The SWORD Builds Itself: LLM-Based Code Generation for Workaround Detection

Johann Müller

Faculty of Business Administration and Economics, Paderborn University, Paderborn, Germany
johann.mueller@uni-paderborn.de

Abstract. Workarounds are goal-driven adaptations of business processes that employees implement to overcome perceived constraints at work. The SWORD framework provides a method for the semi-automated detection of workaround candidates in business process event logs. However, applying SWORD remains resource-intensive and assumes that event logs are sufficiently prepared and comparatively stable. This limits its applicability in dynamic process environments, where concept drift may cause a single event log to contain behavior from multiple process versions. In this paper, we develop and evaluate a code-agent-based extension of SWORD that operationalizes workaround candidate-detection patterns as executable analytics code and applies them under concept drift. Following a Design Science Research approach, we design an IT artifact that first identifies structural process changes, segments event logs into comparatively stable behavioral phases, and then generates and executes SWORD-based detector functions on both complete logs and phase-specific subsets. The artifact is demonstrated using the publicly available BPIC2017 and BPIC2018 event logs. Our contribution is threefold. First, we provide a repeatable workflow for translating SWORD patterns into executable detector logic using a large language model-based code agent. Second, we extend SWORD with concept-drift-aware preprocessing, enabling the comparison of global and phase-specific workaround candidates. Third, we evaluate the artifact's feasibility and limitations, including its ability to recognize patterns that require additional domain knowledge.

Keywords: Workarounds · Workaround Detection · Process Mining · SWORD · Concept Drift · Large Language Models · Code Agents

1 Introduction

Organizations increasingly execute their business processes under dynamic conditions. Technological change, volatile market environments, and changing customer expectations can lead to substantial shifts in process execution [1]. In such settings, employees may develop workarounds to cope with mismatches between prescribed processes and actual work practices. Some of these workarounds may be reflected in event logs recorded by information systems, creating a need for

methods that can analyze workaround behavior under changing process conditions [3].

Process mining provides established techniques for discovering and analyzing process behavior from event logs. In the context of workaround analysis, the [S]emi-automated [WOR]karound [D]etection (SWORD) framework offers a set of patterns for identifying potential workarounds from different perspectives [18]. However, applying SWORD still requires event logs that are sufficiently prepared and suitable for the respective detection patterns [11].

This requirement becomes particularly challenging in dynamic process environments. In process mining, changes in the underlying process during analysis are referred to as concept drift. As a result, a single event log may contain behavior from multiple process versions, which can make aggregated log-level analysis misleading [6]. Existing SWORD-based approaches provide only limited guidance on how to apply workaround-detection patterns when the process itself changes over time.

Prior work shows that LLM-based agents can generate and execute analysis code, structure multi-stage workflows, and support process-mining tasks such as event-log preparation, metric computation, and artifact generation [8,2,15].

However, the current literature lacks a systematic approach that combines workaround detection with the analysis of event logs affected by concept drift. Specifically, there is a gap regarding how SWORD patterns can be applied when processes change fundamentally over time and when event logs contain behavior from multiple process versions. Against this background, this study addresses the following research question:

How can a code-agent-based approach operationalize SWORD patterns as executable and inspectable workaround-candidate detectors for event logs affected by concept drift?

To answer this question, we follow a Design Science Research (DSR) approach and develop a code-agent-based SWORD extension that applies generated detector functions to both complete event logs and drift-aware phase-specific subsets. The artifact is demonstrated and evaluated using the publicly available Business Process Intelligence Challenge 2017 (BPIC2017) and 2018 (BPIC2018) event logs [16,17], which have previously been used to study concept drift in event logs [10,12,7].

The primary contribution of this study is a method-oriented, technical IT artifact that extends SWORD toward concept-drift-aware event-log analysis. The artifact combines drift-aware phase segmentation with LLM-based code generation for SWORD pattern operationalization. Conceptually, the study extends the scope of SWORD's application from prepared, comparatively stable event logs to logs that capture behavior across multiple process versions. Empirically, it demonstrates the feasibility and limitations of this approach using two publicly available BPI Challenge event logs. The contribution is an integrated, inspectable artifact that makes SWORD patterns executable under concept drift.

2 Related Work

Workarounds in business processes help individuals in organizations maintain efficiency, effectiveness, or achieve other important goals despite challenges such as unexpected issues, rigid processes, or conflicting expectations [3]. Depending on their anticipated or realized consequences at the individual, process and organizational levels, workarounds can elicit diverse responses from participants and managers, which range from adoption to prevention or process redesign [4]. Röder et al. [14] show that workarounds can undermine organizational standardization and are often associated with performance losses because they bypass formalized, optimized workflows. At the same time, some deviations may be motivated by the intention to improve speed, flexibility, or the completion of local tasks. This ambivalent role of workarounds highlights that deviations can increase local productivity while also creating inefficiencies, hidden costs, or mismatches elsewhere in the organization. If such interactions between workarounds and process mismatches are not adequately managed, organizations risk an increasing divergence between prescribed processes, actual work practices, and the data recorded in information systems. Such divergence can reduce the business value of process mining, which depends on event data that accurately reflects process execution [3].

To find and analyze deviations and the associated workarounds, there is the process mining method called SWORD framework which can be employed to analyze data mismatches, anomalies, and quality issues in event logs to identify potential workarounds [19]. It consists of 22 patterns classified into four deviation dimensions: control-flow, data, recorded resource, and recorded timestamps. Together, the patterns help identify departures from the prescribed to-be processes, including unintended activities, values that fall outside their usual boundaries, or activities performed by unexpected resources. A concrete example is the control-flow pattern "occurrence of directly repeating activity," which detects when an activity is recorded twice in immediate succession within the same trace. For instance, the trace ⟨A, B, B, C⟩ matches this pattern because activity B is directly repeated. One other example is the SWORD pattern Activity frequency out of bounds. The pattern examines how often an activity occurs within each process case and identifies cases where this frequency falls outside an expected range. For instance, if the activity Request additional documents normally occurs once but is repeated six times in one loan application, the corresponding case may be reported as a potential workaround candidate. Operationalizing the pattern requires at least a case identifier and an activity attribute, as well as a domain-defined or empirically derived reference range. The resulting finding indicates an unusual execution pattern but does not, by itself, establish that a workaround occurred, as this interpretation still requires contextual or domain validation. Löhr et al. [11] propose context-based pattern selection, cross-case patterns, and directed acyclic graphs for impact estimation, but still conclude that the method remains resource-intensive and requires extensive preparation.

LLM-based agents can support analytical workflows by decomposing tasks, generating executable code, and interacting with external tools [20]. In process

mining, prior work discusses LLM applications such as process description, modeling, anomaly detection, root-cause analysis, and process improvement, distinguishing code generation as one central implementation paradigm [5]. Related studies further show that LLMs can generate SQL queries and support event-log extraction, thereby reducing the effort required to prepare process data for analysis [9,15]. Our work follows this code-generation paradigm by using a code-agent-based approach to generate and execute SWORD detector logic for event logs affected by concept drift.

Taken together, prior work shows that LLMs and agents can support process-mining tasks through code generation, SQL query synthesis, and event-log preparation. Yet, existing approaches do not systematically connect these capabilities with established workaround-detection frameworks such as SWORD. In particular, there is limited guidance on how to operationalize SWORD patterns as executable analytics code in a repeatable way, especially for event logs affected by concept drift.

3 Agentic SWORD Analysis under Concept Drift

The SWORD framework was originally developed for semi-automated workaround detection and has since been applied and extended in different organizational contexts [18,19,11]. Although SWORD provides a structured set of workaround-detection patterns, applying these patterns to a concrete event log still requires substantial preparatory and analytical effort [18,11]. An analyst must first understand the event-log schema and map its fields to the information required by each pattern. The analyst must then assess whether additional domain information, such as a process model, is available and translate the pattern definition into corresponding queries or analysis code. Because this procedure must be repeated and validated for each applicable pattern and adapted to different event-log schemas, the main difficulty lies not in executing an existing detector, but in repeatedly translating abstract pattern definitions into log-specific executable logic.

This operationalization challenge becomes more pronounced in the presence of concept drift, as a single event log may contain behavior generated by different process versions [6,1]. To address these challenges, we develop a code-agent-based extension of SWORD that generates executable analytics code for selected SWORD patterns and supports their application to event logs affected by concept drift.

LLMs are used in our artifact to support these steps. They can jointly interpret textual pattern definitions and a compact representation of the event-log schema and generate pattern-specific executable code without requiring a manually implemented detector for every pattern-log combination. Therefore, the study examines whether LLM-based code generation can support the repeated operationalization of SWORD patterns while preserving inspectable detector logic.

Following a DSR approach, the artifact is developed to address the need for repeatable and transparent analytical procedures in workaround detection [13]. We conceptualized and implemented an LLM-based agent that translates SWORD pattern definitions into executable detector logic.

To address evolving process behavior, the pipeline applies an LLM-assisted concept-drift analysis to the time-ordered event log, as illustrated in Fig. 1. The log is partitioned into consecutive fixed-size case windows, for which the system computes behavioral summaries covering control-flow, resource, collaboration, and temporal features. Comparing adjacent windows preserves temporal order and reveals changes that would be obscured in a full-log aggregation [6,1]. The LLM uses these summaries to propose candidate change points and segment the log into comparatively stable behavioral phases. The LLM then compares adjacent window summaries and identifies structural change points. Based on these change points, the log is segmented into comparatively stable behavioral phases. The generated SWORD detectors are subsequently executed on both the complete log and each phase-specific subset, enabling comparison of global and phase-specific workaround candidates. Windowing is used only to identify behavioral changes; the additional full-log execution preserves candidates whose relevant behavior may extend across phase boundaries. The artifact is demonstrated

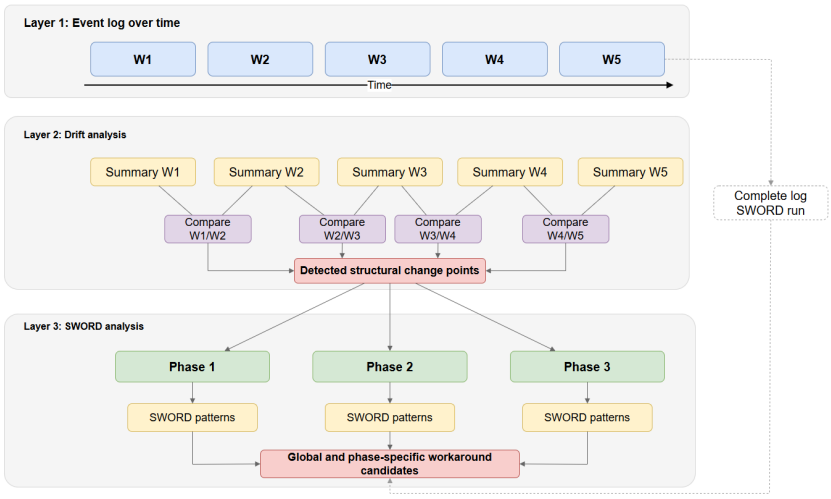

Fig. 1. Concept-Drift-Aware SWORD Agent Pipeline.

on BPIC2017, containing 1,202,267 events across 31,509 loan applications, and BPIC2018, containing 2,514,266 events across 43,809 grant applications [16,17]. Both logs cover processes known to exhibit temporal behavioral changes.

4 SWORD Builds Itself: Operationalizing SWORD with a Code Agent

Figure 2 shows how the agent generates, executes, and stores SWORD detector functions.

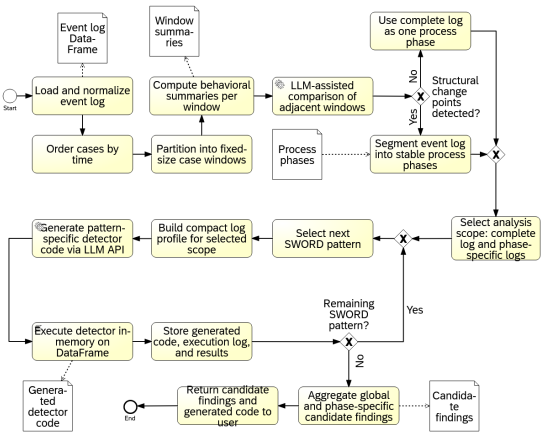

Fig. 2. Workflow of the LLM-based SWORD pattern detection agent.

The artifact's implementation is available in an open-source repository.[1] The code agent receives an event log, selected SWORD patterns, and configuration metadata as input. Since the complete log cannot be passed to the LLM, the agent first creates a compact log profile containing column names, inferred metadata, and representative samples. For each SWORD pattern, the LLM is prompted to generate exactly one Python function, run_detector(df), which receives a pandas DataFrame and returns structured pattern-specific findings. The generated code normalizes common event-log attributes to a canonical schema and is restricted to local execution using pandas, NumPy, PM4Py, and the Python standard library.

The reported runs used the model identifier gpt-5.5. The temperature parameter is not explicitly set; provider defaults therefore apply. The runtime prompts are embedded directly in the implementation and consist of two prompt pairs: one for LLM-assisted concept-drift detection and one for SWORD detector generation. The implementation records LLM, parsing, and execution failures without retrying failed requests. Generated code is instructed to avoid network access and libraries beyond pandas, NumPy, PM4Py, and the Python standard library, but these restrictions are not enforced through an import allowlist or isolated sandbox. Code, configurations, execution logs, change points, and findings are stored for auditability.

[1] `https://github.com/mjohann99/SWORD_LLM.git`

If a pattern cannot be operationalized from the event log alone, the generated function must return an explicit information requirement instead of inferring missing domain knowledge. Otherwise, the function may derive data-driven thresholds from the observed log behavior. The agent executes each detector on the complete log or the identified phases.

In our implementation, several SWORD patterns are operationalized through explicit empirical decision rules rather than generic anomaly labels. For instance, the detector for *Activity frequency out of bounds* computes activity-specific frequencies at the case level and derives Tukey fences from the observed distribution using Q_1, Q_3, the interquartile range IQR, and the bounds $Q_1 - 1.5 \times IQR$ and $Q_3 + 1.5 \times IQR$. The detector for *Time between activities out of bounds* applies the same logic to directly-follows transition durations and computes pair-specific lower and upper bounds in seconds. Similarly, *Duration of trace out of bounds* applies empirical Tukey fences to trace durations. Since detectors are executed separately for each analysis scope, thresholds are scope-specific: phase-level executions produce phase-specific thresholds, whereas full-log executions produce global thresholds.

5 Demonstration and Evaluation on BPIC Event Logs

This section evaluates the artifact on the BPIC2017 and BPIC2018 benchmark logs. The evaluation focuses on phase-aware drift segmentation, the plausibility of the detected drifts, the execution of the SWORD pattern catalog, and the agent's ability to recognize patterns that require missing domain knowledge. Since the benchmark logs do not contain ground-truth labels for workarounds, the resulting findings are interpreted as potential workaround candidates rather than verified workarounds.

5.1 Concept-Drift-Aware Phase Segmentation

To identify periods of behavioral change, both benchmark logs were analyzed chronologically using consecutive case windows and LLM-assisted change-point detection. Each detected phase represents a behaviorally distinct analysis scope described by its time span, number of cases, and number of events. As shown in Table 1, the drift analysis identified seven phases for BPIC2017 and eight phases for BPIC2018. The BPIC2017 phases reflect changes within the loan-application process, while the BPIC2018 phases follow the yearly structure of grant-application handling. Since cases may span longer periods, phase time spans can overlap. The phases should therefore be interpreted as case-window-based behavioral regimes rather than strictly disjoint calendar intervals.

5.2 Plausibility of the Detected Concept Drifts

To assess the plausibility of the detected concept drifts, we compare the identified phase structures with prior drift analyses of the same benchmark logs. This

Table 1. Temporal phases identified in the BPIC2017 and BPIC2018 event logs.

Log	Phase	Cases	Events	Time span
BPIC2017	0	14000	547647	2016-01-01–2017-01-05
BPIC2017	1	5000	185873	2016-06-28–2016-12-20
BPIC2017	2	2000	73928	2016-08-17–2017-01-16
BPIC2017	3	2000	72163	2016-09-06–2017-01-26
BPIC2017	4	500	19207	2016-09-26–2016-11-28
BPIC2017	5	4500	168349	2016-10-01–2017-02-01
BPIC2017	6	3509	135100	2016-11-17–2017-02-01
BPIC2018	0	10458	621483	2014-05-04–2018-01-19
BPIC2018	1	3507	224255	2015-05-11–2018-01-19
BPIC2018	2	1024	64008	2015-05-15–2018-01-19
BPIC2018	3	13377	696407	2016-04-06–2018-01-19
BPIC2018	4	1062	63367	2016-05-17–2018-01-19
BPIC2018	5	13407	778577	2017-04-03–2018-01-19
BPIC2018	6	430	25210	2017-05-13–2018-01-18
BPIC2018	7	544	40959	2017-05-15–2018-01-19

comparison is not intended as a strict ground-truth validation, since the studies differ in feature representations, windowing strategies, and change-point detection procedures. Instead, it serves as an orientation point for evaluating whether the LLM-assisted drift analysis identifies changes in similar temporal regions.

For BPIC2017, prior work reports process-level change points around days 196–200 and 327–330 [10]. Although our analysis does not reproduce these points exactly, Table 1 shows comparable mid-year and late-year changes, particularly the transition from phase 0 to later phases and the start of phase 6 in November 2016. For BPIC2018, prior work reports two major drift points at the beginning of new yearly application periods: a stronger drift between the first and second year and a weaker drift between the second and third year [12]. The phases in Table 1 similarly cluster around the 2014/2015, 2016, and 2017 application periods, supporting the interpretation that BPIC2018 contains multiple process versions rather than one stable process. Compared with prior studies that identify a few prominent change points, the seven BPIC2017 and eight BPIC2018 phases provide more fine-grained candidate behavioral regimes. This granularity creates additional local analysis scopes to determine whether SWORD candidates are globally unusual or specific to particular periods, although the phases should not be interpreted as objectively validated versions of the process.

5.3 Execution of Event-log-Applicable SWORD Patterns

Across the two benchmark logs, the agent returned candidate findings for 19 of the 26 event-log-applicable pattern-log combinations (26 = 13 applicable patterns, times two for both BPI challenges). Coverage was strongest for control-flow, frequency, resource-sharing, and time-related patterns because these can largely be operationalized using standard event-log attributes. In contrast, lower-coverage patterns more often require normative or domain-specific information

Table 2. Applicable SWORD patterns across BPIC2017 and BPIC2018.

SWORD pattern	BPIC2017	BPIC2018
2. Occurrence of recurrent activity sequence	✓	✓
3. Activity frequency out of bounds	✓	✓
7. Occurrence of directly repeating activity	✓	✓
10. Change in value between events	✓	×
12. Activity executed by unauthorized resource	∅	∅
13. Activities executed by multiple resources	✓	✓
14. Activities executed by a single resource	✓	×
17. Occurrence of activity outside of time period	✓	✓
18. Delay between start of trace and activity is out of bounds	✓	✓
19. Time between activities out of bounds	✓	✓
20. Duration of activity out of bounds	✓	∅
21. Duration of trace out of bounds	✓	✓
22. Delay between event and logging is out of bounds	∅	∅

✓: pattern compiled and returned candidate findings; ×: pattern compiled and executed, but returned no candidate findings; ∅: agent returned an information requirement instead of candidate findings.

that is not directly observable in the benchmark logs. In contrast, several patterns with lower coverage require normative or domain-specific information that is not directly observable in the benchmark logs, such as required activities. BPIC2017 produced candidate findings for nearly all applicable patterns, including value changes between events, single-resource traces, and activity-duration deviations. BPIC2018 showed a narrower candidate profile: the corresponding detectors compiled and executed, but no full-log candidates were returned for value changes between events or activities executed by a single resource. In addition, the agent returned information requirements for unauthorized-resource detection, activity-duration detection in BPIC2018, and delay-between-event-and-logging detection, because these patterns require additional information such as authorization rules, reliable activity-duration timestamps, or separate logging timestamps.

5.4 Recognition of Missing Domain Knowledge

We also evaluate whether the agent reports missing information rather than inferring domain knowledge unavailable in the log.

The results show that the agent correctly recognizes several domain-knowledge requirements in both benchmark logs. As shown in Table 2, several patterns consistently return information requirements instead of candidate findings. This indicates that the agent does not infer missing target activities, process models, activity sets, or free-text configurations from the event log.

However, the results also reveal limitations for patterns that require domain-defined reference values. For several empirically bounded patterns, the agent derives relations or thresholds from observed data rather than reporting missing domain knowledge. As shown in Table 3, this occurs for unusual neighbor-

Table 3. Recognition of missing domain knowledge for selected SWORD patterns.

SWORD pattern	17	18	Interpretation
Occurrence of an activity	∅	∅	Requires target activity
Occurrence of activities in an order different from process model	∅	∅	Requires process model
Occurrence of mutually exclusive activities	∅	∅	Requires mutually exclusive sets
Occurrence of unusual neighboring activities	×	×	Agent derives empirical relations
Missing occurrence of activity	∅	∅	Requires required activity
Data object with value outside boundary	×	△	Agent derives empirical boundaries
Specific information in free-text fields	∅	∅	Requires free-text configuration
Activity frequency out of bounds for a resource	×	×	Agent derives resource-frequency bounds
Value frequency out of bounds for a resource	×	×	Agent derives value-frequency bounds

17: BPIC2017; 18: BPIC2018. ∅: missing domain knowledge correctly reported; ×: candidate findings wrongly produced; △: mixed result across scopes.

ing activities, resource-specific frequency bounds, and value-boundary patterns. These outputs may be useful for exploratory anomaly detection, but they do not constitute a strict SWORD operationalization when normative relations or domain-defined bounds are required.

5.5 Discussion

The phase-segmentation results support the initial premise that aggregated analysis may obscure phase-specific deviations. The artifact therefore enables global and phase-specific SWORD candidates to be interpreted in their temporal context.

The evaluation on BPIC2017 and BPIC2018 indicates that LLM-based code generation can operationalize several event-log-applicable SWORD patterns, particularly control-flow, frequency, resource-sharing, and time-related patterns. This suggests that code agents can reduce the manual effort required to implement SWORD detector logic. At the same time, the generated code remains inspectable, which is essential for transparency, reproducibility, and analyst validation.

The findings must nevertheless be interpreted carefully. The benchmark logs do not contain ground-truth workaround labels; thus, the detected findings are potential workaround candidates rather than verified workarounds. Moreover, several SWORD patterns require domain knowledge beyond the event log, such as process models, authorization rules, required activities, or domain-defined thresholds. Although the artifact correctly reports information requirements in

several cases, it sometimes derives empirical thresholds where normative reference values would be required. Finally, the detected process phases should be understood as behaviorally distinct analysis scopes rather than objectively fixed process versions. Drift-aware SWORD analysis therefore supports, but does not replace, expert interpretation and organizational validation.

Because the temperature was not configured, the provider default was used, and sensitivity to alternative settings remains unknown. Higher values could increase variability in generated code, change points, and workaround candidates, primarily affecting stability and reproducibility. Although each stored run remains auditable, controlled repeated experiments are required to assess this effect.

Compared with established drift detectors, the LLM can jointly interpret heterogeneous behavioral summaries without separately specified detection rules, but it provides less controlled, reproducible, and statistically calibrated change-point decisions.

Future work should evaluate the artifact in real organizational settings where domain experts can validate whether candidate findings correspond to actual workarounds. Further research should also integrate domain artifacts such as process models, business rules, role models, authorization matrices, and data dictionaries to support stricter SWORD operationalization. The drift-segmentation component is modular and could be replaced by or combined with an established concept-drift detector. This study did not evaluate such configurations because it focuses on the feasibility of the integrated SWORD workflow rather than on benchmarking drift-detection methods. Comparative and hybrid evaluations, therefore, remain future work.

References

1. Adams, J.N., van Zelst, S.J., Rose, T., van der Aalst, W.M.P.: Explainable concept drift in process mining. Information Systems **114**, 102177 (2023). https://doi.org/10.1016/j.is.2023.102177
2. Antonov, A., Kourani, H., Berti, A., Park, G., van der Aalst, W.M.P.: PMAx: An Agentic Framework for AI-Driven Process Mining. arXiv preprint arXiv:2603.15351 (2026)
3. Bartelheimer, C., Löhr, B., Reineke, M., Aßbrock, A., Beverungen, D.: Workarounds as a Cause of Mismatches in Business Processes—Insights from a Multiple Case Study. Business & Information Systems Engineering **67**(3), 339–356 (2025). https://doi.org/10.1007/s12599-025-00943-5
4. Beerepoot, I., van de Weerd, I.: Prevent, Redesign, Adopt or Ignore: Improving Healthcare Using Knowledge of Workarounds. In: Proceedings of the 26th European Conference on Information Systems (ECIS 2018), Portsmouth, United Kingdom (2018)
5. Berti, A., Kourani, H., H"afke, H., Li, C.-Y., Schuster, D.: Evaluating Large Language Models in Process Mining: Capabilities, Benchmarks, and Evaluation Strategies. arXiv preprint arXiv:2403.06749 (2024)
6. Bose, R.P.J.C., van der Aalst, W.M.P., Žliobaitė, I., Pechenizkiy, M.: Dealing with concept drifts in process mining. IEEE Transactions on Neural Networks

and Learning Systems **25**(1), 154–171 (2014). https://doi.org/10.1109/TNNLS.2013.2278313
7. Denisov, V.V., Belkina, E., Fahland, D.: BPIC'2018: Mining Concept Drift in Performance Spectra of Processes. BPI Challenge 2018 report (2018). https://doi.org/10.4121/uuid:3301445f-95e8-4ff0-981f1f204972
8. Hong, S., Lin, Y., Liu, B., Liu, B., Wu, B., Zhang, C., et al. Data Interpreter: An LLM Agent for Data Science. In: Che, W., Nabende, J., Shutova, E., Pilehvar, M.T. (eds.) Findings of the Association for Computational Linguistics: ACL 2025, pp. 19606–19627. Association for Computational Linguistics, Vienna (2025). https://doi.org/10.18653/v1/2025.findings-acl.1016
9. Jessen, U., Sroka, M., Fahland, D.: Chit-Chat or Deep Talk: Prompt Engineering for Process Mining. arXiv preprint arXiv:2307.09909 (2023)
10. Klijn, E.L., Mannhardt, F., Fahland, D.: Multi-perspective Concept Drift Detection: Including the Actor Perspective. In: Advanced Information Systems Engineering. CAiSE 2024. Lecture Notes in Computer Science, vol. 14663, pp. 141–157. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-61057-8_9
11. Löhr, B., Bartelheimer, C., Köhne, F., Nordlohne, S., Alile, D., Latten, A.: Forging the LongSWORD: Exaptation and Enhancement of the SWORD Framework for Workaround Detection. In: Business Process Management Workshops. Springer Nature Switzerland, Cham (2025). https://doi.org/10.1007/978-3-031-78666-2_24
12. Pauwels, S., Calders, T.: Detecting and Explaining Drifts in Yearly Grant Applications. arXiv preprint arXiv:1809.05650 (2018)
13. Peffers, K., Tuunanen, T., Rothenberger, M.A., Chatterjee, S.: A Design Science Research Methodology for Information Systems Research. Journal of Management Information Systems **24**(3), 45–77 (2007)
14. Röder, N., Wiesche, M., Schermann, M., Krcmar, H.: Why Managers Tolerate Workarounds—The Role of Information Systems. In: Proceedings of the 20th Americas Conference on Information Systems (AMCIS 2014), Savannah, GA, USA (2014)
15. Stein Dani, V., Dees, M., Leopold, H., Busch, K., Beerepoot, I., van der Werf, J.M.E.M., Reijers, H.A.: Event Log Extraction for Process Mining Using Large Language Models. In: Cooperative Information Systems. CoopIS 2024. Springer, Cham (2024)
16. van Dongen, B.: BPI Challenge 2017. 4TU.ResearchData. Dataset (2017). https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b
17. van Dongen, B., Borchert, F.: BPI Challenge 2018. Eindhoven University of Technology. Dataset (2018). https://doi.org/10.4121/uuid:3301445f-95e8-4ff0-98a4-901f1f204972
18. van der Waal, W., Beerepoot, I., van de Weerd, I., Reijers, H.A.: The SWORD is Mightier Than the Interview: A Framework for Semi-automatic WORkaround Detection. In: Di Ciccio, C., Dijkman, R., del Río Ortega, A., Rinderle-Ma, S. (eds.) Business Process Management. BPM 2022, LNCS, vol. 13420, pp. 91–106. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-16103-2_9
19. van der Waal, W., van de Weerd, I., Beerepoot, I., Lu, X., Kappen, T., Haitjema, S., Reijers, H.A.: Putting the SWORD to the Test: Finding Workarounds with Process Mining. Business & Information Systems Engineering **67**, 171–190 (2025). https://doi.org/10.1007/s12599-023-00846-3
20. Xi, Z., Chen, W., Guo, X., He, W., Ding, Y., Hong, B., et al.: The Rise and Potential of Large Language Model Based Agents: A Survey. arXiv preprint arXiv:2309.07864 (2023)