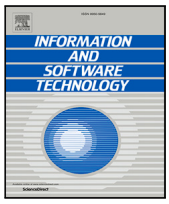
Information and Software Technology 194 (2026) 108079

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



Impact analysis of regulatory requirement changes on business process compliance ☆

Marisol Barrientos [a,*], Karolin Winter [b], Stefanie Rinderle-Ma [a]

[a] TUM School of Computation, Information and Technology, Technical University of Munich, Garching, Germany
[b] Department of Industrial Engineering and Innovation Sciences, Eindhoven University of Technology, Eindhoven, The Netherlands

ARTICLE INFO

Keywords:
Requirement changes
Business process compliance
Requirement formalization
Compliance deviation

ABSTRACT

Context: Ensuring the compliance of business processes with a set of requirements stemming from, for example, laws or regulations, is crucial for companies. When these requirements change, process compliance may be violated, resulting in either non-compliance, where updated requirements are no longer satisfied, or over-compliance, where processes adhere to stricter requirements than necessary. In both cases, this can result in financial or even reputational loss.
Objective: This work provides an automated hybrid approach combining LLM-based and algorithmic steps for continuously analyzing the impact of regulatory requirement changes on business process compliance providing traceability through delta analysis and explanations on compliance deviations.
Method: To address this objective, we present the modular Requirements Change for Process Compliance (RC4PC) approach consisting of three steps. First, we provide a definition of regulatory requirements based on deontic logic allowing to formally represent requirements written in natural language. Second, we extract atomic change operations capturing the differences between an original and a modified requirement. Lastly, we assess the impact of changes on compliance of a process model and provide explanations for compliance deviations. RC4PC concepts are implemented using a combination of LLM-based and algorithmic steps. Change deltas plus explanations provided by the LLM can be taken as input for mitigation actions in case of compliance deviations and contribute to continuous compliance monitoring.
Results: RC4PC is evaluated on three datasets from different domains and in comparison with existing baselines and existing approaches. The results show promising results for each of the steps. Occasional drops in precision, due to inconsistent representations and redundancy, indicate the need for normalization and improved handling of related changes.
Conclusions: RC4PC provides modular and transparent Gen-AI supported approach for analyzing how regulatory requirement changes affect business process compliance. It is domain-independent and can be equipped with existing techniques such as model-checking.

1. Introduction

Business processes are subject to regulatory requirements that are often complex and frequently changing. Such requirements typically originate from diverse sources and are usually articulated in natural language, including legislative texts and regulatory documents [1]. To verify whether a process complies with prescribed requirements, these must be interpreted, formalized, and checked against process models or logs at design and/or runtime [2]. This work focuses on process model compliance at design time. The term *requirement* is used throughout this paper to refer specifically to *regulatory requirements* which correspond to *compliance requirements* in this work.

Most existing compliance verification frameworks, however, do not address the challenge posed by evolving requirements [3,4]. In practice, regulatory change is common and can significantly affect operational processes. During the COVID-19 pandemic, for example, regulations such as mandatory vaccination checks before boarding an airplane were introduced and later rescinded. If process models still enforce a removed vaccination check, this leads to over-compliance, exceeding current requirements [5,6]. Potentially even more severe are

☆ This article is part of a Special issue entitled: 'RegCompliance in SE' published in Information and Software Technology.

* Corresponding author.
E-mail addresses: marisol.barrientos@tum.de (M. Barrientos), k.m.winter@tue.nl (K. Winter), stefanie.rinderle-ma@tum.de (S. Rinderle-Ma).

https://doi.org/10.1016/j.infsof.2026.108079
Received 20 September 2025; Received in revised form 5 February 2026; Accepted 12 February 2026
Available online 14 February 2026

regulatory changes resulting in non-compliance of processes [7], e.g., if an additional "check" is required, but not yet foreseen in the process model. In practice, regulatory requirement changes are often monitored in a manual way, even on a daily basis, and hence create effort and can be error-prone. To address this challenge, this work is guided by the following research questions:

RQ1: How can changes between compliance requirements be represented to support design-time compliance verification and analysis?

RQ2: How can changes between two versions of compliance requirements be automatically identified?

RQ3: How can the identified changes in compliance requirements be used for design-time compliance verification and analysis?

To answer these questions and fill the gap in supporting the management of process compliance in practice, this work introduces the *Requirements Change for Process Compliance (RC4PC)* approach, a design-time analysis approach for identifying changes in a set of requirements and assessing their impact on process compliance. RC4PC aims to detect and explain these compliance deviations. One assumption is that the process model is compliant with the set of requirements at a certain point in time due to the following reasons. (i) Assume that a process model is not compliant before requirement changes. Then it might become (partially) compliant as consequence of the change. This is an important aspect, but beyond the scope of this work. We plan to address it in future work. (ii) We assume compliance requirement change as continuous process, i.e., changes might happen over time, not all at once. In order to keep a clear track of this sequence of changes, we assume a "clean starting point" with a compliant process model.

The paper follows design science research (DSR) [8] for artifact creation and algorithmic engineering (AE) [9] for algorithm creation. Starting from the analysis of a healthcare example from literature [10], RC4PC takes two different versions of a set of requirements in natural language, where some requirements may have been added, deleted, or modified, and formalizes these requirements using a representation based on deontic logic [11] due to being close to (textual) law and change (↦ **RQ1**). This formalization artifact is implemented by a Large Language Model (LLM) due to its strength to deal with text, but under control mechanisms such as controlled vocabularies [12]. For modified requirements, RC4PC extracts, in a second step, atomic change operations (i.e., delta) that capture the modification at different levels of the requirements, such as modified preconditions or norms (↦ **RQ2**). The requirement levels are represented by a four-level hierarchy that enables RC4PC to pinpoint the modifications in a fine-granular way, and enables the analysis of which changes impact process compliance. This second step is designed and implemented as an algorithm in order to calculate the deltas deterministically based on formalized requirements. In a third step, RC4PC calculates the impact of all these change operations on the compliance of a process model, distinguishing between no impact, non-compliance, and over-compliance (↦ **RQ3**). This step is LLM-supported in order to exploit the capabilities of reporting on compliance deviations given a process model and deltas determined in the second step. The evaluation of all artifacts and algorithms is based on three datasets from different domains with corresponding ground truths as baselines to especially evaluate the results achieved by the LLM-based implementations. Moreover, a comparison with existing approaches from literature, i.e., [12,13], is conducted.

The core idea behind RC4PC to determine the *delta* between two sets of requirements is driven by the goals to provide a fine-grained and traceable change analysis, which then serves as input for compliance impact analysis. We distinguish between direct compliance impacts, which arise from changes to a requirement itself, and cascading compliance impacts, which arise indirectly through dependencies between norms across different requirements (e.g., when the fulfillment of one obligation activates another). Our impact analysis focuses on direct impacts; cascading effects are not within the scope of the paper. This reasoning is inspired by delta identification in data snapshots [14] and process changes [15] allowing tracing specific requirement changes to their compliance effects. This provides valuable information for specifying mitigation actions subsequently for both, non-compliance and over-compliance.

The remainder of this paper is structured as follows. Section 2 discusses related work. Section 3 introduces a running example. Section 4 presents the conceptual approach for formalizing requirements, calculating atomic change operations, classifying the operations by the impact they cause, and extracting compliance deviations. Section 5 details how we implemented the provided conceptual approach using LLMs and exact algorithmic components implemented in Python. Section 6 provides the three datasets and the strategy used to evaluate the approach, together with the analysis of the results and a comparison with existing approaches. Section 7 presents a discussion, and Section 8 offers concluding remarks and an outlook.

2. Related work

Our approach for impact analysis of regulatory requirement changes touches upon several research areas as outlined in the following.

Design-time Compliance Verification and Analysis. Design-time verification for business process compliance has been extensively studied, with significant efforts focused on modeling requirements [16,17] as well as modeling compliance violations [18]. Despite the variety of available approaches, ranging from logic-, query-, pattern-, to graph-based [12], the primary focus is either on compliance verification or on modeling requirements, leaving a unified approach that integrates both aspects missing (i.e., an approach that can model /formalize requirements and directly verify their compliance against processes within the same framework) [19]. Moreover, they typically assume that formalized requirements are manually defined [3]. Some exceptions exist that consider natural language input, either in the form of process descriptions [20–22], or compliance requirements [13,23,24]. However, approaches based on process descriptions verify compliance a posteriori against event logs rather than against explicit process models, while approaches relying on process models are typically limited to control-flow and data aspects and do not cover the broader spectrum of compliance requirements. Compliance analysis builds on verification by introducing quantitative measures for the degree of compliance and the costs associated with non-compliance [25] and over-compliance [5]. While these measurements provide a detailed view of the compliance of a process, they generally assume a fixed set of requirements and do not address how the compliance degree or cost estimates should be updated when those requirements change. As a result, it remains difficult to determine which parts of a process become more or less compliant or costly after regulatory updates.

Supporting Requirement Changes. The analysis and handling of changing requirements has been most extensively explored in field of Requirements Engineering [26]. Most of this work has concentrated on supporting legal drafting (e.g., developing the Legislation Editing Open Software (LEOS)[1]) and broader societal impacts, rather than on operational compliance verification. One exception is the work by Abualhaija et al. [26], which proposes a taxonomy for legal changes in the context of software system compliance. While this represents a crucial step, the research remains in early stages and is specifically concerned with regulatory change in software, not directly addressing business process compliance. When it comes to business processes, existing methods for assessing compliance impact only handle the deletion or addition of an entire requirement [27]. However, these approaches often model a replacement simplistically as a deletion followed by an addition. This high-level view fails to capture the specific effects of modifying specific, granular parts within a requirement, leaving a gap for understanding

[1] https://interoperable-europe.ec.europa.eu/collection/justice-law-and-security/solution/leos-open-source-software-editing-legislation, last access 03 February 2026.

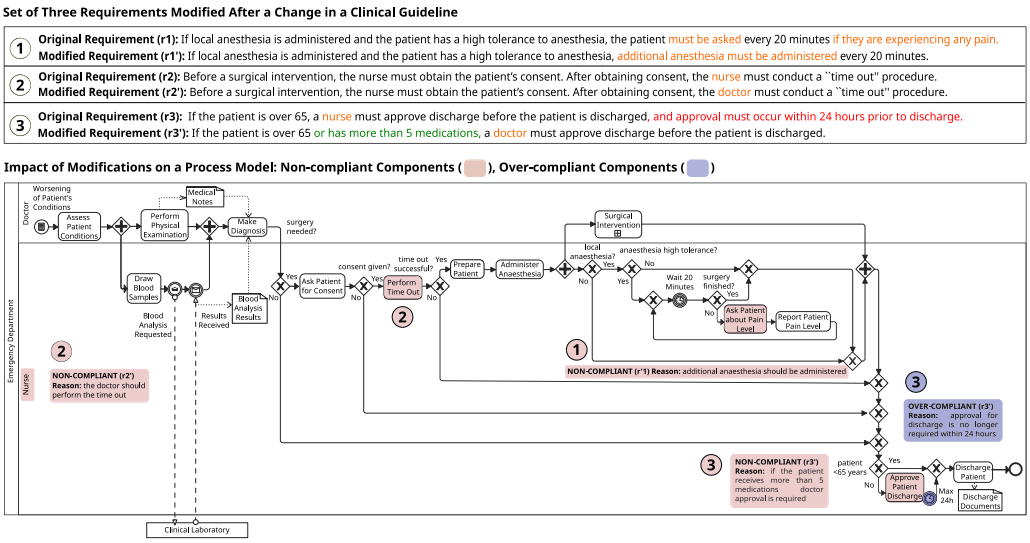


Fig. 1. Modified requirements (top) and their impact on a process model (bottom).

the precise impact of requirement changes on the compliance of a process.

Usage of LLMs for Compliance. Recent papers [28,29] point out that regulatory language is often ambiguous and complex, which makes it difficult to formalize manually, even for experts. This difficulty shows the need for automation. Using LLMs to support the (semi-)automatic extraction and formalization of requirements [30] or in general regulatory modeling [31] is therefore a promising and necessary research direction. LLMs have also been used to extract process models from text [32] or vice-versa, text from process models [33], as well as to derive executable process artifacts such as deployment-ready code from text [34], and these approaches have been systematically reviewed in [35]. While this task differs from design-time compliance verification, both require mapping natural language text to business process elements. It has been also explored the use of LLMs for process model redesign, employing LLMs to identify process change patterns (as cataloged in [36] and formally defined in [37]) from user input and subsequently using this information to guide model modifications [38]. While these approaches focus on determining "how" a process model should be changed in response to user requests, our work concentrates on analyzing "what" is being changed at the level of requirements themselves and which parts of the model become non and over-compliant after that requirement change. By tracing how requirement changes affect the process model, our approach provides concrete explanations for compliance outcomes, addressing the lack of justifications noted by Hassani et al. [39].

3. Running example

To illustrate the problem of requirement changes affecting the compliance of process models, a running example is introduced based on an *Emergency Care* process model (cf. [10]). The process model depicted in Fig. 1 captures the main steps involved in the assessment and treatment of a patient whose condition is worsening. Apart from the process model, the running example also consists of a set of seven requirements inspired by clinical guidelines such as *Safe Surgery Saves Lives* [40]. Among these seven requirements, three are illustrated in Fig. 1 and further analyzed in the subsequent sections. The other four requirements will be considered in the evaluation and are available in the GitHub repository.[2] Each requirement is represented in its original ($r1$, $r2$, $r3$) and updated version ($r1'$, $r2'$, $r3'$), reflecting a change in the requirement.

Our approach assumes that the process model is compliant with the requirements (in this case with r_1–r_3) before changes occur, and is sound, i.e., free of structural and behavioral problems such as deadlocks. Fig. 1 illustrates how changing $r1$–$r3$ into $r1'$–$r3'$ leads to different types of compliance deviations within the process model. Specifically, non-compliance occurs when additional anesthesia is not administered as required by $r1'$ (cf. 1), or when the "time out" is conducted by the nurse instead of the doctor as mandated by $r2'$ (cf. 2). The change in $r3'$ results in both non-compliance, when doctor approval is missing for patients on multiple medications (cf. 3), and over-compliance, when unnecessary nurse approval is still performed within a 24-hour window (cf. 3). Discovering those compliance deviations is crucial; therefore, in the following, we present an approach supporting this complex task.

4. RC4PC conceptual method for analyzing the impact of requirement changes on process compliance

This section presents the conceptual method behind *RC4PC* for analyzing how changes to requirements affect business process compliance. As illustrated in Fig. 2(a), a regulatory change can trigger a requirement change, potentially leading to compliance deviations. Fig. 2(b) outlines the method's three phases. It takes as input a process model and the set of added, deleted, or modified requirements. First, requirements in natural language are formalized (cf. Section 4.1). Second, atomic changes are extracted for modified requirements (cf. Section 4.2); some of these changes may impact compliance more significantly (cf. Section 4.3). Third, all changes are assessed against the process model, unless no relevant changes are found. Each phase is detailed in the following sections.

4.1. Machine-analyzable requirement representation

Managing requirement changes requires a notation that captures the semantics of a requirement precisely and subsequently also the semantics of its change. In this work, we opt for a representation grounded on deontic logic [41], which aligns with the normative nature of compliance requirements. Deontic logic is well suited to expressing obligations, permissions, and prohibitions, concepts that naturally evolve when requirements change (e.g., a previously prohibited action may become permitted). Its expressiveness makes such changes easier to understand and supports the specification of requirements in a conditional and context-sensitive manner [42]. Requirement sets are assumed to be curated and conflict-free. Moreover, our representation allows for modeling common process dimensions, i.e., time, resource,

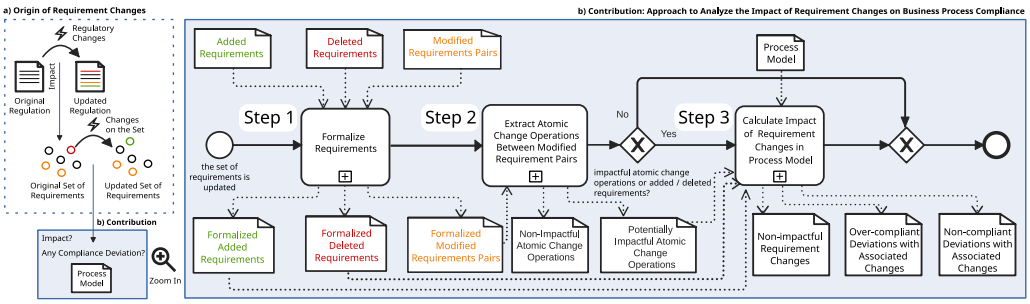


Fig. 2. Origin of requirement changes and the *RC4PC* 3-Step compliance analysis.

data, and control flow. In order to define requirements in our notation, we start with a definition of an *action*.

Definition 4.1 (*Actions*)**.** Let D:= {control-flow, data, resource, temporal} be the set of process dimensions, CP the set of compliance patterns as described in literature for dimensions $d \in D$ (see for control-flow, e.g., [2], data flow [43], resources [21], time [30]), and $\mathcal{X}_{cp}$ be the set of (process) elements affected by compliance pattern $cp \in CP$, e.g., tasks, data elements, resources. The set of all actions is defined as $A :\subseteq D \times CP \times \mathcal{X}_{cp}$.

Action $a \in A$ with a=(control-flow, precedes,{obtain consent, time-out}) (cf. *r2* in the running example Fig. 1), for example, describes a precedes compliance pattern for activities `obtain consent` and `time-out`, i.e., refers to the control-flow. Compliance patterns can be classified based on whether they are sensitive to the order of elements. In particular, order-sensitive patterns, such as `A_followed_by_B` and `A_immediately_followed_by_B`, require that the sequence of activities is preserved. These patterns are commonly found in the control-flow dimension, where changing a constraint from "Activity A must be followed by Activity B" to "Activity B must be followed by Activity A" alters the intended behavior. In contrast, order-insensitive patterns do not rely on the sequence of elements. For example, the resource constraint `static_SoD` (Separation of Duties) specifies that Activity A and Activity B must be executed by different users, regardless of their order in the process specification. Distinguishing between these types of patterns is essential to avoid misclassifying harmless reorderings as meaningful changes.

In general, deontic logic is used to represent norms; norms can be obligations prescribing that a specific action must occur, permissions prescribing that a specific action may occur, i.e., it is neither obligated or prohibited by any other norm, and prohibitions prescribing that a specific action must not occur. A requirement then contextualizes a set of norms with a precondition and temporal validity, formally:

Definition 4.2 (*Norms*)**.** Let M={obligation (O), permission (P), prohibition(F)} be the set of deontic modalities and A be the set of actions. The set of all norms $\mathcal{N}$ is defined as $\mathcal{N} :\subseteq M \times A$. Each norm $n \in \mathcal{N}$ is represented as a pair $n = (m, a)$.

Definition 4.3 (*Requirement*)**.** Let A be the set of actions and $\mathcal{N}$ be the set of norms. Let further P be the set of logical formulas, i.e., a precondition, constructed over A using logical operators LO := {AND, OR, NOT}, i.e., $P := 2^A \mapsto 2^{\text{LO} \times 2^A}$ (nesting of the operators is not permitted). In a precondition all actions in A_{and} must occur, at least one action in A_{or} must occur, and no action in A_{not} may occur. $T = [t_{\text{start}}, t_{\text{end}}]$ denotes the temporal validity of the requirement. If not explicitly indicated, temporal validity is assumed to be unbounded, i.e., $T = (-\infty, +\infty)$ by default. We define the set of all requirements $\mathcal{R}$ as $\mathcal{R} :\subseteq P \times 2^{\mathcal{N}} \times T$.

Remark. Temporal compliance patterns used in actions (see Definition 4.1) constrain the behavior of a process (e.g., periodic execution), and are independent of the temporal validity T of a requirement.

For the preconditions, the set of formulas within a precondition is interpreted as a conjunction. As nesting is not allowed, for example, $p = \{\text{AND}([a_1, a_2]), \text{OR}([a_3, a_4]), \text{NOT}(a_5)\}$ is permitted but $\text{AND}([a_1, \text{OR}([a_2, a_3])])$ is not. Take requirement r_1 from the running example (cf. Fig. 1) in natural language. The actions contained in r_1 are a_1=(data, equals, {anesthesia type, local}), a_2=(data, available, {tolerance, high}), and a_3=(time, periodicity, {check pain, 20min}). Then the formalization based on deontic logic turns out as follows: $r_1 = (p, \{n\}, t)$ with $p = \{\text{AND}(a_1, a_2)\}$, $n = (o, a_3)$, and $t = (-\infty, \infty)$, i.e., r_1 encodes the obligation n to ask the patient about pain every 20 min under the given precondition. Preconditions are treated as factual conditions and do not carry deontic meaning; deontic modalities are defined only over actions in norms.

In Section 5, we will show how to extract the requirement representation automatically from text by prompting an LLM with detailed extraction instructions and a strict schema, ensuring consistent mapping from natural language to our requirement formalization. Alternatively, requirements can be extracted and formalized manually or may already be available for the subsequent steps.

4.2. Requirement change operations

This subsection introduces the operations used to describe changes in compliance requirements ($\mapsto$ **RQ1**). These change operations capture how a set of requirements $R \subseteq \mathcal{R}$, extracted from text-based regulations, may change to reflect new obligations, relaxations, or reinterpretations [26]. Such changes may stem from new legal mandates (e.g., a stricter data privacy law), internal policy updates (e.g., the adoption of remote work rules), or shifts in operational context (e.g., the introduction of new technologies) [44]. In *RC4PC*, changes of R are described along a hierarchy of levels (1–4), cf. Fig. 3. Changes at Level 1, referred to as *high-level changes*, answer the question "what happened to the set of requirements?" by indicating whether a requirement was added, deleted, or modified. We denote the sets of added, deleted, and modified requirements as Add, Del, and Mod, respectively. These sets are assumed to be known, as they can be obtained via requirement traceability systems [45]. For modified requirements $r \in Mod$, we provide a fine-grained analysis based on *atomic changes* at Levels 2–4 explaining "what happened inside the requirement?" by detailing modifications to its content or structure, such as preconditions, norms, temporal validity, actions, or target entities.

Treating a modification as a simple removal followed by an addition would obscure the continuity between versions, which is problematic in legal and regulatory contexts where traceability is essential. Determining the difference between the two versions of requirements first, i.e., the *atomic delta*, preserves continuity, supports automated reasoning, and enhances our understanding of compliance evolution. For instance, consider a requirement r_2 that obligates obtaining the patient's consent before surgery, followed by a "time out" procedure, with both tasks performed by a nurse. This is encoded via norms n_1–n_4, capturing control-flow and resource constraints, as outlined in

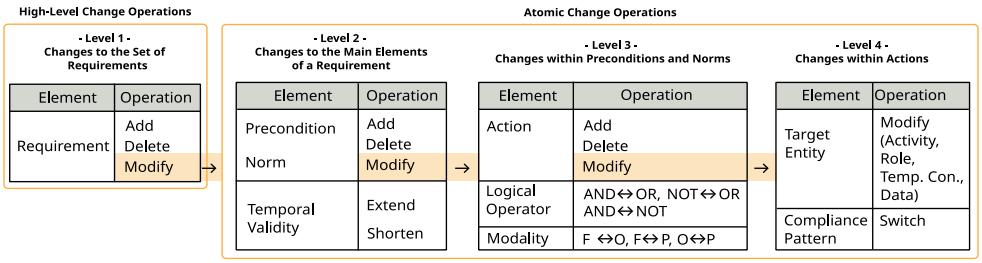


Fig. 3. Hierarchy of high-level and atomic change operations.

Table 1
Example of a requirement formalization and atomic delta (i.e., $\Delta(r_3, r_3')$).

Requirement	Formalization
If the patient is over 65, a nurse must approve discharge before the patient is discharged, and approval must occur within 24 hours prior to discharge. The rule is valid from 2024 to 2030.	r_3 := ({AND({a1})}, {n1 = (O, (control-flow, **precedes**, {"approve_discharge", "patient_discharged"})); n2 = (O, (resource, **performed by**, {"approve_discharge"}, {nurse})); n3 = (O, (time, **time_lag**, {"approve_discharge", "patient_discharged"}, {≤ 24h}))}, T = [2024, 2030]⟩ a1 : (data, **data_in_range**, {age > 65})
If the patient is over 65 or has more than 5 medications, a doctor must approve discharge before the patient is discharged. The rule is valid from 2025 to 2027.	r_3' := ({OR({a1, a2})}, {n1 := (O, (control-flow, **precedes**, {"approve_discharge", "patient_discharged"})); n2 := (O, (resource, **performed by**, {"approve_discharge"}, {doctor}))}, T = [2025, 2027]) a1 : (data, **data_in_range**, {age > 65}) a2 : (data, **data_in_range**, {num_medications > 5})

Change Op	L	Element	Details
Delete	2	Norm	Deleted temporal constraint: `(dimension: temporal, pattern: time_lag, target: {activities: [approve_discharge, patient_discharged], temporal_constraints: [≤ 24h]})`
Shorten	2	Temporal Validity	Shorten validity period from 2024 – 2030 to 2025 – 2027
Add	3	Action	Added action to the logical expression **OR** from precondition: `(dimension: data, pattern: data_in_range, target: {data_values: [num_medications > 5]})`
AND → OR	3	Logical Operator	Logical operator for the precondition `(dimension: data, pattern: data_in_range, target: data_values: [age > 65])` changed from **AND** to **OR**.
Modify role	4	Target Entity	Role performing `approve_discharge` modified: `nurse → doctor`

Definition 4.2. After a regulatory update, the responsibility for the "time out" is reassigned from nurse to doctor. In the updated requirement r_2', n_4 becomes n_4': (O, (resource, *performed by*, "conduct time out", doctor)). Here, $\Delta(r_2, r_2')$ captures this atomic change to the responsible resource, retaining the historical trace of r_2. Fig. 3 shows how the different levels connect, from the overall set of requirements to their internal elements.

Atomic change operations span three levels in order to mirror the structure of a requirement and facilitate the management of complex changes [46,47]. Level 2 includes operations on the main structural elements: preconditions, norms, and temporal validity. Preconditions and norms can be added, deleted, or modified, while temporal validity intervals may be shortened or extended. Level 3 focuses on internal elements of preconditions, such as the content of actions and the logical operators connecting them, and of norms, such as the prescribed actions or their deontic modalities. When an action itself is modified, Level 4 further refines the change by targeting the internal structure of the action, including the compliance pattern (e.g., `precedes`, `followed by`) or the target entity (e.g., a resource, data object). Tables 2 and 3 summarize all change operations, high-level (Level 1) and atomic (Levels 2–4), with examples from the evolution of r_1, r_2, and r_3. Table 1 presents two versions of a requirement, r_3 and r_3'. Changes are highlighted by color: orange indicates modified elements, red marks removed elements, and green shows additions. These differences include a modified resource assignment, removal of a temporal constraint, a shortened validity period, a shift from **AND** to **OR** and the addition of a new data-based action within the precondition. Using Algorithm 1(↦ **RQ2**), the corresponding set of atomic change operations is extracted as $\Delta(r_3, r_3')$ (shown in the lower part of the table). These operations capture the evolution across levels: Level 2 records changes to norms and temporal validity, Level 3 identifies added actions and changes in logical operators, and Level 4 details the modification of the target entity (in this case, a role) within the updated norm.

The *atomic delta* Δ capturing all atomic change operations applied between two versions, r_1 and r_1', is computed by Algorithm 1, which systematically traverses the structure of both requirements to identify atomic changes. It begins by comparing their temporal validity intervals and records any differences, including whether the validity period has been shortened or extended (lines 2–8). It then analyzes changes in preconditions (lines 9–21) by detecting additions and deletions, as well as modifications where structurally similar preconditions differ in actions or logical operators. For example, if logical operators between preconditions change, such as switching from conjunction to disjunction, this is explicitly captured. Finally, the algorithm compares the norms associated with each requirement (lines 22–45), identifying deleted and newly added norms, as well as more subtle changes such as changes in modality (e.g., from obligation to prohibition) and modifications within their actions (e.g., modify target role, switch compliance pattern).

4.3. Impact of changes in requirements on process model compliance

The impact of a requirement change on the compliance of a process model is defined as the set of *compliance deviations* it introduces into a previously compliant process model (↦ **RQ3**). The process in Fig. 1, for example, initially complies with requirements r_1, r_2, and r_3. Two

Algorithm 1 Calculating atomic change operations between requirements

Require: r_1, r_1': formalized requirements (old and new)
Ensure: Δ: list of atomic changes as (*element_type*, *change_operation*, *old_element*, *new_element*)

$\Delta \leftarrow [\,]$
Part 1: Calculating Changes in Temporal Validity
for all $t \in \{\texttt{start}, \texttt{end}\}$ **do**
  if $r_1.\texttt{validity}[t] \neq r_1'.\texttt{validity}[t]$ **then**
    if $R_1.\texttt{validity}[t] < r_1'.\texttt{validity}[t]$ **then**
      $change \leftarrow \texttt{extend\_temporal\_validity}$
    else
      $change \leftarrow \texttt{shorten\_temporal\_validity}$
    $\Delta \leftarrow \Delta \cup \{(\texttt{temporal\_validity}, change, r_1.\texttt{validity}[t], r_1'.\texttt{validity}[t])\}$
Part 2: Calculating Changes in Preconditions
$P_1 \leftarrow r_1.\texttt{preconditions.and} \cup r_1.\texttt{preconditions.or} \cup r_1.\texttt{preconditions.not}$
$P_1' \leftarrow r_1'.\texttt{preconditions.and} \cup r_1'.\texttt{preconditions.or} \cup r_1'.\texttt{preconditions.not}$
for all $p_1 \in P_1$ **do**
  if no $p_1' \in P_1'$ matches p_1 **then**
    $\Delta \leftarrow \Delta \cup \{(\texttt{precondition}, \texttt{delete\_precondition}, p_1, \texttt{None})\}$
for all $p_1' \in P_1'$ **do**
  if no $p_1 \in P_1$ matches p_1' **then**
    $\Delta \leftarrow \Delta \cup \{(\texttt{precondition}, \texttt{add\_precondition}, \texttt{None}, p_1')\}$
  else if $\exists p_1$ such that p_1 matches p_1' but with different actions **then** ▷ For each not equal action do steps from lines 35–45
    $\Delta \leftarrow \Delta \cup \{(\texttt{precondition}, \texttt{modify\_precondition}, p_1, p_1')\}$
if actions in $r_1.\texttt{preconditions}$ are linked with different logical operators than from those in $r_1'.\texttt{preconditions}$ **then**
  $\Delta \leftarrow \Delta \cup \{(\texttt{logical\_operators}, \texttt{switch\_logical\_operators}, r_1.\texttt{preconditions}, r_1'.\texttt{preconditions})\}$
  $\Delta \leftarrow \Delta \cup \{(\texttt{precondition}, \texttt{modify\_precondition}, r_1.\texttt{preconditions}, r_1'.\texttt{preconditions})\}$
Part 3: Calculating Changes in Norms
for all $n_1' \in r_1'$ such that n_1' is not matched in r_1 **do**
  $\Delta \leftarrow \Delta \cup \{(\texttt{norm}, \texttt{add\_norm}, \texttt{None}, n_1')\}$
for all $n_1 \in r_1$ such that n_1 is not matched in r_1' **do**
  $\Delta \leftarrow \Delta \cup \{(\texttt{norm}, \texttt{delete\_norm}, n_1, \texttt{None})\}$
for all matched norms (n_1, n_1') **do**
  if $n_1 \neq n_2$ **then**
    $\Delta \leftarrow \Delta \cup \{(\texttt{norm}, \texttt{modify\_norm}, n_1, n_1')\}$
    if $n_1.\texttt{modality} \neq n_1'.\texttt{modality}$ **then**
      $\Delta \leftarrow \Delta \cup \{(\texttt{modality}, \texttt{switch\_modality}, n_1.\texttt{modality}, n_1'.\texttt{modality})\}$
    if $n_1.\texttt{action} \neq n_1'.\texttt{action}$ **then**
      $\Delta \leftarrow \Delta \cup \{(\texttt{action}, \texttt{modify\_action}, n_1.\texttt{action}, n_1'.\texttt{action})\}$
      if $n_1.\texttt{action.pattern} \neq n_1'.\texttt{action.pattern}$ **then**
        $\Delta \leftarrow \Delta \cup \{(\texttt{c\_pattern}, \texttt{modify\_c\_pattern}, n_1.\texttt{action.pattern}, n_1'.\texttt{action.pattern})\}$
      for all $k \in \{\texttt{activities}, \texttt{data\_values}, \texttt{resources}, \texttt{temporal\_constraints}\}$ **do**
        if $n_1.\texttt{action.target}[k] \neq n_1'.\texttt{action.target}[k]$ **then**
          if $k = \texttt{activities}$ **then**
            $label \leftarrow \texttt{modify\_target\_activity}$
          else if $k = \texttt{data\_values}$ **then**
            $label \leftarrow \texttt{modify\_target\_data}$
          else if $k = \texttt{resources}$ **then**
            $label \leftarrow \texttt{modify\_target\_role}$
          else if $k = \texttt{temporal\_constraints}$ **then**
            $label \leftarrow \texttt{modify\_target\_temporal\_constraint}$
          $\Delta \leftarrow \Delta \cup \{(\texttt{target\_entity}, label, n_1.\texttt{action.target}[k], n_1'.\texttt{action.target}[k])\}$
return Δ

types of deviations may occur: (1) **non-compliance**, where the process allows behavior now prohibited or omits a new obligation (e.g., if r_1 evolves to r_1', the process model is missing a task for administering additional anesthesia), and (2) **over-compliance**. [48] defines over-compliance over a set of traces L that reflect the behavior of a process model. Given a requirement with condition and consequence, L is *"overcompliant iff the consequence is executed in some traces where it is not required"* and *"maximally overcompliant iff the consequence is executed in every trace where it is not required"*. This happens, for example, when the process maintains unnecessary restrictions (e.g., r_3' no longer requires 24-hour approval but the process still enforces it). The change causes **no impact** if the new content is unrelated to any part of the process model, or if the process already aligns with the modified requirement, such as when a requirement is relaxed (e.g., obligation to permission). For example, if $r2'$ relaxes the nurse's obligation to conduct the "time out" to a permission (cf. O→P Table 3), and the process already includes this step, no changes are needed; the action remains compliant. This does not show over-compliance, since it is allowed that the nurse performs that action.

Classification of Requirement Changes. The high-level change operations $Add \cup Del$ and the atomic delta Δ determined by Algorithm 1 are classified along their potential to introduce compliance deviations in a previously compliant process model (see Tables 2 and 3). In some cases, a change impact is determined by its top-level context; for example, adding a new action (cf. `add Action` Table 3) through a norm or precondition inherits the impact of that higher-level addition. In other cases, the effect stems from sub-level modifications, such as switching a norm's modality or logical operator (cf. `switch Modality or LO` Table 3). Table 2 presents the potential impact of operations at Level 1 and Level 2.

At Level 1, coarse-grained changes such as adding, deleting, or modifying entire requirements can immediately alter compliance conditions: introducing new obligations may cause non-compliance, while removing them may lead to over-compliance if the process continues to enforce outdated requirements. Level 2 captures more fine-grained modifications to a requirement's preconditions, norms, and temporal validity. These changes can also lead to deviations. For instance, deleting a precondition (e.g., p_1 in r_1) makes the associated norm apply more broadly, which may result in non-compliance if the process does not satisfy the expanded scope of applicability; conversely, shortening a validity interval (e.g., reducing the period of r_3 from [2024, 2030] to [2025, 2027]) may cause over-compliance if the process still enforces the requirement beyond its intended scope.

Table 2
Impact of change operations in process compliance (levels 1–2).

L	Element	Change Operations	Example of a Change	Impact on Process Compliance
1	Compliance	`add`	Adding r_4: "Before a surgery, blood must be analyzed unless the doctor specifically authorizes skipping this step"	The process becomes **non-compliant** if it allows behaviors the new requirement forbids or omits behaviors it obligates
		`delete`	Deleting r_1	The process becomes **over-compliant** if it still blocks allowed behaviors or requires unnecessary ones
	Requirement	`modify`	Any change occurring in the elements of the sub-levels	Change operations in the **sub-levels** determine it
2	Precondition	`add`	In r_2, add precondition p_1 = (control flow, *existence*, {"make diagnosis"})	The process becomes **over-compliant** if it still blocks allowed behaviors or requires unnecessary ones
		`delete`	In r_1, delete p_1	The process becomes **non-compliant** unless it already meets the required behaviors without the precondition
		`modify`	Any change occurring in the elements of the sub-levels	Change operations in the **sub-levels** determine it
	Norm	`add`	In r_2, add norm n_5 = (O, (control-flow, *precedes*, "administer antibiotics" "surgical intervention"))	The process becomes **non-compliant** if it allows behaviors the new norm forbids or omits behaviors it obligates
		`delete`	In r_2, delete n_4	The process becomes **over-compliant** if it still blocks allowed behaviors or requires unnecessary ones
		`modify`	Any change occurring in the elements of the sub-levels	Change operations in the **sub-levels** determine it
	Tmp. Validity	`extend`	In r_3, extend T to [2023, 2035]	The process becomes **non-compliant** if, after the original end date, it allows behaviors the requirement still forbids or omits behaviors it still obligates
		`shorten`	In r_3, shorten T to [2025, 2027]	The process becomes **over-compliant** if, after the new end date, it still blocks behaviors now allowed or requires actions no longer needed

Table 3 details the potential impact of change operations at Levels 3 and 4, which affect elements contained in preconditions and norms. Level 3 includes changes to actions, logical operators, and modalities. Switching from `OR` to `AND` within a precondition, for example, narrows the scope in which the norm applies. This relaxes the requirement, which may lead to over-compliance if the process continues to enforce the norm in situations where it is no longer required. Changing a modality from permission to obligation (`P→O`) can result in non-compliance if the process does not ensure the required behavior. However, some changes may have no impact, for example, replacing an obligation with a permission (`O→P`) does not affect the process, as the requirement now simply becomes optional. By contrast, if an action becomes permitted only because a requirement was deleted, without being explicitly stated as permitted, this may lead to over-compliance. Level 4 refines the analysis further by targeting elements inside actions, such as the compliance pattern and target entities (e.g., responsible role, data, or temporal constraints). The resulting deviations depend on whether the process has been adjusted to reflect the new semantics introduced by the change.

The classification provided in Tables 2 and 3 can be utilized for determining the impact of requirement changes on process model compliance. First, change operations that will not cause compliance deviations can be detected. Moreover, the classification identifies change operations that might cause compliance deviations, which can be investigated in detail subsequently. Those are cases where the process model (i) permits actions that are now prohibited, (ii) omits actions that have become mandatory, (iii) violates updated control-flow constraints, or (iv) assigns tasks to unauthorized resources, each leading to non-compliance. Over-compliance may also occur if the model retains obsolete constraints, such as outdated temporal rules or redundant role assignments. By leveraging the hierarchical classification and filtering out operations with no expected impact (e.g., modality relaxations), the verification process becomes more efficient and transparent, as it enables a clear mapping between specific types of requirement changes and their potential compliance effects. For instance, if a permission is changed to a prohibition, and the process still allows the prohibited behavior, the deviation can be directly traced back to this specific change in modality. Notably, no deviation arises if a prohibited behavior is structurally impossible (e.g., a forbidden second surgery in a model that never includes such a path).

The *RC4PC* approach aims at a fine-granular analysis of requirement changes and their impact on process model compliance, resulting in a possible optimization of the impact analysis and the ability to pinpoint causes for compliance deviations in a fine-granular way. In Section 5, we will further investigate the options for optimizing compliance impact analysis based on employing Large Language Models (LLMs) for different tasks in the *RC4PC* approach, i.e., the formalization of textual requirements and the compliance deviation analysis exploiting the change operation classification provided in Tables 2 and 3.

5. Implementation of the *RC4PC* approach

This section presents the implementation of the *RC4PC* approach proposed in Section 4 which leverages both, the previous and updated versions of a requirement, along with the extracted delta between them, to identify potential deviations more effectively than by considering only the latest version of the requirement in natural language. Implemented in Python 3.11.13, the three steps (cf. Fig. 2 b) are independent

Table 3
Impact of change operations in process compliance (levels 3–4).

L	Element	Change Operations	Example of a Change	Impact on Process Compliance
3	Action	`add`	Adding a new Norm or Precondition introduces a new Action	Change operations in the **top-levels** determine it
		`delete`	Removing a Norm or Precondition deletes its Action	
		`modify`	Any change occurring in the elements of the sub-levels	Change operations in the **sub-levels** determine it
	Logical Operators	$AND \rightarrow OR$	In r_1, change AND → OR, modifying p_1 to OR($[a_1, a_2]$)	The process becomes **non-compliant** if it allows forbidden behaviors or omits obligated ones
		$OR \rightarrow AND$	In r'_3, change OR → AND, modifying p'_1 to AND($[a'_1, a'_2]$)	The process becomes **over-compliant** if it still blocks allowed behaviors or requires unnecessary ones
		$NOT \leftrightarrow AND$, $NOT \leftrightarrow OR$	In r'_3, modify p'_1 to NOT($[a'_1, a'_2]$), , is an example of $OR \rightarrow NOT$	The process is **non-compliant** if it skips the norm when the negated condition holds, or **over-compliant** if it still enforces the old AND/OR condition, blocking allowed or requiring unnecessary behaviors, and vice versa when AND/OR turn into NOT
	Modality	$O \leftrightarrow F$	In r_2 change the obligation of the nurse conducting the "time out" to a prohibition	The process becomes **non-compliant**
		$P \rightarrow O$, $P \rightarrow F$, $F \rightarrow P$	Allowing nurses to conduct the "time out" procedure, previously forbidden, is an example of $F \rightarrow P$	The process becomes **non-compliant** if it allows forbidden behaviors or omits obligated ones
		$O \rightarrow P$	In r_2 change the obligation of the nurse conducting the "time out" to a permission	No impact (if an action becomes permitted only because a requirement was deleted, without being explicitly permitted, this may result in over-compliance)
4	Target Entity	`modify` *Activity*	In r_2, change activity in n_2 from "conduct time out" to "checklist verification"	The process becomes **non-compliant** if it allows forbidden behaviors or omits obligated ones
		`modify` *Role*	In r_2, change in n_4 responsible role from "nurse" to "doctor"	
		`modify` *Data*	In r_1, change in a_1 from "anesthesia type = local" to "regional"	The process becomes **non-compliant** if the newer value (data, time) is more restrictive, but **over-compliant** if it is more permissive and the process still enforces the old stricter value
		`modify` *Temporal Constraint*	In r_1, change in n_1 periodicity to "10 min"	
	Compliance Pattern	`switch`	In r_2, switch the compliance pattern of n_1 from *precedes* to *followed by*	The process becomes **non-compliant** if the newer pattern is more restrictive, but **over-compliant** if it is more permissive and the process still enforces the old stricter pattern

and executable separately. This separation into three independent steps allows for a modularization of the approach as well as to implement each step with different means. Step 3 could be implemented using exact approaches like model checking. However, as elaborated in the related work section, existing approaches typically do not cover multiple process dimensions at once, which is one aim of our approach. While some approaches provide counterexamples beyond binary results, these are hard to interpret when multiple compliance deviations occur [7]. To obtain more expressive explanations of non- or over-compliance after requirement changes, we therefore partially rely on LLMs. In addition, when using LLMs we do not require a one-to-one mapping between elements from the requirements to elements from the process model, e.g., activity labels could be similar but not the same. When using exact methods, a mapping step is mandatory, when using LLMs this is not necessary. Fig. 4 illustrates the main sub-steps of Steps 1–3. Components that query an LLM are highlighted in purple, i.e., sub-steps 1.2 and 3.2. As shown in Fig. 4, all outputs returned by the LLM are validated against a predefined JSON schema (i.e., sub-steps 1.3 and 3.3). If a response is invalid, e.g., due to missing fields, incorrect data types, or parsing errors, it is logged for manual inspection. The pipeline is configurable (e.g., model, temperature; sub-steps 1.1, 2.1, 3.1) and records execution metadata (e.g., timestamps, parameters; sub-steps 1.4, 2.3, 3.4) for transparency. The full implementation and prompts are available online.[2]

5.1. Implementation of Step 1

To automate Step 1, the transformation of requirements from natural language into a structured formal representation, a large language model (LLM), specifically GPT-4.1, is queried via the OpenAI API.[3] The decision to use GPT-4.1 is based on recent work [49] showing promising results in the formalization of defeasible deontic logic rules from legal text using LLMs. Their results indicate that GPT-based models can perform this task with accuracy similar to rule-based or parser-based approaches, especially when using prompt engineering techniques.

[2] https://anonymous.4open.science/r/Requirements_Change_for_Business_Process_Compliance, last access 03 February 2026.
[3] https://platform.openai.com, last access 12 December 2025.

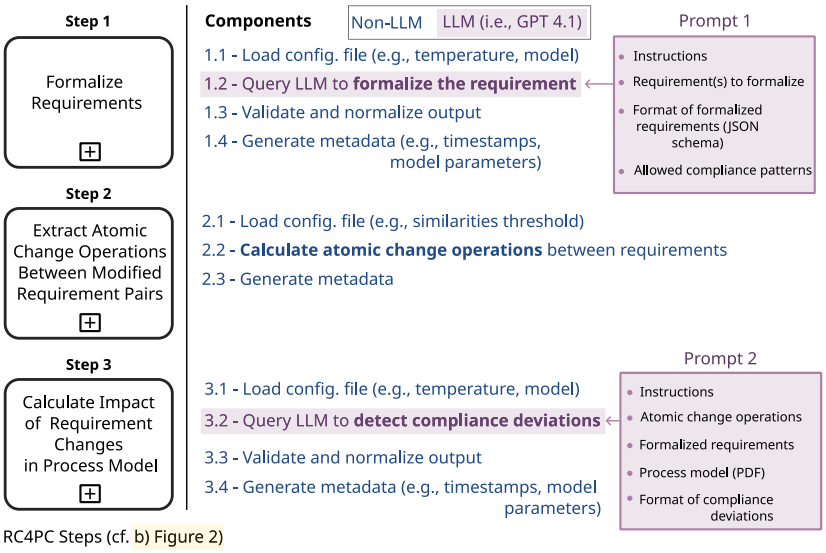


Fig. 4. Implementation pipeline of the RC4PC approach.

In order to support consistent and reliable output from the LLM in Step 1, we apply several prompt engineering techniques. Controlled vocabularies are used to explicitly list the allowed compliance patterns, which reduces ambiguity and prevents the model from generating invalid constructs [50]. An ablation analysis confirms this: when the prompt omits this restriction, the LLM frequently hallucinates non-existent patterns (e.g., `activity_during_activity`) or uses inconsistent naming for the same concept (e.g., `activity_occurs`, `activity_occurrence`, `activity_performs`). Structured formatting is enforced by defining a strict JSON schema [51], and uncertainty is handled explicitly by instructing the LLM to use predefined markers when the requirement is ambiguous [50]. The structure of *Prompt 1*, shown in Fig. 4, guides the transformation of natural language requirements into the structured representation defined in Section 4.1, covering four compliance dimensions: control flow, data, resource, and time. Commentary is disallowed to maintain output structure, and requirements are processed individually, as batching degrades quality.

5.2. Implementation of Step 2

For Step 2, a Python script implements Algorithm 1 (↦ **RQ2**). It takes as input the structured formalizations from Step 1 and compares two versions of a requirement in JSON format (cf. sub-step 2.2 Fig. 4). The script generates a hierarchical delta file (cf. Fig. 3) that captures all additions, deletions, and modifications across preconditions, norms, and temporal validity. Logical changes, such as shifts between conjunctions and disjunctions in preconditions (e.g., `AND` → `OR`), are also detected. Each atomic change is assigned a unique change identifier and classified by type, e.g., precondition added, norm deleted, action modified, switch on the compliance pattern, etc. The comparison includes changes to modalities, compliance patterns, and action attributes (e.g., activities, resources, data values, and temporal constraints). A cosine similarity threshold, computed using the sentence transformer model *all-MiniLM-L6-v2*,[4] ensures semantic alignment between matched elements, while activity labels must match exactly to be considered equal. Metadata is also logged in this step.

[4] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2, last access 03 February 2026.

5.3. Implementation of Step 3

Step 3 evaluates how changes in requirements impact the compliance of a process model (↦ **RQ3**). The inputs include the original and updated requirements (in both natural language and formalized form), the extracted delta, and the process model in PDF format. An ablation analysis shows that using a PDF of the process model leads to clearer results: the LLM better ignores irrelevant details and avoids unwanted identifiers common in raw BPMN XML. While pre-cleaning the XML is possible, it risks removing information relevant to compliance reasoning (cf. Section 7). This supports the idea that, in this context, a picture (i.e., PDF) is indeed worth more than a thousand tags [52], particularly when working with repositories of process models. The PDF is uploaded to OpenAI's file storage using the `client.files.create` method with the `''user_data''` purpose, while the remaining inputs are passed directly in the prompt. GPT-4.1 is queried via the OpenAI API using *Prompt 2* (cf. Fig. 4), which applies the same prompt engineering principles as in Step 1. The model is instructed to assess whether each atomic requirement change triggers a compliance deviation. If a deviation is identified, the output includes its type (i.e., *non-compliance*, *over-compliance*), along with a brief justification and a reference to the affected process element. If there is no impact, this is also explicitly indicated. The output is returned as a structured JSON array to enable automated parsing, and execution metadata, as well as any parsing or validation errors, are recorded for traceability.

6. Evaluation

This section provides the evaluation of the concepts and implementation of the *RC4PC* approach presented in Sections 4 and 5. The datasets used in the evaluation, along with the corresponding results, are available online.[2]

6.1. Datasets

The evaluation comprises three datasets, each containing two versions of a set of compliance requirements and a corresponding business process model. Both the requirements and the business process model are the input of the RC4PC (cf. Fig. 2) The first dataset is the ***Emergencies*** [10], introduced in Section 3. The second dataset is the ***SIM Card***, based on the process model described in [23], which captures the procedure followed by a phone company to onboard a new customer. It includes six requirements, each represented in two versions, derived

Table 4
Overview of the ground truth for each step (highlights in gray).

Step 1: Formalization (total requirements = 36)			Step 2: Change Operation (total change operations = 26)			Step 3: Deviation (total deviations = 25)		
		%		**Change Operation**	%		**Deviation Reason**	%
Precondition	Req. with preconditions	46	L1	Add requirement	8	Non-compliant	Wrong activity	4
	Req. without preconditions	54		Delete requirement	8		Missing activity	16
	Compliance Patterns		L2	Add precondition	8		Wrong role	12
	Data	68		Delete precondition	4		Missing time constraint	4
	Control flow	20		Add norm	4		Weak / Wrong XOR	12
	Time	12		Delete norm	12		Missing XOR	4
	Logical Operators		L3	Change modality	16			
	AND	82		Change logical operator	4			
	OR	12		Delete action	4		Strict time constraint	8
	NOT	6	L4	Change in role	12		Strict order of activities	4
Norm	1 Norm per requirement	68		Switch compliance pattern	8	Over-compliant	Not required activity	24
	>1 Norm per requirement	32		Change in activity	4		Strict XOR	4
	Modality			Change in data	4		Missing XOR	8
	Obligation	78		Change time constraint	4			
	Permission	11						
	Prohibition	11						
	Compliance Patterns							
	Control Flow	73						
	Resource	16						
	Time	11						

from GDPR privacy constraints [53]. The third dataset is the ***Blood Donation***, which is derived from the requirements presented in [5]. The set of requirements in [5] consists of a single version and is based on standards outlined in the Blood Guide published by the European Directorate for the Quality of Medicines & HealthCare.[5] This dataset was selected because, in addition to the guidelines themselves, the Council of Europe provides a comprehensive change log documenting all modifications between editions. These changes are further supported by background documents that explain the scientific rationale behind them.

We evaluate each step of RC4PC independently of the other steps. For this, one of the authors manually created a ground truth dataset containing the expected outputs for each step. For Step 1, the ground truth consists of the formalized requirements, following our representation described in Section 4.1, including elements such as preconditions, norms, and compliance patterns present in each requirement. For Step 2, the ground truth is a set of change operations between the two versions of each requirement, based on the change representation introduced in Section 4.2. For Step 3, the ground truth includes the compliance deviations resulting from the changes identified in Step 2, with each deviation linked to the specific process model elements it affects and an explanation of the reason for the deviation. All annotations are available online.[2]

To reduce potential bias in the annotated ground truth, two independent researchers with expertise in process compliance, who are not authors of the paper, were tasked with formalizing the set of requirements (Step 1), identifying changes between requirement versions (Step 2, simplified), and determining the compliance deviations triggered by those changes (Step 3). The instructions and materials provided to the experts per step, as well as their results and the differences between their annotations and the ground truth, are included in the online repository.[2] For Step 1, the formalizations from the different experts and the ground truth are semantically similar but varied in style. For Steps 2 (simplified) and 3, both experts identify non-compliance in similar ways, but over-compliance allowed for more variation in interpretation. These observations directly influence the evaluation strategy (cf. Section 6.2): for Step 1, we accept alternative formalizations with equivalent meaning, and for Step 3, we limit the LLM's room for interpretation.

Some patterns can be observed across the different datasets. The ***Emergencies*** dataset mainly contains obligations with data-based preconditions and frequent use of control-flow constraints like `A_precedes_B`, sometimes combined with resource constraints such as `performed_by`. Time constraints like `periodicity` appear occasionally. The ***SIM Card*** dataset includes a mix of obligations, permissions, and prohibitions, with fewer explicit preconditions; prohibitions are often linked to `existence_of_A`, and time constraints such as `duration` appear in some cases. The ***Blood Donation*** dataset also contains mostly obligations, often with control-flow patterns like `A_immediately_followed_by_B`; prohibitions are again tied to `existence_of_A`. An overview of the data distribution of the ground truth by step is presented in Table 4, combining all datasets, as the distributions showed little variation between them. With respect to the distribution by step, Step 1 (Formalization) contains 36 requirements, of which 46% include preconditions, mostly data-related, with AND used in 82% of cases. Obligations dominate (78%), and control-flow constraints are the most common pattern (73%). Step 2 (Change Operation) comprises 26 change operations, with modality changes (16%) and norm deletions (12%) being the most frequent, followed by changes in roles and preconditions. Step 3 (Deviations) includes 25 deviations. The most common causes of non-compliance are missing activities (16%), incorrect roles (12%), and weak XOR logic (12%). Over-compliance is mainly due to unnecessary activities (24%).

6.2. Evaluation strategy and analysis of results

This section outlines the evaluation strategy and presents the analysis of results for each of the three steps from the RC4PC approach described in Fig. 4: the automatic formalization of compliance requirements (Step 1), the extraction of atomic change operations between different versions of a requirement (Step 2), and the analysis of their impact on a process model (Step 3). Steps 1 and 3 produce outputs using LLMs, which may contain hallucinations and vary between runs; therefore, the RC4PC approach was run five times independently to study the stability of the results. For Steps 1 (formalized requirements) and 3 (compliance deviations), structural validity was also assessed by verifying whether the output JSON conformed to the schema specified in the prompt. Across the five iterations, all outputs for Steps 1 and 3

[5] https://www.edqm.eu/en/blood-guide, last access 03 February 2026.

Table 5
Overview of results by step and dataset.

Dataset		Step 1 (Formalization)				Step 2 (Change Operation)			Step 3 (Deviations)			
		P	R	F1	SC	P	R	F1		P	R	F1
Emergencies	**Precondition**	0.90	1	0.94					**Dev. Type**	0.96 ± 0.07	0.91 ± 0.19	0.93 ± 0.14
					0.80	0.75	0.85	0.80	**Dev. Reason**	0.85 ± 0.15	0.98 ± 0.04	0.90 ± 0.09
	Norm	0.91	0.91	0.91					**Ref. in Model**	0.94 ± 0.13	1 ± 0.00	0.96 ± 0.07
SIM Card	**Precondition**	0.71	1	0.83					**Dev. Type**	1 ± 0.00	1 ± 0.00	1 ± 0.00
					0.80	1	0.87	0.93	**Dev. Reason**	0.92 ± 0.00	1 ± 0.00	0.96 ± 0.00
	Norm	0.80	0.88	0.84					**Ref. in Model**	1 ± 0.00	1 ± 0.00	1 ± 0.00
Blood Donation	**Precondition**	0.90	0.90	0.90					**Dev. Type**	0.85 ± 0.15	0.85 ± 0.15	0.85 ± 0.15
					0.64	0.75	0.85	0.80	**Dev. Reason**	0.86 ± 0.14	0.86 ± 0.14	0.86 ± 0.14
	Norm	1	1	1					**Ref. in Model**	0.86 ± 0.14	0.86 ± 0.14	0.86 ± 0.14

conformed to the expected format and did not require any additional normalization or manual post-processing. The results from the five executions are available online.[2]

Evaluation Strategy for Step 1. In the evaluation of Step 1, a formalized requirement is considered correct if it semantically aligns with the corresponding representation in the ground truth or captures another valid interpretation of the requirement. This includes the correct representation of preconditions (e.g., actions and their logical structure), norms (with the appropriate modality and compliance pattern), and temporal validity. A true positive is a formalization that matches the ground truth, while a false positive is one that is either incorrect (e.g., wrong content or structure) or not present in the gold standard. Completeness reflects whether all expected elements are present in the output. A false negative is an element from the ground truth that is entirely missing. Based on these definitions, precision (correctness) is calculated as $\mathrm{P} = \frac{TP}{TP+FP}$, and recall (completeness) as $\mathrm{R} = \frac{TP}{TP+FN}$. We also provide the F1-score because it combines precision and recall into a single measure, $F1 = 2 \cdot \frac{\mathrm{P} \cdot \mathrm{R}}{\mathrm{P}+\mathrm{R}}$. However, precision and recall do not indicate how consistently the LLM behaves across repeated runs. To capture this, we measure the stability of Step 1 using self-consistency (SC) metric [54], which reflects how often the LLM produces similar outputs when the same requirement is formalized multiple times. We define it as $SC = \frac{\#(d_{ij} \leq 2)}{\text{total number of pairwise comparisons}}$, where d_{ij} is the distance between two different runs of the same requirement. The term $\#(d_{ij} \leq 2)$ refers to the number of executions in which the distance between two formalizations is 2 or less, meaning that the outputs are identical or differ in a maximum of two elements. We adopted a strict counting strategy when identifying differences between elements, considering variations across all change levels. Differences were counted even when they reflected the same semantic meaning, such as alternative labels for the activity derived from an action of a norm, different ways of expressing temporal constraints (e.g., "within 24 hours" versus "no later than one day"), or the use of different compliance patterns that nonetheless constituted valid interpretations of the requirement. We expected identical results across runs, as the temperature was set to 0 to eliminate stochastic variation. Since a human annotator would consistently formalize the same requirement across multiple attempts, we examined whether the LLM exhibits consistent behavior when formalizing the same requirement multiple times.

Analysis of Results for Step 1. As shown in Table 5 (cf. Step 1), precision, recall, and F1 were consistently high across all three datasets (***Emergencies***, ***SIM Card***, ***Blood Donation***), particularly for identifying norms (average F1 of 0.91). In some cases, redundant preconditions were introduced, such as specifying `existence_of_A` even though the norm `A_followed_by_B` already implies the existence of `A`. In addition, representational inconsistencies affected the quality of some formalizations. Identical time constraints were represented using different activity labels (e.g., `conduct_time_out_procedure` vs. `time_out_procedure`), and roles were sometimes embedded in activity names rather than explicitly assigned (e.g., `nurse_approves_discharge` instead of separating the performer and the action). While redundancy did not distort the semantics, it made some representations unnecessarily verbose, whereas inconsistencies introduced ambiguity and reduced alignment between requirement versions. Stability results further support these observations: the ***SIM Card*** and ***Emergencies*** datasets show the highest stability ($SC = 0.80$), and the ***Blood Donation*** dataset also demonstrates a reasonable level of stability ($SC = 0.64$), indicating that the LLM often produces similar formalizations across runs, with most differences arising from redundant preconditions, small label variations (e.g., "donor_bleeding" vs. "bleeding"), or alternative but equivalent compliance patterns.

Evaluation Strategy for Step 2. For Step 2, P, R, and F1 are computed using the same definitions and formulas introduced for Step 1. To measure P (correctness) and R (completeness), the output is compared against the set of atomic change operations from the ground truth. A true positive is a correctly identified change, a false positive is a change that does not exist in the reference, and a false negative is a missed expected change. Correctness reflects how accurate the detected changes are, while completeness captures whether all expected changes are found. Stability is not assessed in this step, as the algorithm is deterministic and always produces the same output for identical inputs.

Analysis of Results for Step 2. As shown in Table 5 (cf. Step 2), the recall average is close to 0.86, indicating that nearly all expected changes are successfully captured by the system. However, precision is lower, i.e., 0.75 for ***Emergencies***, 1 for ***SIM Card***, 0.75 for ***Blood Donation***, leading to F1-scores between 0.80 and 0.93. This drop in precision occurs primarily due to false positives, where changes were incorrectly inferred from differences in labeling or structure rather than actual semantic modifications. For instance, identical activities labeled (e.g., `conduct_time_out_procedure` vs. `time_out_procedure`) are mistakenly identified as modified actions, and preconditions removed for simplification (e.g., redundant existence checks) are flagged as deletions. In some cases, representational mismatches lead to the detection of changes in temporal constraints, data conditions, or resources, even though the underlying semantics remain unchanged. These issues highlight challenges in norm alignment, that is, determining which norm in the new version corresponds to which in the old norm, particularly when multiple compliance patterns or dimensions (e.g., control-flow, data, resource) are modeled within a single requirement. Despite these challenges, the system successfully detects several relevant changes, including modifications in modality (e.g., obligation to permission), compliance pattern switches, additions or deletions in preconditions, and temporal validity extensions. *The results suggest that while the approach is effective in identifying expected changes, it can be compromised by inconsistencies in modeling practices, reinforcing the need for a mechanism that ensures elements unchanged across versions are modeled consistently.*

Evaluation Strategy for Step 3. To evaluate the quality of the results produced in Step 3 (impact analysis of requirement changes on a process model), we assess the LLM outputs along three aspects: (1) the type of deviation identified, (2) the reason provided for the deviation, and (3) the reference to the affected elements in the process model. For

each requirement change, we first evaluate whether the LLM correctly determines whether the change has any impact on the process model. If it indicates that it has an impact, we assess whether it is assigned the correct deviation type (cf. Table 5 **Dev. Type**). We also evaluate if the LLM provides a correct deviation reason, e.g., noting that a requirement change modifies the resource who should perform `Activity A`, which explains why the deviation occurs (cf. Table 5 **Dev. Reason**). In addition, we check whether the LLM points to the correct elements in the process model that show where the deviation manifests (cf. Table 5 **Ref. in Model**), such as `Activity A` still being assigned to the previous resource. For the deviation type, reason, and reference to the process model, we record true positives (correct type, reason, or reference), false positives (wrong type, reason, or reference) that are not in the ground truth, and false negatives (missing deviation, reason, or reference). Precision, recall, and F1 are then computed for each of the three aspects by averaging the results across the five executions and reporting the corresponding standard deviation to assess the stability of Step 3.

Analysis of Results for Step 3. All the changes introduced solely by representational differences (e.g., alternative modeling of data constraints) or not impacting that model are correctly marked as having no impact. In the ***SIM Card*** dataset, the approach achieves nearly perfect scores (1 ± 0.00) across all three deviation aspects, deviation type, reason, and reference in the process model, confirming its ability to detect when the process fails to meet new obligations (e.g., keeping SIM activation as the customer's responsibility) or enforces obligations that are no longer required (e.g., continuing monitoring or data deletion). The ***Emergencies*** dataset also shows strong results, with F1 scores of 0.93 ± 0.14 for deviation type, 0.90 ± 0.09 for deviation reason, and 0.96 ± 0.07 for reference in model, though the recall for deviation type (0.91 ± 0.19) indicates some variability, due to difficulties in correctly distinguishing over-compliance from non-compliance when a requirement changes from obligatory to optional. The ***Blood Donation*** dataset yields slightly lower but still solid results (0.85 ± 0.15 to 0.86 ± 0.14 across categories), capturing deviations such as the failure to allow doctors to skip unnecessary tests or collecting blood from all patients instead of only adults. This dataset involved a higher number of over-compliant deviations, e.g., continued enforcement of monitoring or restrictive eligibility checks, which resulted in incorrect classifications. Notably, the results show no difference in the quality of deviation detection across the four compliance dimensions, control flow, data, time, and resource, suggesting that the approach is equally applicable across different dimensions. A recurring limitation observed across all three datasets is redundancy. Multiple changes that lead to the same deviation are flagged individually, which affects clarity. Since the LLM was not explicitly tasked with grouping related changes, reducing such redundancy remains an open area for future work. Overall, the results indicate that the approach can accurately detect both, non-compliance and over-compliance, while offering opportunities for improvement in the way deviations are reported.

Summary of Results. Overall, the evaluation shows that *RC4PC* can reliably formalize requirements, detect changes, and identify compliance deviations caused by those changes. While precision is occasionally impacted by representational inconsistencies and redundancy, the results confirm that the identified changes and deviations are both correct and complete. These findings highlight the need to represent unchanged parts of a requirement in a consistent manner (e.g., using the same labels for the same activities), improving the aggregation of related deviations, and further refining the concept of over-compliance.

6.3. Comparison to existing approaches

As *RC4PC* is the first approach for analyzing how changes in requirements impact design-time process compliance, a one-to-one comparison to existing approaches is not possible. The approach is instead compared to design-time compliance verification approaches which are not change-aware. As outlined in Section 2, existing approaches either (i) accept natural-language requirements as input and or (ii) rely on formalized requirements.

Among the **(i) approaches that accept natural language requirements** only [13] is open source, the others [23,24] are not. Consequently, [13] is used to evaluate compliance against both the old and new sets of requirements, with the results contrasted against our ground truth. The evaluation includes the three datasets described in Section 6.1, and the results are available in the project repository.[2] [13] calculates a fitness score that reflects the semantic similarity between a requirement text and a process model. This score is used to filter out irrelevant text fragments, after which the approach checks for compliance deviations (referred to as violations). The method detects non-compliance in only a limited set of cases, specifically: missing activities, incorrect ordering in the control flow dimension, and incorrect roles in the resource dimension, and does not consider over-compliance. It cannot distinguish between `A must happen after B` and `A must happen directly after B`, nor handle resource restrictions like `A cannot be performed by resource X`. Moreover, it focuses solely on obligations, without support for permissions or prohibitions.

Despite these limitations, we applied [13] to our three datasets. In the ***Emergencies*** dataset, which contains five over-compliant and five non-compliant deviations, including two missing activities and two incorrect roles (i.e., 40% of the cases were in scope for detection), the method failed to assign compliance costs to the missing activities and misclassified a role deviation as incorrect ordering os activities. In the ***SIM Card*** dataset, which includes five non-compliant deviations and one over-compliant case, only two missing activities and one incorrect role fall within the method's detection scope (50%). Here, the method reported no semantic similarity for the missing activities and failed to capture the cost of the wrong role assignment. Lastly, in the ***Blood Donation*** dataset, which includes seven over-compliant and two non-compliant deviations (both due to missing activities, i.e., 22.2% in scope), one case was marked as semantically related but received no compliance cost, while the other was treated as unrelated. In both cases, the missing activity was not reflected in the final compliance assessment. One reason for the poor results is that [13] was evaluated on regulation fragments rather than requirements. These fragments contain a larger vocabulary, giving NLP methods more signals to compute similarity scores. In addition, the method only supports a limited set of compliance dimensions, does not handle over-compliance, and has difficulties mapping parts of a requirement to the correct elements in the process model, especially when multiple pools are involved. These factors further reduced its ability to detect deviations.

The modeling languages from **(ii) approaches that rely on formalized requirements** have been compared in, e.g., [12,16,17]. However, there is a lack of benchmarking across their compliance verification solutions. This is likely due to the fact that most of them are not open source (e.g., Regorous, a compliance checker based on Process Compliance Logic [55], or the semantic approach using the LegalRuleML model [56]), or depend on complex and restrictive input formats (e.g., modeling business process with Colored Petri Nets and the requirements with Computation Tree Logic [57], or using a mashup-based framework [58]). In practice, many approaches mainly verify control-flow aspects, with only a few extending the analysis to additional dimensions such as data. A notable exception is [59], which considers data constraints and reports improved performance over NuXMV [60] in mitigating state-space explosion. Moreover, symbolic approaches are affected not only by expressivity limitations but also by a high modeling and formalization effort [3], which limits flexibility in the presence of changing requirements. Since most verification approaches are tightly coupled with the expressiveness of their underlying formalism or modeling notation, their capabilities are inherently limited by what can be formally represented. In [12], the authors examine the

expressiveness of multiple compliance requirement languages, comparing them against a collection of nine requirements. We provide, in our online repository,[2] the formalized representation of each requirement using our notation (see Section 4.1), demonstrating that its semantics are expressive enough to capture the necessary compliance-relevant aspects identified in [12]. This step is included because the way requirements are represented sets the boundaries for how changes can be represented.

7. Discussion

Despite the promising results of *RC4PC* presented in Section 6, it currently has the following conceptual and practical limitations.

7.1. Limitations of the implementation

One limitation brought by the implementation, specifically in Step 1 (i.e., the automatic formalization of requirements), concerns representation inconsistencies between different versions of the same requirement. Although the two versions often share certain fields or elements (e.g., actors, activities, preconditions), these were sometimes formalized differently, both in terms of labeling and compliance patterns. This contradicts the expectation that shared, unchanged parts should be represented in the same way across versions. In the current setup, both versions are formalized at the same time, which can lead to such inconsistencies. In a real-world scenario, the first version might already be formalized independently (e.g., for completeness checking), and an LLM could then be instructed to formalize the second version based on the first. This would encourage the reuse of modeling patterns and help ensure consistency in the representation of unchanged elements.

Related trade-offs also emerged in how process models are handled in Step 3. Previous work on retrieving or querying process models via LLMs [38] relies on simplified abstract representations of process models, such as `Mermaid.js`, instead of using full BPMN 2.0 models. This is enabled by a converter that transforms detailed BPMN files (in `.bpmn` or `.json` format) into `minimal .json`, `.yaml`, or `Mermaid.js`.[6] While such abstractions reduce complexity and make models easier for LLMs to handle, they are not suitable when the goal is to verify process compliance, since critical details, such as time, resources, or control-flow semantics, can be lost in the simplification. At the same time, LLMs are not able to properly process embedded elements in BPMN 2.0 (e.g., resource pools), which motivates the design decision in RC4PC to provide the process model as a PDF in Step 3. This solution still requires improvement because of the high costs of sending PDFs to the LLM. Possible optimizations include converting the PDF to PNG if image processing proves cheaper, or reducing the number of times the process model is sent by emulating prompt caching. Moreover, using PDFs as input could also make the approach more *process-model-format agnostic*, since the visual representation abstracts from the underlying notation. This would allow the same mechanism to be applied to other process abstractions, such as Petri Nets [61] or Process Structure Trees [5], which could be explored in future work.

The choice of BPMN as format for the process models is mainly motivated by BPMN being the standard notation and being understandable for users. This is especially relevant if non-technical users, e.g., domain experts, are supposed to assess the compliance analysis output, including the model [32]. However, BPMN offers a rich palette of modeling constructs with only partially defined operational semantics. Currently, the LLM deals with this aspect. However, a formal representation such as Process Structure Trees as output format would add a clear semantics of the process model and compliance analysis results. Hence, we plan to integrate *RC4PC* into our conversational process modeling approach using Process Structure Trees that provide guarantees of soundness and block-structuredness (see autobpmn.ai).

Beyond choices related to the representation of process models, recent work has emphasized the importance of process model understanding [62]. Tasks such as translating text into a model [33] (or code [34]) and vice versa [32] (cf. Section 2) implicitly assess how well LLMs comprehend process semantics. However, these tasks have traditionally focused almost exclusively on control-flow, leaving other dimensions such as data flow or resource assignment, underexplored. More recent studies begin to address this gap by explicitly investigating LLM's ability to reason about data- and resource-related aspects of processes [62]. These results show that LLMs benefit from their broad pre-training and perform well on general-domain benchmarks. Other studies indicate that providing LLMs with explicit process knowledge, such as process rules, improves their ability to follow process logic and reduces errors during execution [63]. In a related direction, unsupervised evaluation based on round-trip correctness has been proposed to assess whether models preserve semantics across transformations, rather than relying on lexical similarity or pattern-based matching alone [64].

Usage of LLMs in general. We opt for LLM-based implementation of Steps 1 and 3 due to their strengths in dealing with textual information, including the provision of feedback about compliance deviations to users. Step 1 could also benefit from LLM robustness against noise in regulatory documents. Studies such as [65] show that robustness depends on different factors such as zero-shot detector, length of the text, and formal/yinformal writing style. Despite the need to investigate this point further, the question remains how robust alternative approaches are. Aside noise, also ethical issues with LLMs arise, *"including bias and fairness, privacy and data security, misinformation and disinformation, transparency and accountability, intellectual property and plagiarism, access and inequality"* [66]. Several quality metrics for the results of LLM-generated process models and descriptions have been proposed, e.g., [32], also from regulatory texts such as GDPR [67]. We assess the stability of the results in this work, but we also need to employ and possibly suggest further quality metrics, especially in the area of regulatory compliance.

7.2. Limitations of the datasets

This study represents the first attempt to investigate compliance deviations resulting from changes in requirements within process models, and, as such, no dat sets was available containing multiple versions of the same set of requirements. The experimental setup assumed a one-to-one evolution, where a single requirement could evolve into a newer version of itself; however, more complex evolution scenarios, such as one requirement splitting into several, or multiple requirements being consolidated into one, were not addressed. The use of real-world datasets and the analysis of requirement changes over time could enable the identification of such cases, facilitate the discovery of recurring change patterns and their associated compliance deviations, and support the prediction of which requirements are more likely to change and which parts of a process model are most susceptible to their impact.

7.3. Requirement representation challenges

The deontic-based requirement representation adopted in this work does not account for hierarchical relationships or inter-dependencies between requirements, primarily due to the difficulty of extracting such structures using LLMs, as highlighted in previous work on formal rules extraction [49]. In practice, requirements often form defeasible hierarchies within the same regulatory version. For example, in a customer service process, a general requirement like "respond to all

[6] https://github.com/com-pot-93/bpmn-converter/tree/main, last access 03 February 2026.

complaints within 5 days" may be overridden by a more specific one such as "respond to urgent complaints within 2 days". The general rule is defeasible and defeated when the specific condition applies, forming a hierarchy between requirements within the same version. Similar limitations arise in contrary-to-duty scenarios [68], where secondary obligations become applicable only after a violation of a primary one (e.g., sending an apology after a response deadline has been missed). Such cases are currently handled using preconditions to indicate when a requirement applies, but this may not be sufficient for more complex or interdependent requirements. While this supports basic applicability distinctions, it does not fully capture richer normative structures such as explicit priority relations, defeasibility, or obligations with in-force intervals. Before adding such complexity, it is essential to first show that the change hierarchy and *RC4PC* approach help users detect compliance deviations and understand their causes.

The change hierarchy proposed in RC4PC was designed to be independent of any specific requirement representation. Although it is demonstrated using our deontic-based representation, the change hierarchy can be applied to other requirement representations, such as Abstract Syntax Trees (ASTs) [5], Process Compliance Language (PCL) [69], or Declare [70]. The end goal is to enable compliance change management in a formalism-agnostic way. While this level of abstraction was taken into account during the hierarchy's design, the current evaluation is limited to the proposed requirement representation, which constrains the generalizability of the empirical findings. Extending it to other formalisms remains a promising direction for future work.

7.4. Implications of assuming initial process compliance

One limitation arises because RC4PC assumes that the initial business process model is fully compliant with the original version of the requirements. This assumption is essential to isolate deviations introduced by the requirement change from those that might have already existed. To address this situation, RC4PC could be easily extended to handle cases where initial compliance is unknown. A preliminary step could be added, which involves using a compliance-checking algorithm to identify any pre-existing deviations between the process model and the initial requirements. After this assessment, RC4PC can be applied to identify new or resolved deviations that result specifically from the updated requirements. For example, if a timing constraint is relaxed, an activity that was previously non-compliant may become compliant in the updated version. While these results are promising, the method's effectiveness also depends on how accurately changes in requirements are identified and interpreted, as discussed next.

7.5. Challenges in interpreting and tracing requirement changes

A challenge lies in identifying what has been added, deleted, or modified in the set of requirements. To reduce complexity for RC4PC, this information is assumed to be known, specifying which requirements were added, which were deleted, and which were modified. This assumption was introduced since distinguishing between a modified requirement vs. a case where one requirement is deleted, and another quite similar one is added is not always straightforward. For example, if a requirement originally states that *Activity A must be completed within 2 h* and is later replaced by *Activity A must be completed within 4 h*, this would typically be classified as a modification. In contrast, if the new requirement instead refers to *Activity B* with a different purpose or context, it may be more appropriate to treat the original as deleted and the new one as added. Defining clear criteria for this differentiation is essential, as it directly impacts the interpretation of compliance deviations and the traceability of requirement evolution.

Closely related to this is the open question of how to determine which requirement has evolved into which, especially in the absence of explicit versioning information. In practice, this is often addressed through tracking systems that maintain links between different versions of the same requirement. For this reason, the question was not in the scope of this paper. Nonetheless, it remains relevant, particularly when individual requirements encapsulate multiple norms. As observed in this paper, determining which specific norms have evolved into others presents a similar challenge and deserves further investigation in future work. For example, consider a requirement stating: *Activity A must be completed within 2 h and approved by a supervisor*. If the new version states: *Activity A must be completed within 4 h and logged in the system*, it becomes unclear whether the time constraint was modified and the approval obligation was removed, or whether the entire requirement was replaced by a fundamentally different one with distinct normative content.

In addition, while requirement evolution was assumed to be known in this work, the relationship between individual requirements and the elements of the process model was not. Ideally, a tracking system should not only be applied to requirements themselves but should also establish explicit links between requirements and process model elements. Without such connections, it is not possible to state with certainty whether a process model becomes over-compliant after a change due to a lack of knowledge on whether a specific part of the model was implemented to cover a regulatory requirement or whether it reflects an inherent aspect of the process that cannot be relaxed. For example, a process might include a safety check before starting a machine. If the corresponding requirement is removed, the process model may appear being over-compliant. However, this step may still be indispensable for physical safety reasons, meaning it cannot be eliminated without compromising the system's functioning.

8. Conclusion

This paper presents RC4PC, an approach for automatically analyzing the impact of regulatory requirement changes on business process compliance by combining requirement formalization, hierarchical change representation, and impact analysis to identify and reason about compliance deviations, i.e., non-compliance and over-compliance. The approach leverages LLMs to reduce manual effort in repeatedly interpreting and re-formalizing changing regulatory requirements. The evaluation across three domains demonstrates that RC4PC correctly identifies the compliance deviations associated with changes, although inconsistencies in representation occasionally reduced precision. More precisely, all changes which had no compliance impact were correctly identified, nearly all compliance deviations were correctly detected, i.e., correct deviation type, reason, and reference on the process model, and the changes referred to all four process dimensions (i.e., control flow, time, data, resources), and the results did not favor any particular dimension. On the downside, in some cases, the same part of the requirement was formalized using different compliance patterns and activity labels, even though it did not change. To extend its applicability, future work will address more complex forms of requirement evolution, where a requirement splits into multiple new ones, or multiple old requirements are consolidated into one. With access to real-world datasets, RC4PC could support the discovery of recurring change patterns and enable predictive compliance analysis. Possible extensions also include user studies to evaluate usability and interpretability, the development of mitigation strategies for non-compliant cases, and optimization techniques for addressing over-compliance.

CRediT authorship contribution statement

Marisol Barrientos: Writing – original draft, Software, Investigation, Formal analysis, Data curation. **Karolin Winter:** Writing – review & editing, Supervision, Conceptualization. **Stefanie Rinderle-Ma:** Writing – review & editing, Supervision, Project administration, Funding acquisition, Conceptualization.

Declaration of Generative AI in the Manuscript Preparation Process

During the preparation of this work, the authors used GPT-4.1 for spell checking and formatting tables. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work has been partly funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), Germany – project number 514769482.

Data availability

A link to the data is provided in the paper.

References

[1] M. Hashmi, G. Governatori, H. Lam, M.T. Wynn, Are we done with business process compliance: state of the art and challenges ahead, Knowl. Inf. Syst. 57 (1) (2018) 79–133.
[2] L.T. Ly, F.M. Maggi, M. Montali, S. Rinderle-Ma, W.M.P. van der Aalst, Compliance monitoring in business processes: Functionalities, application, and tool-support, Inf. Syst. 54 (2015) 209–234.
[3] H.A. López, T.T. Hildebrandt, Three decades of formal methods in business process compliance: A systematic literature review, 2024, CoRR arXiv:2410.10906.
[4] S. Rinderle-Ma, K. Winter, J. Benzin, Predictive compliance monitoring in process-aware information systems: State of the art, functionalities, research directions, Inf. Syst. 115 (2023) 102210.
[5] J. Loebbecke, J. Mangler, S. Rinderle-Ma, Tree-based compliance verification: Bridging the gap between compliance requirements and process execution, in: Conceptual Modeling, 2025, pp. 185–203.
[6] C. Sai, K. Winter, E. Fernanda, S. Rinderle-Ma, Detecting deviations between external and internal regulatory requirements for improved process compliance assessment, in: Advanced Information Systems Engineering, 2023, pp. 401–416.
[7] F.M. Maggi, A. Marrella, G. Capezzuto, A. Armas-Cervantes, Explaining non-compliance of business process models through automated planning, in: C. Pahl, M. Vukovic, J. Yin, Q. Yu (Eds.), Service-Oriented Computing - 16th International Conference, ICSOC 2018, Hangzhou, China, November 12-15, 2018, Proceedings, in: Lecture Notes in Computer Science, Vol. 11236, Springer, 2018, pp. 181–197.
[8] J. vom Brocke, A. Hevner, A. Maedche, Introduction to design science research, in: Design Science Research. Cases, Springer, 2020, pp. 1–13.
[9] J. Mendling, H. Leopold, H. Meyerhenke, B. Depaire, Methodology of algorithm engineering, ACM Comput. Surv. 58 (4) (2025).
[10] L. Pufahl, F. Zerbato, B. Weber, I. Weber, BPMN in healthcare: Challenges and best practices, Inf. Syst. 107 (2022) 102013.
[11] J. Hage, Donald Nute (ed.), defeasible deontic logic, Artif. Intell. Law 8 (1) (2000) 75–91.
[12] A. Zasada, M. Hashmi, M. Fellmann, D. Knuplesch, Evaluation of compliance rule languages for modelling regulatory compliance requirements, Software 2 (1) (2023) 71–120.
[13] K. Winter, H. van der Aa, S. Rinderle-Ma, M. Weidlich, Assessing the compliance of business process models with regulatory documents, in: Conceptual Modeling, 2020, pp. 189–203.
[14] W. Labio, H. Garcia-Molina, Efficient snapshot differential algorithms for data warehousing, in: Very Large Data Bases, 1996, pp. 63–74.
[15] S. Rinderle, M. Reichert, M. Jurisch, U. Kreher, On representing, purging, and utilizing change logs in process management systems, in: Business Process Management, 2006, pp. 241–256.
[16] A.M. Mustapha, O.T. Arogundade, S. Misra, R. Damasevicius, R. Maskeliunas, A systematic literature review on compliance requirements management of business processes, Int. J. Syst. Assur. Eng. Manag. 11 (3) (2020) 561–576.
[17] M. Hashmi, G. Governatori, A methodological evaluation of business process compliance management frameworks, in: Asia Pacific Business Process Management, 2013, pp. 106–115.
[18] S. Dunzer, A. Liessmann, M. Stierle, M. Matzner, Conceptualizing business process violations, in: Process Mining Workshops, 2024, pp. 1–14.
[19] N. Adams, A. Augusto, M. Davern, M. La Rosa, Addressing the contemporary challenges of business process compliance, Bus. Inf. Syst. Eng. (2025) 1–24.
[20] M. Barrientos, K. Winter, J. Mangler, S. Rinderle-Ma, Verification of quantitative temporal compliance requirements in process descriptions over event logs, in: Advanced Information Systems Engineering, 2023, pp. 417–433.
[21] H. Mustroph, M. Barrientos, K. Winter, S. Rinderle-Ma, Verifying resource compliance requirements from natural language text over event logs, in: Business Process Management, 2023, pp. 249–265.
[22] H. Mustroph, K. Winter, S. Rinderle-Ma, Social network mining from natural language text and event logs for compliance deviation detection, in: M. Sellami, M. Vidal, B.F. van Dongen, W. Gaaloul, H. Panetto (Eds.), Cooperative Information Systems - 29th International Conference, CoopIS 2023, Groningen, the Netherlands, October 30 - November 3, 2023, Proceedings, in: Lecture Notes in Computer Science, Vol. 14353, Springer, 2023, pp. 347–365.
[23] X. Sun, S. Yang, C. Zhao, D. Yu, Design-time business process compliance assessment based on multi-granularity semantic information, J. Supercomput. 80 (4) (2024) 4943–4971.
[24] J. Chen, S. Pang, M. Xi, T. Zhao, S. Deng, J. Yin, Service regulation analysis framework for service design time: A case study of internet healthcare service, IEEE Trans. Serv. Comput. 17 (5) (2024) 2876–2889.
[25] R. Lu, S. Sadiq, G. Governatori, Compliance aware business process design, in: Business Process Management Workshops, 2007, pp. 120–131.
[26] S. Abualhaija, M. Ceci, N. Sannier, D. Bianculli, L.C. Briand, D.A. Zetzsche, M. Bodellini, AI-enabled regulatory change analysis of legal requirements, in: Requirements Engineering Conference, 2024, pp. 5–17.
[27] T. Seyffarth, S. Kühnel, Maintaining business process compliance despite changes: a decision support approach based on process adaptations, J. Decis. Syst. 31 (3) (2022) 305–335.
[28] E. Kempe, S. Semsar, A. Massey, S. Sampath, C. Seaman, Modeling, analyzing and communicating regulatory ambiguity: An empirical study, in: IEEE/ACM Workshop on Multi-Disciplinary, Open, and RElevant Requirements Engineering, 2024, pp. 28–34.
[29] G. Schumann, J.M. Gómez, Detection of conflicts, contradictions and inconsistencies in regulatory documents: A literature review, in: Intelligent Data Science Technologies and Applications, 2024, pp. 81–88.
[30] M. Barrientos, K. Winter, S. Rinderle-Ma, Automatic extraction and formalization of temporal requirements from text: A survey, in: Enterprise Design, Operations, and Computing, 2024, pp. 259–278.
[31] S. de Kinderen, K. Winter, Towards taming large language models with prompt templates for legal GRL modeling, in: Enterprise, Business-Process and Information Systems Modeling, 2024, pp. 213–228.
[32] N. Klievtsova, J. Benzin, T. Kampik, J. Mangler, S. Rinderle-Ma, Conversational process modelling: State of the art, applications, and implications in practice, in: Business Process Management Forum, 2023, pp. 319–336.
[33] N. Klievtsova, J. Mangler, T. Kampik, S. Rinderle-Ma, Utilizing process models in the requirements engineering process through Model2Text transformation, in: G. Liebel, I. Hadar, P. Spoletini (Eds.), 32nd IEEE International Requirements Engineering Conference, RE 2024, Reykjavik, Iceland, June 24-28, 2024, IEEE, 2024, pp. 205–217.
[34] F. Monti, F. Leotta, J. Mangler, M. Mecella, S. Rinderle-Ma, NL2ProcessOps: Towards LLM-guided code generation for process execution, in: A. Marrella, M. Resinas, M. Jans, M. Rosemann (Eds.), Business Process Management Forum - BPM 2024 Forum, Krakow, Poland, September 1-6, 2024, Proceedings, in: Lecture Notes in Business Information Processing, Vol. 526, Springer, 2024, pp. 127–143.
[35] S. de Kinderen, Q. Ma, J. Silva Mercado, K. Winter, The interim experiment report: A systematic account of the experimental design of large language model-based text-to-model approaches, in: H.-G. Fill, Y. Wautelet, J. Ralyté, J. Zdravkovic (Eds.), The Practice of Enterprise Modeling, Springer Nature Switzerland, Cham, 2026, pp. 102–120.
[36] B. Weber, M. Reichert, S. Rinderle-Ma, Change patterns and change support features - enhancing flexibility in process-aware information systems, Data Knowl. Eng. 66 (3) (2008) 438–466.
[37] S. Rinderle-Ma, M. Reichert, B. Weber, On the formal semantics of change patterns in process-aware information systems, in: Conceptual Modeling, 2008, pp. 279–293.
[38] N. Klievtsova, T. Kampik, J. Mangler, S. Rinderle-Ma, Conversationally actionable process model creation, in: Cooperative Information Systems, 2024, pp. 39–55.
[39] S. Hassani, M. Sabetzadeh, D. Amyot, J. Liao, Rethinking legal compliance automation: Opportunities with large language models, in: Requirements Engineering Conference, 2024, pp. 432–440.
[40] F. Venneri, L.B. Brown, F. Cammelli, E.R. Haut, Safe surgery saves lives, in: Textbook of Patient Safety and Clinical Risk Management, Cham, 2021, pp. 177–188.
[41] R. Hilpinen, Deontic logic, in: The Blackwell Guide To Philosophical Logic, John Wiley & Sons, Ltd, 2017, pp. 159–182.
[42] M. Hashmi, G. Governatori, M.T. Wynn, Normative requirements for regulatory compliance: An abstract formal framework, Inf. Syst. Front. 18 (3) (2016) 429–455.

[43] T. Voglhofer, S. Rinderle-Ma, Collection and elicitation of business process compliance patterns with focus on data aspects, Bus. Inf. Syst. Eng. 62 (4) (2020) 361–377.
[44] S. Abualhaija, M. Ceci, N. Sannier, D. Bianculli, D.A. Zetzsche, M. Bodellini, Toward automated change impact analysis of financial regulations, in: Workshop on Software Engineering Challenges in Financial Firms, 2024, pp. 31–32.
[45] R. Etezadi, S. Abualhaija, C. Arora, L.C. Briand, Classification or prompting: A case study on legal requirements traceability, 2025, CoRR arXiv:2502.04916.
[46] D. Leffingwell, D. Widrig, Managing Software Requirements: a Unified Approach, Addison-Wesley Longman Publishing Co. Inc., 1999.
[47] S. Jayatilleke, R. Lai, A systematic review of requirements change management, Inf. Softw. Technol. 93 (2018) 163–185.
[48] J. Loebbecke, S. Rinderle-Ma, A framework for assessing overcompliance and undercompliance in business processes, in: Cooperative Information Systems, 2025.
[49] E. Horner, C. Mateis, G. Governatori, A. Ciabattoni, Toward robust legal text formalization into defeasible deontic logic using LLMs, 2025.
[50] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, D.C. Schmidt, A prompt pattern catalog to enhance prompt engineering with ChatGPT, 2023, CoRR arXiv:2302.11382.
[51] P. Törnberg, Best practices for text annotation with large language models, 2024, CoRR arXiv:2402.05129.
[52] H. Wei, Y. Sun, Y. Li, DeepSeek-OCR: Contexts optical compression, 2025, CoRR arXiv:2510.18234.
[53] S. Agostinelli, F.M. Maggi, A. Marrella, F. Sapio, Achieving GDPR compliance of BPMN process models, in: CAiSE Forum 2019, 2019, pp. 10–22.
[54] X. Wang, J. Wei, D. Schuurmans, Q.V. Le, E.H. Chi, S. Narang, A. Chowdhery, D. Zhou, Self-consistency improves chain of thought reasoning in language models, in: Learning Representations, OpenReview.net, 2023.
[55] G. Governatori, S. Shek, Regorous: a business process compliance checker, in: Artificial Intelligence and Law, ACM, 2013, pp. 245–246.
[56] G. Governatori, M. Hashmi, H. Lam, S. Villata, M. Palmirani, Semantic business process regulatory compliance checking using LegalRuleML, in: Knowledge Engineering and Knowledge Management, 2016, pp. 746–761.
[57] H. Groefsema, N.R.T.P. van Beest, A. Armas-Cervantes, Efficient conditional compliance checking of business process models, Comput. Ind. 115 (2020) 103181.
[58] Á. Bernal, F. Montero, C. Cabanillas, P. Fernandez, M. Resinas, STATUS: a low-code business process compliance management system, in: BPM Best Dissertation Award, Doctoral Consortium, and Demonstration & Resources Forum, CEUR-WS.org, 2024, pp. 141–145.
[59] S.C. Tosatto, H. Burke, N. van Beest, H. Groefsema, A first and fast symbolic approach for data-aware business process compliance checking, Inf. Softw. Technol. (2026) 108047.
[60] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, S. Tonetta, The nuXmv symbolic model checker, in: A. Biere, R. Bloem (Eds.), Computer Aided Verification - 26th International Conference, CAV 2014, Held As Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings, in: Lecture Notes in Computer Science, Vol. 8559, Springer, 2014, pp. 334–342.
[61] C.G. Cassandras, S. Lafortune (Eds.), Petri nets, in: Introduction To Discrete Event Systems, Springer US, Boston, MA, 2008, pp. 223–267.
[62] H. Kourani, A. Berti, J. Henrich, W. Kratsch, R. Weidlich, C. Li, A. Arslan, D. Schuster, W.M.P. van der Aalst, Leveraging large language models for enhanced process model comprehension, 2024, CoRR arXiv:2408.08892.
[63] S. Kaltenpoth, A. Skolik, O. Müller, D. Beverungen, A step towards cognitive automation: Integrating LLM agents with process rules, in: A. Senderovich, C. Cabanillas, I. Vanderfeesten, H.A. Reijers (Eds.), Business Process Management - 23rd International Conference, BPM 2025, Seville, Spain, August 31 - September 5, 2025, Proceedings, in: Lecture Notes in Computer Science, Vol. 16044, Springer, 2025, pp. 308–324.
[64] M. Allamanis, S. Panthaplackel, P. Yin, Unsupervised evaluation of code LLMs with round-trip correctness, in: Forty-First International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024, OpenReview.net, 2024.
[65] J. Wu, R. Zhan, D.F. Wong, S. Yang, X. Yang, Y. Yuan, L.S. Chao, DetectRL: Benchmarking LLM-generated text detection in real-world scenarios, in: Advances in Neural Information Processing Systems, 2024.
[66] J. Jiao, S. Afroogh, Y. Xu, C. Phillips, Navigating LLM ethics: advancements, challenges, and future directions, AI Ethics 5 (6) (2025) 5795–5819.
[67] C. Sai, S. Rinderle-Ma, Agentic generation of process models from regulatory texts, in: Cooperative Information Systems, 2025.
[68] J. Hansen, G. Pigozzi, L.W.N. van der Torre, Ten philosophical problems in deontic logic, in: G. Boella, L.W.N. van der Torre, H. Verhagen (Eds.), Normative Multi-Agent Systems, 18.03. - 23.03.2007, in: Dagstuhl Seminar Proceedings, Vol. 07122, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.
[69] G. Governatori, A. Rotolo, A conceptually rich model of business process compliance, in: Asia-Pacific Conference on Conceptual, Vol. 110, 2010, pp. 3–12.
[70] W.M. Van Der Aalst, M. Pesic, DecSerFlow: Towards a truly declarative service flow language, in: Workshop on Web Services and Formal Methods, 2006, pp. 1–23.