# Detecting Context-Aware Deviations in Process Executions

Gyunam Park, Janik-Vasily Benzin, and Wil M. P. van der Aalst

Process and Data Science Group (PADS), RWTH Aachen University, Germany
gnpark@pads.rwth-aachen.de, janik.benzin@rwth-aachen.de,
wvdaalst@pads.rwth-aachen.de

**Abstract.** A deviation detection aims to detect deviating process instances, e.g., patients in the healthcare process and products in the manufacturing process. A business process of an organization is executed in various contextual situations, e.g., a COVID-19 pandemic in the case of hospitals and a lack of semiconductor chip shortage in the case of automobile companies. Thus, *context-aware deviation detection* is essential to provide relevant insights. However, existing work 1) does not provide a systematic way of incorporating various contexts, 2) is tailored to a specific approach without using an extensive pool of existing deviation detection techniques, and 3) does not distinguish *positive* and *negative* contexts that justify and refute deviation, respectively. In this work, we provide a framework to bridge the aforementioned gaps. We have implemented the proposed framework as a web service that can be extended to various contexts and deviation detection methods. We have evaluated the effectiveness of the proposed framework by conducting experiments using 255 different contextual scenarios.

**Keywords:** Context-aware deviation detection · Context · Deviation detection · Process mining

## 1 Introduction

Deviation detection in process executions aims to identify anomalous executions by distinguishing deviating behaviors from normal behaviors. A range of deviation detection techniques for business processes has been proposed [4]. The techniques are categorized as supervised and unsupervised ones. The former defines normal behavior to identify deviations of recorded process executions with respect to the specified normal behavior, whereas the latter identifies deviations without such normal behaviors. Since many businesses lack the specification of normal behavior, unsupervised deviation detection techniques recently gained more attention [4].

As a process is executed in a specific *context* (e.g., COVID-19 Pandemic) that affects the behavior of the execution, it is indispensable to consider the context when detecting deviations [2]. In this regard, *context-aware deviation detection* aims to classify a trace (i.e., a sequence of events by a process instance) to ① *context-insensitive normal* meaning the trace is normal regardless of context, ②

*context-insensitive deviating* meaning the trace is deviating regardless of context, ③ *context-sensitive normal* meaning the trace is deviating without considering context but normal when considering context, and ④ *context-sensitive deviating* meaning the trace is normal without considering context but deviating when considering context.

Few approaches have been developed to (indirectly) solve the context-aware deviation detection problem [4]. For instance, Pauwels et al. [15] extend Bayesian networks to learn conditional probabilities for organizational contexts such as roles of resources. Warrender et al. [17] propose a sliding-window based approach that considers time-related context. Mannhardt et al. [12] conceptualize context as data attributes of process instances.

However, each approach is tailored to consider limited aspects of contexts, not providing a systematic way to extend the approach to consider various aspects of contexts. Given a large space of possibly relevant contexts proposed in studies on contexts (cf. Subsect. 2.2), we need a systematic framework to integrate context to deviation detection.

Moreover, a framework to integrate a large number of existing deviation detection methods with different strengths and weaknesses on varying assumptions is missing. Instead, the existing work is confined to a single method and inherits the methods' unique set of properties.

Furthermore, existing techniques do not distinguish *positive* and *negative* contexts. The former justifies deviations. For instance, COVID-19 Pandemic in a healthcare process explains the long waiting time for admission, e.g., due to the sudden increase in the number of patients. The latter refutes non-deviations. "Crunch time" in a video game industry denies a normal throughput time of the game development process, e.g., with the compulsory overwork by employees. Existing work considers only negative contexts when integrating context into deviation detection.

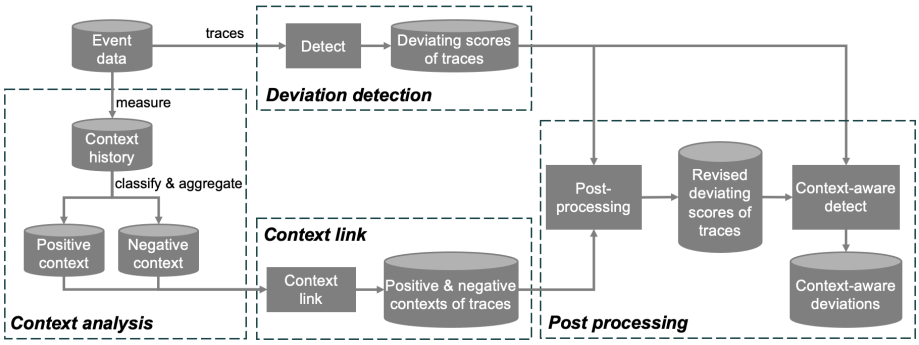



**Fig. 1:** An overview of the framework for context-aware deviation detection

In this paper, we propose a framework based on *post-processing mechanism* to systematically support the context-aware deviation by integrating the extensive existing deviation detection methods and contexts. As shown in Fig. 1, the framework consists of four components. First, *deviation detection* computes deviating

scores of traces, with which we can classify *non-context deviating* and *non-context normal* traces. Next, *context analysis* computes *positive* and *negative* contexts by aggregating *context history*. Afterwards, *context link* connects the context to traces. Next, *post-processing* increases the deviation score of a trace with the positive context of the trace and decreases it with the negative context. Using the revised deviation score, we classify traces as *context-normal* and *context-deviating*. Finally, we label a trace as one of ①-④ based on the *non-context* and *context* classifications.

To summarize, this paper provides the following contributions:

- We propose a framework to solve the context-aware deviation detection problem while integrating the existing deviation detection methods and contexts.
- We extend the context conceptualization with *positive* and *negative* contexts that carry dedicated semantics for deviation detection.
- We implement a flexible and scalable web service supporting the framework and evaluate the effectiveness of the framework with 225 simulated scenarios.

The remainder is organized as follows. We discuss the related work in Sect. 2. Then, we present the preliminaries in Sect. 3. Next, we introduce the context-awareness in Sect. 4 and a framework for integrating contexts and deviation detection in Sect. 5. Afterward, Sect. 6 introduces the implementation of a web application, and Sect. 7 evaluates the effectiveness of the proposed framework. Finally, Sect. 8 concludes the paper.

## 2 Related Work

In this section, we introduce existing literature on unsupervised deviation detection of process executions and the context of business processes.

### 2.1 Unsupervised Deviation Detection

Unsupervised deviation detection is categorized into 1) process-centric, 2) profile-based, 3) process-agnostic and interpretable, and 4) process-agnostic and non-interpretable methods.

**Process-centric.** [3] computes the conformance of traces to a process model and classifies non-conforming traces as deviating. [5] refines the concept of likelihood graphs by mining small likelihood graph signatures from event data. A deviation is determined by comparing the execution likelihood of a trace with respect to a set of mined signatures and a reference likelihood. [8] discovers process models using genetic algorithms and conducts conformance checking using token-based replay to detect deviating traces.

**Profile-based.** [11] iteratively samples more normal sets of traces and *profiles* each trace against the more normal set of traces. The result is a sorted list of traces according to their profiles in the last iteration, which is used to partition the event data into a set of normal traces and a set of deviating traces using a deviation threshold.

**Process-agnostic and interpretable.** [15] extends Bayesian networks and defines a conditional likelihood-based score using the extended bayesian network on traces. All traces are then sorted according to the score, and the first $k$ are returned as deviating traces. [6] uses *association rules*. A set of anomaly detection association rules specifying normal behavior is mined from the event data. A trace is detected as deviating if its aggregate support is below the aggregate support of its most similar trace in the event data with respect to the set of anomaly detection association rules. [17] uses a sliding window-based approach to extract frequency information over those windows. If a trace contains infrequent windows, then it is deviating.

**Process-agnostic and non-interpretable.** [13] encodes traces in event data using one-hot encoding and train *autoencoder neural network* with them. The deviation of a trace is determined using the error the autoencoder makes in predicting the trace. [14] further develops the application of neural networks to event data by training a recurrent neural network to predict the next event in integer-encoding based on the current event in a trace. The aggregate likelihood of predicting the correct events is used to detect deviations.

Some of the unsupervised deviation detection methods provide room for handling limited kinds of context but take method-dependent approaches such that neither a general integration nor support for a systematic extension of context is provided. In this work, we provide a general framework to integrate various unsupervised deviation detection techniques with different strengths, weaknesses, and assumptions to systematically extend them with contexts.

### 2.2 Context

In pervasive computing, especially for developing adaptive services, context is conceptualized as the lower level of the abstraction of raw data [18]. Another higher-level abstraction, called situation, is introduced to map one or multiple contexts to semantically richer concepts such as users' behaviors.

In business processes, a context is a multitude of concepts that affect the behavior and performance of the process. [2] derives four levels of context that should be considered during the analysis of processes to improve the quality of results. [16] extends it and provides an ontology of contexts in BPM by conducting an extensive literature review of the context in BPM.

More ontological approaches have been proposed to specify context and situations. Generally, they categorize contexts into *intrinsic* and *relational*. [7] differentiate between *intrinsic* and *relational* context whereby intrinsic context is essential to the nature of the entity and relational context is inherent to the relation of multiple entities. [9] develops a two-level framework for structuring context, which is more coarse-grained than the four levels of [2].

In this work, we merge relevant contexts of the earlier work and their categorizations into an integrative context ontology that is aimed at extracting context from event data.

## 3 Preliminaries

**Definition 1 (Event).** *Let $\mathbb{U}_e$ be the universe of events, Let $\mathbb{U}_{att}$=$\{act, case, time, \dots\}$ be the universe of attribute names. For any $e \in \mathbb{U}_e$ and $att \in \mathbb{U}_{att}$ : $\#_{att}(e)$ is the value of attribute att for event e, e.g., $\#_{time}(e)$ indicates the timestamp of event e.*

**Definition 2 (Trace).** *A trace is a finite sequence of events $\sigma \in \mathbb{U}_e^*$ such that each event appears only once, i.e., $\forall_{1 \leq i < j \leq |\sigma|}\, \sigma(i) \neq \sigma(j)$. Given $\sigma \in \mathbb{U}_e^*$ and $e \in \mathbb{U}_e$, we write $e \in \sigma$ if and only if $\exists_{1 \leq i \leq |\sigma|}\, \sigma(i)$=e. We define $elem \in \mathbb{U}_e^* \rightarrow \mathcal{P}(\mathbb{U}_e)$ with $elem(\sigma)$=$\{e \in \sigma\}$.*

**Definition 3 (Event Log).** *An event log is a set of traces $L \subseteq \mathbb{U}_e^*$ such that each event appears at most once in the event log, i.e., for any $\sigma_1, \sigma_2 \in L$ such that $\sigma_1 \neq \sigma_2 : elem(\sigma_1) \cap elem(\sigma_2)$=$\emptyset$. Given $L \subseteq \mathbb{U}_e^*$, we denote $E(L)$=$\bigcup_{\sigma \in L} elem(\sigma)$.*

**Definition 4 (Time Window).** *Let $\mathbb{U}_{time}$ be the universe of timestamps. $\mathbb{U}_{tw} = \{(t_s, t_e \in \mathbb{U}_{time} \times \mathbb{U}_{time} \mid t_s \leq t_e\}$ is the set of all possible time windows. $duration \in \mathbb{U}_{tw} \rightarrow \mathbb{R}$ maps a time window to a real valued representation of the difference between the its start and end in the granularity of seconds.*

For $tw$=$(t_s, t_e)$, $\pi_s(tw)$=$t_s$ and $\pi_e(tw)$=$t_e$. For instance, $tw_1$=(2022-01-01 00:00:00, 2022-01-08 00:00:00) is a time window where $\pi_s(tw_1)$=2022-01-01 00:00:00, $\pi_e(tw_1)$=2022-01-08 00:00:00, and $duration(tw_1)$=604800 (seconds). Note that, in the remainder, we denote 604800 as *week*.

A time span of an event log with length $l$ is a collection of non-overlapping time windows of the event log that have the equal duration of $l$.

**Definition 5 (Time Span).** *Let $l \in \mathbb{R}$ be a time span length. Let $L \in \mathbb{U}_e^*$ be an event log. $t_{min}(L) = min_{e \in E(L)} \#_{time}(e)$, $t_{max}(L) = max_{e \in E(L)} \#_{time}(e)$, and $n_l(L) = \lceil (t_{max}(L) - t_{min(L)})/l \rceil$. $span_l(L) = \{(t_{min}(L) + (k-1) \cdot l, t_{min}(L) + k \cdot l) \mid 1 \leq k \leq n_l(L)\}$. For any $e \in E(L)$, $tw_{l,L}(e) = tw$ s.t. $\pi_s(tw) \leq \#_{time}(e) \leq \pi_c(tw)$.*

Assume that event log $L$ contains traces that consist of events between 2022-01-01 00:00:00 and 2022-01-15 00:00:00. $t_{min}(L)$ = 2022-01-01 00:00:00, $t_{max}(L)$ = 2022-01-15 00:00:00, and $n_{week}(L) = 2$. $span_{week}(L)$ contains two time windows $tw_1$ = (2022-01-01 00:00:00, 2022-01-08 00:00:00) and $tw_2$=(2022-01-08 00:00:00, 2022-01-15 00:00:00).

## 4 Context-Aware Deviation Detection

In this section, we introduce a context-aware deviation detection problem and explain an ontology of contexts for context-aware deviation detection.

### 4.1 Context-Aware Deviation Detection Problem

First, a deviation detection problem is to compute a function that labels traces either with label *deviating* or with label *normal*. All known deviation detection methods implicitly or explicitly use some form of scoring of traces *score* that is a mapping of traces to some real number (cf. Subsect. 2.1). A threshold $\tau$ is used to decide the label. We conceptualize deviating traces as traces scored above $\tau$.

**Definition 6 (Deviation Detection).** *Let $L$ be an event log. Let $\mathbb{S}$=[0, 1] be a range of all possible score values and $\tau \in \mathbb{S}$ be a threshold value. A score function score $\in L \rightarrow \mathbb{S}$ maps traces to score values. $detect_{score} \in L \rightarrow \{d, n\}$ is a deviation detection using score such that, for any $\sigma \in L$, $detect_{score}(\sigma) = d$ if $score(\sigma) > \tau$. $detect_{score}(\sigma) = n$ otherwise.*

Instead of the two-class labeling problem, a context-aware deviation detection problem is a four-class labeling problem. Table 1 describes the four classes with two dimensions: *non-context* and *context*. The non-context deviating ($d$) and normal ($n$) correspond to the two classes of the deviation detection problem, whereas context-deviating ($d_c$) and context-normal ($n_c$) indicate that a trace is deviating and normal, respectively, when considering context. First, *context-insensitive deviating* (i.e., $d{\Rightarrow}d_c$) indicates that a trace is both non-context deviating and context-deviating. Second, *context-sensitive deviating* (i.e., $n{\Rightarrow}d_c$) denotes that a trace is non-context normal, but context-deviating. Third, *context-sensitive normal* (i.e., $d{\Rightarrow}n_c$) indicates that a trace is non-context deviating, but context-normal. Finally, *context-insensitive normal* (i.e., $n{\Rightarrow}n_c$) denotes that a trace is both non-context normal and context-normal.

**Table 1:** Four classes in a context-aware deviation detection problem

| $\sigma \in \mathbb{U}_e^*$ | | **Context** | |
|---|---|---|---|
| | | Deviating ($d_c$) | Normal ($n_c$) |
| **Non-context** | Deviating ($d$) | Context-insensitive deviating ($d{\Rightarrow}d_c$) | Context-sensitive normal ($d{\Rightarrow}n_c$) |
| | Normal ($n$) | Context-sensitive deviating ($n{\Rightarrow}d_c$) | Context-insensitive normal ($n{\Rightarrow}n_c$) |

**Definition 7 (Context-Aware Deviation Detection Problem).** *Given $L \subseteq \mathbb{U}_e^*$, compute a function that labels traces with context-insensitive deviating, context-sensitive deviating, context-sensitive normal, or context-insensitive normal, i.e., c-detect $\in L \rightarrow \{d{\Rightarrow}d_c, n{\Rightarrow}d_c, d{\Rightarrow}n_c, n{\Rightarrow}n_c\}$.*

### 4.2 Context-Awareness

Based on existing work on contexts of business processes introduced in Subsect. 2.2, we provide context ontology for context-aware deviation detection in Fig. 2. First, *intrinsic* context is inherent to an event. The intrinsic contexts *resource* and *data* correspond to the organizational and data perspectives for a

single event. The *waiting time* context represents the average waiting time of an event. Thus, the information of *waiting time* contexts can be used to capture unusually long delays for events.

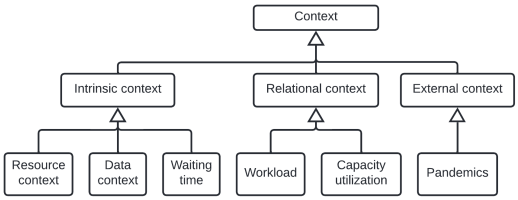



**Fig. 2:** An ontology of business process context for deviation detection [2] [16] [7] [9].

Next, *relational* context is inherent to the relation of multiple events. The relational contexts *workload*, *waiting time* and *capacity utilization* represent context information that is measured (extracted) by relating multiple events of the data. The *workload* context represents event counts of various selections for a given time window. The *capacity utilization* context represents workloads of resources or locations of events by counting the respective events that were recorded during the time window of the context, e.g., the capacity utilization of a finance department. Therefore, the information of *capacity utilization* contexts can be used to capture unusually high workloads of resources.

Finally, *external* context is not directly attributable to events, but still affects them. The external context *pandemics* represents the outbreak of infectious disease, e.g., COVID-19 pandemic. As an external context is not directly measurable on event data, either additional data has to be used, or it has to be represented by another measurable relational context caused by the external context, e.g., a hygienic products shop experiences exceptionally large demand during the first worldwide outbreak of Corona pandemic such that the *workload* context captures the unusual demand increase and, thus, the external context *pandemic*.

## 5 Framework for Context-Aware Deviation Detection

This section introduces a framework based on *post-processing mechanism*. We explain each of the four components described in Fig. 1 with a running example: 1) deviation detection, 2) context analysis, 3) context link, and 4) post processing.

### 5.1 Running Example

Fig. 3 shows a running example of an order management process. It describes events of the process for two weeks under 1) the context of high workload (i.e., many events during the week) in *week 1* and 2) the context of overwork (i.e., many events during the weekend) in *week 2*. The context of high workload is considered as a positive context, i.e., the context justifies deviating traces in *week 1*, producing more context-normal traces. In contrast, we consider the context of

overwork as a negative context, i.e., the context refutes normal traces in *week 2*, producing more context-deviating traces in *week 2*.

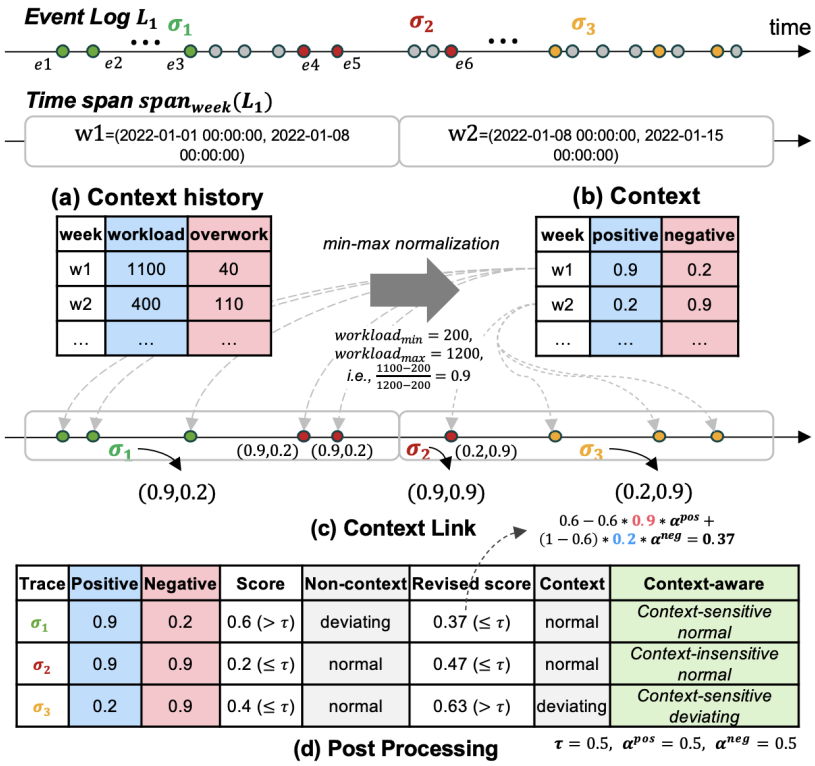



**Fig. 3:** A running example of context-aware deviation detection for the time window *week 1 (w1)* and *week 2 (w2)*. **(a)** The context history of $L_1$ in *w1* shows *workload* of 1100 (total number of events in *w1*) and *overwork* of 40 (total number of events during weekend in *w1*), respectively. **(b)** Assume *workload* is a positive measure, $workload_{max}$=1200, and $workload_{min}$=200. By aggregating positive (blue) and negative (red) measures in *w1* with *min-max normalization*, we compute the context in *w1*, i.e., positive context of 0.9 and negative context of 0.2. **(c)** We first connect the context to events (as denoted by gray dotted lines) and then connect the context to a trace by computing the maximum positive and negative contexts of its events. $\sigma_2$ has the positive context of 0.9 (i.e., the maximum positive context of its events) and the negative context of 0.9 (i.e., the maximum negative context). **(d)** The non-context deviating score of $\sigma_1$ is 0.6 ($> \tau$, i.e., non-context deviating), but its revised deviation score is 0.37 ($\leq \tau$, i.e., context-normal). Thus, $\sigma_1$ is context-sensitive normal.

### 5.2 Context Analysis

We analyze context in two steps. First, we compute *context history* based on event logs. A *context history* describes the value of different measures (e.g., *workload* and *overwork*) in different time windows.

**Definition 8 (Context History).** *Let* $\mathbb{U}_{measure}$=$\{workload, overwork, \dots\}$ *be the universe of measure names.* $\mathbb{U}_{ch} = \mathbb{U}_{tw} \nrightarrow (\mathbb{U}_{measure} \nrightarrow \mathbb{R})$ *is the universe of context history. Let* $L$ *be an event log and* $l \in \mathbb{R}$ *a time span length.* $ch_l(L) \in \mathbb{U}_{ch}$ *is the context history in* $L$ *with time span of* $l$*.*

Fig. 3(a) shows the context history of $L_1$ with time span length *week*, i.e., $ch_{week}(L_1)$. It contains the measures of *workload* and *overwork*. For instance, $ch_{week}(L_1)(w1)(workload) = 1100$ and $ch_{week}(L_1)(w1)(overwork) = 40$.

A context consists of *positive* and *negative* context scores. They describe the overall positive/negative contexts in a time window with a value ranging from 0 to 1, respectively. The closer the value is to 1, the stronger the respective context is. We compute the context in a time window using context measures in the context history of the time window. To this end, we 1) normalize context measures in the time window, 2) distinguish positive and negative context measures, 3) aggregate positive and negative context measures with different weights (i.e., the importance of measures).

**Definition 9 (Context).** *Let* $L$ *be an event log and* $l \in \mathbb{R}$ *a time span length.* $type \in \mathbb{U}_{measure} \rightarrow \{pos, neg\}$ *maps measures to pos and neg,* $w \in \mathbb{U}_{measure} \rightarrow \mathbb{R}$ *maps measures to weights, and* $norm \in \mathbb{U}_{measure} \rightarrow (\mathbb{R} \rightarrow [0,1])$ *maps measures to normalization functions that assign values ranging from 0 to 1 to measure values.* $ctx_{l,L} \in span_l(L) \nrightarrow [0,1]^2$ *is a context such that, for any* $tw \in dom(ctx_{l,L})$*,* $ctx_{l,L}(tw) = (pc, nc)$ *with*

- $pc = \sum_{m \in dom(ch^{tw}_{l,L}) \wedge type(m)=pos} {}^{w(m) \cdot norm(m)(ch^{tw}_{l,L}(m))}/_{w(m)}$ *and*
- $nc = \sum_{m \in dom(ch^{tw}_{l,L}) \wedge type(m)=neg} {}^{w(m) \cdot norm(m)(ch^{tw}_{l,L}(m))}/_{w(m)}$

*, where* $ch^{tw}_{l,L} = ch_l(L)(tw)$*.*

The example in Fig. 3 assumes $norm_1$, $type_1$, and $w_1$. First, $norm_1$ uses *min-max normalization* for each measure, e.g., with the maximum *workload* of 1200, the minimum *workload* of 200, the maximum *overwork* of 120, and the minimum *overwork* of 20. Moreover, $type_1$ classifies *workload* as a positive context measure and *overwork* as a negative context measure, i.e., $type_1(workload)$=*pos* and $type_1(overwork)$=*neg*. Finally, $w_1$ assigns the weights of 10 and 5 to *workload* and *overwork*, respectively, i.e., $w_1(workload)$=10 and $w_1(overwork)$=5.

Fig. 3(b) shows context $ctx_{week,L_1}$. The positive context in time window *w1* is ${}^{w_1(workload) \cdot norm_1(workload)(1200)}/_{w_1(workload)}$=${}^{10 \cdot 0.9}/_{10}$=0.9. The negative context in *w1* is ${}^{w_1(overwork) \cdot norm_1(overwork)(200)}/_{w_1(overwork)}$=${}^{5 \cdot 0.2}/_{5}$=0.2. Note that, in the example, the weight does not play its role since we only use one positive and one negative context measure.

### 5.3 Linking Context to Traces

To connect context to traces, we first link context to events. An event is connected to the context of the time window that the event belongs to.

**Definition 10 (Context-Event Link).** *Let $L$ be an event log and $l \in \mathbb{R}$ a time span length. A context-event link, $elink_{l,L} \in E(L) \rightarrow [0,1]^2$, maps events to positive and negative contexts such that, for any $e \in E(L)$, $elink_{l,L}(e)$=$ctx_{l,L}(tw_{l,L}(e))$.*

As depicted in Fig. 3(c) by gray dotted lines, $e1$, $e2$, and $e3$ by $\sigma_1$ and $e4$ and $e5$ by $\sigma_2$ are connected to $ctx_{week,L_1}(w1)$, i.e., $elink_{week,L_1}(e1)$=$ctx_{week,L_1}(w1)$=(0.9, 0.2), etc.

The context of a trace is determined by the context of its events. In this work, we define the maximum positive and negative context of the events of a trace as the context of the trace.

**Definition 11 (Context-Trace Link).** *Let $L$ be an event log and $l \in \mathbb{R}$ a time span length. $tlink_{l,L} \in L \rightarrow [0,1]^2$ maps traces to positive and negative contexts s.t., for any $\sigma \in L$, $tlink_{l,L}(\sigma)$=$(\max(\{pc \in [0,1] \mid \exists_{e \in elem(\sigma)}\ (pc, nc)$=$elink_{l,L}(e)\})$ $, \max(\{nc \in [0,1] \mid \exists_{e \in elem(\sigma)}\ (pc, nc)$=$elink_{l,L}(e)\}))$.*

As shown in Fig. 3(c), $\sigma_1$ has the positive context of 0.9 and negative context of 0.2, i.e., $tlink_{week,L_1}(\sigma_1)$=$(0.9, 0.2)$, since the maximum positive context of its events, i.e., $e_1$, $e_2$, and $e_3$, is 0.9 and the maximum negative context is 0.2. $tlink_{week,L_1}(\sigma_2)$=$(0.9, 0.9)$, since the maximum positive context of its events, i.e., $e_4$, $e_5$, and $e_6$, is 0.9 and the maximum negative context is 0.9.

### 5.4 Post Processing

Post-processing function revises the non-context deviating score of a trace using the positive and negative context of the trace. The positive context decreases the deviating score, whereas the negative context increases it.

**Definition 12 (Post Processing).** *Let $L$ be an event log, $l \in \mathbb{R}$ a time span length, and score a score function. $post_{l,L,score} \in L \times [0,1]^2 \rightarrow [0,1]$ maps a trace, a positive degree, and a negative degree to revised score such that, for any $\sigma \in L$, $\alpha^{pos} \in [0,1]$, and $\alpha^{neg} \in [0,1]$, $post_{l,L,score}(\sigma, \alpha^{pos}, \alpha^{neg})$=$score(\sigma) - score(\sigma) \cdot \alpha^{pos} \cdot pc + (1 - score(\sigma)) \cdot \alpha^{neg} \cdot nc$ where $(pc, nc)$=$tlink_{l,L}(\sigma)$.*

In Fig. 3(d), $\sigma_1$ has the deviation score of 0.6, i.e., $score_1(\sigma_1)$=0.6. Given $\sigma_1$, $\alpha^{pos}$=0.5 and $\alpha^{neg}$=0.5, $post_{week,L_1,score_1}$ revises the deviating score to a new score of 0.37, i.e., $0.6 - 0.6 \cdot 0.5 \cdot 0.9 + (1 - 0.6) \cdot 0.5 \cdot 0.2$=0.37.

Finally, a context-aware detection function labels traces with the four context-aware classes described in Table 1, based on the non-context deviating score and revised deviating score.

**Definition 13 (Context-Aware Detection).** *Let $L$ be an event log and $l \in \mathbb{R}$ a time span length. Let score be a score function. Let $\alpha^{pos}, \alpha^{neg} \in [0,1]$ be positive and negative degrees and $\tau \in \mathbb{S}$ be a threshold. c-detect $\in L \rightarrow \{d{\Rightarrow}d_c, n{\Rightarrow}d_c, d{\Rightarrow}n_c, n{\Rightarrow}n_c\}$ maps traces to context-aware labels such that for any $\sigma \in L$:*

$$c\text{-}detect(\sigma)=\begin{cases} d{\Rightarrow}d_c & \text{if } detect_{score}(\sigma){=}d \text{ and } post_{l,L,score}(\sigma,\alpha^{pos},\alpha^{neg}) > \tau \\ n{\Rightarrow}d_c & \text{if } detect_{score}(\sigma){=}n \text{ and } post_{l,L,score}(\sigma,\alpha^{pos},\alpha^{neg}) > \tau \\ d{\Rightarrow}n_c & \text{if } detect_{score}(\sigma){=}d \text{ and } post_{l,L,score}(\sigma,\alpha^{pos},\alpha^{neg}) \leq \tau \\ n{\Rightarrow}n_c & \text{if } detect_{score}(\sigma){=}n \text{ and } post_{l,L,score}(\sigma,\alpha^{pos},\alpha^{neg}) \leq \tau \end{cases}$$

As shown in Fig. 3(d), given $\tau$=0.5, $\alpha^{pos}$=0.5, and $\alpha^{neg}$=0.5, $c\text{-}detect(\sigma_1)$= $d{\Rightarrow}n_c$ since $detect_{score_l}(\sigma_1)$=$d$ and $post_{week,L_1,score_l}(\sigma_1,\alpha^{pos},\alpha^{neg})$=0.37 $\leq \tau$. Furthermore, $c\text{-}detect(\sigma_3)$=$n{\Rightarrow}d_c$ since $detect_{score_l}(\sigma_2)$=$n$ and $post_{week,L_1,score_l}($ $\sigma_2,\alpha^{pos},\alpha^{neg})$=0.63 $> \tau$.

## 6 Implementation

The framework for context-aware deviation detection is implemented as a cloud-based web service with a dedicated user interface. The implementation is available at `https://github.com/janikbenzin/contect` along with the source code, a user manual, and a demo video. It consists of four functional components: (1) context analysis, (2) deviation detection, (3) context-aware deviation detection, and (4) visualization.

**Fig. 4:** A screenshot of *Scatter* visualization. By varying the degree of positive and negative context, we can deduce the adequate degree of positive and negative context to be used for the context-aware deviation detection.

First, the context analysis component supports the computation of the context history and context. The context introduced in Fig. 2 have been implemented including *workload*, *weekend*, *waiting time*, and *capacity utilization*.

Second, the deviation detection component implements four deviation detection methods that correspond to representatives of four respective categories introduced in Subsect. 2.1. For process-centric methods, we adapt the two-step approach in [8] by using *Inductive* miner [10] for process discovery and *alignment* [1] for conformance checking. For profile-based approaches, *Profiles* [11] has been implemented, while *ADAR* [6] and *Autoencoder* [13] have been implemented as process-agnostic & interpretable/non-interpretable approaches, respectively. Next, the context-aware deviation detection component implements the post processing and the context-aware deviation detection function.

Finally, the visualization component supports an analysis view for each deviation detection method. Each analysis view consists of three visualizations: *tabular*, *scatter*, and *calendar*. *Tabular* visualizes the most deviating traces by sorting them based on the deviation score, the proximity to being relabelled as context-normal, etc. *Scatter* shows a 3D-scatter plot of the deviation score, positive context, and negative context, as shown in Fig. 4. As the number of deviating traces can be large, the k-Mediods clustering algorithm is applied to all deviating traces such that the user can analyze the medoids to understand the whole space of deviating traces more efficiently (depicted as first to fourth and seventh legend entry in Fig. 4). Moreover, by varying the positive and negative degrees, we can analyze the effect of the context on the deviation detection. *Calendar* visualizes the context over time by aggregating contexts by time and plotting them over the time span.

## 7 Evaluation

This section evaluates the proposed framework using the implementation in Sect. 6. To this end, we conduct four case studies using deviation detection methods: *Inductive*, *Profiles*, *ADAR*, and *Autoencoder*. In each case study, we compare the performance of context-aware deviation detection and context non-aware deviation detection in 225 different simulated scenarios. In the rest of this section, we first introduce a detailed experimental design and then report the results.

### 7.1 Experimental Design

As depicted in Fig. 5, the evaluation follows a four step pipeline: *data generation*, *simulation scenario injection*, *framework application*, and *evaluation of results*.

First, the data generation uses CPN Tools[1] to simulate an order management process. Next, we inject four different types of deviating events into the generated event data and label them as non-context deviating: 1) *Rework* randomly adds an event to a trace with the activity that has already occurred, 2) *Swap* randomly swaps the timestamp of two existing events, 3) *Replace resource* randomly replaces the resource of an event with a different resource, and 4) *Remove* randomly

[1] `www.cpntools.org`

removes an existing event from the data. To understand the effect of the amount of deviations on the classification result, the evaluation injected 2%, 5%, or 10% deviations equally distributed among the four types.

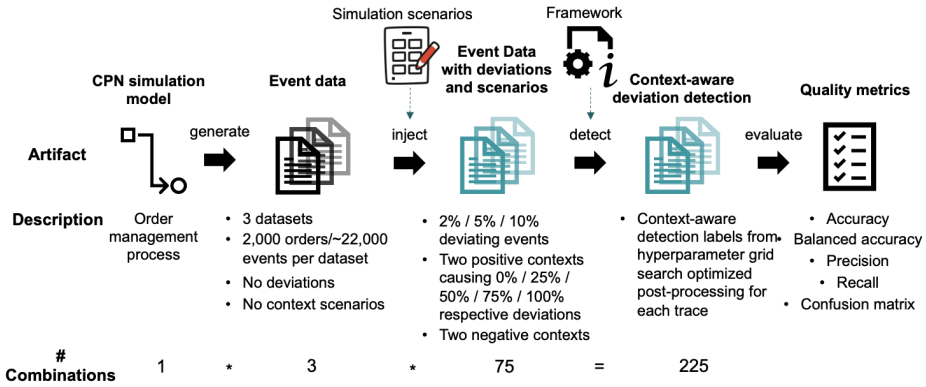



**Fig. 5:** An overview of the experimental design

Afterward, we inject four contextual scenarios as follows.

1. For *workload* scenario, we randomly select a week and add additional orders in the week. We consider it as a positive context and, thus, the non-context deviating events of the selected week are relabelled to context-normal.
2. For *capacity utilization performance* scenario, we randomly assign vacations and sick leaves to resources, lowering the capacity of the process. It is considered as a positive context, and non-context deviating events associated with the reduced capacity resource are relabelled to context-normal.
3. For *waiting time*, all events of randomly chosen days are randomly delayed. It is considered a negative context, and all of the delayed events that are non-context normal or context-normal are labeled as context-deviating.
4. For *overwork* scenario, we shift the random percentage of events during weekdays to Saturday and Sunday. It is regarded as a negative context, and all shifted events that are non-context normal or context-normal are relabelled to context-deviating.

To determine the strength of the relationship between positive contexts and deviations, we use *% context attributable* parameter that determines how many traces are affected by positive contextual scenarios, i.e., non-context deviating events are relabelled to context-normal. We include it as the second parameter for experiments with values ranging from 0% to 100% as depicted in Fig. 5.

225 experiments per case study $(3 * 3 * 5 * 5)$ result from the parameters as shown in Fig. 5, i.e., three event datasets, three % events deviating parameters and the five % context attributable parameters per positive contextual scenario.

Next, we apply the proposed framework and compute context-aware detection results. Hyperparameter grid search is applied to find the best combination of positive and negative degrees for the post function.

**Table 2:** Evaluation results from four case studies

| | | Context-non-aware deviation detection $\alpha^{pos} = \alpha^{neg} = 0$ | Context-aware deviation detection $\alpha^{pos}, \alpha^{neg}$ optimized | Difference |
|---|---|---|---|---|
| Inductive | Accuracy | 0.389118 | 0.426846 | 0.037728 |
| | Avg. class accuracy | 0.326856 | 0.311832 | -0.015024 |
| | Precision | 0.248691 | 0.293496 | 0.044805 |
| | Recall | 0.389118 | 0.426846 | 0.037728 |
| Autoencoder | Accuracy | 0.385035 | 0.425686 | 0.040651 |
| | Avg. class accuracy | 0.311249 | 0.312451 | 0.001202 |
| | Precision | 0.235668 | 0.369101 | 0.133433 |
| | Recall | 0.385035 | 0.424996 | 0.039961 |
| *Profiles* | Accuracy | 0.363995 | 0.406368 | 0.042373 |
| | Avg. class accuracy | 0.293880 | 0.292083 | -0.001797 |
| | Precision | 0.220972 | 0.332658 | 0.111686 |
| | Recall | 0.363995 | 0.404011 | 0.034061 |
| *ADAR* | Accuracy | 0.351544 | 0.395066 | 0.043522 |
| | Avg. class accuracy | 0.291969 | 0.289284 | -0.002685 |
| | Precision | 0.229760 | 0.334021 | 0.104261 |
| | Recall | 0.351544 | 0.385152 | 0.033608 |

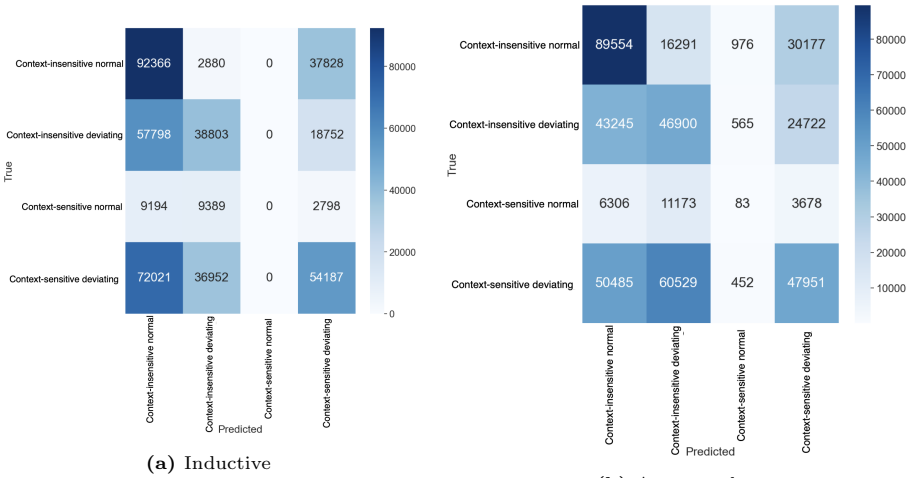
### 7.2 Experimental Results

First, we report average results for each case study in Table 2, showing that the consideration of positive/negative context is effective in the context-aware deviation detection. The first column in Table 2 shows the performance of context-non-aware deviation detection with $\alpha^{pos}$ and $\alpha^{neg}$ both set to 0. The second column in Table 2 shows the performance of context-aware deviation detection with positive $\alpha^{pos}$ and negative degree $\alpha^{neg}$ both optimized through the hyper-parameter grid search. The third column shows the performance difference of the proposed approach with respect to the baseline.

In the case study using *Inductive*, the accuracy of *0.389118* is improved by *0.037728* to *0.426846*, the average class accuracy of *0.326856* is slightly reduced by *0.015024* to *0.311832*, the precision of *0.248691* is boosted by *0.044805* to *0.293496* and the recall of *0.389118* is upgraded by *0.037728* to *0.425686*. The other three case studies also show performance improvements in terms of accuracy, precision, and recall similar to *Inductive* and a decrease in average class accuracy. In particular, the results are significantly more precise with the framework's context-aware deviation detection than for deviation detection.

Second, Fig. 6 shows two confusion matrices in Fig. 6 for *Inductive* and *Autoencoder*, summing the confusion matrix of each experiment. The confusion matrix for *Autoencoder* is representative for *Profiles* and *ADAR*, showing similar results. The context-awareness generally improves the performance in all case studies by improving the detection of *context-sensitive deviating* traces, but not by detection of *context-sensitive normal* traces. With respect to *context-sensitive normal*, the framework's context-awareness has most of the time does not correctly predict the *context-sensitive normal* traces (0 out of *9,194 + 9,389 + 2,798 = 21,381 context-sensitive normal* traces for *Inductive* and *83* out of *6,306 + 11,173 + 83 + 3,678 = 21,240* traces for *Autoencoder*). With respect to *context-sensitive*

*deviating*, the framework's context-awareness performs significantly better for the *context-sensitive deviating* traces with *54,186* of *72,021 + 36,952 + 0 + 54,187 = 163,160* correctly predicted traces (*Inductive*) and with *47,951* of *50,485 + 60,529 + 452 + 47,951 = 159,417* correctly predicted traces (*Autoencoder*).




**(a)** Inductive

**(b)** Autoencoder

**Fig. 6:** Confusion matrices summed over all 225 experiments of the respective context-aware deviation detection method

## 8 Conclusion

In this paper, we proposed a framework to support context-aware deviation detection. The proposed framework can incorporate any existing unsupervised deviation detection methods with varying strengths and weaknesses and enhance them with various contextual aspects. We have implemented the framework as an extensible web service with a dedicated user interface. Moreover, we have evaluated the effectiveness of the framework by conducting experiments using representative deviation detection methods in different contextual scenarios.

This work has several limitations. First, the proposed framework introduces several parameters that possibly affect the detection results, e.g., the negative and positive degree of *post* function, the threshold of *score* function, etc. Second, the framework is dependent on the performance of the deviation detection method. Third, using an event log as the input, the framework only indirectly measures external contexts.

Besides addressing the above limitations, in future work, we plan to extend the framework to support the root cause analysis of context-aware deviations. We can analyze the relevant context of context-aware deviating instances and trace back the relevant context measure, e.g., high workload. Moreover, we plan to extend the framework to consider contexts of different time window lengths, e.g., context in *week*, *day*, and *hour*. Another direction of future work is to develop different post functions to improve the performance of the context-aware deviations.

## References


1. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdiscip. Rev. Data Min. Knowl. Discov. **2**(2), 182–192 (2012)
2. van der Aalst, W.M.P., Dustdar, S.: Process mining put into context. IEEE Internet Computing **16**(1), 82–86 (2012)
3. Bezerra, F., Wainer, J.: Algorithms for anomaly detection of traces in logs of process aware information systems. Information Systems **38**(1), 33–44 (2013)
4. Böhmer, K., Rinderle-Ma, S.: Anomaly detection in business process runtime behavior - challenges and limitations. CoRR **abs/1705.06659** (2017)
5. Böhmer, K., Rinderle-Ma, S.: Multi instance anomaly detection in business process executions. In: et al, J.C. (ed.) BPM 2017. LNCS, vol. 10445, pp. 77–93 (2017)
6. Böhmer, K., Rinderle-Ma, S.: Mining association rules for anomaly detection in dynamic process runtime behavior and explaining the root cause to users. Information Systems **90**, 101438 (2020)
7. Dockhorn Costa, P., Almeida, J.P.A., Ferreira Pires, L., van Sinderen, M.: Situation specification and realization in rule-based context-aware applications. In: Indulska, J., Raymond, K. (eds.) Distributed Applications and Interoperable Systems. pp. 32–47 (2007)
8. Jalali, H., Baraani, A.: Genetic-based anomaly detection in logs of process aware systems. World Academy of Science, Engineering and Technology **64**(4), 304–309 (2010)
9. Kronsbein, D., Meiser, D., Leyer, M.: Conceptualisation of contextual factors for business process performance. Lecture Notes in Engineering and Computer Science **2210** (2014)
10. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: et al, N.L. (ed.) Business Process Management Workshops - BPM 2013 International Workshops. LNBIP, vol. 171, pp. 66–78 (2013)
11. Li, G., van der Aalst, W.M.P.: A framework for detecting deviations in complex event logs. Intelligent Data Analysis **21**(4), 759–779 (2017)
12. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P.: Balanced multi-perspective checking of process conformance. Computing **98**(4), 407–437 (2016)
13. Nolle, T., Luettgen, S., Seeliger, A., Mühlhäuser, M.: Analyzing business process anomalies using autoencoders. Mach. Learn. **107**(11), 1875–1893 (2018)
14. Nolle, T., Luettgen, S., Seeliger, A., Mühlhäuser, M.: BINet: Multi-perspective business process anomaly classification. CoRR **abs/1902.03155** (2019)
15. Pauwels, S.: An anomaly detection technique for business processes based on extended dynamic Bayesian networks. Proceedings of the ACM Symposium on Applied Computing **Part F1477**, 494–501 (2019)
16. Song, R., Vanthienen, J., Cui, W., Wang, Y., Huang, L.: Towards a comprehensive understanding of the context concepts in context-aware business processes. In: Betz, S. (ed.) S-BPM ONE 2019. pp. 5:1–5:10 (2019)
17. Warrender, C., Forrest, S., Pearlmutter, B.A.: Detecting intrusions using system calls: Alternative data models. In: 1999 IEEE Symposium on Security and Privacy. pp. 133–145 (1999)
18. Ye, J., Dobson, S., McKeever, S.: Situation identification techniques in pervasive computing: A review. Pervasive and Mobile Computing **8**(1), 36–66 (2012)