTM models were evaluated together with the new procedure-based model. The Bi-LSTM-Attention method was trained for 30 epochs, using a minimum batch size of 200, and $P_T = 1e-3$. The two LSTM layers used 128 neurons, and the input layer used a value of $m = 5$. The MS split LSTM method had the same configuration and trained one LSTM for each of the five microservices. The AdaBoost model used 10 LSTM classifiers ranging from 2-3 hidden layers, 32-200 neurons, and $m = 5$. There are no hyper-parameters to tune for the new procedure-based method that can learn the behavior quickly with scanty training data. To investigate how

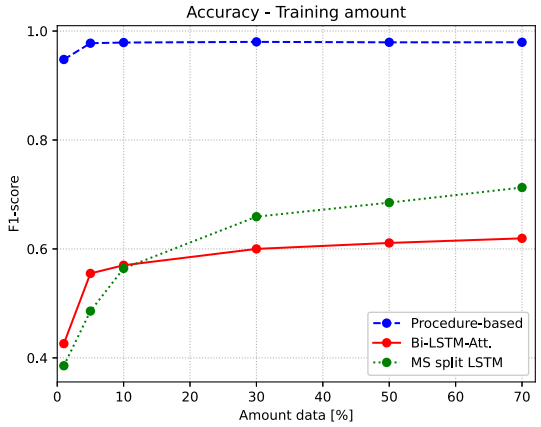

Fig. 19. Accuracy for an increasing amount of training data.

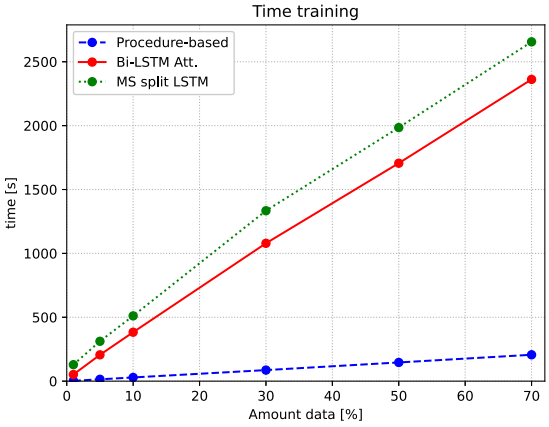

Fig. 20. Training time needed for an increasing amount of training data.

the amount of training data affects accuracy, the F1-score was compared as the amount of data was increased; see Fig. 19.

While the procedure-based method reaches its maximum at five percent data usage, the baseline methods continue to improve up to 70% of data usage. The time needed to train on historical data is also an essential factor, and the training time increased linearly for all methods; see Fig. 20. The training time of the Adaboost method was too large to include in this evaluation, and this is also why it is not seen in Fig. 19, Fig. 20, and Fig. 21.

A detailed view of the accuracy, time spent in training, and anomalies found are presented in Table II, where 70% of data were used for training.

The baseline methods have high accuracy when predicting single events, and the validation accuracy was greater than 0.9 for all three baseline methods at the end of training. This accuracy concerns the prediction of single events, but the accuracy will be much lower when predicting a whole sequence of events, which on average, includes 150 events. The baseline method with the highest F1-score found more than half of the anomalies and many FP cases. Keeping the number of FP cases low is essential for troubleshooters who don't want to waste time analyzing normal sequences. As the FP cases are numerous, another version of the Bi-LSTM-attention method with $P_T = 1e{-}8$ is evaluated. The AdaBoost and MS split methods used the same low threshold. Notable is that accuracy increases

TABLE II
EVALUATION DATA FOR EACH METHOD

Method	Precision	Recall	F1-score	TP	FP	training time [s]
Bi-LSTM-attention low thres.	0.979	0.453	0.619	498	11	2360
Bi-LSTM-attention	0.819	0.727	0.771	800	176	2360
AdaBoost LSTM	0.959	0.494	0.652	543	23	6300
MS split LSTM	0.998	0.554	0.712	610	1	2650
Procedure-based	**0.998**	**0.960**	**0.979**	**1057**	**1**	**210**

TABLE III
ANOMALY TYPES INTRODUCED AND FOUND FOR EACH METHOD

Name	Anomalies	Bi-LSTM	Bi-LSTM low thres.	MS split LSTM	AdaBoost	Procedure-based
One event removed	100	65	15	39	11	**93**
End of sequence removed	100	72	28	0	36	**92**
Two events removed	100	74	20	64	17	**94**
Duplicate event	100	47	14	51	15	**89**
Add a known extra event	100	88	64	82	76	**89**
Add one new unknown event	100	100	100	99	100	**100**
Replace event with unknown event	100	100	100	100	100	**100**
Replace event with known event	100	94	64	90	80	**100**
Sub-procedure missing	100	55	37	4	34	**100**
Duplicate sub-procedure	100	90	56	81	72	**100**
New sub-procedure	100	15	0	0	2	**100**
Total number of anomalies	1100	800	498	610	543	**1057**

when one Bi-LSTM-attention method is used for each MS. The great advantage of the procedure-based approach is also apparent; almost all anomalies are found and only one FP case was reported.

There is also a considerable improvement in training time, where the deep learning approaches need at least 2,360 seconds of CPU time, and the procedure-based only needs 210 seconds. Table III presents a more detailed analysis of the anomalies that each method could detect.

The effect of the MS-split LSTM is more apparent in this table, where more anomalies are detected for most anomaly types. Still, a drawback is that it cannot identify some anomalies where a larger part of the events are missing. Another disadvantage is that the MS-split LSTM method cannot detect anomalies with insufficient input events. The LSTM methods can also more easily detect additional events than missing events. This is part of the LSTM nature, where events occurring close after the predicted event have a high probability. The procedure-based method can easily detect changes in sub-procedure grouping and new events. Still, some of the first five anomaly types do not seem to be detected and required further analysis. It was then discovered that all of these anomalies were introduced such that the resulting sequence was the same as the normal scenario seen during training. The procedure-based method was very robust during our cross-validation, and the ROC-AUC curve for all results can be seen in Fig. 21.

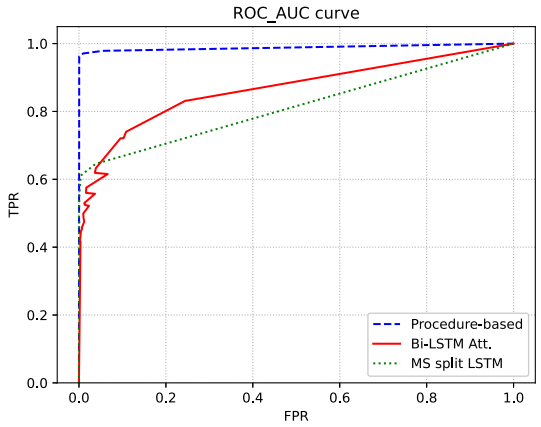

Fig. 21. ROC-AUC curve for procedure-based, Bi-LSTM-attention, and MS split LSTM methods.

D. Observability

In the evaluation of the *observability-score* there are two aspects of the methods that is more important than others; How the anomalies and behavior is presented to the end-users, and the number of FP/TP cases each method find. Each FP case requires extra time in the analysis and without a TP case it is difficult to identify the deviating behavior. Since all baseline methods present the behavior and anomalies in the same way, only the BI-LSTM-attention with low threshold were selected due to its low number of FP cases. To make a fair evaluation between the base line method and the procedure-based method, the analysis were performed for sequence where both methods found a TP case. A test group of 4 developers with mixed experience was selected to answer the questions stated in Section IV-D. A random anomaly was selected from each anomaly type and all developers answered the five observability questions from Section IV-D as quickly as possible. The troubleshooter was told to answer them in the order stated in Section IV-D. Each troubleshooter took the test twice with different anomalies, and the mean value with one standard deviation is given for each score and method in Table IV.

All troubleshooters improved the *observability-score* using the procedure-based method. The ability to visualize and compare procedures greatly increased the system's understanding. Without the visual support in the Bi-LSTM-attention method, the troubleshooters manually searched for similar

TABLE IV
OBSERVABILITY-SCORE FOR EACH METHOD (EXPERIENCED TROUBLESHOOTER)

Method	Hist. proc.	Failing proc.	Failing MS	Normal event	Anomaly type
Bi-LSTM-attention low thres.	9.5 ±0.7	9.2±0.9	8.5 ±1.1	6.5 ±2.9	4.3 ±3.5
Procedure-based	10.0 ±0	10.0 ±0	10.0 ±0	9.6 ±0.4	7.7 ±3.3

patterns in other UE's and compared them event by event to distinguish normal from abnormal sequences. Several troubleshooters also had to conduct the documentation and code to identify what kind of procedure occurred in the system log. The results also show that the procedure-based method can help provide a fast understanding of the system, where the three first questions could be answered within three minutes.

VI. DISCUSSION AND LIMITATIONS

This work simulates anomalies we have seen during our 20 years in the telecom industry. The anomalies introduced were nevertheless not telecom-specific, as they represent an abnormal sequence of events in the system log. We have already observed how our trace guidelines and procedure-based method are able to improve the anomaly detection in Cloud RAN production environment [39]. We believe it can improve anomaly detection and observability in many other complex systems as well.

The real applications come with both simpler or more complex combinations of anomalous events in the log compared with the ones that are simulated in this work. Despite, we feel confident that the procedure-based method will achieve a high accuracy in production environment by observations from its high performance in the various simulated anomalies and the integrated Cloud RAN solution.

We chose to compare our new procedure-based method with recent state-of-the-art LSTM methods in this work. Other deep learning methods may have surpassed these baseline methods in accuracy or CPU time to some extent, but it would most likely not change our conclusions. We sought to investigate if instrumenting the application with a larger, systemized mindset could improve observability and ML-based anomaly detection. The new trace guidelines may guide developers in the right direction and aid other researchers in developing methods that can further enhance anomaly detection and observability.

A new *observability-score* was defined to measure how well a troubleshooter understood the ongoing procedures and whether it is possible to differentiate between normal and abnormal behavior. There could also have been system-specific questions about the testbed, but we limited our investigation to more general questions relevant to other large distributed systems. The study could have included more troubleshooters to gain greater statistical certainty in the *observability-score*. Still, the results are satisfying, and the percentage of novice developers would have increased if a larger group of developers had conducted the test. It is worth mentioning that developers thought both methods enhanced the system's observability. Most systems do not have ML-based behavior analysis, and without LSTM or procedure-based support, the developers have to search through the logs to detect the anomalies manually.

The methods in this work use supervised learning to learn the behavior from sequences without anomalies. The methods could also be unsupervised if anomalies are kept low. The procedure-based method would, in this case, need a new hyper-parameter to tune the probability that a procedure or sub-procedure appears and then filter out those that occur more infrequently. Normal sequences occurring seldom may increase the FP rate. Still, we believe that it would be beneficial to use the procedure-based approach as the number of unique sequences can be kept low when sequences are separated using procedures, objects, and components.

Evaluating some of the LSTM models required hours of training and testing for each experiment, and we chose only to verify our results using 5-folded cross-validation. The advantages of procedural learning in RAN are still apparent, and no further validation would have changed that fact.

The proposed trace guidelines could easily be implemented to improve the 5G testbed, but there may be large distributed systems where the procedure cannot be determined for some events. Such cases could be a sign of bad design or architecture, and perhaps such systems would benefit from a redesign and new trace guidelines.

The extra procedure information stored in the system log increases the log size and can be a problem for systems that already experience problems with log storage. The evaluation in this work uses LTTng to store data, which is known to be a highly effective tracing framework. Still, there can be issues with the storage of data if a large number of procedures are handled at the same time. We recommend that those inspired by our work first evaluate if the system can handle the extra amount of data added by the procedural information.

Our new procedural-based method has been integrated and evaluated in existing pipelines for 5G cloud RAN, and the results are promising. The anomaly detection accuracy has increased, and fewer false positives anomalies are detected. Still, there is a need to improve the visualization support to help end users understand where anomalies occur in the procedures.

Our new procedure-based method has together with the improved trace guideline, laid the foundation for a newly published master's thesis at Umeå University[5] [39]. The thesis concludes that the procedure-based method is working well in Cloud RAN continuous integration pipelines and can improve the anomaly detection accuracy. This strengthen our conviction that we need to structure the logs in more generic ways to help both the end users and the machine learning methods. Future collaborations with industry partners will explore if there are any foreseen disadvantages with this new procedure-based method.

[5]https://www.umu.se/en/

The guidelines stated in this work were used to add a procedure context for a large distributed 5G testbed. Based on our experience designing other large systems, we believe our guidelines can improve observability and anomaly detection for many other systems. It is, though, important that the procedural information is added in cooperation with system experts and developers that know how the procedures affect the system. Once the instrumentation is carried out, the procedure-based methods can automatically learn the behavior of the systemized procedures. This also helps developers and troubleshooters learn the behavior and map it to the system's documentation.

VII. Conclusion and Future Work

This work presents how a procedure-based trace guideline can improve ML-based anomaly detection and observability in an advanced distributed 5G testbed. The extra procedure context enabled the new procedural-based method to learn the behavior of smaller pieces of the system and then combine them to realize better how all pieces work together. The results were astonishing: where previous state-of-the-art deep learning methods needed hours of training to detect some of the anomalies, the procedure-based method could, within minutes, detect all of the anomalies. The procedural information also enhances the system's observability, where all troubleshooters better understood the behavior of the large distributed 5G testbed and identified the anomalies faster.

We believe this work can provide some insight into how instrumentation should be carried out to allow machine learning methods to learn the behavior of large systems and improve observability more easily. In future research, we will investigate whether these guidelines can also aid in detecting the root cause of anomalies in large distributed systems.

Acknowledgment

Thanks to TietoEvry for funding part of this research.

References

[1] Z. Li, T.-H. Chen, and W. Shang, "Where shall we log? Studying and suggesting logging locations in code blocks," in *Proc. 35th IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2020, pp. 361–372.

[2] J. Zhu et al., "Tools and benchmarks for automated log parsing," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng. Softw. Eng. Pract.*, 2019, pp. 121–130.

[3] J. Soldani and A. Brogi, "Anomaly detection and failure root cause analysis in (MICRO) service-based cloud applications: A survey," *ACM Comput. Surveys*, vol. 55, no. 3, pp. 1–39, 2022.

[4] B. Li et al., "Enjoy your observability: An industrial survey of microservice tracing and analysis," *Empirical Softw. Eng.*, vol. 27, no. 1, pp. 1–28, 2022.

[5] N. Bosch and J. Bosch, "Software logging for machine learning," 2020, *arXiv:2001.10794*.

[6] C. Cassé, P. Berthou, P. Owezarski, and S. Josset, "A tracing based model to identify bottlenecks in physically distributed applications," in *Proc. Int. Conf. Inf. Netw.*, 2022, pp. 226–231.

[7] C. Cassé, P. Berthou, P. Owezarski, and S. Josset, "Using distributed tracing to identify inefficient resources composition in cloud applications," in *Proc. IEEE 10th Int. Conf. Cloud Netw.*, 2021, pp. 40–47.

[8] X. Yu, P. Joshi, J. Xu, G. Jin, H. Zhang, and G. Jiang, "CloudSEER: Workflow monitoring of cloud infrastructures via interleaved logs," *ACM SIGARCH Comput. Architect. News*, vol. 44, no. 2, pp. 489–502, 2016.

[9] Z. S. Ageed et al., "Comprehensive survey of big data mining approaches in cloud systems," *Qubahan Acad. J.*, vol. 1, no. 2, pp. 29–38, 2021.

[10] X. Zhang et al., "Robust log-based anomaly detection on unstable log data," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2019, pp. 807–817.

[11] A. Bento, J. Correia, R. Filipe, F. Araujo, and J. Cardoso, "Automated analysis of distributed tracing: Challenges and research directions," *J. Grid Comput.*, vol. 19, no. 1, pp. 1–15, 2021.

[12] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, "A survey on automated log analysis for reliability engineering," *ACM Comput. Surveys*, vol. 54, no. 6, pp. 1–37, 2021.

[13] M. Landauer, S. Onder, F. Skopik, and M. Wurzenberger, "Deep learning for anomaly detection in log data: A survey," 2022, *arXiv:2207.03820*.

[14] R. B. Yadav, P. S. Kumar, and S. V. Dhavale, "A survey on log anomaly detection using deep learning," in *Proc. 8th Int. Conf. Rel. Infocom Technol. Optim. (ICRITO)*, 2020, pp. 1215–1220.

[15] X. Zhao, Z. Jiang, and J. Ma, "A survey of deep anomaly detection for system logs," in *Proc. IEEE Int. Joint Conf. Neural Netw. (IJCNN)*, 2022, pp. 1–8.

[16] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in *Proc. ACM Special Interest Group Security Audit Control Conf. Comput. Commun. Security*, 2017, pp. 1285–1298.

[17] T. Sundqvist, M. Bhuyan, J. Forsman, and E. Elmroth, "Boosted ensemble learning for anomaly detection in 5G RAN," in *Proc. IFIP Int. Conf. Artif. Intell. Appl. Innov.*, 2020, pp. 15–30.

[18] W. Gan, J. C.-W. Lin, H.-C. Chao, and J. Zhan, "Data mining in distributed environment: A survey," *Wiley Interdiscipl. Rev. Data Min. Knowl. Disc.*, vol. 7, no. 6, 2017, Art. no. e1216.

[19] R. Fonseca, G. Porter, R. H. Katz, and S. Shenker, "X-Trace: A pervasive network tracing framework," in *Proc. 4th USENIX Symp. Netw. Syst. Design Implement.*, 2007, p. 9.

[20] Y. Gan et al., "SEER: Leveraging big data to navigate the complexity of performance debugging in cloud microservices," in *Proc. 24th Int. Conf. Architect. Support Program. Lang. Oper. Syst.*, 2019, pp. 19–33.

[21] S. Lima, J. Correia, F. Araujo, and J. Cardoso, "Improving observability in event sourcing systems," *J. Syst. Softw.*, vol. 181, Nov. 2021, Art. no. 111015.

[22] A. Janes, X. Li, and V. Lenarduzzi, "Open tracing tools: Overview and critical comparison," 2022, *arXiv:2207.06875*.

[23] J. Halvorsen, J. Waite, and A. Hahn, "Evaluating the observability of network security monitoring strategies with TOMATO," *IEEE Access*, vol. 7, pp. 108304–108315, 2019.

[24] A. Tagliabue et al., "LION: LiDAR-inertial observability-aware navigator for vision-denied environments," in *Proc. Int. Symp. Exp. Robot.*, 2020, pp. 380–390.

[25] R. Drouilly, P. Rives, and B. Morisset, "Semantic representation for navigation in large-scale environments," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, pp. 1106–1111.

[26] S. Kabinna, C.-P. Bezemer, W. Shang, M. D. Syer, and A. E. Hassan, "Examining the stability of logging statements," *Empirical Softw. Eng.*, vol. 23, no. 1, pp. 290–333, 2018.

[27] T. Sundqvist, M. Bhuyan, and E. Elmroth, "Uncovering latency anomalies in 5G RAN—A combination learner approach," in *Proc. 14th Int. Conf. Commun. Syst. Netw.*, 2022, pp. 621–629.

[28] X. Li, P. Chen, L. Jing, Z. He, and G. Yu, "SwissLog: Robust anomaly detection and localization for interleaved unstructured logs," *IEEE Trans. Depend. Secure Comput.*, vol. 20, no. 4, pp. 2762–2780, Jul./Aug. 2023.

[29] A. Pi, W. Chen, S. Wang, and X. Zhou, "Semantic-aware workflow construction and analysis for distributed data analytics systems," in *Proc. 28th Int. Symp. High Perform. Parallel Distrib. Comput.*, 2019, pp. 255–266.

[30] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[31] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder–decoder approaches," 2014, *arXiv:1409.1259*.

[32] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

[33] R. Yang, D. Qu, Y. Gao, Y. Qian, and Y. Tang, "NLSALog: An anomaly detection framework for log sequence in security management," *IEEE Access*, vol. 7, pp. 181152–181164, 2019.

[34] S. Gu, Y. Chu, W. Zhang, P. Liu, Q. Yin, and Q. Li, "Research on system log anomaly detection combining two-way slice GRU and GA-attention mechanism," in *Proc. 4th Int. Conf. Artif. Intell. Big Data*, 2021, pp. 577–583.
[35] L. Yang et al., "Semi-supervised log-based anomaly detection via probabilistic label estimation," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng.*, 2021, pp. 1448–1460.
[36] S. Huang et al., "Hitanomaly: Hierarchical transformers for anomaly detection in system log," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 4, pp. 2064–2076, Dec. 2020.
[37] C. Zhang, X. Wang, H. Zhang, H. Zhang, and P. Han, "Log sequence anomaly detection based on local information extraction and globally sparse transformer model," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 4, pp. 4119–4133, Dec. 2021.
[38] D. Yu, X. Hou, C. Li, Q. Lv, Y. Wang, and N. Li, "Anomaly detection in unstructured logs using attention-based bi-LSTM network," in *Proc. 7th IEEE Int. Conf. Netw. Intell. Digit. Content*, 2021, pp. 403–407.
[39] C. Elfving, "Temporal & sequential learner for system log anomaly detection: Development and evaluation of a new machine learning model for 5G RAN system log-based anomaly detection," 2023.

Tobias Sundqvist (Member, IEEE) received the Ph.D. degree in computer science and engineering from Umeå University, Sweden. He has been developing the radio access network for the last 20 years and his specialty is anomaly detection and observability in large distributed systems.

Monowar Bhuyan (Member, IEEE) received the Ph.D. degree in computer science and engineering from Tezpur University, India, in 2014. He has been an Assistant Professor with the Department of Computing Science, Umeå University, Sweden, since January 2020, and one of the research group leaders at Autonomous Distributed Systems Lab. Before this, he worked with the Nara Institute of Science and Technology, Japan; Umeå University; Assam Kaziranga University, India; and Tezpur University from January 2009 to December 2019. He has published over 70 papers in leading international journals and conference proceedings and has written a book with Springer. His experience leading/co-leading research projects attracted over 25 MSEK national, international, and European Union grants. His primary research areas include machine learning, anomaly detection, systems security, and distributed systems. He is also a member of the ACM.

Erik Elmroth is a Professor of Computing Science with Umeå University. He has been the Head and the Deputy Head of the Department of Computing Science for 13 years and the deputy director for a national supercomputer center for another 13 years. He has established Umeå University research on distributed systems, addressing autonomous management systems for virtual computing infrastructures, such as clouds and edge environments, see http://www.cloudresearch.org. His experience with management and executive groups in large-scale research projects includes highlights, such as the 550 m euro Wallenberg AI, autonomous systems and software program, and the strategic research area eSSENCE. He has developed two research strategies for the Nordic Council of Ministers. International experiences include a year at NERSC, Lawrence Berkeley National Laboratory, University of California at Berkeley, Berkeley, and one semester at the Massachusetts Institute of Technology, Cambridge, MA, USA. He has also been a member of the Swedish Research Councils Committee for research infrastructure and the Chair of its expert panel on eScience as well as the Chair of the Board of the Swedish National Infrastructure for Computing. He is a lifetime member of the Swedish Royal Academy of Engineering Sciences and the Vice Chair of its division for Information Technology.