Software and Systems Modeling
https://doi.org/10.1007/s10270-023-01087-4

REGULAR PAPER

Correlating contexts and NFR conflicts from event logs

Mandira Roy[1] · Souvick Das[3] · Novarun Deb[2] · Agostino Cortesi[3] · Rituparna Chaki[1] · Nabendu Chaki[1]

Received: 19 March 2022 / Revised: 16 January 2023 / Accepted: 17 January 2023

Abstract
In the design of autonomous systems, it is important to consider the preferences of the interested parties to improve the user experience. These preferences are often associated with the contexts in which each system is likely to operate. The operational behavior of a system must also meet various non-functional requirements (NFRs), which can present different levels of conflict depending on the operational context. This work aims to model correlations between the individual contexts and the consequent conflicts between NFRs. The proposed approach is based on analyzing the system event logs, tracing them back to the leaf elements at the specification level and providing a contextual explanation of the system's behavior. The traced contexts and NFR conflicts are then mined to produce Context-Context and Context-NFR conflict sequential rules. The proposed Contextual Explainability (ConE) framework uses BERT-based pre-trained language models and sequential rule mining libraries for deriving the above correlations. Extensive evaluations are performed to compare the existing state-of-the-art approaches. The best-fit solutions are chosen to integrate within the ConE framework. Based on experiments, an accuracy of 80%, a precision of 90%, a recall of 97%, and an F1-score of 88% are recorded for the ConE framework on the sequential rules that were mined.

Keywords Sequential rule mining · Context correlation · Context-NFR conflict correlation · Goal models

1 Introduction

In modern-day autonomous systems, multiple stakeholders are involved in the elicitation of requirements (both functional and non-functional), as each of them is concerned with specific performance issues. There are negotiation frameworks to facilitate a common platform for the stakeholders (like Theory W [1]), so that the system can be delivered with the desired feature set. This requires a couple of important issues to be addressed, (i) the need to understand the correlation between stakeholder preferences and the contexts associated with the requirements; (ii) to correlate the possible NFR conflicts [2–4] that can arise in different contexts (identified previously). In the recent works [5,6], it is shown how NFRs and contexts play a crucial role in the operational behavior of a system. There is a strong need to correlate the two for improving the overall experience of the end-user. For example, let us consider the Uber online cab booking application. We can identify two NFRs—*Fast Response Time* and *Feature Availability*, which can be associated with such an application. The NFR *fast response time* aims to reduce the waiting time of service delivery for the end-user, and the NFR *feature availability* talks about the number of features (or services) that are being delivered to the end-user. These two NFRs might be in conflict in case of a particular environmental context. For example, it may not be possible to deliver all resource-intensive services while achieving fast response times with limited available bandwidth (say, due to heavy

Communicated by Iris Reinhartz-Berger.

✉ Mandira Roy
mrcomp_rs@caluniv.ac.in

Souvick Das
souvick.das@unive.it

Novarun Deb
novarun_deb@iiitvadodara.ac.in

Agostino Cortesi
cortesi@unive.it

Rituparna Chaki
rchaki@ieee.org

Nabendu Chaki
nabendu@ieee.org

[1] University of Calcutta, Kolkata, India

[2] Indian Institute of Information Technology, Vadodara, India

[3] Ca' Foscari University, Venice, Italy

Published online: 25 February 2023

rain). In such a situation, solution architects may choose to compromise on *feature availability* and deliver a limited set of essential services while achieving *fast response times* for the same. Establishing correlations between NFR conflicts and contexts allow system engineers to design the application servers in a manner where multiple different apps for the same service provider (for example, Uber and Uber Lite) need not be maintained. Based on the environmental context, the server offers a different quality of service and, perhaps, only a subset of the functionalities to the end-user. This gives the end-user a seamless experience of service consumption under different environmental contexts.

The objective of this research is to design a novel framework for identifying correlations between contexts and NFR conflicts within a system by processing its event logs. The motivation of building this framework is to predict the NFR conflicts that might occur in the future when certain environmental contexts are activated. The proposed *contextual explainability* framework establishes sequential rules between—(i) individual contexts and (ii) corresponding NFR conflicts—that happened within the systems' operational environment. We develop a transformer-based language model that maps the events recorded in the system's event-logs with the system's requirement goal model. The event log only records the activities performed and not the NFRs or contexts. This mapping helps to infer, which contextual parameters were activated during system operations *vis-à-vis* the NFRs which were observed to be in conflict. We assume that the functional requirements are explicitly annotated with context information and the variation points [7,8] are identified in the requirements goal model.

Our research methodology can be summarized as follows. Based on the literature review, we first design a framework for Contextual Explainability. Its architecture is depicted in Fig. 1. This framework maximizes the reuse of existing pre-trained language models and sequential rule-mining libraries. In the next step, we experimentally compared the results for the alternative solutions with respect to accuracy, precision, recall etc. This is done by adopting different BERT-based transformers and different sequential rule mining algorithms.

The proposed Contextual Explainability framework consists of the following sequence of steps:

- Initially, we inspect the system event log which lists all the activities performed by different resources (humans, servers, databases, etc.) while delivering the services provided by the system.
- Next, we deploy a BERT-based Transformer Model (BTM), pre-trained over existing event logs. This BTM identifies the leaf goals operationalized for a particular execution thread using semantic similarity. Based on the leaf goals identified, we observe the contexts that are activated at the variation points and the NFR conflicts that occur.
- The above process is repeated to identify the contexts and NFR conflicts associated with all the execution threads in the event log.
- These data are then fed into the Sequential Rule Mining Framework of the SPMF library [9] for mining two types of sequential rules—(i) between individual contexts; and (ii) between contexts and NFR conflicts.

It is interesting to note that our approach involves three different research domains—namely, goal-oriented requirements engineering, sequential rule mining, and process re-engineering. This work provides scope for academics from these three domains to explore the possibility of combining their knowledge and expertise to deliver end-to-end solutions for real-world problems.

The main advantages of adopting our Contextual Explainability (ConE) model within the software engineering specification activity, either in an evolutionary or in a DevOps lifecycle, are twofold:

1. ConE provides insights into the possible set of contexts that could be activated simultaneously in the operations environment.
2. ConE provides insights into the non-functional requirements that may not be satisfied when certain contexts are activated in the operational environment.

The structure of the paper is as follows. Section 2 discusses existing related work in the literature. Section 3 provides the background. Section 4 details our *ConE* framework. Section 5 describes the experimental environment. Section 6 discusses the experimental results. Section 7 highlights the different threats to validity of this research. Section 8 provides a comparative evaluation of our approach with existing works in the literature, and Sect. 9 concludes.

2 Related works

The number of empirical research on explanations and explainable systems has been increasing in a variety of significant themes. Incorporating explanations can assist users in understanding why a system has given specific outcomes. This in turn reduces opacity and makes decision-making more visible. Explanations are often considered a way to overcome a system's lack of transparency [10]. Selvaraju et al. [11] and Tintarev et al. [12] have investigated the impact of explanations on quality aspects such as acceptability, trust and effectiveness, while Kulesza et al. [13] explored how aspects of explanations impact on the understandability of a system. Chazette et al. [14] have focused on the

relationship between usability and explainability as usability significantly influences software quality. They proposed a set of lightweight user-centered activities to support the design of explanations that can be integrated into the requirements engineering phase. Chazette et al. in [15] contribute to the development of standard catalogs and semantics, allowing the discussion and analysis of explainability during the RE process. The authors proposed a model to analyze the impacts of explainability across different quality dimensions. Sadeghi et al. [16] present a taxonomy of explanation needs that classify scenarios that require explanations. The taxonomy can be used to guide the requirements elicitation for explanation capabilities of interactive intelligent systems. For each leaf node in the taxonomy, the software system should produce an explanation. The explanation of different cases helps to analyze the system's interaction. Beaudounin et al. [17] combine three steps to define the right level of explainability in a given context. The first step defines the contextual factors, the second step examines the technical tools available, and the third one is a function of the first two steps and chooses the right levels of global and local explanation.

Recently, concepts such as interpretable machine learning and explainable artificial intelligence have evolved, and factors that enhance the intelligibility of machine learning algorithms have become a trending area of research [18,19]. In those domains, the use of explanations often focuses on understanding the mechanisms of the learned models during visualization [20–22] and decision making [11,23]. In [24] Dam et al. argued for explainable software analytic issues to assist human understanding of machine prediction as a crucial aspect to gaining trust from software practitioners. Zevenbergen et al. [25] collected five use cases for explainability in machine learning-based systems, along with relevant motivating scenarios. The concept of embedding explanations in software systems has previously been extensively researched in the domain of knowledge-based systems [26]. Vultureanu-Albişi et al. [27] aim at explainability in recommendations while taking the user's context into consideration. The authors [27] presented an explainable recommendation system that is quantitative and qualitative in nature and gives context-based explanations of recommendations.

A qualitative comparison of our approach with existing works in the literature in presented in Table 6 in Sect. 8.

3 Preliminaries

In this section we briefly illustrate some of the preliminary concepts that will help the reader to better understand the objective and results of this research work.

3.1 Goal model concepts

Goal models are an abstract way of eliciting, modeling and analyzing software requirements at an early phase. Primarily, goal models are comprised of the following components as stated in [28]:

- *Actors or Agents*: " An actor is an active entity that carries out actions to achieve goals by exercising its know how."
- *Goals*: " A goal is a condition or state of affairs in the world that the actor would like to achieve."
- *Tasks*: " A task specifies a particular way of doing something. When a task is specified as a subcomponent of a (higher) task, this restricts the higher task to that particular course of action."
- *Resources*: " A resource is an entity (physical or informational) that is not considered problematic by the actor."
- *Soft-goals*: " A soft-goal is a condition in the world that the actor would like to achieve, but unlike in the concept of (hard) goal, the criteria for the condition being achieved are not sharply defined a priori and are subject to interpretation."
- *Decomposition or Refinement Links*: These links are used to decompose a goal into sub-goals or tasks. There are two types of decomposition links: *AND (Task Decomposition Link)* and *OR (Means-end Link)*.
- *Dependency Links*: An actor may be depended on another actor to achieve goals, perform tasks or furnish resources.
- *Contribution Links*: Contribution links are used to represent the varying degrees of impact that one softgoal or goal has upon another softgoal.

The different components of a goal model help requirement engineers and developers to visualize higher-level strategic objectives at an early phase. Contextual goal models [7] further help in associating operational contexts with goals or requirements of the system. An example case study consisting of hard and soft goals for a patient healthcare system is added in Appendix-I, and the goal model corresponding to it is available in our GitHub repository: https://github.com/ConEModel/PHA-Goal-Model.

3.2 Non-functional requirements

NFRs characterize an important quality aspect of a software system. NFRs do not exist independently and are often related to different functional requirements. In addition, NFRs are interrelated, i.e., one NFR may have a positive or negative impact (conflicts) on the fulfillment of another NFR [29]. In a multi-stakeholder system, there may exist multiple such conflicting NFRs and for each NFR there may be more than one operationalization strategy [30]. These different operationalization strategies evolve due to the different contexts in

which the system may operate. Several research efforts have been made to identify NFR conflicts at an early software development phase [29,31]. However, there is limited work that tries to identify the NFR conflicts arising dynamically when a system operates in different contexts. This research work is aimed toward identifying these correlations between contexts and NFR conflicts.

3.3 Sequential rule mining

Sequential rule mining algorithms [32,33] establish associations between the items in a sequence. A sequential rule maps sequential relationship among items. It shows that if some item(s) occurs in a sequence, what are the other item(s) that may follow, based on some support and confidence values [32]. Support gives a measure of what portion of the input database in sequential rule mining satisfies the mined rule. Confidence gives the conditional probability of an item to occur given that another item has occurred. There are many different sequential rule mining algorithms like ERMiner [32], CMRules [34], RuleGrowth [33], CMDeo [34] and RuleGen [35]. Each of them differs by its performance characteristics. In this research work, we have explored different sequential rule mining algorithms for deriving context-context correlations and contexts-NFR conflict correlations.

3.4 Sentence embedding and semantic similarity

Representation of words and sentences as vectors in a low-dimensional space enables us to incorporate various deep learning NLP techniques to overcome different challenging tasks such as semantic similarity, named entity recognition, key phrases extraction and many more. The quality of information inference, using a language model, is determined by the following three characteristics:

1. The corpus on which the model is trained.
2. The architecture of the model.
3. The amount of contextual knowledge the model focuses on.

The BERT model is built on the transformer architecture, and it is trained on gigabytes of data from various sources (mostly from Wikipedia and Book Corpus) in an unsupervised fashion [36]. In a nutshell, the training is performed by masking nearly 15% of the words in a corpus and allowing the model to predict masked words. As the model is trained to predict, it also learns to generate an efficient internal representation of words as word embedding. The BERT model is a bi-directional model that explores text representation from both directions in order to obtain a clearer understanding of the context. In this regard, we consider the BERT model as it sets a new benchmark performance on semantic textual similarity among state-of-the-art models [36,37]. Furthermore, it is also possible to fine-tune BERT-based transformer models on task-specific datasets.

4 The ConE framework

In this section, we introduce the overall idea and workflow of our contextual explainability framework (refer to Fig. 1 using a simple case study. We take the help of a Patient Healthcare Assistance (PHA) system (refer to Appendix-1) to illustrate the ConE framework. The basic assumptions for applying our framework are as follows:

1. The availability of domain-specific event logs. Such logs serve the purpose of reusing vocabulary for training our transformer-based language model and other NLP downstreaming tasks.
2. The availability of a goal model representation of the system requirements.
3. A contextual knowledge base that identifies the variation points in the goal model and defines the strategy to be followed under specific contexts.

There are research works in different domains where researchers have used event logs for their simulations [38,39]. Many of these event logs have been made publicly available and can be used for experimentation purposes. The availability of goal models might be a challenge, particularly for legacy systems, where goal models had not been used for requirements elicitation. In such situations, requirement engineers may have to represent the existing requirements using goal modeling concepts. However, the annotation of goal models with the context knowledge base is possible. Research works of Ali et al. [7] have shown how such annotated goal models can be created.

The architecture of the framework is depicted in Fig. 1. The end-to-end flow of activities and outcomes of the framework are shown using the red arrows. The different subprocesses of the framework are shown as light-blue boxes with a numeric tag in orange. The yellow artifacts represent intermediate process outputs fed as input to a subsequent phase. The green artifacts are external inputs required for particular processes. Different machine learning models used and developed as part of the framework are shown in red boxes. The following subsections elaborate on each process activity of the framework with the help of this running example of PHA system.

4.1 Pre-training BERT-based transformer model

The application of a pre-training BERT-based transformer model is marked in Fig. 1 as activity (1). We consider the BERT-based transformer model [36,40] and pre-train it for

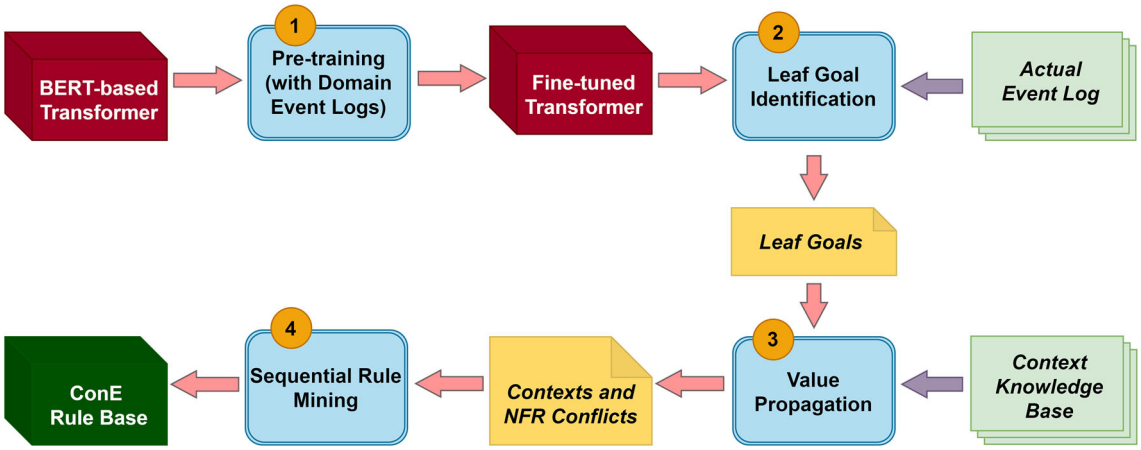


Fig. 1 The Contextual Explainability (ConE) Framework

the particular domain in which we want to deploy our framework. The pre-training process uses existing event logs of the target application domain as the training data. The outcome of this process is a pre-trained BERT-based Transformer Model (BTM) which will be used in the next activity. For the PHA system case study, the BTM is trained using event logs from the healthcare domain.

4.2 Leaf goal identification

This is the sub-process (2) of the ConE framework shown in Fig. 1. The pre-trained BTM, derived from the first step, is further fine-tuned to be used in the goal identification process. This is done by using semantic similarity, as shown in Fig. 2. In the very first step of the fine-tuning process, the goals and event logs are fed into the pre-trained BTM. The transformer model accepts these inputs and generates token embedding. Pooling strategies generate fixed-sized sentence embedding (from token embedding) to perform certain NLP tasks—like classification, semantic similarity, and named entity recognition. There are two possible pooling strategies—*MEAN Pooling* and *MAX Pooling*—that are used for sentence representation tasks. Among these two strategies, *MEAN Pooling* performs better for sentence representation [41]. In case of *MEAN Pooling*, the instances of fixed-size sentence embedding are obtained by averaging the hidden state of encoding layer on the time axis. Finally, the cosine similarity is measured for these fixed-size sentence embedding to determine the semantic similarity between events and goals. The highest similarity measure determines the leaf level goal corresponding to a particular activity in the event log.

It is worth mentioning that cosine similarity is generally used as a metric that measures the angle between vectors where the magnitude of the vectors is not considered.

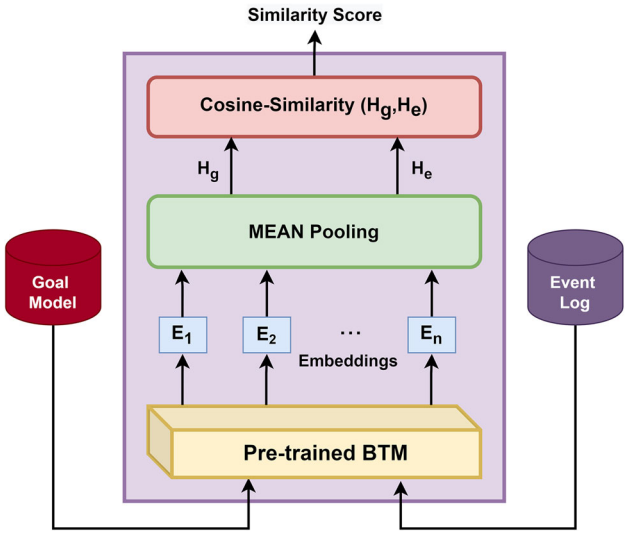


Fig. 2 Architecture of fine-tuning pre-trained BTM for calculating Semantic Similarity

Sentences could be of uneven length. In this situation, the semantic similarity between two sentences can be affected if we consider the spatial distance measures. As cosine similarity measures the angle between two vectors, it is preferable for measuring semantic similarity than the approaches that use spatial distance measures such as Euclidean distance [41,42]. This is the reason for choosing the cosine similarity measure.

For example, Fig. 3 illustrates a part of the goal model specification of our PHA system. Table 1 shows a part of the event log provided as input to the fine-tuning process described above. Column 6 in Table 1 lists the leaf level goals from Fig. 3 correspond to the activity listed in column 2 of the same table.

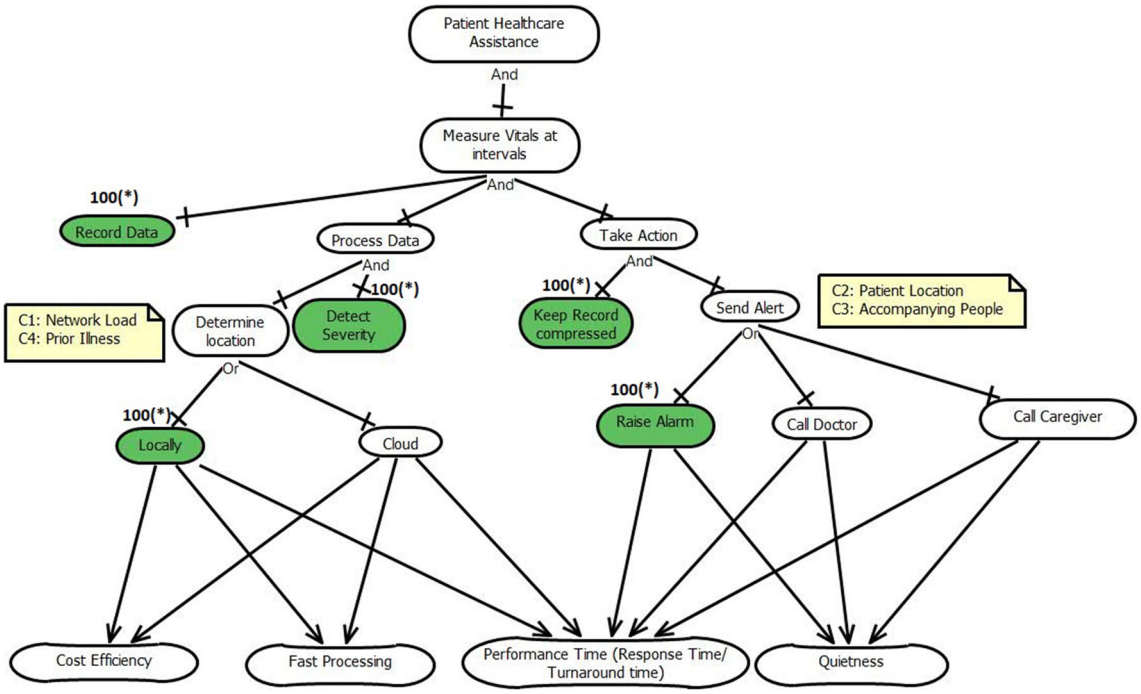


Fig. 3 The goal model specification with its associated context knowledge base

Table 1 Some entries of the event log with respect to a particular thread

Thread ID	Activity	Type	Date	Time	Leaf goal
1	Blood pressure measurement	Complete	02/04/2021	11:03	Record data
1	Local authentication	Complete	02/04/2021	11:05	Locally
1	Data analysis	Complete	02/04/2021	11:07	Detect severity
1	Results compressed and stored	Complete	02/04/2021	11:09	Keep record compressed
1	Alarm raised	Complete	02/04/2021	11:11	Raise alarm

4.3 Value propagation

In sub-process (3) of the ConE framework, the leaf goals identified in the previous step are used to discover the contexts and NFR conflicts activated for each event thread in the event log. The steps performed in activity (3) are as follows:

1. We obtain the corresponding leaf goals by fine-tuning the BTM for each thread of entries in the event log. These leaf goals are then marked with +100 satisfaction values within the goal model (refer to Fig. 3). In subsequent steps, we observe the propagation of satisfaction values from the leaf goals through goal decomposition and contribution links.
2. We have used the execution view of jUCMNav [43] tool to observe the propagation of satisfaction values. At first, we have manually assigned a +100 evaluation value (in the execution strategy of the tool) to each of the leaf goals identified. The jUCMNav tool then propagates the satisfaction values toward the root goal and through the contribution links toward the softgoals (or NFRs) [44].
3. Next, we refer to our pre-existing *Context Knowledge Base* (refer to Table 2) that contains the various contexts and their values associated with different leaf goals. We have associated contexts only with the variation points [7,8] within the goal model (i.e., where goals have OR decomposition). This can be observed in Fig. 3, where we see that each OR-decomposed goal has some contexts associated with it. Each variation point within the goal model gives rise to multiple possible execution strategies. The particular strategy that is chosen depends on contexts activated at run time and their corresponding values.

Table 2 Context knowledge base

Context	Values	Leaf goals
Network load (C1)	Low (C1.1)	Cloud
	Medium (C1.2)	Locally
	High (C1.3)	Locally
Patient location (C2)	Home (C2.1)	Raise alarm
	Old-age-home (C2.2)	Call Caregiver
	Hospital (C2.3)	Call Doctor
Accompanying people (C3)	Relative (C3.1)	Raise alarm
	Caregiver (C3.2)	Call Caregiver
	None (C3.3)	Call Doctor
Prior Illness (C4)	None (C4.1)	Locally
	Yes (C4.2)	Cloud

Next, we perform the following two tasks:

(a) First we identify the specific contexts that have been activated with respect to the leaf goals identified in activity (2) of the framework. Thus, for each event thread, we can trace out the different environmental contexts that were activated, which eventually helps to reason about the system behavior.
(b) Second, we identify the different NFR conflicts for each event thread. When a leaf goal has opposing contributions to two different softgoals, there is a possibility of conflicts existing between them. The jUCMNav tool marks such softgoal (or NFR) conflicts with distinguishing colors.

These two observations are used to correlate, how the occurrence of various contexts affects the satisfaction of different NFRs.

In Fig. 4a, the leaf level goals identified in the rightmost column of Table 1 are fully satisfied by assigning the value +100. Figure 4a shows the propagation of satisfaction values from these leaf goals in jUCMNav's execution mode. In Fig. 4a, we make the following observations from this value propagation:

1. At the *{C1,C4}* variation point, the context parameters have the values *C1.2: Network Load = Medium* and *C4.1: Prior Illness = None*.
2. At the *{C2,C3}* variation point, the context parameters have the values *C2.1: Patient Location = Home* and *C3.1: Accompanying Person = Relative*. The values of the context are derived from the pre-existing *Context Knowledge Base* (refer to Table 2) that we assumed for this example.
3. The leaf goal *Locally* gives rise to two NFR conflicts—*{Cost Efficiency, Fast Processing}* and *{Performance Time, Fast Processing}*.
4. The leaf goal *Raise Alarm* results in only one NFR conflict—*{Performance Time, Quietness}*. In this case the soft-goal *Performance Time* is satisfied, but the soft-goal *Quietness* is affected.

Similarly in Fig. 4b, we consider another scenario where leaf goals *Cloud* and *Call Doctor* are satisfied. Here we can make the following observations:

1. At the *{C1,C4}* variation point, the context parameters have the values *C1.1: Network Load = Low* and *C4.2: Prior Illness = Yes*.
2. At the *{C2,C3}* variation point, the context parameters have the values *C2.3: Patient Location = Hospital* and *C3.3: Accompanying Person = None*.
3. The leaf goal *Cloud* gives rise to two NFR conflicts—*{Fast Processing, Cost Efficiency}* and *{Fast Processing, Performance Time}*. However, the observation here is that only the soft-goal *Fast Processing* is satisfied in this case. The soft-goals *Cost Efficiency* and *Performance Time* that were satisfied in the previous case (Fig. 4a) are both denied here.
4. The leaf goal *Call Doctor* results in only one NFR conflict—*{Quietness, Performance Time}*. Unlike the previous case (Fig. 4a) here soft-goal *Quietness* is satisfied, but *Performance Time* is affected.

We already know, from the goal model, how the contributions of leaf goals to the associated soft-goals result in conflicts among the NFRs. The ConE framework further helps to identify how the satisfaction of these conflicting NFRs is affected by several contextual parameters arising in the systems operational environment. The mapping of execution trace to contextual goal models helps in identifying when the conflict actually occurs and the contexts responsible for degradation of different quality of service parameters.

The data, which are inferred from an individual thread in the event log, are recorded. The process is repeated for every thread of activities that have been recorded in the event log.

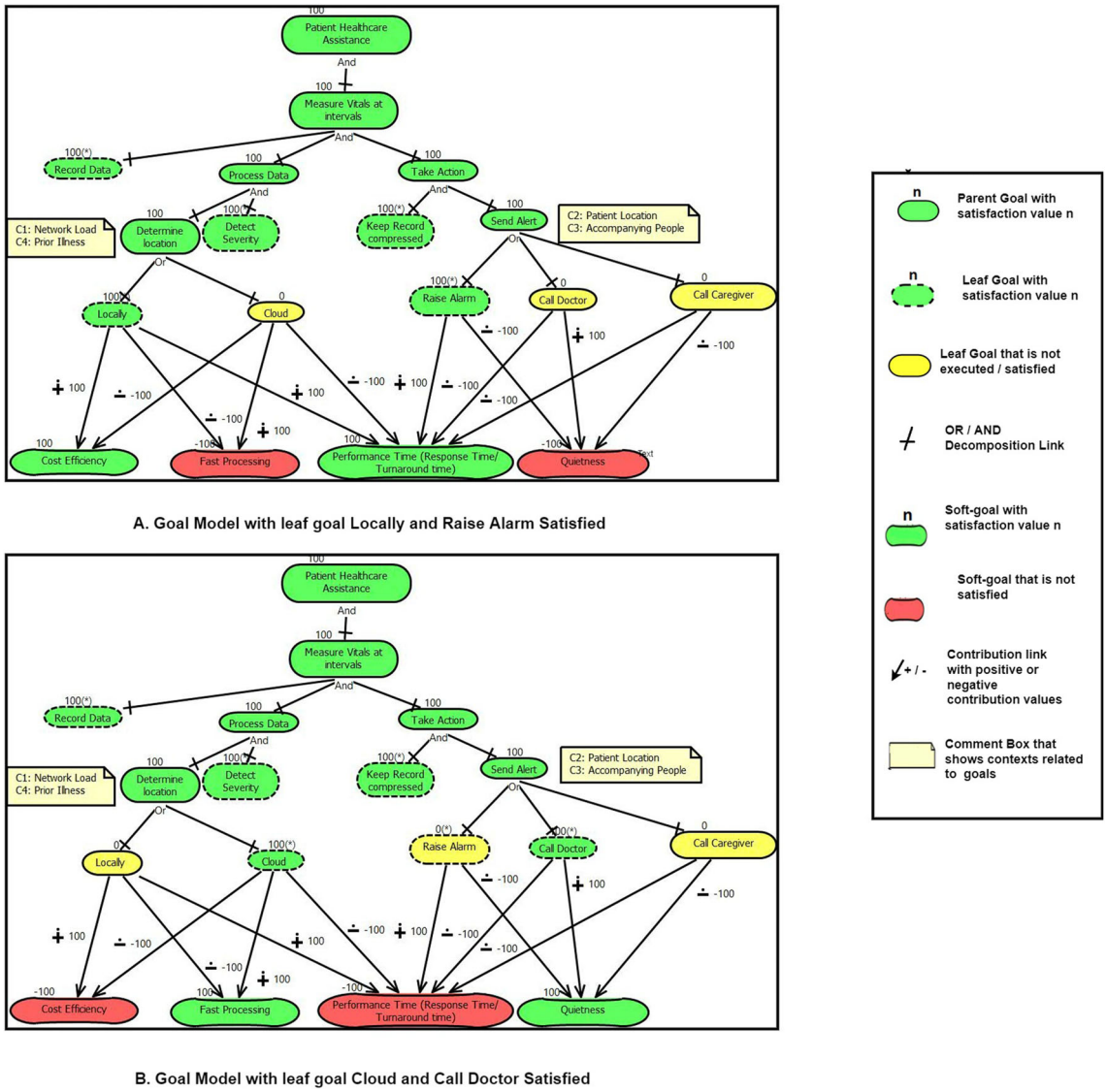


Fig. 4 Propagation of satisfaction and contribution values using jUCMNav

The derived sets of inferences serve as input for the next step of Sequential Rule Mining.

4.4 Sequential rule mining

This is sub-process (4) of the ConE framework shown in Fig. 1. The set of identified contexts and NFR conflicts (as obtained from previous step) associated with each thread in the event log serves as input for the sequential rule mining activity.

The sequential rule miner produces the main deliverable of this research—the *Contextual Explainability (ConE) Model*. The *ConE Model* mines the following two types of sequential rules:

1. Given a context C_i, what are the other contexts that are also activated in the operations environment?
2. Given a context C_i, which are the NFR conflicts that occur in the operations environment?

We take the derived dataset of contexts and NFR conflicts (from activity (3)) and feed it into a sequential rule miner (for instance ERMiner [32]). Table 3 shows one such sample input dataset. Thread 1 in Table 3 corresponds to the thread shown in Table 1.

We feed two input datasets in the sequential rule miner to derive the two types of rules, respectively. The first input consists of sequences representing the set of contexts that are activated for a particular thread in the event log. This dataset is captured by `Column 2` of Table 3. In the second input, each sequence consists of both the contexts and the NFR conflicts that have been activated for a particular thread. This dataset is derived from `Columns 2 and 4` of Table 3.

Table 3 Derived dataset of contexts and NFR conflicts fed as input to the sequential rule mining algorithms

Thread ID	Contexts	Associated NFRs	NFR conflicts
1	C1.2: Network Load = Medium C4.1: Prior Illness = None C2.1: Patient Location = Home C3.1: Accompanying Person = Relative	Cost Efficiency Fast Processing Performance Time (Turnaround time) Quietness	NC1: Cost Efficiency and Fast Processing NC2: Fast Processing and Performance Time NC3: Quietness and Performance Time
2	C1.1: Network Load = Low C4.2: Prior Illness = Yes C2.2: Patient Location = Old-age-home C3.2: Accompanying Person = Caregiver	Cost Efficiency Fast Processing Performance Time Quietness	NC1: Cost Efficiency and Fast Processing NC2: Fast Processing and Performance Time
3	C1.2: Network Load = Medium C4.1: Prior Illness = None C2.1: Patient Location = Home C3.1: Accompanying Person = Relative	Cost Efficiency Fast Processing Performance Time Quietness	NC1: Cost Efficiency and Fast Processing NC2: Fast Processing and Performance Time NC3: Quietness and Performance Time
4	C1.3: Network Load = High C4.2: Prior Illness = Yes C2.1: Patient Location = Home C3.2: Accompanying Person = Caregiver	Cost Efficiency Fast Processing Performance Time Quietness	NC1: Cost Efficiency and Fast Processing NC2: Fast Processing and Performance Time NC3: Quietness and Performance Time

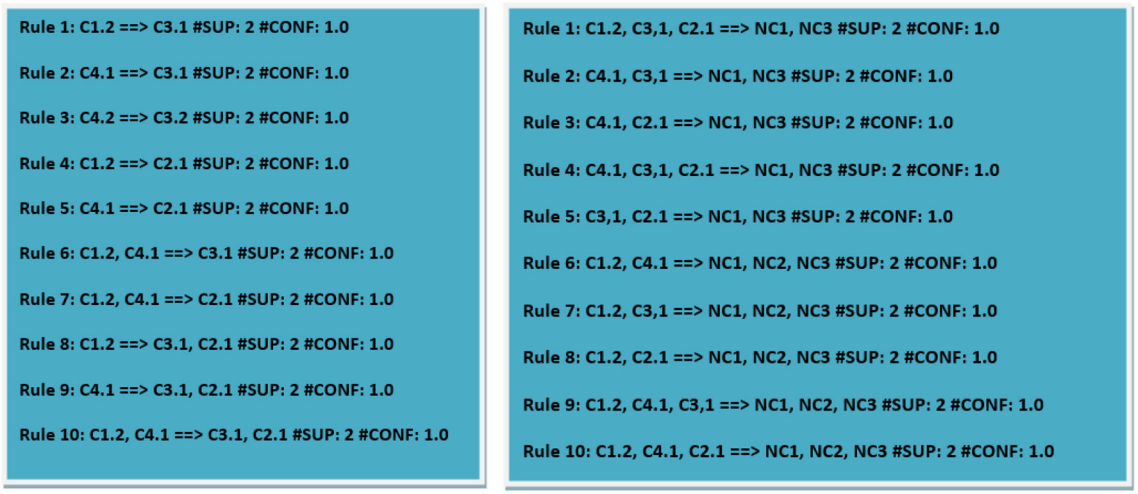
ConE rule base

The *ConE* rule base consists of two types of correlations:

1. ***Context-Context Correlations***: Each correlation $X \Rightarrow Y$ implies that whenever context(s) X has occurred subsequently context(s) Y has also been activated in the operations environment. In Fig. 5a, we have shown the rules that have been mined by a sequential rule mining algorithm (ERMiner [32]). The correlations are supported by the corresponding support and confidence values. For example Rule 2 in Fig. 5a shows that whenever the context *Accompanying Person* has the value *Relative* (context C4.1), the context *Prior Illness* has value *None* (context C3.1). Similarly, in Rule 3 in Fig. 5a, we find that whenever the same context *Accompanying Person* has the value *Caregiver* (context C4.2), the context *Prior Illness* has value *Yes* (context C3.2). Each of these rules shows how different contexts are correlated based on their values observed in the systems operational environment.
2. ***Context-NFR conflict(s) Correlations***: Each correlation $X \Rightarrow Y$ implies whenever context(s) X has been activated, NFR conflict(s) Y has also been observed in the operations environment. Figure 5b shows the Context-NFR conflict correlations that have been mined by a sequential rule mining algorithm (ERMiner [32]). For example in Rule 1 in Fig. 5b we find that whenever contexts *Network Load, Accompanying Person,* and *Patient Location* have values *Medium* (C1.2), *Relative* (C3.1) and *Home* (C2.1), respectively, the NFR pairs $\langle$*Cost Efficiency*,*Fast Processing*$\rangle$ (NC1) and $\langle$*Quietness*,*Performance Time*$\rangle$ (NC3) are in conflict. The reader may correlate the data in Table 3 and Fig. 5a and b to have a better understanding of the rules.

The sequential rule mining algorithm shows how contexts with different values are correlated among themselves and also results in different NFR conflicts. The rules mined always satisfy the specified support and confidence threshold. We also observe that in Fig. 5a some of the rules can be formed from the combination of other rules. Each of these rules has the same support values, but it is not always the case (as observed in our experimental Sect. 6). The results generated in Fig. 5a are based on a very small example; hence, they have the same support values. We do not remove these rules that are formed from the composition of other rules as in real-life scenarios they may have different support values. The support values determine the significance of each of these rules.

The *Context-Context Correlations* serve as the basis for predicting the system behavior based on context activation in the operations environment. The *Context-NFR conflicts*



(a) Context-Context Correlations (b) Context-NFR conflict Correlations

Fig. 5 ConE Rule Base

Correlations are used to identify how different contextual factors contribute to the triggering of different NFR conflicts. This serves the system analyst in understanding whether the NFRs are satisfied as per their priorities. These observations can be useful for the developers for refactoring the system to meet the desired needs. Further, the prediction of context-NFR conflict correlation helps to build a smart system that can adjust its quality parameters based on past predictions.

With respect to the motivating case study of the Uber app (explained in Sect. 1), the context-NFR conflict correlation would be captured as -

$$C1 \Rightarrow NC1$$

where $C1$ refers to the environmental context *heavy rain* and $NC1$ refers to the conflicts between the NFRs *fast response time* and *feature availability*. Uber system designers can deploy the application server architecture to address this correlation. Whenever an end-user tries to use the Uber cab booking service in heavy rains ($C1$), a lightweight version of the app is loaded to minimize the impact of the NFR conflict ($NC1$).

5 Setting the scene for the experimental evaluation

Our empirical evaluation aims to assess the ConE framework on a significant case study.

5.1 Case study selection criteria

1. *Rationale*. The PHA system was chosen as we have been exploring and using this case study in all of our recent works. We have a clear understanding of the requirements of the system. We also found several healthcare system event logs that could be freely accessed for training our models.
2. *Objective*. The case study was adopted for illustrating the workflow of the proposed methodology. Every subprocess of the proposed framework has been elaborated with examples from our case study.
3. *Research Question*. Given the event logs of healthcare services that have been delivered, can we correlate the requirement contexts with the NFR conflicts that affect the quality of these services?
4. *Method of Data Collection*. The training data for our pre-training process were obtained from freely accessible online repositories and archives. Possible contextual parameters for our framework were obtained from the datasets generated by the ITRA project [45–47]. These datasets were generated from interviews conducted with healthcare service providers distributed over different locations.
5. *Selection of Data*. The datasets were generated by aggregating interviews conducted with all the different stakeholders involved with the project. The interviews were conducted once every week over a period of six months.

5.2 Experimental environment

1. *Data Acquisition*: We have carried out a literature survey and found several research works [38,48] that pro-

pose Deep Learning models trained on electronic health records or clinical datasets. Two databases Cerner Health Facts[1] and Truven Health MarketScan[2] are well-known datasets for electronic health records, but they are not publicly accessible. On the other hand, MIMIC-III [49] is an openly accessible popular dataset for the clinical domain and hence, used in our work. We obtained an initial set of healthcare system requirements from the ITRA project [45–47]. These data were further extrapolated using requirements collected from healthcare experts and stakeholders of the project over a period of time. The interviews were conducted by researchers who have a sound understanding of FRs and NFRs. The NFRs were observed or collected in iterations after different deployments of the project. The contextual parameters affecting the system were also collected by observing the different geographical locations in which it was deployed. The obtained set of requirements have been represented as contextual goal models.

2. *Data Generation for Testing*: The FRs obtained from the ITRA project were used for generating the event threads of our healthcare event logs. Based on the variation points in the goal model, we have created different execution threads in the log. The activities in the event log are created by considering different possible combinations of the contextual parameter values associated with the variation points. The same thread of activities is replicated multiple times to mine sequential rules having sufficiently high confidence and support values. Some event threads were created using random combinations of activities to obtain a fair distribution of sequential rules with low confidence and support values as well.
3. *Data Analysis*: The testing dataset was generated through multiple iterations. A careful analysis of each data increment was performed to detect the introduction of data bias due to repetition of certain event threads. Replications of specific event threads were introduced (or deleted) from the event logs for removing these biases even if they get introduced during the data generation process.
4. *Ground Truth Estimation*: There was an unavailability of ground truth in terms of identifying the leaf goals corresponding to the entries in the event logs. We have identified the leaf goals[3] corresponding to each activity in the event log manually. This served as the ground truth for our experiments and helped in selecting the best BTM for our framework.
5. *Model Selection*: For the purpose of pre-training, we have experimented with five different BERT-based transformer models. A detailed methodology and a comparative analysis of these models are provided in Sects. 6.1 and 6.2, respectively. For sequential rule mining tasks, we identified five standard sequential rule mining algorithms existing in the state of the art and compared their performance characteristics in Sect. 6.5. However, only two representative algorithms were used to document the efficiency analysis.

[1] https://sc-ctsi.org/resources/cerner-health-facts.

[2] https://www.ibm.com/products/marketscan-research-databases/databases.

[3] https://github.com/ConEModel/PHA-Goal-Model.

6 Experimental results

In this section, we elaborate the experiments performed to demonstrate the functioning of the *ConE* framework. All experiments are performed on a workstation with 11th Generation Intel Core i7 processor (3.0 GHz), 16GB RAM and Windows 11 operating system.

6.1 Pre-training BERT-based transformer models

In our pre-training, at the foundation level, we use five BERT-based transformer models—BERT [36], RoBERTa [40], DistilBERT [50], PUBER [51] and FiBER [51]. Each of them has a default architecture of 1024 hidden layers, each with 24 transformer blocks and 16 self-attention heads. We train these models on healthcare event logs and name them as *LogBERT*, *LogRoBERTa*, *LogDistilBERT*, *LogPUBER* and *LogFiBER*, respectively. The pre-trained models can be obtained by the following steps:

1. The first step is to build the domain-specific vocabulary. In order to build the vocabulary from the alphabet of single byte, we have used the default *WordPiece* tokenizer with 30,000 token vocabulary.
2. Once the vocabulary is prepared, we start training the language model. The vocabulary is then used for word embedding and masking.
3. As the FiBER model is based on BERT, we train it on a task of *Masked Language Modeling* [36] that masks some percentage of the input tokens randomly and then predicts those masked tokens.
4. The final hidden vectors corresponding to the masked tokens are fed into a softmax layer. The training environment is described as follows.
   (a) Batch size is set to 32.
   (b) Number of training steps is 100,000.
   (c) Learning rate is kept as 2e-5.

These settings enable us to obtain a bidirectional pre-trained domain-specific language model. Once this model is developed, it is ready to be used for different downstream tasks. Figure 6 shows the architecture for obtaining the Log-

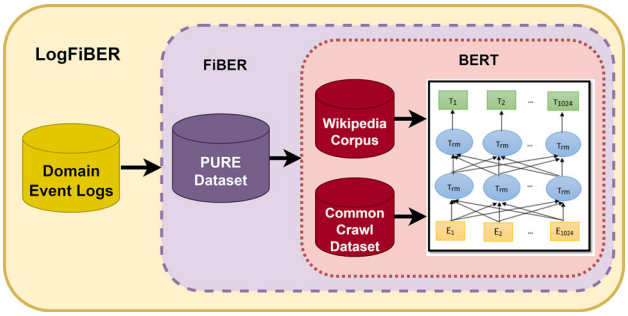


Fig. 6 The LogFiBER Transformer Model Architecture

FiBER transformer model. The architecture remains the same for all other pre-trained models. The pre-training datasets, models and results are available in our GitHub repository.[4]

6.2 Comparing BERT-based transformer models

We have synthetically created an event log (refer to Sect. 6.3) that serves as the test data for the five pre-trained models. In this section, we show the comparison between the different pre-trained models by measuring their accuracy, precision, recall and F1-score.

Accuracy

Figure 7a shows the accuracy achieved by the pre-trained BTMs mentioned earlier. LogFiBER outperforms the other models and achieves the highest accuracy of 83.33% in identifying the leaf goals. LogBERT scores 76.80%, which is better than the remaining models. LogDistilBERT and LogPUBER give 71.42% and 62.71% of accuracy, respectively. LogRoBERTa gives the lowest score of 61.01%.

Precision

Figure 7b shows the precision achieved by the pre-trained BTMs. LogFiBER model achieves 84.61% of precision, whereas highest precision of 89.28% can be seen with LogBERT. LogRoBERTa and LogDistilBERT show 85.96% and 83.33% of precision, respectively. LogPUBER shows poor performance with 63.79% of precision.

Recall

Figure 7c shows the recall achieved by the pre-trained BTMs. LogFiBER model gives the highest recall of 98.21%. LogPUBER and LogRoBERTa both achieve the 97% mark with 97.36% and 97.29%, respectively. LogBERT and LogDistilBERT show similar recall measures of 83.87% and 83.33%, respectively.

F1-score

Figure 7d shows the analysis of F1-Score of the pre-trained BTMs. LogFiBER model achieves the highest F1-Score of 90.90%. LogBERT model stands second among them with 86.66% of F1-Score. LogPUBER and LogRoBERTa show F1-Score measures of 77.08% and 75.78%, respectively. LogDistilBERT achieves 83.33% of F1-Score.

The *ConE* framework aims to identify the complete set of correlations that exist between the contexts and NFR conflicts of a particular system. In order to achieve completeness, accuracy and recall are more significant than precision as a metric for choosing the language model. The accuracy and recall of *LogFIBER* are better than that of *LogBERT* by a factor of 6.5% and 15%, respectively. F1-score combines the recall and precision metrics and, here also, *LogFIBER* outperforms *LogBERT* by a factor of 3% (approx.). Thus, we select LogFiBER for the subsequent phases.

6.3 Synthetic event log creation

We have created a goal model for a PHA system. The goal model shown in Fig. 3 is a part of the larger goal model. The complete goal model is available in our GitHub repository[4]. We have built our event log taking this goal model into consideration. Table 4 shows the different characteristics of our goal model and event logs.

The steps for building the event log were as follows:

1. First, for each leaf goal in the goal model, we determine a list of activities to be performed for satisfying that goal. For example, if we consider the goal *Keep Record Compressed* in Fig. 3, the sequence of activities that are performed to achieve this goal are—(a) fetching the record, (b) compressing the record and (c) storing the record in a suitable location. We have considered an existing hospital event log [52] while determining the activities for each goal.
2. We continue by adding sets of activities to threads. Each thread in the event log consists of activities related to multiple goals. We have considered the temporal relationships among the leaf goals while adding the activities in the log. For example, the activities for *Record Data* must appear before the activities for *Keep Record Compressed* (refer to Fig. 3) in the log for the sake of achievability of final goals.
3. We try to infer some temporal ordering from the interactor dependencies for sequencing the events in the event log. For example, let us consider two activities involving two different actors in our PHA system—patient (actor)

[4] https://github.com/ConEModel/Pre-training-Data.

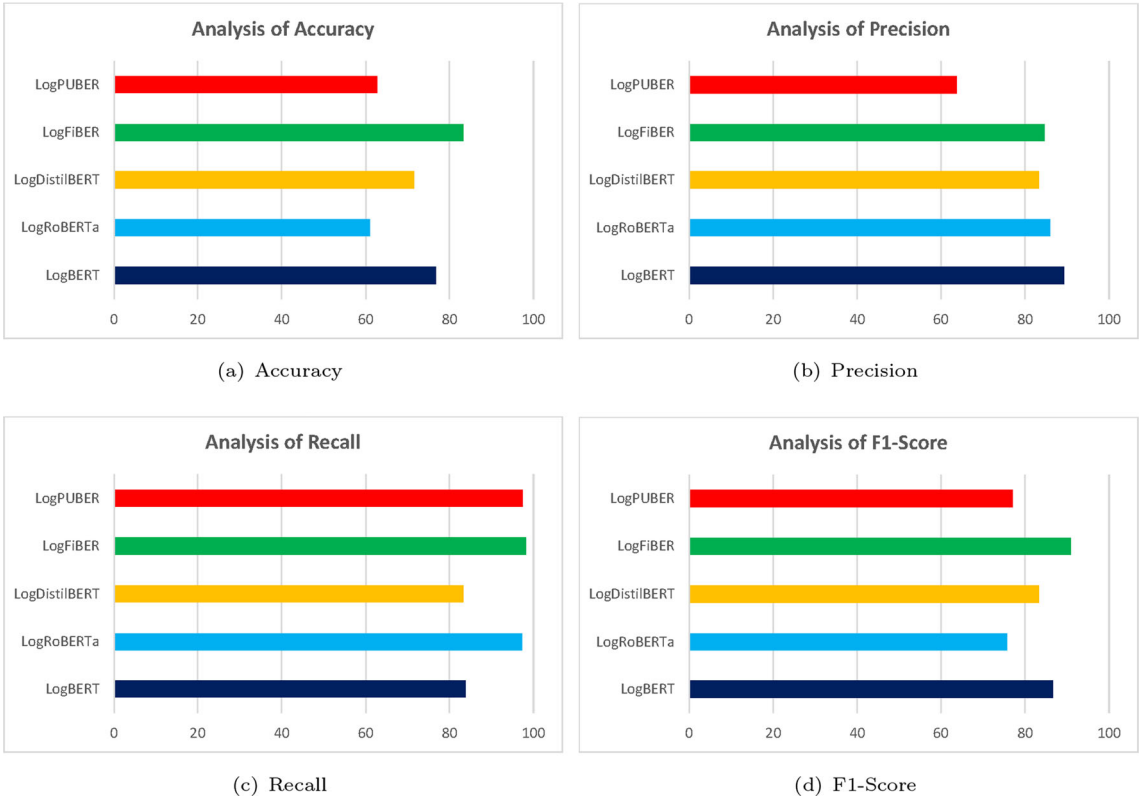


Fig. 7 Analysis of Different Models

orders medicine online and the pharmacist (actor) gets it delivered. Then the activity of the patient is recorded at a time instant earlier than the activity of the pharmacist to keep the event log consistent. As observed from our studies, popular requirement goal models such as i* [28] do not capture temporal ordering explicitly.

4. We have interleaved the activities corresponding to mutually exclusive goals. This brings variation while maintaining correctness in the ordering of the events.
5. The same set of entries are repeated in different threads to populate the event log (refer to Table 4).

Table 4 Characteristics of goal model and synthetic event log

Parameter	Value / Range
No. of leaf goals	48
No. of actors in the goal model	4
No. of inter-actor dependencies	10
No. of activities for each leaf goal	3–9
Total no. of threads	70,000
Total no. of activities in the log	1,116,518
Thread repetition	500–10,000

The event log created manually is available in our GitHub repository.[5]

6.4 Simulation

The comparative analysis in Sect. 6.2 shows that LogFiBER outperforms other models in terms of accuracy, recall and F1-score. We consider the leaf goals identified by the LogFiBER transformer model for performing the subsequent step of our *ConE* framework. The identified leaf goals are activated in the goal model in jUCMNav [43] execution mode to observe the propagation of satisfaction values (through goal decomposition and contribution links) for each thread in the event log. The final outcome is the identification of contexts and NFR conflicts. Activity (3) is performed against two sets of leaf goals:

1. **Actual Leaf Goals**: This is the actual set of leaf goals corresponding to each activity in the event log identified manually. We determine the contexts and NFR conflicts that are activated in the operations environment based on this set. `Note:` *These rules mined subsequently are treated as the ground truth for our experimental process.*

[5] https://github.com/ConEModel/Event-Logs.

2. **Predicted Leaf Goals**: This is the predicted set of leaf goals identified by the LogFiBER transformer model.

The process described in Sect. 4.3 is performed for both actual and predicted leaf goals. The list of contexts and NFR conflicts activated in the operations environment is recorded for each of the 70,000 threads.

We have two input datasets for context-context rule mining—one based on actual leaf goals and another based on predicted leaf goals. They are named *Actual Context Dataset* (C_A) and *Predicted Context Dataset* (C_P). Similarly, we have two input datasets for context-NFR conflict rule mining—one based on actual leaf goals and another based on predicted leaf goals. They are named *Actual Context NFR conflict Dataset* (CN_A) and *Predicted Context NFR conflict Dataset* (CN_P). These four datasets are available in our GitHub repository.[6] These four sets of input sequences are subjected to different sequential rule mining algorithms. The results are discussed in the next section.

It is to be noted that these four input datasets are sparse in nature. This is due to the manually created event logs. Each thread in the event log contains only a small number of entries. This results in few associated contexts and NFR conflicts for each thread resulting in a sparse dataset.

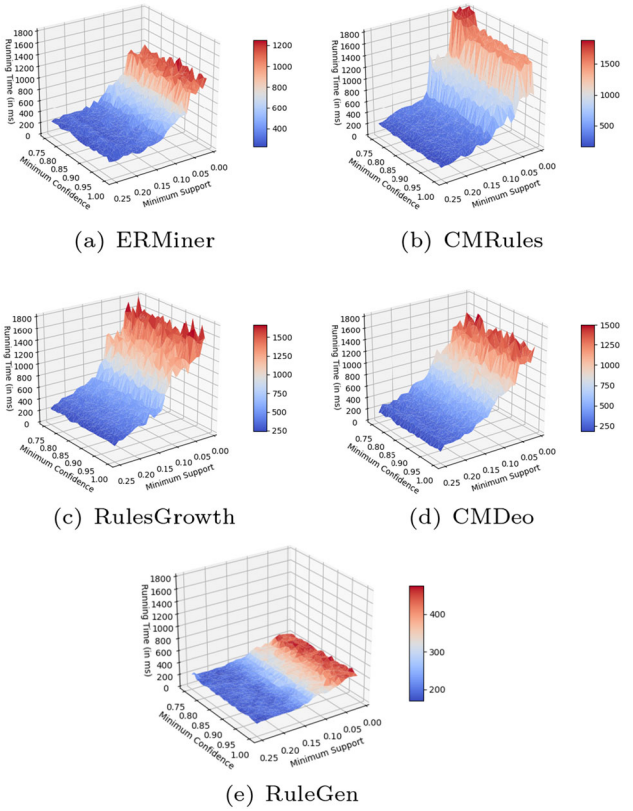


Fig. 8 Running Time for Context-Context Rule Generation

6.5 Performance characteristics

We have mined sequential rules from our derived dataset using 5 algorithms ERMiner [32], CMRules [34], RuleGrowth [33], CMDeo [34] and RuleGen [35]. Typically, the rules mined by the algorithms ERMiner, CMRules, RuleGrowth and CMDeo are in the same format, but RuleGen mines a different form of sequential rules.

The sequential rules mined by ERMiner [32], CMRules [34], RuleGrowth [33] and CMDeo [34] are in the form of $X \Rightarrow Y$, where X and Y are two sets of items. Here X and Y are disjoint and unordered sets of items. In the case of RuleGen [35] the sequential rules are in the form of $X \Rightarrow Y$, such that X and Y are ordered itemsets and X is a subset of Y. In our application scenario for context-context rule generation, both X and Y represent the sets of contexts. However, for context-NFR conflict rule generation, X represents contexts and Y represents NFR conflicts.

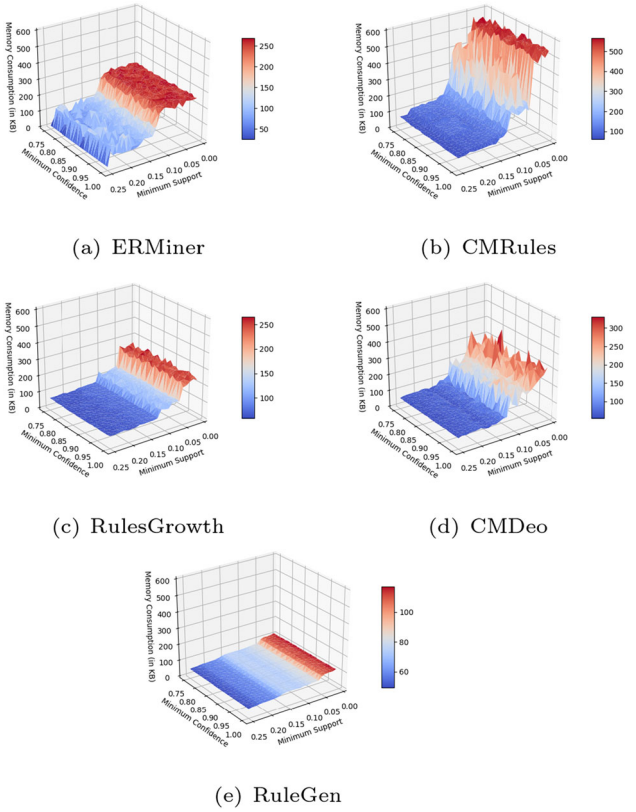
Context-context rule generation

Context-Context Correlations are created by mining rules using all 5 algorithms on both datasets C_A and C_P. The rules mined for dataset C_A (let us assume them to be RC_A) represent the actual set of context-context sequential rules for our event log. The rules mined for dataset C_P (let us assume them to be RC_P) represent the context-context sequential rules mined based on the leaf-goals predicted by our LogFiBER transformer model. The five algorithms (as mentioned above) consider the sequential datasets C_A or C_P, a minimum support value and a minimum confidence value. Since we have a sparse dataset, it is observed that the first set of sequential rules was obtained for minimum support of 0.25 and a minimum confidence value 0.75. None of the five algorithms gave any output for our sparse dataset when the support value is higher than 0.25. We generated sequential rules with minimum support in the range [0.25, 0.01] in steps of −0.01. The minimum confidence interval was set to [0.75, 1.0] in steps of +0.01. When dataset C_A is mined, each algorithm generates 650 files, one for each combination of support and confidence values. Similarly, for dataset C_P, each algorithm generates 650 files.

Figures 8 and 9 show the running time and memory consumption (documented in the data sheets[7]) of different algorithms for generating Context-Context sequential rules for dataset C_A. The running time and memory consumption required for mining rules from both datasets C_A and C_P are almost similar, and hence, the surface plots for dataset C_P are not shown.

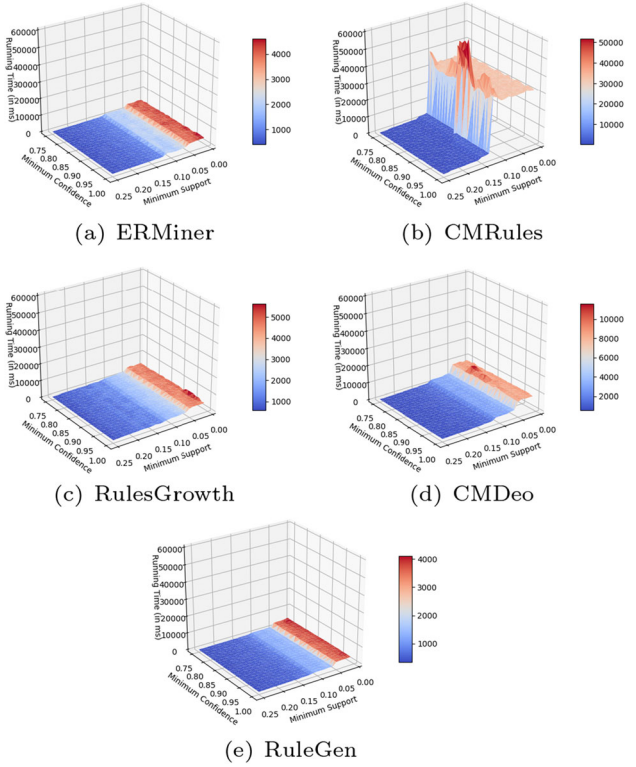
[6] https://github.com/ConEModel/SequentialRuleMiningDataSets.

[7] https://github.com/ConEModel/SequentialRuleMiningResults.



(a) ERMiner (b) CMRules (c) RulesGrowth (d) CMDeo (e) RuleGen

Fig. 9 Memory Consumption for Context-Context Rule Generation



(a) ERMiner (b) CMRules (c) RulesGrowth (d) CMDeo (e) RuleGen

Fig. 10 Running Time for Context-NFR conflict Rule Generation

Context-NFR conflict rule generation

Context-NFR conflict Correlations are created by mining rules using all 5 algorithms on datasets CN_A and CN_P. The rules mined for dataset CN_A (let us assume them to be RCN_A) represent the actual set of Context-NFR conflict sequential rules for our event log. The rules mined for dataset CN_P (let us assume them to be RCN_P) represent the Context-NFR conflict sequential rules mined based on the leaf-goals predicted by our LogFiBER transformer model. We have generated sequential rules with minimum support in the range [0.25, 0.01] in steps of −0.01. The minimum confidence interval was set to [0.75, 1.0] in steps of +0.01. When dataset CN_A is mined, each algorithm generates 650 files, one for each combination of support and confidence values. Similarly, for dataset CN_P, each algorithm again generates 650 files.

Figures 10 and 11 show the running time and memory consumption (documented in the data sheets[7]) of different algorithms for generating Context-NFR conflict sequential rules for dataset CN_A. The running time and memory consumption required for mining rules from both datasets CN_A and CN_P are similar, and hence, the surface plots for dataset CN_P are not shown. All Context-Context and Context-NFR conflict correlations mined by all the 5 algorithms are available in our GitHub repository[7].

6.6 Efficiency analysis

In this section, we discuss and compare the efficiency of two sequential rule mining algorithms—ERMiner and RuleGen. We have mentioned previously that ERMiner [32], CMRules [34], RuleGrowth [33] and CMDeo [34] belong to the same category of sequential rule mining algorithms. We have observed that both Context-Context and Context-NFR conflict correlations mined by these algorithms are exactly the same (results are available in our GitHub repository[7]). We also ran these four algorithms on the test datasets provided in the SPMF library [9] and found that they generate exactly the same set of sequential rules. In Figs. 8, 9, 10 and 11 we observe that ERMiner has the best time performance (this is also documented in the SPMF library [9]) and memory requirements are just second best to RulesGrowth. Based on this comparative performance analysis of ERMiner, CMRules, CMDeo and RulesGrowth, we select ERMiner as the representative of the four algorithms. We have measured the efficiency of the rules mined by ERMiner for both *Actual* and *Predicted* datasets. Since the same output is generated by the remaining three algorithms, the efficiency analysis yields the same result (refer to our GitHub repository[7]). We

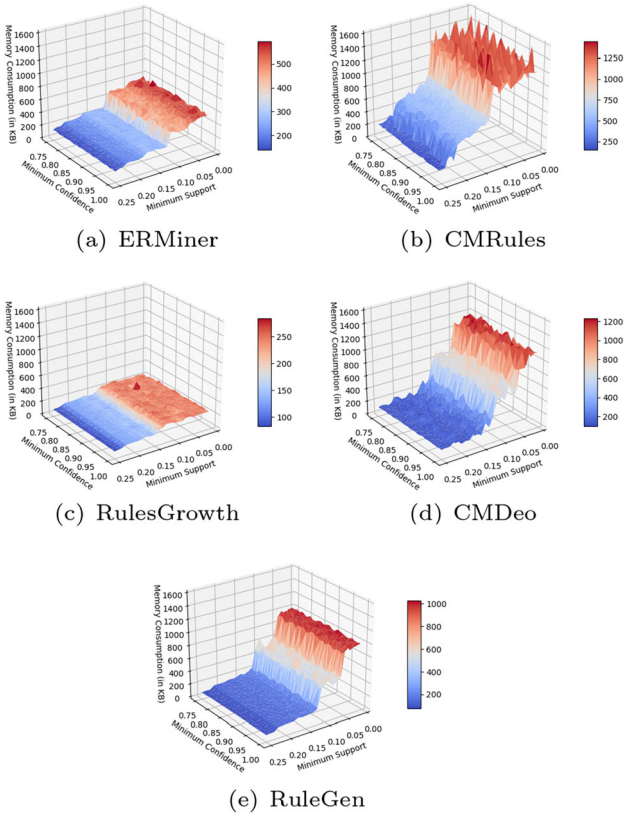


Fig. 11 Memory Consumption for Context-NFR conflict Rule Generation

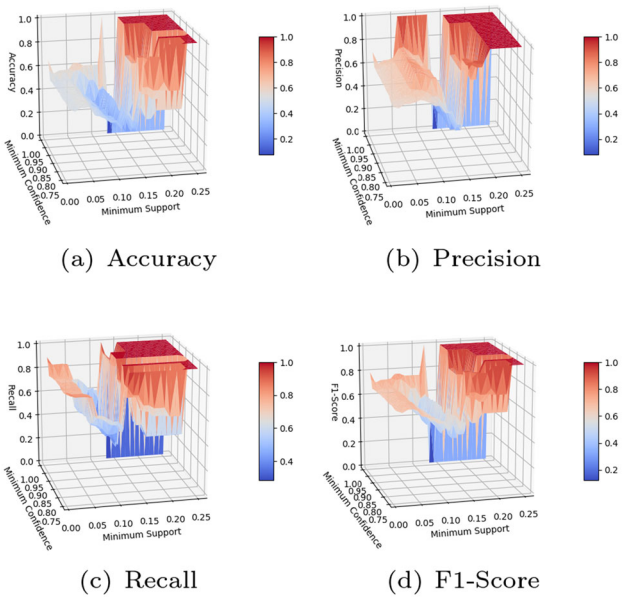


Fig. 12 Context-Context Rule Mining Efficiency for ERMiner

also measure the efficiency of the RuleGen algorithm as it represents a different category of sequential rule mining.

We have measured the accuracy, precision, recall and F1-score of the rules mined from the predicted datasets (C_P and CN_P). The rules mined from the actual datasets (C_A and CN_A) serve as the ground truth. We define the confusion matrix to measure these parameters, for the mined sequential rules as follows:

1. **True Positive(TP)**: All Context-Context correlations that are present in both RC_A and RC_P, are marked as true positive. Similarly, all Context-NFR conflict correlations that are present in both RCN_A and RCN_P, are marked as true positive.
2. **False Positive(FP)**: All Context-Context correlations that are present only in RC_P, but not in RC_A, are marked as false positive. Similarly, all Context-NFR conflict correlations that are present only in RCN_P, but not in RCN_A, are marked as false positive. These false positives occur due to the incorrect leaf goals identified in activity (2) of the proposed *ConE* framework.
3. **False Negative(FN)**: All Context-Context correlations that are present in RC_A, but absent in RC_P, are marked as false negatives. Similarly, all Context-NFR conflict correlations that are present in RCN_A, but absent in RCN_P, are marked as false negatives. Identifying incorrect leaf goals adds some invalid contexts or NFR conflicts, due to which some actual correlations are missed.
4. **True Negative(TN)**: True negative implies something that is not present in actual is not identified in the predictions as well. In our measurements, there is no scope of mined sequential rules to be incorrect or invalid. Hence, we have kept the true negatives as zero.

The values of these parameters, as obtained for the ERMiner and RuleGen algorithms, are available in our GitHub repository.[8] The accuracy, precision, recall and F1-score are computed using the parameters TP, FP, FN and TN as mentioned in the literature [53,54].

Efficiency of ERMiner:

Figure 12 shows how accuracy, precision, recall and F1-score vary for different support and confidence values when Context-Context correlations are mined by ERMiner. In this case, average recall is 75.02%, average precision is 72.75%, average F1-score is 70.73% and average accuracy is 59.23%. The low accuracy is due to the imbalanced dataset. Although our LogFiBER transformer model has precision and recall of 84.61% and 98.21%, respectively, there are incorrect leaf goal identifications that introduce invalid contexts against different threads. This results in invalid rules being mined while missing some of the actual rules.

Figure 13 shows how accuracy, precision, recall and F1-score vary for different support and confidence values when

[8] https://github.com/ConEModel/RuleComparison.

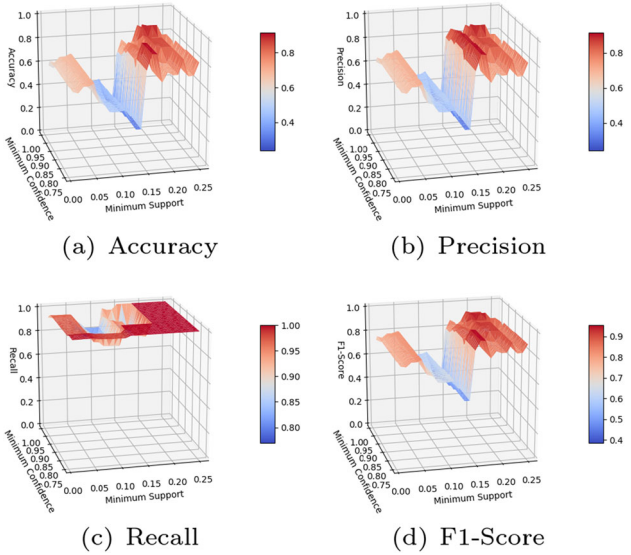


Fig. 13 Context-NFR conflict Rule Mining Efficiency for ERMiner

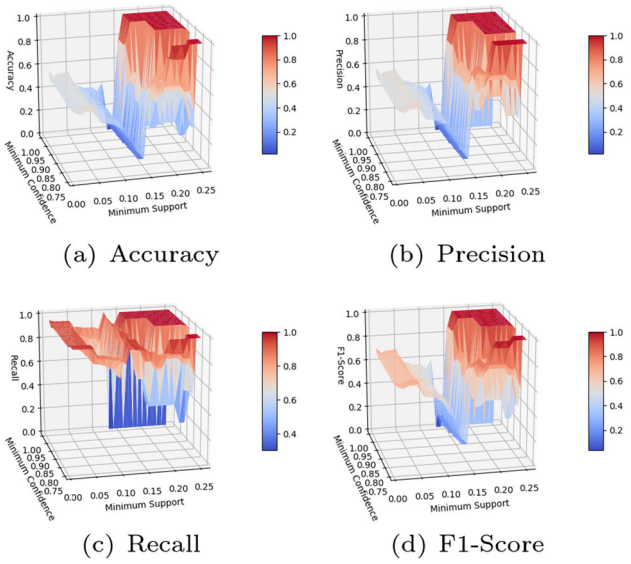


Fig. 14 Context-Context Rule Mining Efficiency for RuleGen

Context-NFR conflict correlations are mined by ERMiner. In this case, average recall is 95.15%, average precision is 65.12%, average F1-score is 76.03% and average accuracy is 63.29%. Here recall is high, but precision is comparatively low which implies that the number of false negatives is low, and false positives are high. This is because for each incorrect leaf goal prediction by our LogFiBER transformer model, both incorrect contexts and NFR conflicts are added. Hence, two different sets of errors are introduced, which results in a number of incorrect sequential rules being mined.

Efficiency of RuleGen:

Figure 14 shows how accuracy, precision, recall and F1-score vary for different support and confidence values when

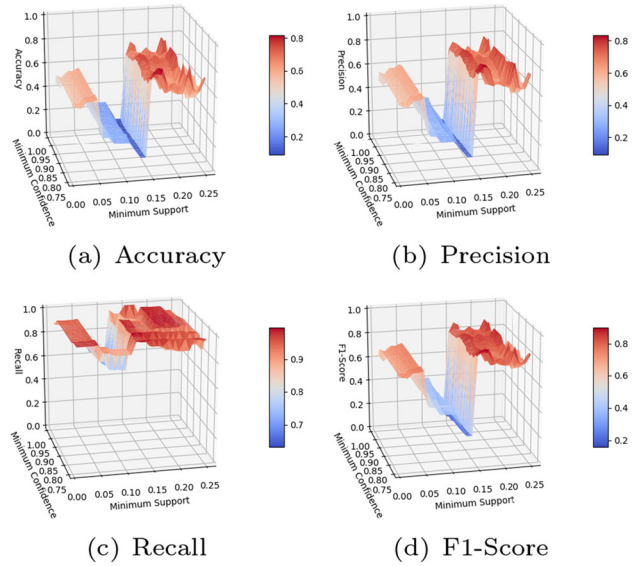


Fig. 15 Context-NFR conflict Rule Mining Efficiency for RuleGen

Context-Context correlations are mined by RuleGen. In this case, average recall is 84.4%, average precision is 58.4%, average F1-score is 65.2% and average accuracy is 53.7%. In RuleGen, the itemsets in the mined sequential rules are ordered. So for the same set of items (in this case contexts) multiple rules are mined with different orders with different support values. Thus, each invalid set of contexts is repeated multiple times in this way, increasing the number of false positives and resulting in low precision and accuracy.

Figure 15 shows how accuracy, precision, recall and F1-score vary for different support and confidence values when Context-NFR conflict correlations are mined by RuleGen. In this case, average recall is 89.5%, average precision is 51%, average F1-score is 62.3% and average accuracy is 48.4%. Here also for each incorrect leaf goal prediction by our LogFiBER transformer model, both incorrect contexts and incorrect NFR conflicts are added. For each set of invalid contexts, multiple incorrect correlations with different support values are mined, resulting in large false positives. This reduces the precision and accuracy.

6.7 Impact of data bias

We observe from Figs. 12, 13, 14 and 15 that the surface plots of accuracy, precision and F1-score encounter a sudden drop when the minimum support value reaches 0.14. This happens due to—(i) the biases in our datasets and (ii) incorrect leaf goal identifications. The biased sequences that are responsible for this sudden drop have a frequency of 9900 in our dataset. When the minimum support value is 0.15 (frequency of 10,500) and above, these incorrect rules are not mined. However, as soon as the support value becomes 0.14 (fre-

Table 5 Summary of experimental results

Average Values (in %)	Accuracy		Precision		Recall		F1-score	
Support Values	> 0.14	≤ 0.14	> 0.14	≤ 0.14	> 0.11	≤ 0.11	> 0.14	≤ 0.14
ERMiner (Context-Context)	79.89	43.00	90.27	58.98	84.84	62.53	85.36	59.24
ERMiner (Context-NFR)	78.23	51.56	79.92	53.51	97.28	92.45	87.59	66.95
RuleGen (Context-Context)	74.90	37.02	82.14	39.80	84.55	84.23	81.74	52.16
RuleGen (Context-NFR)	65.49	35.01	68.21	36.94	94.11	83.71	78.78	49.34

quency of 9800), these incorrect rules satisfy the threshold and gives rise to multiple false positives. This results in the sudden drop in accuracy, precision and F1-score. Decreasing the minimum support values further, the values gradually increase as there is an increase in true positives. Thus, for the remaining results we observe no sudden drop in values due to a large number of false positives.

The surface plots for recall, as observed in Figs. 12, 13, 14 and 15, also encounter a drop when the minimum support value reaches 0.11. The reason for this same as mentioned above. The biased sequences that are responsible for these drops have a frequency of 7,838 in our dataset. When the minimum support value is 0.12 (frequency of 8,400) and above, these incorrect rules are not mined. However, as soon as the support value becomes 0.11 (frequency of 7,700), these incorrect rules satisfy the threshold and gives rise to multiple false positives. This in turn reduces the number of true positives and increases the number of false negatives. Hence, we observe these drops in recall. Decreasing the minimum support values further, the recall value gradually increases as there is an increase in true positives.

6.8 Summary

In Table 5 we have provided a comprehensive summary of the average recall, precision, F1-score and accuracy of ERMiner and RuleGen algorithms in mining the two types of correlations. The measurements are based on the absolute count of the sequential rules generated and number of correct matches in the activity-goal mapping. Further we have rounded off these values to four significant digits.

In the previous section we have discussed how the presence of bias in the input dataset results in the drop of accuracy, precision, recall and F1-score at support values 0.14, 0.14, 0.11 and 0.14, respectively. In Table 5, for each parameter, we have shown two averages—for support values below and above the bias threshold, respectively. For example, in Table 5, we can observe that ERMiner (for Context-Context correlation) has a high precision of 90.27% for all support values greater than 0.14. However, its precision drops to 58.98% when support value is less than or equal to 0.14.

The ConE framework proves to be effective in terms of accuracy, precision, recall and F1-score for support values that are greater than the respective bias thresholds. The existence of this bias in the generated event logs has been explained in the previous section. In real life datasets, we do not expect such data bias to exist. Thus, we can conclude that we have been able to successfully address the research question mentioned in Sect. 5.1.

7 Threats to validity

The first possible threat to validity is related to the validity of the data used for experimental evaluation. Empirical research needs to be supported by some case study or experiments. This research work uses actual event logs for training and synthetically generated event logs for testing our models. There are publicly available event logs, but they are not supplemented with corresponding goal models. Also, creating a goal model for the entire system (represented by the real logs) is not feasible as the goal model becomes very complex and often non-deterministic. Hence for this purpose synthetic event logs are created by taking reference from actual event logs.

The second possible threat to validity is related to the research validity of this proposed model. The potential usage of this research is aimed toward better development of software systems whose behavior depends upon environmental contexts. Although our experimental evaluation yields meaningful correlations between contexts and NFR conflicts, the framework still needs to be tested within real-world environments to better validate its utility. However, the scope of this empirical research work is limited to proposing the framework and conducting the necessary supporting experiments. A real-life case study for this research comes within the scope of future work.

The third possible threat to validity is related to the goal model of the PHA system that is manually created. There is no deterministic way of identifying a single goal model specification of a system. For this research, we have reused the goal model specifications that we have developed for this case study in our recent previous works. Based on the

acceptance of these works in the requirements engineering community, we can possibly assume that our goal model representation does not pose a threat to the validity of the framework.

The fourth possible threat to validity is related to the availability of domain-specific event logs for training the language model. We have found multiple works in the literature where event logs are used for the simulation of different domains. These logs are publicly available, but we cannot claim the availability of event logs for every domain of applications. However, the event logs used for training are system generated and will not be very difficult to obtain for the target system in which we want to deploy the ConE framework.

The second threat regarding the validity of the proposed model can also be correlated to the threat associated with the acceptability of the framework by system engineers. The third threat may give rise to internal validity threats where the correlations being derived by our framework may get biased by the goal model specification. An additional threat is related to the specification of the threshold values that are very much dependent on the nature of the input data set, whether it is sparse or dense. The mined correlations might not be an accurate representation of the real-world environment that we are studying. External validation threats associated with the generalization of correlations mined in our case study to other real-world settings are subject to further empirical explorations and collection of evidence.

8 Comparative evaluation

In this section, we compare our contextual explainability approach with other explainability approaches from the literature (refer to Table 6). We have carefully selected works that have tried to address the issues of system behavior explainability. The details of the research works in Table 6 are briefly illustrated in Sect. 2. These works are focused on applications of AI approaches for system behavior explainability.

Table 6 shows that there are limited works that have addressed Context-NFR conflict correlations in their proposed approach. Limited attention is given to the operational contexts of the system and explored their correlations. These correlations form the basis for predicting future system behavior. Chazette et al. [14] have tried to correlate contexts and NFR but that is limited to only specific types of NFR and they have not explored conflicting scenarios. Predicting NFR conflicts and contexts correlation from systems operational behavior can help the system designer in adapting system behavior in certain contexts. Such system can alert the user of the conflicts that may arise in different contexts of system usage.

Also, none of them have used system event logs in deriving the correlations. ConE provides a novel approach that

Table 6 Qualitative comparison with existing works

Research Work	Context-Context Correlation	Context-NFR conflict Correlation	Experimental Evaluation	Usage of Event Logs	Research Domain
Chazette et al. [14]	Not addressed	Identified the link among explainability and NFRs like usability	Qualitative analysis by using open-coding approach	No	RE
Sadeghi et al. [16]	Not addressed	Not addressed	GitHub Repository has been provided with explainability cases	No	Software Systems
Zevenbergen et al. [25]	Not addressed	Not addressed	Not done	No	AI
Vultureanu-Albişi et al. [27]	User's context has been considered	Not addressed	Qualitative and Quantitative analysis have been carried out	No	AI
Beaudouin et al. [17]	User's context and operational context have been considered	Not addressed	Not done	No	AI
Chazette et al. [15]	User's context has been considered	Not addressed	Not done	No	RE
ConE	Environmental Context-Context correlations addressed	Context-NFR conflict correlations are addressed	Performed with synthetic data	Yes	RE and AI

identifies system behaviors from the event logs while correlating environmental contexts with other contexts and NFR conflicts.

9 Conclusion

There has been much research in the domain of explainable systems that aims to help end-users understand the changing behavior of a system under different contexts. However, the present state of the art does not explore the correlations between contexts and system NFRs in detail. The main contribution of this paper has been to provide contextual explainability of system behavior by tracing back events in the event log to their corresponding goals in the goal model. The novelty of the proposed *ConE* framework lies in the approach to provide explainable system behavior in terms of the environmental contexts, capturing information on how they activate other contexts, and on their correlation with NFR conflicts.

As part of our future work, we aim to extend this research work in different directions. One such exploration could be to observe whether the sequential rules being mined are actually being repeated in future execution threads of the system. There is also a need to assess the impact of the ConE model in different operational environments and application domains, in particular, related to IoT [55,56] and robotic systems [57,58]. Another interesting research direction could be to observe how the sequential rules generated by the ConE framework may be impacted when system design undergoes changes due to evolving requirements. Another future work is the external validation of the ConE framework in some real, industrial settings.

Acknowledgements The work was partially supported by SPIN-2021 Ressa_Rob, funded by Ca'Foscari University and by iNEST-Interconnected NordEst Innovation Ecosystem, funded by PNRR (Mission 4.2, Investment 1.5), NextGeneration EU—Project ID: ECS 00000043.

Appendix: I

This section briefly describes the functional and non-functional requirements of our use case of the Patient Healthcare Assistance (PHA) system.

Functional Requirements

1. A patient healthcare assistance system that is designed to monitor patient health remotely.
2. Such a system will have a front end that runs as an application on smartphones or desktops. At the back end, the application will forward the data to the edge node or cloud server for processing.
3. The cloud server will be responsible for maintaining patient health records.
4. The edge nodes will be lightweight and have limited processing capability.
5. The patient must register himself/herself to a healthcare application system.
6. The patient is expected to have wearable sensor devices in their body. There may be one or more wearable sensor devices in patients' bodies.
7. The wearable sensor device(s) will periodically record patient vitals.
8. The data recorded by the sensor will be forwarded to the application in the smartphone of the patient. It then forward the data for processing.
9. The data can be processed at edge node or at cloud server depending upon the type of analysis to be performed.
10. Cloud server should keep a record of all data processing results at all times for all patients registered in the system.
11. Based on the data processing of the vitals recorded if it finds some abnormality, then it must take an action.
12. Based on the severity of the condition and patient location different actions can be taken-
    (a) When the patient is located at home supported by some caregiver or family member, the system may raise an alarm. This requires an alarm system to be installed in the home where the patient resides.
    (b) When the patient is alone at home, then it may notify a doctor to visit the patient or send an ambulance.
    (c) When the patient is outside, it may send an ambulance and also may notify and family members.
13. The application will also provide the following types of assistance to be patient -

(a) Placing online medicine order:

(i) The patient must be logged in before placing an order.
(ii) After logging the patient can place an order for medicine by selecting the medicine and adding them to the cart.
(iii) The patient must upload a prescription for the medicines.
(iv) After uploading the prescription, payment has to be made.
(v) After payment the order will be successfully placed.
(vi) The cloud server will maintain a record of all the purchases made by the patients at all times.

(b) Booking a test online:

(i) The patient must be logged in before booking the test.
(ii) After logging the patient can select the tests required.
(iii) The patient must upload a prescription for the tests.
(iv) After uploading the prescription, payment has to be made.
(v) After payment the system must notify the patient whether the technician will be visiting or the patient has to go to the lab.
(vi) The cloud server will maintain a record of all the tests booked by the patients and test results after the test is conducted.

(c) Online Doctor Assistance -

(a) A patient must register himself/herself in the application to avail of online doctor assistance.
(b) The patient can upload his/her past medical history in the system.
(c) The patient can submit his/her symptoms in the system and book a doctor's appointment.
(d) The patient will have the provision to select the doctor from the application.
(e) The system must notify the patient of the appointment one-day in-prior.
(f) The doctor can provide an online prescription to the patient.

Non-functional Requirements

The following list of NFRs has been considered for our case study. For the purposes of brevity, there may not be formal quantification metrics associated with them.

1. *Efficiency*: The processing of a patient's vitals should be performed as fast as possible.
2. *Cost Efficiency*: The collection of vitals, processing and storage of results should be performed cost-effectively.
3. *Security*: The patient record must be transferred and stored in an encrypted format.
4. *Space Efficiency*: Processed results, patient history and other data should be stored in a compressed format for space efficiency.
5. *Response Time*: The system must respond quickly to any abnormality recorded by the sensor.
6. *Availability*: The system must ensure that patient with abnormal vitals must be attended by a caregiver or doctor.
7. *Quietness*: The system must respond to a patient abnormality in a way that it minimizes disturbances in the neighborhood.
8. *Authorization*: The system must authorize the patient before he/she can place a medicine order, book a test and book a doctor's appointment.
9. *Authentication*: The system must authenticate user details before giving access to the system.
10. *Confidentiality*: A patient must not access any other patient's medical record.
11. *Access Control*: An authorized caregiver will have limited access to the patients' records.
12. *Privacy*: The patients' medical records will only be accessible to the concerned doctor, involved in the treatment of that patient.
13. *Ease of Use*: The application interface must be user-friendly such that any age group must be comfortable to use it.
14. *Safety*: Online medicine orders will be delivered only after the pharmacy verifies the prescription uploaded by the patient.
15. *Privacy*: The system must ensure that the pharmacy maintains patients' privacy while handling their orders.
16. *Latency*: The medicine order should not be delivered any later than 2 days.

References

1. Boehm, B., Ross, R.: Theory-W software project management principles and examples. IEEE Trans. Softw. Eng. **15**(7), 902–916 (1989). https://doi.org/10.1109/32.29489
2. Roy, M., Deb, N., Cortesi, A., Chaki, R., Chaki, N.: NFR-aware prioritization of software requirements. Syst. Eng. **24**(3), 158–176 (2021). https://doi.org/10.1002/sys.21572
3. Roy, M., Deb, N., Cortesi, A., Chaki, R., Chaki, N.: Requirement-oriented risk management for incremental software development. Innov. Syst. Softw. Eng. **17**(3), 187–204 (2021). https://doi.org/10.1007/s11334-021-00406-6
4. Roy, M., Deb, N., Cortesi, A., Chaki, R., Chaki, N.: CARO: a conflict-aware requirement ordering tool for DevOps. In: Proceedings of the IEEE International Conference on Requirements Engineering, pp. 442–443 (2021). https://doi.org/10.1109/RE51729.2021.00061
5. Gupta, M., Abdelsalam, M., Khorsandroo, S., Mittal, S.: Security and privacy in smart farming: challenges and opportunities. IEEE

Access **8**, 34564–34584 (2020). https://doi.org/10.1109/ACCESS.2020.2975142
6. Samin, H.: Priority-awareness of non-functional requirements under uncertainty. In: 2020 IEEE 28th International Requirements Engineering Conference (RE), pp. 416–421 (2020). https://doi.org/10.1109/RE48521.2020.00061
7. Ali, R., Dalpiaz, F., Giorgini, P.: A goal-based framework for contextual requirements modeling and analysis. Requir. Eng. **15**, 439–458 (2010). https://doi.org/10.1007/s00766-010-0110-z
8. Botangen, K.A., Yu, J., Yongchareon, S., Yang, L.H., Bai, Q.: Specifying and reasoning about contextual preferences in the goal-oriented requirements modelling (2018). https://doi.org/10.1145/3167918.3167945
9. Fournier-Viger, P. et al.: The SPMF open-source data mining library version 2. In: Proceedings of the 19th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2016) Part III, Springer LNCS **9853**, pp. 36–40 (2016). https://www.philippe-fournier-viger.com/spmf/
10. Doran, D., Schulz, S., Besold, T.R.: What does explainable AI really mean? A new conceptualization of perspectives. arXiv:1710.00794 (2017)
11. Selvaraju, R.R., et al.: Grad-cam: visual explanations from deep networks via gradient-based localization. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 618–626 (2017). https://doi.org/10.1109/ICCV.2017.74
12. Tintarev, N., Masthoff, J.: Evaluating the effectiveness of explanations for recommender systems. User Model. User-Adapt. Interact. **22**(4–5), 399–439 (2012). https://doi.org/10.1007/s11257-011-9117-5
13. Kulesza, T. et al.: Too much, too little, or just right? Ways explanations impact end users' mental models. In: 2013 IEEE Symposium on Visual Languages and Human Centric Computing, pp. 3–10 (2013). https://doi.org/10.1109/VLHCC.2013.6645235
14. Chazette, L., Schneider, K.: Explainability as a non-functional requirement: challenges and recommendations. Requir. Eng. **25**(4), 493–514 (2020). https://doi.org/10.1007/s00766-020-00333-1
15. Chazette, L., Brunotte, W., Speith, T.: Exploring explainability: a definition, a model, and a knowledge catalogue. In: 2021 IEEE 29th International Requirements Engineering Conference (RE), pp. 197–208 (2021). https://doi.org/10.1109/RE51729.2021.00025
16. Sadeghi, M., Klös, V., Vogelsang, A.: Cases for explainable software systems: characteristics and examples. In: 2021 IEEE 29th International Requirements Engineering Conference Workshops (REW), pp. 181–187 (2021). https://doi.org/10.1109/REW53955.2021.00033
17. Beaudouin, V. et al.: Flexible and context-specific AI explainability: a multidisciplinary approach. Available at SSRN 3559477 (2020)
18. Koh, P.W., Liang, P.: Understanding black-box predictions via influence functions. In: International Conference on Machine Learning, pp. 1885–1894 (2017)
19. Ribeiro, M. T., Singh, S., Guestrin, C.: "Why should i trust you?" Explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1135–1144 (2016). https://doi.org/10.18653/v1/N16-3020
20. Hamel, L.: Visualization of support vector machines with unsupervised learning. In: 2006 IEEE Symposium on Computational Intelligence and Bioinformatics and Computational Biology, pp. 1–8 (2006). https://doi.org/10.1109/CIBCB.2006.330984
21. Jakulin, A., Možina, M., Demšar, J., Bratko, I. & Zupan, B.: Nomograms for visualizing support vector machines. In: Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, pp. 108–117 (2005). https://doi.org/10.1145/1081870.1081886
22. Možina, M., Demšar, J., Kattan, M., Zupan, B.: Nomograms for visualization of Naive Bayesian classifier. In: European Conference on Principles of Data Mining and Knowledge Discovery, pp. 337–348 (2004). https://doi.org/10.1007/978-3-540-30116-5_32
23. Kim, B., Glassman, E., Johnson, B., Shah, J.: iBCM: Interactive Bayesian case model empowering humans via intuitive interaction. CSAIL Technical Reports (July 1, 2003 - present) (2015). http://hdl.handle.net/1721.1/96315
24. Dam, H.K., Tran, T., Ghose, A.: Explainable software analytics. In: Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results, pp. 53–56 (2018). https://doi.org/10.1145/3183399.3183424
25. Zevenbergen, B., Woodruff, A., Kelley, P.G.: Explainability case studies. arXiv:2009.00246 (2020)
26. Konstan, J.A., Riedl, J.: Recommender systems: from algorithms to user experience. User Model. User-Adapt. Interact. **22**(1), 101–123 (2012). https://doi.org/10.1007/s11257-011-9112-x
27. Vultureanu-Albişi, A., Bădică, C.: Explainable collaborative filtering recommendations enriched with contextual information. In: 2021 25th International Conference on System Theory, Control and Computing (ICSTCC), pp. 701–706 (2021). https://doi.org/10.1109/ICSTCC52150.2021.9607106
28. Yu, E.S.-K.: Modelling Strategic Relationships for Process Reengineering. University of Toronto, Toronto (1996)
29. Mairiza, D., Zowghi, D., Nurmuliani, N.: Towards a catalogue of conflicts among non-functional requirements. In: ENASE—Proceedings of the 5th International Conference on Evaluation of Novel Approaches to Software Engineering, Athens, Greece, pp. 20–29 (2010)
30. Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering, vol. 5 (2000). https://doi.org/10.1007/978-1-4615-5269-7
31. Carvalho, R.M.: Dealing with conflicts between non-functional requirements of ubicomp and IoT applications. In: 25th IEEE International Requirements Engineering Conference, RE, Lisbon, Portugal, pp. 544–549 (2017). https://doi.org/10.1109/RE.2017.51
32. Fournier-Viger, P., et al.: Erminer: sequential rule mining using equivalence classes. In: Advances in Intelligent Data Analysis XIII, pp. 108–119 (2014). https://doi.org/10.1007/978-3-319-12571-8_10
33. Fournier-Viger, P., Wu, C.W., Tseng, V.S., Cao, L., Nkambou, R.: Mining partially-ordered sequential rules common to multiple sequences. IEEE Trans. Knowl. Data Eng. **27**(8), 2203–2216 (2015). https://doi.org/10.1109/TKDE.2015.2405509
34. Fournier-Viger, P., Faghihi, U., Nkambou, R., Nguifo, E.M.: CMRules: mining sequential rules common to several sequences. Knowl.-Based Syst. **25**(1), 63–76 (2012)
35. Zaki, M.J.: SPADE: an efficient algorithm for mining frequent sequences. Mach. Learn. **42**, 31–60 (2001). https://doi.org/10.1023/A:1007652502315
36. Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805 (2018)
37. Das, S., Deb, N., Cortesi, A., Chaki, N.: Sentence embedding models for similarity detection of software requirements. SN Comput. Sci. **2**(69), 1–11 (2021). https://doi.org/10.1007/s42979-020-00427-1
38. Alsentzer, E. et al.: Publicly available clinical BERT embeddings (2019). https://doi.org/10.48550/ARXIV.1904.03323
39. Lu, Y., Chen, Q., Poon, S.K.: A deep learning approach for repairing missing activity labels in event logs for process mining (2022). https://doi.org/10.48550/ARXIV.2202.10326
40. Liu, Y. et al.: RoBERTa: a robustly optimized BERT pretraining approach. arXiv:1907.11692 (2019)
41. Reimers, N., Gurevych, I.: Sentence-BERT: sentence embeddings using Siamese BERT-networks. arXiv:1908.10084 (2019)

42. Sitikhu, P., Pahi, K., Thapa, P., Shakya, S.: A comparison of semantic similarity methods for maximum human interpretability, vol. **1**, p. 1–4 (IEEE, 2019)
43. jUCMNav 7.0.0: University of Ottawa, ACC. September (2016) https://github.com/JUCMNAV
44. Amyot, D., Shamsaei, A.: Towards Advanced Goal Model Analysis with jUCMNav. Lecture Notes in Computer Science, vol. 7518, pp. 201–210. Springer, Berlin (2012)
45. Banerjee, S., Sarkar, A.: Ontology-driven approach towards domain-specific system design. Int. J. Metadata Semant. Ontol. **11**(1), 39–60 (2016). https://doi.org/10.1504/IJMSO.2016.078110
46. Banerjee, S., Sarkar, A.: Ontology driven conceptualization of context-dependent data streams and streaming databases. Comput. Inf. Syst. Ind. Manag., 240–252 (2017)
47. Banerjee, S., Sarkar, A.: Ontology driven meta-modelling of clinical documents. Int.J. Healthc. Technol. Manag. **16**, 271 (2017). https://doi.org/10.1504/IJHTM.2017.088869
48. Rasmy, L., Xiang, Y., Xie, Z., Tao, C., Zhi, D.: Med-BERT: pretrained contextualized embeddings on large-scale structured electronic health records for disease prediction. NPJ Digit. Med. **4**(1), 1–13 (2021)
49. Johnson, A.E., et al.: MIMIC-III, a freely accessible critical care database. Sci. Data **3**(1), 1–9 (2016)
50. Sanh, V., Debut, L., Chaumond, J., Wolf, T. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv:1910.01108 (2019)
51. Das, S., Deb, N., Cortesi, A., Chaki, N.: Sentence embedding models for similarity detection of software requirements. SN Comput. Sci. **2**(2), 1–11 (2021). https://doi.org/10.1007/s42979-020-00427-1
52. Pika, A., et al.: Privacy-preserving process mining in healthcare. Int. J. Environ. Res. Public Health **17**, 1612 (2020). https://doi.org/10.3390/ijerph17051612
53. Goutte, C., Gaussier, E.: A probabilistic interpretation of precision, recall and F-score, with implication for evaluation. Lect. Notes Comput. Sci. **3408**, 345–359 (2005). https://doi.org/10.1007/978-3-540-31865-1_25
54. Dalianis, H. (2018) Evaluation metrics and evaluation. In: Clinical Text Mining: Secondary Use of Electronic Patient Records. pp. 45–53. https://doi.org/10.1007/978-3-319-78503-5_6
55. Ferrara, P., Mandal, A.K., Cortesi, A., Spoto, F.: Static analysis for discovering IoT vulnerabilities. Int. J. Softw. Tools Technol. Transf. **23**(1), 71–88 (2021). https://doi.org/10.1007/s10009-020-00592-x
56. Mandal, A.K., Panarotto, F., Cortesi, A., Ferrara, P., Spoto, F.: Static analysis of android auto infotainment and on-board diagnostics II apps. Softw. Pract. Exp. **49**(7), 1131–1161 (2019). https://doi.org/10.1002/spe.2698
57. White, R., Christensen, H.I., Caiazza, G., Cortesi, A.: Procedurally provisioned access control for robotic systems. IEEE Int. Conf. Intell. Robots Syst. (2018). https://doi.org/10.1109/IROS.2018.859446
58. Dieber, B., et al.: Penetration testing ROS. Stud. Comput. Intell. **831**, 183–225 (2020). https://doi.org/10.1007/978-3-030-20190-6_8

Mandira Roy is presently a Research Scholar, in Computer Science and Engineering Department of University of Calcutta, India. She has completed Masters' in Computer Science and Engineering from the University of Calcutta, in the year 2020. She is pursuing her research studies in the domain of requirements engineering and software engineering. She has published in both journals and conferences. One of which was published as a research article in ACSS International Symposium (2021), where the same has been awarded as the best paper. Consequently, an extended version of the said article was published in the Springer Innovations in Systems and Software Engineering.

Souvick Das is a Research Associate at the Department of Environmental Science, Informatics, and Statistics, of Ca' Foscari University, Venice, Italy and is currently pursuing Ph.D. in Computer Science and Engineering from University of Calcutta, India. He has received B.Sc and M.Sc in Computer Science from West Bengal State University, India in the year 2012 and 2014 respectively. He has qualified UGC NET-JRF in the year of 2016.

Novarun Deb is an Assistant Professor at the Department of Computer Science and Engineering in the Indian Institute of Information Technology (IIIT) Vadodara. He has worked as a research associate as part of his post-doctoral studies at the Department of Informatics and Statistics, Ca' Foscari University in Venice, Italy. He was awarded Ph.D. from the Department of Computer Science and Engineering, University of Calcutta, India. His active research interests are in the domain of Requirements Engineering and Software Engineering, and the application of Natural language Processing and Artificial Intelligence in these domains.

Agostino Cortesi is a full professor of computer science at Ca' Foscari University, Venice, Italy. He has extensive experience in the area of static analysis and software verification techniques, with particular emphasis on security applications. He published more than 150 papers in high level international journals (including ACM TOPLAS, IEEE TSE, SCP, TCS, IEEE RAL, ESWA etc.) and proceedings of international conferences (including ACM POPL, ACM PLDI, IEEE LICS, ICALP, VMCAI, SAS, ACM SAC etc.). His current h-index is 21 according to Scopus, and 30 according to Google Scholar. Currently, he serves as co-Editor in Chief of the book series "Services and Business Process Reengineering" published by Springer-Nature.

Rituparna Chaki a full professor in the A K Choudhury School of IT, University of Calcutta, India, since 2013. She is actively involved in the research related to the domain of wireless networking for over last twelve years. Her field of research encompasses optical network topology to adhoc network routing, wds for WSN. She is also actively involved in the promotion of Internet of Things and has recently authored a book on IoT security by CRC press. She is an active member of ACM India and is currently chairing the ACM-Kolkata professional Chapter. Besides wireless networking and IoT, she is also involved in research related to Systems Software, Software security and testing. She has well over 150 international publications to her credit. Prof. Chaki has delivered invited lecture in different Universities, and Conferences in India and abroad. She is also a Visiting Professor in the AGH University of Science & Technology, Poland since October 2013.

Nabendu Chaki is a Professor and Head in the Department of Computer Science & Engineering, University of Calcutta, Kolkata, India. He is sharing the responsibility of the Series Editor for the Springer Nature book series on Services and Business Process Reengineering jointly with Professor Agostino Cortesi of Venice, Italy. Besides editing more than 50 conference proceedings with Springer, Dr. Chaki has authored 8 texts and research books with reputed publishers including CRC Press, Springer Nature, etc. He has published more than 250 Scopus Indexed research articles in Journals and Conferences (h-index is 16 in SCOPUS base). Nabendu has served as a Visiting Professor in different places including US Naval Postgraduate School, in California, and in different Universities in Europe. He has been the founder Chair of ACM Professional Chapter in Kolkata and served in that capacity for 3 years during 2014–2017.