applied sciences

MDPI

Article

A Hybrid Framework for LLM-Assisted Concept Drift Analysis in Process Mining

Zineb Errahal * and Hanae Sbai

Mathematics, Computer Science and Application Laboratory, Faculty of Sciences and Techniques of Mohammedia, Hassan II University of Casablanca, Mohammedia 28806, Morocco; hanae.sbai@fstm.ac.ma
* Correspondence: zineb.errahal-etu@etu.univh2c.ma

Abstract

Business processes evolve continuously over time. Variations in workload, organizational practices, seasonality, or external conditions may alter the way a process is executed, making concept drift (CD) an important issue in process mining (PM). Existing drift detection methods locate changes but say little about what happened and offer no natural-language interaction, which limits their use by non-experts. Large Language Models (LLMs) show strong capabilities in natural language, but their output may lack support from the event data. We propose a hybrid framework for drift detection in PM by combining PM techniques and LLM capabilities. The event log is first divided into temporal non-overlapping windows of configurable size (40 for the synthetic log, 80 for the real-life log). Sublogs are abstracted and then compared by an LLM to extract candidate changes, which are validated by Kolmogorov–Smirnov and Mann–Whitney tests ($\alpha = 0.05$). Validated changes are stored in a Change Log and organized into a classified Change Tree, which is used as the knowledge source of a Retrieval-Augmented Generation (RAG) component for natural-language exploration. On a synthetic log, the injected Risk Assessment insertion was validated ($p < 0.001$), and ProDrift localized a drift at the same position. On the real-life SEPSIS log, three changes were reported and compared with ProDrift results.

Keywords: process mining; concept drift; Large Language Models; retrieval-augmented generation; process change analysis

1. Introduction

Business processes continuously evolve in response to organizational, technological, and operational changes. To remain effective in such dynamic environments, organizations must be able to adapt their operational processes accordingly [1]. These processes are often supported by information systems that connect people, applications, and process models [2]. During process execution, information systems record activities as event logs, providing valuable information about how processes actually behave. Process mining (PM) uses these event logs to discover processes, check conformance, and improve their execution [3].

Many PM techniques assume that the underlying process remains relatively stable. In practice, however, process behavior may evolve over time. In PM, concept drift (CD) refers to such changes in the underlying process. Its analysis involves detecting when a change occurs, localizing the affected behavior, and understanding the nature of the observed evolution from event data [1]. Detecting and analyzing these changes is important because

Check for updates

Academic Editors: Kingsley Okoye and Julius Nganji

Received: 24 September 2026
Revised: 20 October 2026
Accepted: 21 October 2026
Published: 23 October 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

an outdated view of the process may affect the reliability of process models and the conclusions derived from them.

Researchers have proposed several approaches for CD analysis in PM [1,4]. They include window-based techniques, statistical hypothesis testing, and distribution-based methods that compare process behavior over different time periods. Statistical tests such as the Kolmogorov–Smirnov (KS) test and the Mann–Whitney U (MW) test can, for example, provide evidence of changes in event data. Other studies focus on configurable processes and event-log preparation [5,6], while others investigate how detected changes can be represented and visualized [7]. These methods provide useful mechanisms for detecting process evolution. Still, interpreting statistical or structural evidence and connecting it to concrete changes in process behavior can require substantial PM expertise.

This creates a practical gap. The people who understand a business process best are not always familiar with specialized PM tools or statistical analysis. Natural-language interaction offers one possible way to make such analyses easier to explore. Instead of navigating only through technical indicators, users could also describe, query, and investigate process changes using language closer to the way domain experts reason about their processes.

Large Language Models (LLMs) provide new possibilities in this direction. Recent models have shown strong capabilities in understanding, generating, and reasoning over textual information [8]. Their use in PM is also receiving increasing attention. Berti et al. [8], for instance, proposed textual abstractions of PM artifacts that can be provided to LLMs.

Other studies address different questions. Teubner et al. [9] discuss the risks and the potential benefits of using LLMs for business and information systems research. Ceravolo et al. [10] review how machine learning methods must be adapted to the characteristics of event data, recommending that generic models cannot be applied to process data without care. Berti and Qafari [11] test several ways of prompting LLMs on PM artifacts, including the generation of database queries that check hypotheses against the event log. Berti et al. [12] propose benchmarks and evaluation strategies to demonstrate how well LLMs perform PM tasks. Finally, Kampik et al. [13] present Large Process Models, a solution that combines the language abilities of LLMs with the analytical precision of PM and knowledge-based systems.

However, none of these studies addresses CD analysis in PM, and the question of how the changes that an LLM describes between two periods can be checked against the event data remains open.

Using an LLM for this task also introduces an important problem. LLM outputs are probabilistic, and a model may describe a process change that appears plausible without sufficient support in the underlying event data. A change generated by an LLM should therefore not automatically be treated as a confirmed drift. It first needs to be checked against measurable process evidence. This motivates an approach in which the LLM helps identify and describe candidate changes, while quantitative methods provide an independent validation step. Indeed, previous research has emphasized that realizing business value from AI requires more than the adoption of AI technologies alone. Its effective use depends on complementary Business Process Management (BPM) capabilities that enable organizations to integrate AI into the analysis, management, and improvement of their business processes [14].

In this paper, we propose a hybrid framework for LLM-assisted CD analysis in PM. We divide the event log into consecutive sublogs, abstract them, and compare them using an LLM. We initially treat the changes identified during these comparisons as candidates rather than as confirmed drifts.

We then assess each candidate directly from the event data of the same two windows. We compare the frequency of the activities or relations that it involves using the KS and

MW tests, and we validate the candidate only if the change is significant and goes in the expected direction. In this way, the LLM contributes to describing what may have changed without becoming a standalone CD mechanism, in line with the vision of Kampik et al. [13].

The framework progressively organizes the resulting information into structured knowledge about process evolution. A Change Log records the identified changes together with their temporal information and the available evidence. The framework then consolidates the changes into a Change Tree that captures their occurrences, characteristics, and evolution over time. These artifacts are also used as the knowledge source for a Retrieval-Augmented Generation (RAG) layer. Users can therefore explore the analysis through natural-language questions, while the generated answers remain grounded in the evidence produced by the framework.

The main contributions of this work are as follows:

- A hybrid approach in which an LLM identifies candidate changes from consecutive process windows.
- A candidate-level validation mechanism that verifies each proposed change with two statistical tests on the corresponding activity or relation frequencies with an additional check of the expected direction is produced.
- A structured knowledge representation based on a Change Log and a Change Tree. These artifacts connect process changes with their temporal information, validation evidence, occurrences, and drift characteristics.
- A RAG-based exploration layer that uses the structured drift information as its knowledge source, allowing users to investigate process evolution and its supporting evidence through natural-language questions.

The framework does not aim to replace existing CD methods or human expertise. Instead, it combines complementary roles: LLMs support the identification and exploration of process changes, statistical analysis provides validation evidence, and structured representations preserve the resulting knowledge for subsequent interpretation.

The remainder of this paper is organized as follows. Section 2 introduces the main concepts related to PM, CD, LLMs, and RAG. Section 3 reviews related work on CD analysis and the use of LLMs in PM. Section 4 (Materials and Methods) presents the proposed framework, including candidate change identification, statistical validation, structured representation, and RAG-based exploration. Section 5 presents the experimental evaluation, and Section 6 discusses the results. Finally, Section 7 concludes the paper and outlines directions for future work.

2. Background

This section introduces the main concepts needed to understand the proposed framework. We first present PM and business process changes, followed by CD and Change Mining (CM). We then introduce LLMs and RAG, with a focus on the concepts that are directly used in this work.

2.1. Process Mining

PM is a discipline that combines data mining, process modeling, and process analysis to extract knowledge from event logs. Its main objective is to understand how processes are actually executed and to use this information for process discovery, conformance checking, and process improvement [15].

PM mainly relies on event logs, which record the execution of activities within a process. Each event is generally associated with a case and an activity, while additional information such as timestamps, resources, or other process-related attributes may also be recorded. These data provide the basis for reconstructing and analyzing process behavior.

Several formats have been introduced to represent event data. XES is widely used for conventional event logs, where events are associated with individual cases. More recently, OCEL 2.0 has extended this representation to object-centric event logs, allowing an event to be related to multiple objects of different types [16].

Based on event data, PM is commonly organized around three main tasks. Process discovery derives a process model from observed behavior. Conformance checking compares observed executions with expected or modeled behavior. Process enhancement uses information extracted from event data to extend or improve an existing process model.

Many traditional PM analyses assume that the underlying process remains relatively stable during the observation period. In practice, this assumption does not always hold. Business processes evolve as organizations, systems, and operating conditions change. Understanding this evolution is therefore an important part of process analysis.

2.2. Business Process Changes

A business process consists of a set of related activities and other process elements organized according to a given structure. A process change may affect these elements, the relationships between them, or the way they are executed. Such changes can occur at design time or during process execution [17].

Process changes are often associated with adaptation or re-engineering initiatives introduced to address new business requirements. They may affect different process perspectives, including activities, data, and resources [18]. In practice, a change may involve adding or removing an activity, modifying an existing behavior, or changing the order in which activities are executed.

Not every process change is explicitly documented by the information system. In many situations, its effect can only be observed through the execution data produced by the process. This makes event logs an important source for studying how process behavior evolves over time.

2.3. Concept Drift and Change Mining

CM focuses on extracting and analyzing information about process changes. When explicit Change Logs are available, they provide a useful source for understanding what was modified and how the process evolved [17]. However, many information systems do not record such changes explicitly [19]. In these situations, process evolution can instead be investigated from the behavior recorded in event logs.

CD broadly refers to changes in the underlying data distribution or in the relationships between variables over time, which may affect the performance and validity of models operating in evolving environments [20,21]. In the PM context, the objective is not limited to determining whether the process has changed. It also involves locating the change, identifying the affected behavior, and understanding how that behavior evolves.

CD is commonly described through four main patterns [1,22]:

- **Sudden drift:** a process behavior is replaced by another behavior within a relatively short period.
- **Recurring drift:** a previously observed behavior disappears and later reappears, often because of periodic or seasonal conditions.
- **Gradual drift:** old and new process behaviors coexist for a period while the new behavior progressively becomes dominant.
- **Incremental drift:** the process evolves through a sequence of smaller changes that progressively lead to a different behavior.

Different methods have been proposed to detect and analyze these patterns. Bose et al. [1], for example, introduced techniques for detecting, localizing, and characterizing pro-

cess changes through event-log analysis. Statistical methods and sliding-window strategies are frequently used to compare process behavior observed during different periods. Other approaches also address drift classification and localization [23].

Recent work has also investigated process deviations from an anomaly-detection perspective. Vitale et al. [24] proposed a PM-based feature extraction approach for detecting control-flow anomalies, including skipped, unknown, and wrongly ordered activities. Their approach combines alignment-based conformance checking with feature extraction and dimensionality reduction, while preserving the explainability of the detected deviations. Although anomaly detection and CD address different problems, this work highlights the value of connecting detected behavioral deviations with measurable process evidence rather than relying only on an isolated detection signal.

CD analysis therefore involves several related questions: when did the process change, what part of the process was affected, and how did the observed behavior evolve? Answering these questions usually requires combining evidence obtained from event data with information about the structure and execution of the process. These aspects are particularly important in this work, where we progressively identify, validate, and characterize the detected changes.

2.4. Large Language Models

LLMs are language models trained on large collections of textual data and designed to process and generate natural language. Their ability to represent contextual information allows them to perform tasks such as question answering, summarization, information extraction, and reasoning over textual descriptions [25].

The behavior of an LLM depends not only on the model itself but also on how information and instructions are provided to it. Prompting is therefore an important part of LLM-based applications. Instructions can define the role of the model, specify the task to perform, constrain the expected response, and provide contextual information needed for the analysis.

Another important limitation is the context window. Only a bounded amount of information can be processed within a single request. This becomes particularly relevant when working with event logs, which may contain thousands of traces and events. Directly providing a complete log to an LLM may therefore be impractical. Data abstraction and segmentation provide possible ways to reduce the amount of information processed at each step.

Generation parameters can also influence model behavior. Temperature, for example, controls the variability of generated responses. Lower values generally produce more deterministic outputs, which can be useful when the LLM is used for analytical tasks rather than creative generation [26]. The specific model and inference parameters used in our experiments are described later in the experimental setup.

Despite their capabilities, LLMs have important limitations. Their outputs are probabilistic and may contain information that appears reasonable but is not supported by the provided data. This issue is particularly important in CD analysis, where an incorrectly identified change may lead to a wrong interpretation of process evolution. In the proposed framework, we therefore treat LLM outputs as candidate changes rather than confirmed drifts. We perform the validation separately, using evidence extracted from the underlying event data.

2.5. Retrieval-Augmented Generation

RAG combines information retrieval with LLM-based generation. Instead of relying only on information encoded in the model parameters, a RAG system retrieves information

relevant to a user query from an external knowledge source and provides it as additional context to the LLM [27].

A typical RAG pipeline first divides the available information into retrievable units. These units are represented using embeddings and stored in a searchable index. When a user submits a question, the most relevant content is retrieved based on its similarity to the query. The retrieved information is then included in the context used by the LLM to generate the response.

This approach is particularly useful for knowledge-intensive applications, where answers should remain connected to a specific source of information. Grounding generation in retrieved evidence can reduce unsupported responses, although it does not completely eliminate them [27]. The quality of the final answer still depends on the information stored in the source and on the ability of the retrieval mechanism to select the relevant evidence.

Recent work has also shown that the organization of retrieved knowledge can play an important role in RAG-based reasoning. Zhang et al. [28] proposed TreeQA, where complex questions are decomposed through a hierarchical logic tree and reasoning is supported by evidence from structured and unstructured knowledge sources. Their results suggest that structured and verifiable reasoning paths can improve the reliability and interpretability of RAG-based question answering. Although the tree structure serves a different purpose in our work, this observation is relevant to our framework. Rather than directly retrieving isolated drift observations, we organize validated process changes into a Change Tree before making them available to the RAG layer. The tree therefore acts as a structured representation of process-evolution knowledge from which drift-related evidence can be retrieved and explained.

In the context of this work, RAG plays a specific role. The knowledge source is not built directly from the raw event log. Instead, it relies on the structured information produced during CD analysis. The Change Log records candidate and validated changes together with their supporting information, while the Change Tree organizes related changes and their characteristics over time. These artifacts provide the knowledge used by the retrieval layer, allowing users to query process evolution and its supporting evidence through natural language.

The concepts introduced in this section provide the background needed to understand the proposed approach. The next section reviews existing work on CD in PM and recent studies exploring the use of LLMs for process analysis.

3. Related Work

This section reviews the two research directions that are most closely related to our work. We first discuss existing approaches for CD analysis in PM. We then examine recent studies that use LLMs for PM and BPM. The section concludes by positioning our framework at the intersection of these two areas.

3.1. Concept Drift in Process Mining

CD has been studied in PM as a way to detect and understand changes in process behavior over time. Early work by Bose et al. [1] established several of the main challenges of this problem, including change-point detection, localization, and characterization. Their work also showed the importance of considering different drift patterns, such as sudden, gradual, recurring, and incremental changes.

A common strategy is to divide an event log into temporal windows and compare the behavior observed across these windows. Differences can then be evaluated using statistical tests, process relations, or other behavioral measures. Carmona and Gavaldà [4], for example, investigated online techniques for dealing with evolving process behavior.

Such approaches are useful when changes must be detected while the process is still running rather than only through retrospective analysis.

Other studies have focused on improving the preparation and organization of process information before drift analysis. The works presented in [5,6] investigated ways of supporting change analysis through event-log preparation and configurable process information. Beyond detection itself, representing the resulting changes is also important. The Change Tree proposed in [7], for instance, provides a structured way to organize process modifications and support their analysis.

Elkhawaga et al. [29] introduced the CONDA-PM framework based on a systematic review of existing approaches. The framework organizes CD analysis according to aspects such as drift types, affected process perspectives, and levels of granularity, while highlighting areas that remain insufficiently addressed by existing techniques. Sato et al. [30] subsequently provided a broader survey of CD detection in PM, systematizing existing approaches and highlighting remaining challenges in the field.

More recent research has continued to address the problem from different perspectives. De Sousa et al. [31] proposed an integrated approach for detecting and localizing CDs using trace clustering in both batch and streaming settings. Adams et al. [23] proposed explainable approaches to better localize and characterize detected changes. Behavior-oriented methods, in turn, attempt to capture changes that may not be visible through simple activity-frequency comparisons [32].

These studies have provided solid methods for identifying process evolution, but each family of approaches comes with trade-offs. Window-based statistical approaches, such as the one of Bose et al. [1], test whether features extracted from two windows differ significantly. They are well founded and efficient, but their results depend on the window size and on features that must be chosen in advance. Their output is a change point with a set of significant features rather than a description of what changed. Maaradji et al. [33] present ProDrift, which reduces the dependence on a fixed window, but they report a drift only after a detection delay. Ostovar et al. [34] produced process-tree fragments for characterization, yet an expert is still needed to interpret the result tree. Generic drift detectors from the data-stream literature [20] monitor changes in data distributions or in the error of a learner; they are domain independent, but they ignore the semantics of the process. Explainable approaches, such as the one by Adams et al. [23], go further by detecting drifts in several process perspectives and testing cause-effect relations between them. However, their explanations remain statistical relations and charts. In short, current tools are good at telling *when* and *where* a process changed, but they offer little semantic explanation of *what* changed, and none of them supports natural-language questions. Turning their results into an understandable account of how the process evolved still requires substantial PM knowledge.

This distinction is important for our work. We do not seek to replace these established detection principles. Instead, we retain data-driven analysis as a validation mechanism and investigate how LLMs can help to identify candidate changes, organize the resulting evidence, and make it easier to explore.

3.2. Large Language Models in Process Mining

The growing capabilities of LLMs have led to increasing interest in their use for PM and BPM, particularly for process enhancement and improvement [35]. At the same time, the growing availability of process data and advances in AI are creating new opportunities for systems that support process analysis, improvement, and innovation [36].

One of the main challenges when applying LLMs to PM is the size and structure of process data. Event logs and process models cannot always be provided directly to an LLM,

especially when they contain large numbers of traces or process elements. Berti et al. [8] addressed this problem by introducing textual abstractions for PM artifacts, including event logs, Directly Follows Graphs (DFGs), and process variants. These abstractions make process information more suitable for prompting and allow LLMs to answer questions or reason over process-related information. Process abstractions have also been investigated as an interface between process models and LLMs. Kourani et al. [37] systematically evaluated how different process model abstractions, prompting strategies, and LLMs affect process model comprehension. Their work further introduced a conversational assistant that enables users to query process models through natural language.

Related work has explored the automatic ingestion and interpretation of event data. Berti and Aalst [16] used LLMs to support the parsing of event logs from CSV data, reducing some of the manual effort required before process analysis. Benchmark-oriented studies have also started to evaluate more systematically what LLMs can and cannot do in process mining. Berti et al. [12], for example, examined different models and prompting strategies across several PM tasks, showing promising capabilities but also important variations, depending on the model, prompt, and task.

LLMs have also been investigated as interfaces for process knowledge. Conversational interaction can allow users to query event data or process artifacts without directly manipulating specialized analysis tools. Approaches such as that of Berti et al. [38] explore how reasoning models can be configured for process-related tasks, while other studies investigate conversational process exploration and natural-language interaction [39]. These directions are particularly relevant for domain experts, since they offer a different way to access process information. Another group of studies focuses on process modeling and explanation. Kourani et al. [40] investigated the ability of LLMs to generate business process models and evaluated the resulting models using structural and semantic criteria. Other work has examined declarative process models and the ability of LLMs to reason about constraints [41]. Fahland et al. [42] studied process explanations generated with LLM support and considered how these explanations are perceived by users.

At a broader level, Kampik et al. [13] introduced the vision of *Large Process Models*. Their proposal combines generative AI with symbolic process knowledge and considers the use of such models across several stages of the BPM lifecycle. This direction is particularly interesting because it moves beyond using an LLM only as a text generator and instead considers how process knowledge and generative models can complement each other.

RAG has recently started to appear in PM applications as well. For example, Casciani et al. [43] combine retrieval with LLMs for next-activity prediction. Relevant process information is retrieved and provided as context before prediction, illustrating how external process evidence can be used to ground an LLM-based task.

These studies also reveal limitations of generative approaches that are central to our problem. LLM outputs are probabilistic and may report some errors. Vidgof et al. [44] point to reliability issues and to the limited context window of LLMs as challenges for BPM, and Yao et al. [25] note that LLMs may produce hallucinated content, i.e., content that sounds plausible but is not supported by the input. Benchmarks such as the one by Berti et al. [12] confirm that the quality of the answers varies with the model, the prompt, and the task. In addition, event logs must be abstracted before being given to an LLM, so part of the information is lost. An LLM also cannot measure by itself whether a difference between two abstractions is significant. For drift analysis, this means that a change described by an LLM may be plausible but not supported by the event data. Few of the studies above check LLM outputs against the event log; Berti and Qafari [11] generate database queries to test hypotheses on the log, but not for the analysis of process change.

More directly related to our problem, recent work has started to combine statistical CD analysis with AI-driven reasoning [45]. This is an important development because it shows that LLM-supported interpretation is also becoming relevant to CD analysis itself. Our work follows this emerging direction but focuses on a different processing chain. We treat the changes identified from consecutive abstracted sublogs as candidates, validate them against the underlying event data, consolidate them into structured knowledge artifacts, and then use them as the evidence source for RAG-based exploration.

Table 1 summarizes representative studies and highlights the variety of roles currently assigned to LLMs in PM and BPM. The table is not intended to be exhaustive but rather to illustrate the diversity of LLM applications in PM.

Table 1. Representative Large language Model (LLM) applications in Process Mining (PM).

Reference	Main Idea	Target Task	LLM Contribution	Model Used	Method	PM Scope
[43]	Uses RAG-enhanced LLMs to improve next-activity prediction.	Predictive monitoring	Uses retrieved traces and contextual information during prediction.	Llama, Mistral, Qwen, GPT-4o-mini	RAG and in-context learning.	Predictive process monitoring
[37]	Investigates LLMs and process abstractions for process-model comprehension.	Process-model comprehension	Supports natural-language querying and reasoning over process models.	Multiple LLMs	Process abstractions, prompting strategies, and conversational assistance.	Analysis and comprehension
[40]	Evaluates LLM capabilities for process-model generation.	Process modeling	Generates formal process models from textual descriptions.	GPT-4, Claude 3.5, Gemini, Llama	Prompt engineering, benchmarking, and iterative refinement.	Discovery and modeling
[42]	Studies the quality and usefulness of LLM-supported process explanations.	Process explanation	Combines process information into natural-language explanations.	GPT-4	SAX4BPM framework and user study.	Analysis and explainability
[41]	Evaluates LLM capabilities for declarative process specifications.	Declarative PM	Translates between natural language and declarative process specifications.	Multiple LLMs	Bidirectional translation and quantitative evaluation.	Declarative process analysis
[38]	Investigates the configuration of large reasoning models using PM.	Process reasoning	Uses reasoning models to perform process-related analytical tasks.	Large reasoning models	Benchmark and case study.	Process analysis
[13]	Introduces the vision of Large Process Models for BPM.	BPM lifecycle support	Supports generation, querying, and reasoning over process knowledge.	General-purpose LLMs	Neuro-symbolic architecture.	Discovery, conformance, and prediction
[39]	Proposes a user-centric framework for using LLMs in PM.	User-oriented PM assistance	Provides role-specific insights and recommendations for PM users.	LLM-based	User-centric framework and case study.	Analysis and decision support
[16]	Uses LLMs to support the ingestion of CSV event logs for PM.	Event-log preparation	Supports the interpretation and transformation of CSV data for process analysis.	LLM-based	LLM-assisted CSV parsing.	Data preparation

Table 1. *Cont.*

Reference	Main Idea	Target Task	LLM Contribution	Model Used	Method	PM Scope
[8]	Introduces textual abstractions of process artifacts for LLM interaction.	Process querying	Supports natural-language querying and reasoning over process abstractions.	GPT-4	Process abstractions and prompting.	Discovery, conformance, and analysis
[12]	Evaluates LLM capabilities across different PM tasks.	Evaluation and bench-marking	Provides direct insights, code generation, and hypothesis generation.	GPT-4, Gemini	Comparative evaluation framework.	Discovery, analysis, and redesign
[44]	Explores opportunities for LLMs across the BPM lifecycle.	BPM lifecycle support	Supports summarization, interaction, and process-related assistance.	GPT family, OPT, BLOOM	Lifecycle-oriented exploratory study.	Identification, discovery, analysis, and monitoring

3.3. Research Gap and Positioning of This Work

The literature shows two research directions that are increasingly moving closer to each other. CD research provides established mechanisms for detecting and localizing changes from event data. LLM-oriented process mining research, in contrast, has mainly explored how process information can be represented, queried, generated, or explained through natural language. Recent work has begun to connect statistical drift analysis with LLM-supported reasoning, but this intersection is still at an early stage.

One important challenge remains. Although an LLM can identify plausible differences between two process representations, these differences are not necessarily supported by the underlying event log. Considering them directly as process drifts may therefore lead to unreliable conclusions. On the other hand, statistical methods can indicate that a change has occurred but usually provide little information about what actually changed or how analysts can examine the evidence.

The goal of our framework is to combine the strengths of these two approaches. Firstly, the LLM compares consecutive abstracted sublogs and identifies candidate process changes. We then test each candidate on the event data of the same two windows, using the frequency of the activities or relations that it involves. We validate it only if the change is significant and goes in the expected direction. The two components therefore provide complementary views of the same temporal transition: the LLM describes possible behavioral changes, while the data-driven component checks whether each of them is supported by measurable process evolution. Another key aspect of the proposed framework is how it handles this information. Instead of keeping statistical results and textual LLM outputs as isolated artifacts, the framework stores the detected changes and their supporting evidence in a structured Change Log and organizes them into a classified Change Tree. Together, these representations preserve occurrences, temporal information, evidence, and drift characteristics. They provide a structured view of process evolution that also serves as the knowledge base for the RAG component.

This positioning defines the main idea of the proposed hybrid framework: combining the flexibility of LLM-based analysis with an independent data-driven assessment and a persistent structured representation of process evolution.

4. Materials and Methods

This section presents the proposed method and the materials used for its evaluation. The framework combines PM, LLM-based analysis, statistical validation, and RAG. Its main purpose is to identify changes in process behavior while keeping a clear distinction between changes suggested by the LLM and changes supported by the event data.

The method follows a progressive analysis. First, we divide the event log into temporally ordered sublogs. We transform consecutive sublogs into textual process abstractions, and an LLM compares them to identify candidate changes. In parallel, we analyze the same window pairs directly from the event data, using structural and performance-related evidence. We then organize the resulting information into structured representations that users can explore through a RAG-based interface.

4.1. Problem Formulation and Framework Overview

Let an event log L contain a set of process traces:

$$L = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}, \tag{1}$$

where each trace σ_i represents one process execution and is composed of an ordered sequence of events:

$$\sigma_i = \langle e_1, e_2, \ldots, e_m \rangle. \tag{2}$$

Each event contains at least a case identifier, an activity name, and a timestamp. Additional attributes may also be available depending on the event log.

A process may evolve while it is being executed. As a result, traces recorded during one period may exhibit a different behavior from traces observed later. The objective of the proposed framework is to identify these differences, determine whether they are supported by the event data, and preserve the resulting information for further analysis.

To capture this temporal evolution, we divide the event log into k ordered sublogs:

$$L = \{L_1, L_2, \ldots, L_k\}. \tag{3}$$

Each L_i represents the behavior observed during one analysis window. The framework compares consecutive windows:

$$(L_1, L_2), (L_2, L_3), \ldots, (L_{k-1}, L_k). \tag{4}$$

For a transition from L_{i-1} to L_i, we denote the observed process change by

$$\Delta_i = \delta(L_{i-1}, L_i), \tag{5}$$

where δ represents the change in process behavior between two consecutive windows.

In practice, Δ_i is not directly known. The role of the framework is therefore to obtain evidence about this change through several complementary steps.

Figure 1 presents the overall architecture.

The analysis starts from the raw event log. Temporal segmentation produces smaller sublogs that preserve the chronological evolution of the process. The framework converts these sublogs into textual abstractions and provides them to the LLM. The model compares two consecutive process windows and proposes candidate changes. At this stage, a candidate is not considered a confirmed drift. This distinction is central to the proposed method. Each candidate change is then compared against the event data of the two windows from which it was generated. For each candidate, a different feature is computed according to its change type described by the LLM (the frequency of the activity for insertions, skips and replacements, and the frequency of the directly follows relation for reorderings), and the comparison is done with the KS and MW tests. This provides evidence that does not depend on the LLM output. The framework then stores the change information generated by the LLM and the data-driven evidence in the Change Log and consolidates them into a Change Tree. These structured artifacts form the knowledge base used by the final RAG component.

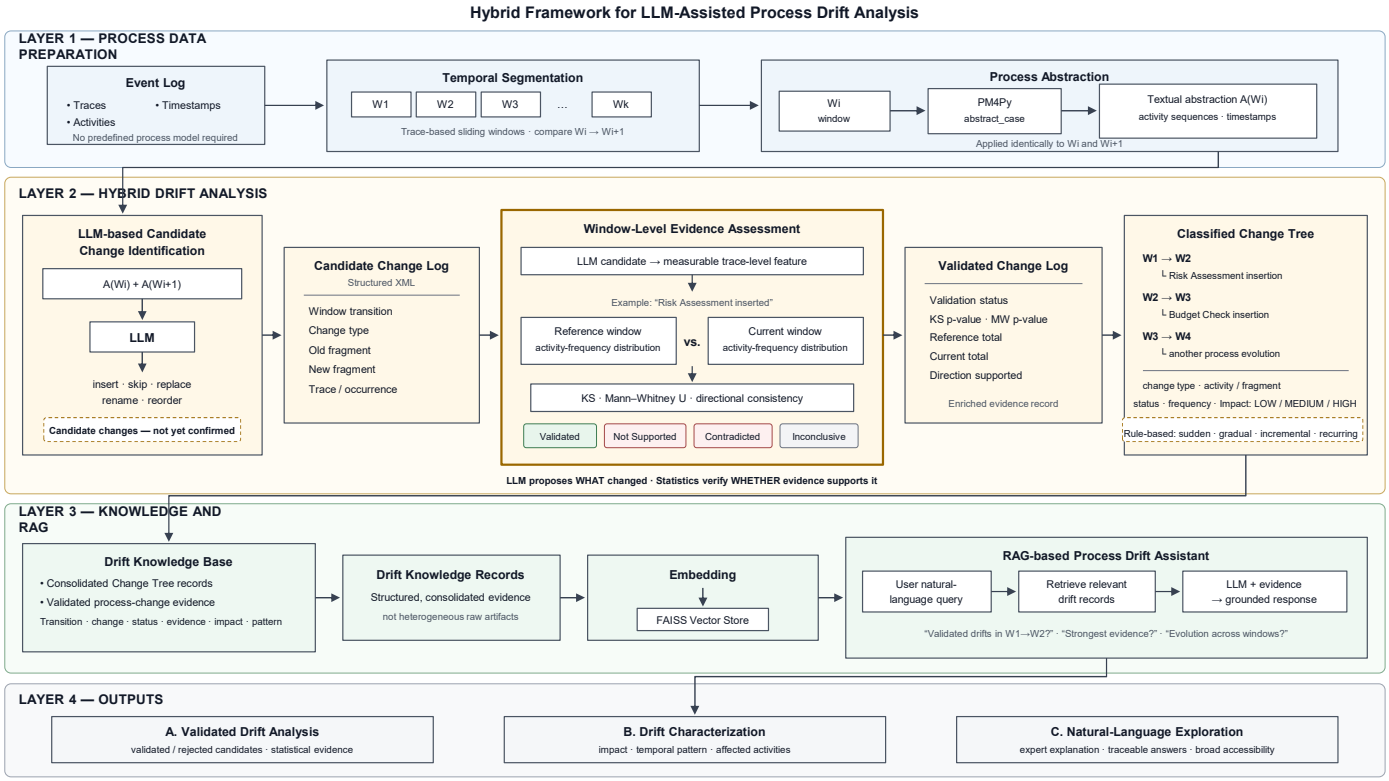


Figure 1. Overall architecture of the proposed hybrid framework for LLM-assisted Concept Drift (CD) analysis.

In this way, the framework assigns different roles to generative and data-driven components. The LLM helps describe what may have changed, while the algorithmic component assesses whether the corresponding temporal transition contains measurable evidence of process evolution.

4.2. LLM-Guided Process Change Extraction

Processing complete event logs directly with an LLM is often impractical. Real-life logs may contain thousands of traces and events, resulting in long and repetitive representations. The framework therefore works with temporal sublogs and compact textual abstractions.

Before creating the windows, we order the traces according to their temporal information. The window size determines how many traces are included in each sublog, while the step controls the progression of the analysis through the log. These parameters can be adjusted according to the size and characteristics of the analyzed process.

Following the abstraction principles introduced in [11], we transform each sublog into a textual representation suitable for LLM processing. In the current implementation, we mainly use `pm4py.llm.abstract_case`. The abstraction preserves the activity sequence and temporal information associated with each trace.

Let $A(L_i)$ denote the textual abstraction generated from sublog L_i. For each pair of consecutive windows, the LLM receives the abstractions of the reference and current sublogs. Candidate changes can then be represented as

$$C_i = \mathcal{G}(A(L_{i-1}), A(L_i)), \tag{6}$$

where $\mathcal{G}$ denotes the LLM-based comparison and C_i is the set of candidate changes proposed for the transition from L_{i-1} to L_i.

We do not prompt the model to return a simple drift/no-drift decision. Instead, we ask it to describe explicit behavioral differences. These may include activity insertion, skipping, replacement, renaming, or changes in activity ordering.

We use a structured prompt to make the comparison more consistent. It defines the role assigned to the LLM, provides analysis guidelines, and specifies the expected output format. Figure 2 illustrates part of this prompt.



Figure 2. Example of the structured prompt used for candidate change identification.

The framework records the generated candidates in a structured *Change Log*. For each window transition, it stores the proposed change type, the affected process fragment, trace information, timestamps, and the corresponding sublogs.

Figure 3 shows an example of this XML representation.

The entries contained in the Change Log represent candidate changes identified by the LLM. They are not considered sufficient evidence of drift on their own. The framework therefore complements them with an independent assessment of the corresponding window transition, as described next.

4.3. Evidence-Based Assessment and Change Structuring

The data-driven component operates at the level of two consecutive windows, rather than validating each candidate generated by the LLM individually. Its purpose is to determine whether the transition from L_{i-1} to L_i contains measurable evidence of process evolution. The current implementation combines structural indicators with a performance-related statistical signal.

```
<changes>
  <change>
    <reference_sublog_id>1</reference_sublog_id>
    <compared_sublog_id>2</compared_sublog_id>
    <type>skip_activity</type>
    <change_new />
    <change_old>Release A</change_old>
  </change>
  <change>
    <reference_sublog_id>1</reference_sublog_id>
    <compared_sublog_id>2</compared_sublog_id>
    <type>insert_activity</type>
    <change_new>Release B</change_new>
    <change_old />
  </change>
  <change>
    <reference_sublog_id>1</reference_sublog_id>
    <compared_sublog_id>2</compared_sublog_id>
    <type>reorder_pair</type>
    <change_new>Admission NC-&gt;Release B</change_new>
    <change_old>Release B-&gt;Admission NC</change_old>
  </change>
  <change>
    <reference_sublog_id>1</reference_sublog_id>
    <compared_sublog_id>2</compared_sublog_id>
    <type>insert_activity</type>
    <change_new>Admission IC</change_new>
    <change_old />
  </change>
  <change>
    <reference_sublog_id>1</reference_sublog_id>
    <compared_sublog_id>2</compared_sublog_id>
    <type>skip_activity</type>
    <change_new />
    <change_old>Admission NC</change_old>
  </change>
  <change>
    <reference_sublog_id>1</reference_sublog_id>
    <compared_sublog_id>2</compared_sublog_id>
    <type>insert_activity</type>
    <change_new>IV Liquid</change_new>
    <change_old />
  </change>
  <change>
    <reference_sublog_id>1</reference_sublog_id>
    <compared_sublog_id>2</compared_sublog_id>
    <type>insert_activity</type>
    <change_new>IV Antibiotics</change_new>
    <change_old />
  </change>
  <change>
    <reference_sublog_id>1</reference_sublog_id>
    <compared_sublog_id>2</compared_sublog_id>
    <type>insert_activity</type>
    <change_new>LacticAcid</change_new>
    <change_old />
  </change>
  <change>
    <reference_sublog_id>1</reference_sublog_id>
    <compared_sublog_id>2</compared_sublog_id>
    <type>insert_activity</type>
    <change_new>Leucocytes</change_new>
    <change_old />
  </change>
  <change>
    <reference_sublog_id>1</reference_sublog_id>
    <compared_sublog_id>2</compared_sublog_id>
    <type>insert_activity</type>
    <change_new>CRP</change_new>
    <change_old />
  </change>
  <change>
```

Figure 3. Example of the structured XML Change Log generated from consecutive process windows.

We first order the traces of the event log by the timestamp of their first event. We then split them into consecutive, non-overlapping windows (sublogs) of n traces, except that the final window may contain fewer traces. The processing loop assigns the window identifiers, rather than inferring them from the LLM output, to avoid potential LLM hallucinations. The window size is a configurable parameter that can be adjusted according to the characteristics of the analyzed process. The SEPSIS experiment uses $n = 80$ traces with a step of 80 traces. For activity insertion and skipping, the framework compares the frequency of the concerned activity per trace between the two windows. For replacement and renaming, it examines the frequencies of both the old and new activities. For activity reordering, the analysis considers the frequencies of the corresponding directly-follows relations. Each feature produces one observation per trace. Table 2 summarizes the features and expected directions.

Table 2. Trace-level features and expected directions for each candidate change type.

Change Type	Feature(s)	Expected Direction
Insertion	Frequency of the proposed activity	Increase
Skipping	Frequency of the affected activity	Decrease
Replacement, renaming	Frequencies of old and new activities	Old decreases; new increases
Reordering	Frequencies of old and new directly follows relations	Old decreases; new increases

For each candidate change c, we compare the corresponding feature distributions in the reference and current windows using the two-sample KS test and MW test. Both are non-parametric. The KS test compares empirical distributions, whereas MW examines whether observations in one sample tend to have higher ranks than observations in the other. Both tests are two-sided. Statistical support for a candidate is defined as

$$S_i = \mathbb{I}(p_{KS} < 0.05) + \mathbb{I}(p_{MW} < 0.05), \tag{7}$$

where p_{KS} and p_{MW} are the p-values returned by the two tests. The significance level is fixed at $\alpha = 0.05$, a conventional threshold in hypothesis testing and not a universal boundary, as explained by Lakens et al. [46]. The tests provide statistical support for a candidate; the direction of the observed change is checked separately before validation.

In addition to statistical significance, the direction of the observed change is also considered. The observed evolution must also be consistent with the direction implied by the proposed change. For example, for activity insertion, an increase in the frequency of the inserted activity in the current window is expected, whereas, for activity skipping candidate, a decrease is required. For replacement and renaming, the expected evolution consists of a decrease in the old activity together with an increase in the new activity. Similarly, a reordering candidate requires a decrease in the original directly follows relation and an increase in the proposed relation.

We assess the direction from the total frequency of each relevant feature in the reference and current windows. For a candidate involving two features, both expected directions must hold.

Let $D(c)$ indicate whether the expected direction of candidate c is supported by the observed data. The validation status is then defined as

$$V(c) = \begin{cases} \text{validated}, & S(c) = 1 \wedge D(c) = 1, \\ \text{contradicted}, & S(c) = 1 \wedge D(c) = 0, \\ \text{not_supported}, & S(c) = 0. \end{cases} \tag{8}$$

A candidate classified as `not_supported` lacks statistical evidence under the adopted rule. This does not necessarily mean that the LLM proposal is incorrect: a small or localized change may remain undetected. The status `inconclusive` is used when the required data or candidate fragment cannot be resolved. The framework retains candidates with their status and available evidence, rather than removing them from the Change Log.

Algorithm 1 presents the procedure of candidate validation applied to the Change Log.

Algorithm 1 Statistical validation of the candidate Change Log

Require: Windows $W_1, \ldots, W_K$; candidate sets $C_1, \ldots, C_{K-1}$; $\alpha = 0.05$
Ensure: Annotated Change Log C^*

$C^* \leftarrow \emptyset$
for $k = 1$ to $K - 1$ **do**
  for all $c \in C_k$ **do**
    if the windows or required candidate information cannot be resolved **then**
      $V(c) \leftarrow$ inconclusive
    else
      Extract the trace-level feature(s) associated with c
      Apply KS and MW tests to each computable feature comparison
      Check the expected direction $D(c)$
    end if
    Store c, $V(c)$, and available statistical evidence in C^*
  end for
end for
return C^*

This decision should be interpreted as evidence that the process behavior changed between the two windows. It does not, by itself, prove that every candidate change proposed by the LLM is correct. The LLM and the data-driven component therefore play complementary roles: the former describes possible behavioral differences, whereas the latter provides an independent indication of process evolution for the same temporal region.

The framework preserves the resulting information in the Change Log, together with the window identifiers, detected change information, and available evidence. It subsequently transforms the Change Log into a hierarchical representation called the *Change Tree*. It groups repeated observations according to the affected process fragment and change type, while retaining their temporal information. Figure 4 shows an example of the resulting structure.

Each consolidated change stores its occurrences, affected traces, temporal span, associated evidence, and other descriptive information. The Change Tree therefore provides a structured representation of the process changes identified across the analyzed windows rather than a direct statistical validation of each individual LLM statement.

The validated changes are characterized according to their temporal evolution. The other groups keep their status label.

The final stage supports the natural-language exploration of this structured knowledge. Instead of indexing the complete raw event log, the RAG component uses information derived from the Change Log and the classified Change Tree. The framework converts each consolidated change into a textual knowledge record. The record contains its temporal location, evidence status, statistical evidence, occurrences, impact, and drift characterization.

The framework embeds these records and stores them in a `FAISS` index. When a user submits a question, it retrieves the most relevant records and supplies them as context to the LLM. The model therefore generates its answer from process-change information produced during the previous stages.



```
<?xml version="1.0" encoding="UTF-8"?>

<classifiedChangeTree>
  <type name="insert_activity" total_frequency="43">

    <group
      oldfragment=""
      newfragment="Risk Assessment"
      frequency="40"
      validation_status="validated"
      validation_feature="inserted_activity_frequency_per_trace"
      target_activity="Risk Assessment"
      ks_pvalue="4.472e-16"
      mw_pvalue="4.762e-15"
      direction_supported="True"
      reference_total="0"
      current_total="35"
      impact="HIGH"
      drift_pattern="INCREMENTAL"
      unique_traces="40">

      <occurrence
        reference_sublog_id="1"
        compared_sublog_id="2"
        trace_id="PO_0041"
        event_index="2"
        date="2024-01-06T14:12:00Z"/>

      <occurrence
        reference_sublog_id="1"
        compared_sublog_id="2"
        trace_id="PO_0042"
        event_index="2"
        date="2024-01-06T17:04:00Z"/>

    </group>

  </type>
</classifiedChangeTree>
```


Figure 4. Example of the classified Change Tree containing consolidated process changes and their associated evidence.

This design preserves the distinction between the change descriptions generated by the LLM and the evidence computed for the corresponding temporal windows. For example, the user can ask which activities were inserted during a given period, which changes had the largest impact, or what evidence was observed around a particular change.

Figure 5 presents the implemented prototype.

4.4. Experimental Setup

We evaluated the framework on one controlled synthetic event log and one real-life event log. The two datasets serve different purposes. The synthetic log provides known process changes, while the real-life log allows the complete pipeline to be applied to less controlled and more heterogeneous behavior. Table 3 summarizes the main characteristics of the two logs. We computed the statistics on the XES files used as input to the framework.

Table 3. Characteristics of the event logs used in the evaluation.

Characteristic	Purchase Order (Synthetic)	SEPSIS Cases (Real-Life)
Traces (cases)	160	1050
Events	1403	15,214
Time period	1–22 January 2024	November 2013–June 2015
Window size n (traces)	40	80
Injected changes	4 (Table 4)	none (no ground truth)
Source	generated by the authors	4TU.ResearchData [47]

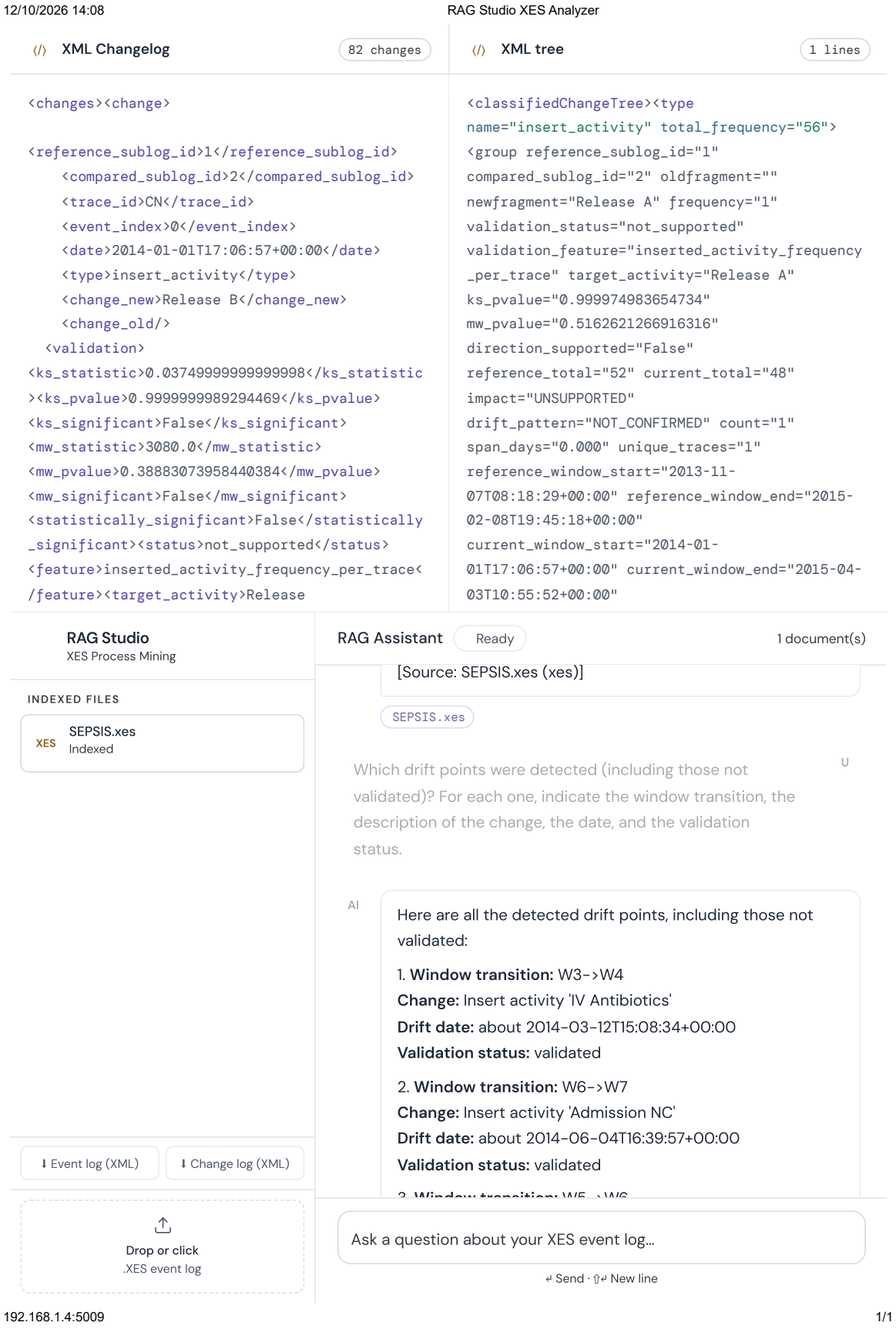


Figure 5. Prototype integrating the Change Log, classified Change Tree, and RAG-based exploration.

4.4.1. Synthetic Purchase Order Log

The first dataset is a synthetic Purchase Order event log. We introduced several controlled modifications into the process: activity insertions of different strengths, and a decrease in the frequency of an activity that becomes optional. We use the insertion of *Budget Check* and *Risk Assessment* in the following sections to illustrate the analysis, because these changes have known locations and expected behavior.

Figure 6 presents the process model of the synthetic Purchase Order process showing the controlled changes introduced into the event log.

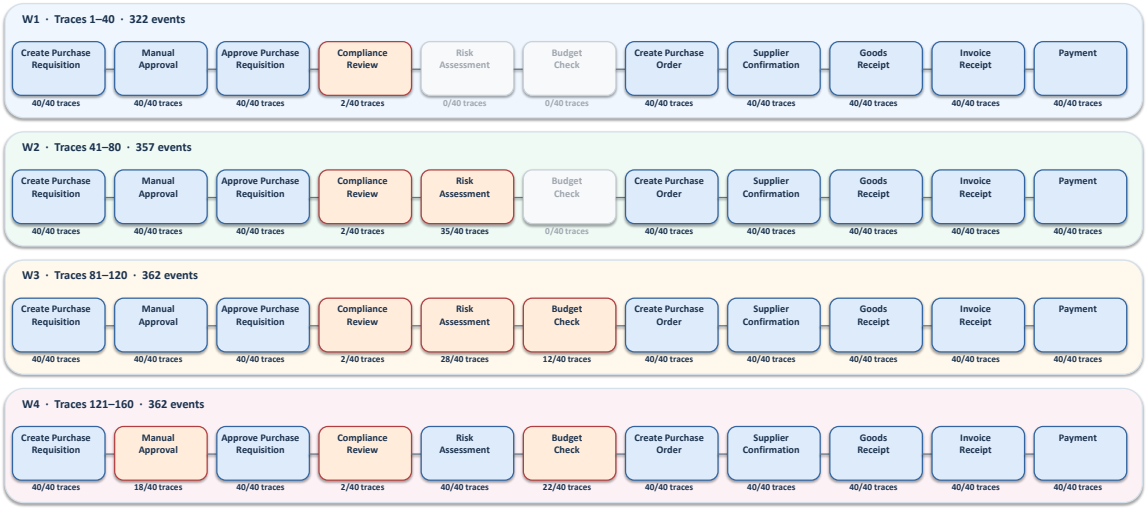


Figure 6. Process model of the synthetic Purchase Order process showing the controlled changes introduced into the event log.

The base process has nine activities, executed in sequence: *Create Purchase Requisition*, *Manual Approval*, *Approve Purchase Requisition*, *Compliance Review*, *Create Purchase Order*, *Supplier Confirmation*, *Goods Receipt*, *Invoice Receipt*, and *Payment*. The log contains 160 traces, ordered in time and organized in four consecutive segments of 40 traces. Each segment corresponds to one window of the analysis, so that each comparison between two windows contains known changes:

- W1, traces 1–40 (from 1 January 2024): the baseline behavior.
- W2, traces 41–80 (from 6 January 2024): a strong insertion of *Risk Assessment* after *Approve Purchase Requisition*, present in 35 of the 40 traces.
- W3, traces 81–120 (from 11 January 2024): a partial insertion of *Budget Check* at the same position, present in 12 traces. In the other traces, *Risk Assessment* is still executed.
- W4, traces 121–160 (from 16 January 2024): *Manual Approval* becomes optional and is executed in only 18 traces, against 40 in W3. At the same time, *Budget Check* becomes more frequent (22 traces, executed after *Risk Assessment*).

Table 4 lists the expected changes for each comparison.

Table 4. Controlled changes injected into the synthetic Purchase Order log (number of traces containing the activity, before → after, out of 40 per window).

Comparison	Change	Type	Traces
W1 → W2	Risk Assessment	insertion (strong)	0 → 35
W2 → W3	Budget Check	insertion (partial)	0 → 12
W3 → W4	Manual Approval	skip (decrease)	40 → 18
W3 → W4	Budget Check	frequency increase	12 → 22
all	Compliance Review (control)	no change	2 → 2

4.4.2. SEPSIS Cases Log

The second dataset is the publicly available *SEPSIS Cases* event log [47]. It contains real-life hospital process executions and presents more heterogeneous behavior than the synthetic dataset. In contrast to the controlled *Purchase Order* log, explicit ground-truth drift annotations are not available. We therefore use this dataset to examine how the complete pipeline behaves on a real-life event log rather than to calculate ground-truth detection accuracy.

The log was recorded by the ERP system of a Dutch hospital and was published by Mannhardt in 2016 on 4TU.ResearchData (DOI: https://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460). Each case describes the clinical pathway of one patient with potential sepsis. The log contains 39 data attributes, such as the group responsible for an activity, test results, and checklist information. The framework does not use these attributes: it relies only on the case identifier, the activity name, and the timestamp.

We also point out one limitation of this log. To protect patient privacy, the timestamps were randomized by the publisher, while the time between events within the same trace was kept unchanged [47]. The order of the cases over time, and therefore the absolute dates of the detected changes, must be read with caution.

4.4.3. Preprocessing and Windowing

Both logs are imported from XES files using `PM4Py` and kept only the three standard attributes: `case:concept:name`, `activity:concept:name`, and `time:timestamp`. We then sorted the traces by the timestamp of their first event and split them into consecutive non-overlapping windows of $n = s$ traces, as described in Section 4.3. Each pair of consecutive windows gives one comparison, so a log with K windows gives $K - 1$ comparisons. For SEPSIS, we used $n = s = 80$, which gives 14 windows (13 of 80 traces and a last one of 10 traces) and 13 comparisons. This value is a compromise. A smaller window gives a finer location of the changes, but the statistical tests then work on very small samples and lose power. A larger window gives more reliable tests, but it can merge several changes into the same comparison and reduces the number of comparisons. The last SEPSIS window contains only 10 traces, so the results of the last comparison are less reliable and are interpreted with care. For the synthetic log, we used $n = 40$, so that the window boundaries coincide with the four segments of the log.

4.4.4. Implementation and Configuration

We implemented the framework in `Python` using `Flask` and `PM4Py`. We use the LLM-oriented functionalities available in `PM4Py` to produce textual event-log abstractions [48].

We generated candidate changes with `GPT-4o-mini` through the OpenAI API. We called the model with the alias `gpt-4o-mini` and not with a dated snapshot. We set the temperature to 0.2 to limit unnecessary variation in the output, and the maximum output length to 16,000 tokens. For each pair of windows, the LLM receives the abstractions of the two windows and is asked to list the changes between them in a fixed XML format. The full prompt is given in Appendix A.

For the RAG component, the knowledge base contains one text section per consolidated change record, per structural drift record, per window, and per trace of the event log. Each section is indexed as a single chunk; a section longer than 6000 characters is split at line boundaries, without overlap. We embed the chunks with the OpenAI model `text-embedding-3-small` and index them with `FAISS` (exact search, cosine similarity). For each question, the context given to the LLM consists of two sections that are always included, namely, the overview of all change candidates and the summary of the event log, followed by the $k = 5$ most similar chunks. When the question names a case, the section

of that case is added as well. The same LLM generates the answers, with a temperature of 0.2 and a maximum of 16,000 tokens. The statistical tests use the implementations of `SciPy` (`ks_2samp` and `mannwhitneyu`, two-sided, default settings). Table 5 lists the tools and versions, and Table 6 summarizes the parameters.

Table 5. Software and versions used in the experiments.

Component	Tool	Version
Programming language	Python	3.12.8
Process mining	PM4Py	2.7.13.1
Statistical tests	SciPy	1.18.0
Web interface	Flask	3.1.0
Vector index	FAISS (CPU)	1.12.0
LLM	GPT-4o-mini (OpenAI)	alias `gpt-4o-mini`
Embedding model	text-embedding-3-small (OpenAI)	-
Baseline	ProDrift	2.5

Table 6. Parameters of the framework.

Stage	Parameter	Value
Windowing	window size n (traces)	80 (SEPSIS), 40 (synth.)
	step, overlap	n, none
LLM generation	temperature, max. tokens	0.2, 16,000
Validation	tests	KS and MW, two-sided
	significance level α	0.05
	support rule	$p_{KS} < \alpha \vee p_{MW} < \alpha$
	validation rule	support and expected direction
RAG	chunking	one chunk per section; split at 6000 char., no overlap
	embedding model	`text-embedding-3-small`
	always-included sections	change overview, event log summary
	retrieved chunks k	5
	temperature, max. tokens	0.2, 16,000

4.4.5. Baseline: ProDrift

We also analyzed the SEPSIS log with ProDrift 2.5, a reference tool for process drift detection [33,34]. ProDrift is not part of our framework: we ran it separately, as a standalone tool, with its default settings and without any tuning, since there is no ground truth for SEPSIS on which to tune it. We used its event-based mode, which reads the log as a stream of events and reports the index and the date of each detected drift. ProDrift also characterizes each drift by comparing the process trees discovered before and after it, and describes the differences as a list of statements on activities and fragments [34]. This comparison provides an external temporal reference for the detected process evolution. ProDrift is not used as a direct substitute for the proposed framework, since the two approaches do not produce the same type of output. The comparison therefore focuses on their common detection-related information, while the structured characterization and RAG-based exploration are evaluated separately.

5. Experimental Evaluation

We evaluated the proposed framework using both synthetic and real-life event logs. The synthetic experiment provides a controlled setting in which the introduced process changes are known in advance, while the real-life experiment is used to examine the applicability of the framework to more complex process behavior. In addition, we analyzed the

synthetic log with ProDrift to provide an external point of comparison with an established CD detection tool.

5.1. Evaluation Results

On the synthetic Purchase Order log, the framework identified the controlled process modifications introduced during log generation. One of the main changes concerned the insertion of the *Risk Assessment* activity. For the corresponding window transition, the data-driven component also indicated process evolution. Both the KS and MW tests returned highly significant results ($p < 0.001$) for the compared case-duration distributions, while the structural analysis captured changes in the observed process behavior.

The LLM-based analysis also identified a later change involving *Budget Check*. The results illustrate the complementary role of the two components: the LLM describes candidate behavioral changes, while the structural and statistical analysis provides an independent assessment of the temporal windows in which these changes are observed.

The framework records the resulting change information and associated evidence in the Change Log and organizes them within the Change Tree. These representations preserve information about the identified changes, their occurrences, window-level evidence, and temporal characteristics. They also constitute the structured knowledge used by the RAG component to answer drift-related questions.

We applied the same pipeline to the SEPSIS event log. The framework identified candidate behavioral changes, computed structural and statistical evidence for the corresponding window transitions, and generated the same structured representations used in the synthetic experiment. Since no explicit ground-truth drift annotations are available for this log, these results should not be interpreted as a measure of detection accuracy. Instead, they show that the proposed pipeline can be applied beyond the controlled synthetic setting and can handle more complex real-life process behavior.

5.2. Comparison with ProDrift

To provide an external point of comparison, we analyzed the synthetic Purchase Order event log and the real-life SEPSIS log using ProDrift. The aim is not to establish a direct benchmark between the two approaches, since their detection mechanisms are different, but to examine whether they identify process evolution within comparable temporal regions.

5.2.1. Synthetic Purchase Order Log

We executed ProDrift in event-based mode using a window size of 40, ADWIN window adaptation, a noise-filter threshold of 0.1, and low detection sensitivity. Under this configuration, ProDrift reported a single drift at event 294. This point falls within the same general temporal region in which the proposed framework identifies the emergence of the *Risk Assessment* activity.

While ProDrift provides the location of the detected drift, the proposed framework associates the observed evolution with an explicit behavioral change and preserves the evidence computed for the corresponding temporal transition. The proposed framework also identified a later change involving *Budget Check*, whereas ProDrift reported no additional drift under the tested configuration. Table 7 summarizes the main differences between the two approaches.

Figure 7 presents the outputs obtained from the two approaches on the same event log. The correspondence around the first temporal region provides an additional indication that the observed process evolution is not specific to the LLM-based analysis.

Table 7. Comparison between ProDrift and the proposed framework on the synthetic Purchase Order event log.

Aspect	ProDrift	Proposed Framework
Detection	Drift at event 294	*Risk Assessment* identified in the corresponding temporal region
Change information	Drift location	Explicit behavioral change and affected activity
Validation	Internal detection mechanism	Structural and statistical evidence
Additional output	Drift visualization	Change Log, Change Tree, and RAG-based exploration

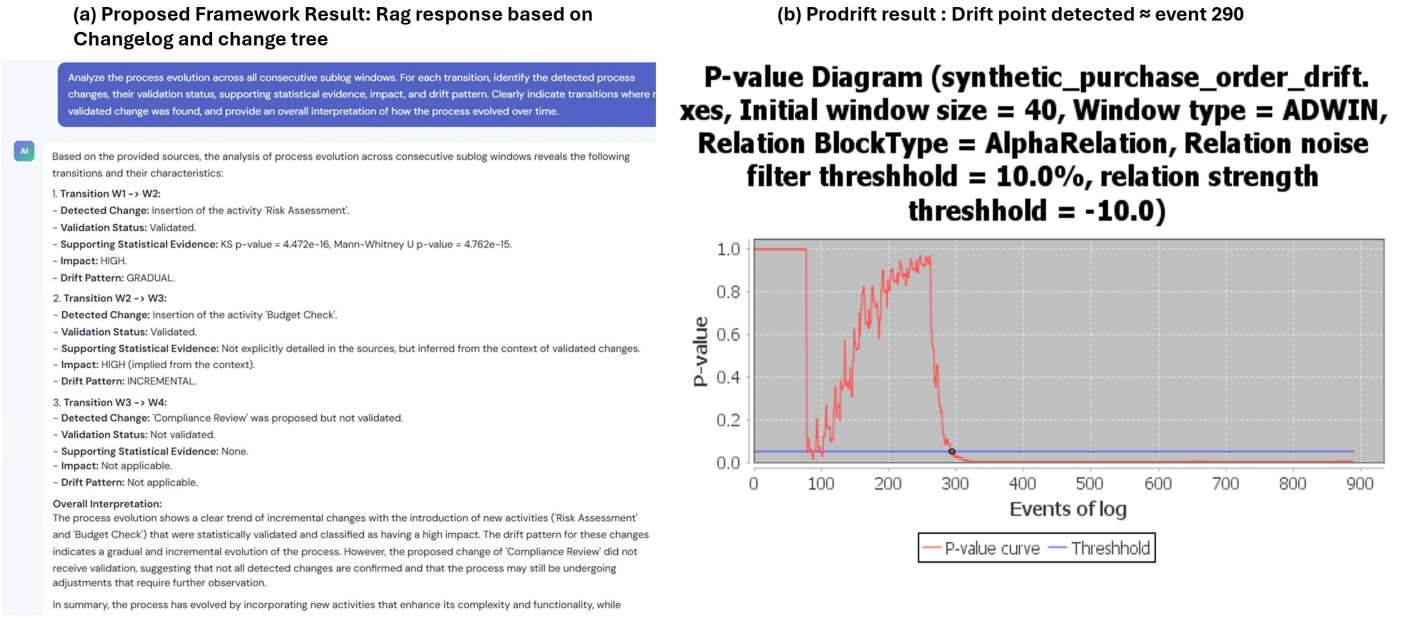


Figure 7. Comparison between the outputs of the two approaches on the synthetic Purchase Order event log.

5.2.2. SEPSIS Cases Log

On SEPSIS, ProDrift reported five drifts (Table 8). Each drift was reported after reading about 640 events beyond the drift point (562 events for the fourth one). This detection delay matters for the comparison.

Table 8. Drifts reported by ProDrift on the SEPSIS log.

Drift Number	Event	Date	Reported After	Main Differences Between the Trees Before and After
D1	2273	20 February 2014	2913	*Admission NC* becomes optional; *IV Liquid* is no longer skippable; *Return ER* restructured
D2	4315	10 April 2014	4955	*IV Liquid* becomes skippable; *Admission IC* appears in a choice with *Admission NC*
D3	5023	29 April 2014	5663	Same changes as D2; *Release C* and *Release D* inserted
D4	10,058	7 September 2014	10,620	*Admission NC* moved (from a parallel branch to an optional one); *Release D* inserted; *Return ER* moved
D5	11,384	11 October 2014	12,024	*Admission NC* placed after *IV Antibiotics* and no longer optional; *IV Liquid* skippable; *Release D* inserted

On the same log, our framework reported five candidate changes (Table 9). Three of them were validated: the insertion of *IV Antibiotics* between W3 and W4, the decrease in

Admission NC between W5 and W6, and its increase between W6 and W7. The two other candidates concern *Release D*. The framework marked them as *not supported* because *Release D* occurs in only one or two traces per window (24 occurrences in the whole log), so that no significant frequency change can be observed between two windows of 80 traces.

Table 9. Candidate changes reported by the proposed framework on the SEPSIS log (windows of 80 traces), in chronological order.

Drift Number	Change	Windows	Start of Current Window	p_{MW}	Status
F1	Insertion of *IV Antibiotics*	W3→W4	event 3155 12 March 2014	0.040	validated
F2	Insertion of *Release D*	W4→W5	event 4374 11 April 2014	0.566	not supported
F3	Skip of *Admission NC*	W5→W6	event 5522 9 May 2014	0.021	validated
F4	Insertion of *Admission NC*	W6→W7	event 6606 4 June 2014	0.010	validated
F5	Skip of *Release D*	W8→W9	event 9097 14 August 2014	1.000	not supported

Table 10 gives, for each candidate of our framework, the closest ProDrift drift and the distance between the two positions.

The two tools give a consistent picture of how the SEPSIS process evolved. Four of the five ProDrift changes have a candidate of our framework, and the candidates concern the same activities compared to ProDrift results; only the last drift (D5) has no corresponding candidate. Two of these correspondences are confirmed by the statistical validation. The closest one is particularly clear: between W5 and W6, our framework observes that *Admission NC* drops from 79% to 61%, while ProDrift reports, ten days earlier (D3), that *Admission NC* is no longer mandatory and is now in a choice with *Admission IC*. The two tools describe the same change, one through the frequency of the activity and the other through the structure of the model. Our framework also captures what happens next: *Admission NC* becomes frequent again between W6 and W7 (from 61% to 80%), a return that does not appear as a separate ProDrift drift.

Table 10. Closest ProDrift drift for each candidate of the proposed framework on the SEPSIS log (one window corresponds to about 1160 events).

Our Framework	ProDrift	Distance	Correspondence on the Activities
F1: insertion of *IV Antibiotics* (validated)	D1	882 events (20 days)	Same part of the process: treatment before admission (*IV Liquid*, *IV Antibiotics*)
F2: insertion of *Release D* (not supported)	D2	59 events (1 day)	*Release D*, reported as inserted by ProDrift in D3
F3: skip of *Admission NC* (validated)	D3	499 events (10 days)	Same activity and same meaning: *Admission NC* becomes less frequent (ours) and optional, in a choice with *Admission IC* (ProDrift)
F4: insertion of *Admission NC* (validated)	D3	1583 events (36 days)	Same activity: return of *Admission NC* after F3, not reported as a separate ProDrift drift
F5: skip of *Release D* (not supported)	D4	961 events (24 days)	Same activity: *Release D*

The candidates on *Release D* illustrate the complementary role of the two components of our framework. The LLM proposed changes on this activity exactly where ProDrift reports its insertion (D2–D3 and D4), which shows that the comparison of abstracted

sublogs captures the same structural differences. However, *Release D* is too rare in SEPSIS for this change to be statistically confirmed between two windows of 80 traces. The validation step therefore keeps these candidates as not supported, with their evidence, rather than reporting them as confirmed drifts. The candidates are still available in the Change Log and can be explored through the RAG component. Finally, recall that the timestamps of SEPSIS were randomized by the publisher (Section 4.4); the dates above are used to compare the two tools, not to relate the changes to real events in the hospital.

The comparison should not be interpreted as evidence that one approach is more accurate than the other. ProDrift focuses primarily on locating process drift, whereas the proposed framework aims to identify what may have changed, assess the corresponding temporal transition from the event data, and organize the result for subsequent interpretation. The experiments remain limited to a controlled synthetic process and one real-life event log. A broader evaluation across additional processes, drift types, window configurations, and LLMs will therefore be needed to assess the generalizability of the approach.

5.3. Sensitivity of the Validation Step

To test sensitivity, we varied the window size n and the significance level α of the statistical tests. The result shows that the window size n has a stronger effect on the results than the significance level α. On SEPSIS, the number of confirmed changes decreased from 15 with $n = 40$ to 3 with $n = 200$, at $\alpha = 0.05$. Smaller windows capture more local changes, but they are also more sensitive to differences between a few traces. Larger windows provide more data for testing, although they may hide short changes. Changing α also affects the decision boundary. A stricter threshold (0.01) rejects weaker candidates, while a higher threshold (0.10) accepts more borderline changes. We therefore used $\alpha = 0.05$ as a practical compromise, rather than an optimal value for every dataset. Since our experiments cover only the synthetic Purchase Order and the SEPSIS logs, we cannot yet conclude that the same settings will work equally well across business domains. The window size should be selected according to the log characteristics, and the sensitivity of the validation results should be checked before applying the framework to a new domain.

6. Discussion

6.1. Framework Design and Parameter Sensitivity

The main purpose of this work is to show why combining LLMs with statistical tests is useful. In fact, the LLM has the potential to identify possible changes and describes them in simple words, such as an inserted, skipped, or replaced, activity. However, these descriptions are not always supported by the real data or the real behavior of the process, and this is due to LLM hallucinations, which can lead to false positives. The statistical tests help to address this issue by checking each candidate against the original event log. In one SEPSIS log, 54 of 64 candidates were rejected as not supported after statistical validation. However, statistical validation has its limits. In fact, some changes involve rare activities and are difficult to confirm with small samples. For example, the LLM identified changes in *Release D* near to a drift reported by ProDrift, but the activity appeared in only one or two traces per window, so the statistical test failed to validate it. Other changes may affect the order or timing of activities without changing their frequency. Such cases require different features for validation. We therefore keep rejected candidates in the Change Log, along with their statistical results, so they can be examined later.

By analyzing sensitivity, we found that window size has a strong influence on the results. Small windows help to locate changes more precisely, but they may also produce detections based on a few unusual traces. Larger windows provide more data for testing, yet they can hide short changes. The significance level had a smaller effect. A stricter threshold

($\alpha = 0.01$) missed some weaker changes, while $\alpha = 0.10$ included more borderline cases (Section 5.3).

6.2. Comparison with Existing Approaches and Limitations

ProDrift [33] locates structural changes, while our framework describes candidate changes and checks them against event-log evidence. On the SEPSIS log, the changes reported by our framework had a close correspondence with what Prodrift localized. Explainable CD approches [23] study possible causes of changes but do not provide a natural-language interaction, while LLM-based PM [12] supports the natural-language analysis of process artifacts. Recent work also combines statistical analysis with AI-based reasoning [45]. This is the study closest to ours, but it does not provide a structured representation of the changes or a natural-language interface.

Our contribution is different. We validate individual LLM-generated candidates and store their evidence in a Change Log and Change Tree. The RAG component then allows analysts to explore these findings through natural-language questions.

Beyond PM, this approach could also support the analysis of changes in industrial and IoT data. For example, ref. [49] addresses fault diagnosis under different operating conditions. These approches focus on model adaptation, while our framework could complement them by adding a new layer that helps identifying and explaining changes in the observed data.

We evaluated our framework using two logs with GPT-4o-mini. Since LLM outputs may vary between runs, the same experiment may produce different candidates. Statistical validation, however, gives the same result for the same candidate and windows, which justifies the combination of LLMs and statistical tests. The framework is also modular: the LLM can be replaced by another model, and the statistical tests can be adapted to the log characteristics. As future work, we plan to test the framework on more datasets, with different LLMs and repeated runs. We also plan to improve the validation of rare changes and explore its use with IoT and sensor data.

7. Conclusions and Future Work

Our framework addressed a key challenge in CD analysis: identifying process changes while providing explanations supported by event-log evidence. Traditional qalgorithmes for drift detection can identify changes, but can not provide information about their nature on natural language or does not provide natural-language iteraction. In parallel, LLMs shoe a strong capability to offer more readable descriptions, but usully suffer from hallucinations.

The main scientific contribution of our work is a new hybrid framework for CD localization and characterization in PM using LLMs. It combines the LLM-based generation of candidate changes with statistical validation using the KS and MW tests. This combinaison reduces the risk of unsupported LLM interpretations or LLM hallucinations while preserving their ability to describe process changes.

The framework then organizes the resulting information into a structured XML Change Log and a classified Change Tree. These artifacts preserve all the information about the link between a detected change, its occurrences, validation evidence, temporal information, and drift characteristics. They also form the knowledge data source that feeds the RAG layer, allowing users to explore process evolution through natural-language questions while keeping the generated answers connected to the evidence produced during the analysis.

The experiments provide evidence of the framework's potential. On the synthetic Purchase Order log, by using non overlapping windows ($W = 40$, $S = 40$), the statistical validation step confirmed all injected changes with $\alpha = 0.05$. In comparison, ProDrift reported only one drift.

For real-life SEPSIS log, the window size and the step used are $W = 80$ and $S = 80$. Three of five reported candidates were statistically validated. Four of the five drifts reported by ProDrift had a correspondance with the result of our framework. These results suggest that the framework can identify meaningful process changes while providing additional evidence for their interpretation.

The sensitivity analysis showed that window size has a stronger effect on the results than the significance level. Smaller windows can reveal local changes but may produce more weak detections. Larger windows provide more data for testing, although they can hide short changes. For practical use, analysts should select the window size according to the event-log characteristics, use $\alpha = 0.05$ as an initial threshold, and review the statistical evidence before accepting a change.

Several directions remain open. First, the framework should be evaluated on benchmarks with known drift locations and types, including gradual and recurring drifts, and with several LLMs and repeated runs, in order to report precision and recall. Second, the RAG component should be evaluated quantitatively. Third, the framework knowledge base could be enriched with domain-specific ontologies to produce more contextual explanations. Finally, we are extending the framework to IoT and sensor data, in order to apply it to industrial process monitoring.

Author Contributions: Conceptualization, Z.E. and H.S.; methodology, H.S.; software, Z.E.; validation, H.S.; formal analysis, Z.E.; investigation, Z.E.; writing—original draft preparation, Z.E.; writing—review and editing, Z.E. and H.S.; supervision, H.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The synthetic Purchase Order event log is publicly available in the 4TU Research Data repository. The raw SEPSIS Cases event log used in the experiments is available from the authors upon request.

Acknowledgments: During the preparation of this manuscript, the authors used ChatGPT (GPT-6, OpenAI) to assist with language editing, reformulation, and improvement to the clarity and organization of the manuscript. The authors reviewed and edited the generated output and take full responsibility for the content of this publication.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

PM	Process Mining
CD	Concept Drift
LLM	Large Language Model
RAG	Retrieval-Augmented Generation
BPM	Business Process Management
DFG	Directly-Follows Graph
KS	Kolmogorov–Smirnov
MW	Mann–Whitney U

Appendix A. Prompt Used for Drift Candidate Generation

This appendix reports the prompt used to generate the candidate Change Log from each pair of consecutive sublogs. The placeholders `{reference_sublog}` and

{current_sublog} are replaced at run time by the abstracted traces of the reference and current windows, and {sublog_id_ref} and {sublog_id_cur} by their window identifiers. We query the model with a temperature of 0.1. We do not trust the window identifiers returned by the LLM: the comparison loop overwrites them deterministically before the statistical validation of each candidate.

Listing A1. System Message: Instructions for XML-Based Change Log Generation

```
You are an expert PM assistant that outputs ONLY well-formed XML. Never include markdown
    fences, code blocks, comments, or explanations. Do not add an XML declaration. Output
    exactly one root element <changes>...</changes>.
```

Listing A2. User Message: Comparing Reference and Current Sublogs

```
Task
----
Compare the CURRENT abstracted sublog with the REFERENCE abstracted sublog and produce a
    candidate change log.

The REFERENCE sublog represents the previous behavior.
The CURRENT sublog represents the behavior to analyze.

Important scientific role
-------------------------
You are NOT the final drift detector.
You only propose candidate process changes.
Each candidate will later be validated statistically.

Inputs
------
[REFERENCE_SUBLOG]
{reference_sublog}

[CURRENT_SUBLOG]
{current_sublog}

STRICT OUTPUT SPEC
------------------
- Output EXACTLY one root element: <changes> ... </changes>.
- For EACH change, include in this order:

<change>
<reference_sublog_id>{sublog_id_ref}</reference_sublog_id>
<compared_sublog_id>{sublog_id_cur}</compared_sublog_id>
<trace_id>...</trace_id>
<event_index>...</event_index>
<date>ISO-8601</date>
<type>skip_activity|insert_activity|rename_activity|replace_activity|reorder_pair</type>
<change_new>...</change_new>
<change_old>...</change_old>
</change>

CHANGE TYPES EXPLAINED:
- skip_activity: Activity existing in the reference behavior was skipped or decreased in the
    current sublog.
- insert_activity: Extra activity was added or increased in the current sublog.
- rename_activity: Activity name changed between reference and current sublog.
- replace_activity: Different activity performed instead of the expected one.
- reorder_pair: Two activities changed their order.

RULES:
```

```
- Detect only meaningful behavioral differences.
- For insert_activity: change_old = "" and change_new = inserted activity.
- For skip_activity: change_old = skipped activity and change_new = "".
- Escape &, <, >.
- No attributes, CDATA, comments, or extra tags.
- If nothing detected, return <changes></changes>.
```

References

1. Bose, R.P.J.C.; van der Aalst, W.M.P.; Žliobaitė, I.; Pechenizkiy, M. Handling Concept Drift in Process Mining. In *Proceedings of the Advanced Information Systems Engineering*; Mouratidis, H., Rolland, C., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; pp. 391–405. [CrossRef]
2. Van Der Aalst, W.M.P. Process-Aware Information Systems: Lessons to Be Learned from Process Mining. In *Transactions on Petri Nets and Other Models of Concurrency II*; Jensen, K., Van Der Aalst, W.M.P., Eds.; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5460, pp. 1–26. [CrossRef]
3. Van Der Aalst, W.; Adriansyah, A.; De Medeiros, A.K.A.; Arcieri, F.; Baier, T.; Blickle, T.; Bose, J.C.; Van Den Brand, P.; Brandtjen, R.; Buijs, J.; et al. Process Mining Manifesto. In *Business Process Management Workshops*; Daniel, F., Barkaoui, K., Dustdar, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; Volume 99, pp. 169–194. [CrossRef]
4. Carmona, J.; Gavaldà, R. Online Techniques for Dealing with Concept Drift in Process Mining. In *Advances in Intelligent Data Analysis XI*; Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J.M., Mattern, F., Mitchell, J.C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., et al., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7619, pp. 90–102. [CrossRef]
5. Hmami, A.; Sbai, H.; Fredj, M. A New Framework to Improve Change Mining in Configurable Process. In Proceedings of the 3rd International Conference on Networking, Information Systems & Security, Marrakech, Morocco, 31 March–2 April 2020; pp. 1–6. [CrossRef]
6. Hmami, A.; Sbai, H.; Fredj, M. Change Mining in Business Process Variability: A Comparative Study. In Proceedings of the 2019 4th World Conference on Complex Systems (WCCS), Ouarzazate, Morocco, 22–25 April 2019; pp. 1–5. [CrossRef]
7. Hmami, A.; Sbai, H.; Fredj, M. Applying Change Tree Method in Configurable Process Mining. In Proceedings of the 2022 2nd International Conference on Innovative Research in Applied Science, Engineering and Technology (IRASET), Meknes, Morocco, 3–4 March 2022; pp. 1–6. [CrossRef]
8. Berti, A.; Schuster, D.; van der Aalst, W.M.P. Abstractions, Scenarios, and Prompt Definitions for Process Mining with LLMs: A Case Study. *arXiv* **2023**. [CrossRef]
9. Teubner, T.; Flath, C.M.; Weinhardt, C.; Van Der Aalst, W.; Hinz, O. Welcome to the Era of ChatGPT et al.: The Prospects of Large Language Models. *Bus. Inf. Syst. Eng.* **2023**, *65*, 95–101. [CrossRef]
10. Ceravolo, P.; Junior, S.B.; Damiani, E.; Van Der Aalst, W. Tuning Machine Learning to Address Process Mining Requirements. *IEEE Access* **2024**, *12*, 24583–24595. [CrossRef]
11. Berti, A.; Qafari, M.S. Leveraging Large Language Models (LLMs) for Process Mining (Technical Report). *arXiv* **2023**, arXiv:2307.12701. [CrossRef]
12. Berti, A.; Kourani, H.; Hafke, H.; Li, C.Y.; Schuster, D. Evaluating Large Language Models in Process Mining: Capabilities, Benchmarks, and Evaluation Strategies. *Enterp.-Bus.-Process Inf. Syst. Model.* **2024**, *511*, 13–21. [CrossRef]
13. Kampik, T.; Warmuth, C.; Rebmann, A.; Agam, R.; Egger, L.N.P.; Gerber, A.; Hoffart, J.; Kolk, J.; Herzig, P.; Decker, G.; et al. Large Process Models: A Vision for Business Process Management in the Age of Generative AI. *KI-Künstl. Intell.* **2025**, *39*, 81–95. . Correction in *KI-Künstl. Intell.* **2025**, *39*, 345–347. [CrossRef]
14. Zebec, A.; Indihar Štemberger, M. Creating AI Business Value through BPM Capabilities. *Bus. Process Manag. J.* **2024**, *30*, 1–26. [CrossRef]
15. R'bigui, H.; Cho, C. The State-of-the-Art of Business Process Mining Challenges. *Int. J. Bus. Process Integr. Manag.* **2017**, *8*, 285–303. [CrossRef]
16. Berti, A.; Aalst, W.M. *CSV-PM-LLM-Parsing: Automatic Ingestion of CSV Event Logs for Process Mining Using LLMs*; CEUR-WS.org: Aachen, Germany, 2024.
17. Song, W.; Jacobsen, H.A. Static and Dynamic Process Change. *IEEE Trans. Serv. Comput.* **2018**, *11*, 215–231. [CrossRef]
18. Kaes, G.; Rinderle-Ma, S. On the Similarity of Process Change Operations. In *Advanced Information Systems Engineering*; Dubois, E., Pohl, K., Eds.; Springer International Publishing: Cham, Switzerland, 2017; Volume 10253, pp. 348–363. [CrossRef]
19. Bose, R.P.J.C.; Van Der Aalst, W.M.P.; Zliobaite, I.; Pechenizkiy, M. Dealing With Concept Drifts in Process Mining. *IEEE Trans. Neural Netw. Learn. Syst.* **2014**, *25*, 154–171. [CrossRef]
20. Gama, J.; Žliobaitė, I.; Bifet, A.; Pechenizkiy, M.; Bouchachia, A. A Survey on Concept Drift Adaptation. *Acm Comput. Surv.* **2014**, *46*, 1–37. [CrossRef] [PubMed]
21. Van Der Aalst, W. *Process Mining: Data Science in Action*; Springer: Berlin/Heidelberg, Germany, 2016. [CrossRef]

22. Hmami, A.; Sbai, H.; Fredj, M. Handling Sudden and Recurrent Changes in Business Process Variability: Change Mining Based Approach. *Int. J. Adv. Comput. Sci. Appl.* **2021**, *12*, 632–640. [CrossRef]
23. Adams, J.N.; Van Zelst, S.J.; Rose, T.; Van Der Aalst, W.M. Explainable Concept Drift in Process Mining. *Inf. Syst.* **2023**, *114*, 102177. [CrossRef]
24. Vitale, F.; Pegoraro, M.; Van Der Aalst, W.M.; Mazzocca, N. Control-Flow Anomaly Detection by Process Mining-Based Feature Extraction and Dimensionality Reduction. *Knowl.-Based Syst.* **2025**, *310*, 112970. [CrossRef]
25. Yao, Y.; Duan, J.; Xu, K.; Cai, Y.; Sun, Z.; Zhang, Y. A Survey on Large Language Model (LLM) Security and Privacy: The Good, The Bad, and The Ugly. *High-Confid. Comput.* **2024**, *4*, 100211. [CrossRef]
26. Arora, C.; Sayeed, A.I.; Licorish, S.; Wang, F.; Treude, C. Optimizing Large Language Model Hyperparameters for Code Generation. *arXiv* **2024**. [CrossRef]
27. Gao, Y.; Xiong, Y.; Gao, X.; Jia, K.; Pan, J.; Bi, Y.; Dai, Y.; Sun, J.; Wang, M.; Wang, H. Retrieval-Augmented Generation for Large Language Models: A Survey. *arXiv* **2024**. [CrossRef]
28. Zhang, X.; Zhao, F.; Liu, Y.; Chen, P.; Wang, Y.; Wang, X.; Ma, D.; Xu, H.; Chen, M.; Li, H. TreeQA: Enhanced LLM-RAG with Logic Tree Reasoning for Reliable and Interpretable Multi-Hop Question Answering. *Knowl.-Based Syst.* **2025**, *330*, 114526. [CrossRef]
29. Elkhawaga, G.; Abuelkheir, M.; Barakat, S.I.; Riad, A.M.; Reichert, M. CONDA-PM—A Systematic Review and Framework for Concept Drift Analysis in Process Mining. *Algorithms* **2020**, *13*, 161. [CrossRef]
30. Sato, D.M.V.; de Freitas, S.C.; Barddal, J.P.; Scalabrin, E.E. A Survey on Concept Drift in Process Mining. *Acm Comput. Surv.* **2022**, *54*, 1–38. [CrossRef]
31. De Sousa, R.G.; Meira Neto, A.C.; Fantinato, M.; Peres, S.M.; Reijers, H.A. Integrated Detection and Localization of Concept Drifts in Process Mining with Batch and Stream Trace Clustering Support. *Data Knowl. Eng.* **2024**, *149*, 102253. [CrossRef]
32. Sun, S.; Li, Q. A Behavior Change Mining Method Based on Complete Logs with Hidden Transitions and Their Applications in Disaster Chain Risk Analysis. *Sustainability* **2023**, *15*, 1655. [CrossRef]
33. Maaradji, A.; Dumas, M.; Rosa, M.L.; Ostovar, A. Detecting Sudden and Gradual Drifts in Business Processes from Execution Traces. *IEEE Trans. Knowl. Data Eng.* **2017**, *29*, 2140–2154. [CrossRef]
34. Ostovar, A.; Leemans, S.J.J.; Rosa, M.L. Robust Drift Characterization from Event Streams of Business Processes. *Acm Trans. Knowl. Discov. Data* **2020**, *14*, 1–57. [CrossRef]
35. Abbasi, M.; Nishat, R.I.; Bond, C.; Graham-Knight, J.B.; Lasserre, P.; Lucet, Y.; Najjaran, H. A Review of AI and Machine Learning Contribution in Business Process Management (Process Enhancement and Process Improvement Approaches). *Bus. Process Manag. J.* **2025**, *31*, 1414–1452. [CrossRef]
36. Moder, L.; Fehrer, T.; Röglinger, M. Design Principles for Process Improvement and Innovation Systems. *Bus. Process Manag. J.* **2026**, *32*, 198–219. [CrossRef]
37. Kourani, H.; Berti, A.; Hennrich, J.; Kratsch, W.; Weidlich, R.; Li, C.Y.; Arslan, A.; Van Der Aalst, W.M.; Schuster, D. Leveraging Large Language Models for Enhanced Process Model Comprehension. *Decis. Support Syst.* **2026**, *200*, 114563. [CrossRef]
38. Berti, A.; Kourani, H.; Park, G.; Aalst, W.M.P.V.D. Configuring Large Reasoning Models Using Process Mining: A Benchmark and a Case Study. *TechRxiv* **2025**. [CrossRef]
39. Vijay, V.; Jayaraman, M.; Khaleel, M.A. Enhancing Process Mining through LLMs: A User-Centric Approach. In *Proceedings of the 2024 IEEE 8th International Conference on Information and Communication Technology (CICT)*; IEEE: New York, NY, USA, 2024; pp. 1–6. [CrossRef]
40. Kourani, H.; Berti, A.; Schuster, D.; Van Der Aalst, W.M.P. Evaluating Large Language Models on Business Process Modeling: Framework, Benchmark, and Self-Improvement Analysis. *Softw. Syst. Model.* **2025**, *25*, 1151–1186. [CrossRef]
41. Fionda, V.; Ielo, A.; Ricca, F. Are Large Language Models Fluent in Declarative Process Mining? In Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, Montreal, QC, Canada, 16–22 August 2025; pp. 4500–4508. [CrossRef] [PubMed]
42. Fahland, D.; Fournier, F.; Limonad, L.; Skarbovsky, I.; Swevels, A.J. How Well Can a Large Language Model Explain Business Processes as Perceived by Users? *Data Knowl. Eng.* **2025**, *157*, 102416. [CrossRef]
43. Casciani, A.; Bernardi, M.L.; Cimitile, M.; Marrella, A. Enhancing next Activity Prediction in Process Mining with Retrieval-Augmented Generation. *Inf. Syst.* **2026**, *137*, 102642. [CrossRef]
44. Vidgof, M.; Bachhofner, S.; Mendling, J. Large Language Models for Business Process Management: Opportunities and Challenges. *arXiv* **2023**. [CrossRef]
45. Meira Neto, A.C.; Fantinato, M.; Peres, S.M. An Integrated and Multi-Perspective Framework for Concept Drift Localization and Characterization Combining Statistical Rigor with AI-Driven Reasoning. 2025. Available online: https://ssrn.com/abstract=5679334 (accessed on 21 September 2026).
46. Lakens, D.; Adolfi, F.G.; Albers, C.J.; Anvari, F.; Apps, M.A.J.; Argamon, S.E.; Baguley, T.; Becker, R.B.; Benning, S.D.; Bradford, D.E.; et al. Justify Your Alpha. *Nat. Hum. Behav.* **2018**, *2*, 168–171. [CrossRef]
47. Mannhardt, F. *Sepsis Cases—Event Log*; Eindhoven University of Technology: Eindhoven, The Netherlands, 2016. [CrossRef]

48. Berti, A. PM4Py.LLM: A Comprehensive Module for Implementing PM on LLMs. *arXiv* **2024**, arXiv:2404.06035. [CrossRef]
49. Wang, X.; Jiang, H.; Dong, Y.; Mu, M. Spatial-Channel Collaborative Multi-Scale Graph Interaction Deep Transfer Learning for Unsupervised Rotating Machinery Fault Diagnosis. *Eng. Appl. Artif. Intell.* **2026**, *176*, 114691. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.