MDPI

Review

Computational Scalability in Process Mining: A Systematic Map and Thematic Synthesis

Alessandro Berti *

Process and Data Science (PADS), RWTH Aachen University, Ahornstraße 55, 52074 Aachen, Germany
* Correspondence: a.berti@pads.rwth-aachen.de

Abstract

Process mining turns event data into insight about how processes operate and how they can improve. Making that insight available for larger datasets and continuously arriving events requires methods that use computation and memory effectively. This review brings together a systematic literature map and a synthesis of selected full texts to explain how computational strategies support event data engineering, process discovery, conformance checking and predictive process monitoring. It organizes methods around four ways of changing computational work: reducing it, parallelizing it, moving it and reusing earlier results. Across these tasks, useful scalability depends on the required answer, the structure of the workload and the costs of preparing, executing and maintaining an analysis. Reuse can make repeated analyses cheaper; decomposition can reduce difficult searches; and continuous operation requires explicit policies for retaining and updating information. These connections provide a practical framework for comparing methods and identifying which experiments would make their benefits clearer. The review helps researchers and practitioners connect computational choices to the process insights they need.

Keywords: process mining; high performance computing; big data; systematic literature review; distributed computing; event streams

1. Introduction

Process mining uses recorded event data to discover process behavior, compare it with a reference model through conformance checking and predict the future development of ongoing executions [1]. Process discovery constructs a model of observed behavior; conformance checking assesses agreement with a given model; prediction estimates future activities, durations or outcomes. Obtaining these results also requires extracting, correlating and organizing event data. Computational feasibility matters across this analysis pipeline: an answer is useful only if the data and computation can be handled within the available resources and the required time.

The scalability challenge differs across tasks and workloads. Additional events increase data access and preparation work, while a larger activity alphabet can expand process discovery problems and a complex reference model can enlarge conformance checking search spaces. Object-centric analyses also depend on relationships between events and objects. Continuous operation adds the need to maintain information as new events arrive. Computational difficulty therefore depends on event count together with task-specific structure and operating conditions.

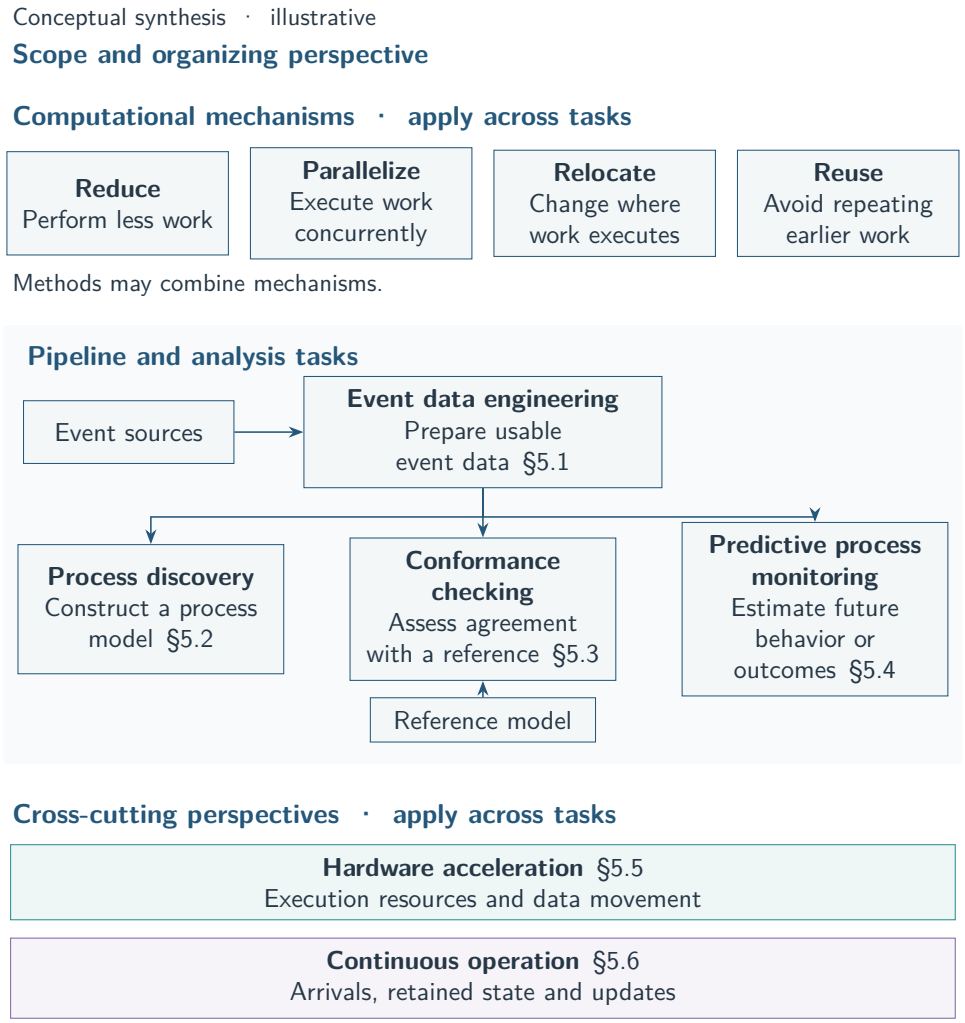

Figure 1. Conceptual synthesis: scope and organizing perspective of the review. Event data engineering supports process discovery, conformance checking and predictive process monitoring as distinct analyses. Conformance checking also receives a reference model. Across these analyses, methods reduce, parallelize, relocate or reuse computational work. Hardware acceleration and continuous operation add execution and state management considerations. Arrows show the inputs required by each analysis, which can be performed separately.

Methods address these challenges by changing the amount, allocation, location or repetition of computational work. They *reduce* work through pruning or compact representations, *parallelize* operations that can execute concurrently, *relocate* computation to suitable execution locations, and *reuse* prepared data or earlier results. These mechanisms are complementary. Algorithmic and memory improvements on one computer belong within this scope alongside high performance computing and distributed data systems; adding hardware is one possible response to a bottleneck.

Choosing among these possibilities requires specifying the required result and the conditions of use. A method may preserve an exact answer, restrict the supported inputs or estimate a different output. Preparation can be worthwhile for repeated requests even when it increases the cost of the first analysis. The total benefit of accelerating an operation depends on the surrounding stages: decomposed process discovery can shorten local model construction while introducing model reduction that takes days [2]. Meaningful comparison therefore aligns the answer requirements, the workload and all relevant stages of computation.

Table 1. Positioning against three purposefully selected reviews. Cells summarize the identified author versions and their contribution to the comparison.

Dimension	Process discovery review [3]	Encoding review [4]	Drift survey [5]	Present review
Task scope	Automated process discovery and model languages.	Encodings across tasks; anomaly detection benchmark.	Process level drift and online process discovery.	Computational contributions across process mining tasks.
Mechanisms	Algorithm designs, decomposition and process discovery runtime.	Representation families and encoding resource costs.	Detection, windows, forgetting and maintained abstractions.	Reduce, parallelize, relocate and reuse computational work.
Event data	Preparation within process discovery; filtering in benchmark.	Encoding is the central preparation problem.	Features, event order and stream windows.	Preparation, storage, querying and downstream analysis.
Continuous operation	Incremental methods discussed; fixed log benchmark.	Partial traces discussed; fixed log benchmark.	Central focus on memory, latency and adaptation.	Common operational profile across relevant tasks.
Analytical properties	Model constructs, soundness and empirical quality.	Expressivity and empirical prediction quality.	Drift detection, localization and characterization.	Output, assumptions, guarantees, diagnostics and history.
Evidence	Review plus direct implementation benchmark.	Review plus encoding time/memory benchmark.	Full paper synthesis and evaluation critique.	Search map and full text synthesis of published experiments.

Checked versions: process discovery, arXiv v3 (2018); encoding, arXiv v1 (2023); drift, DOI-linked arXiv v1 (2021). Citations identify the canonical publications.

This review connects those requirements to methods across event data engineering, process discovery, conformance checking and predictive process monitoring. Hardware acceleration and continuous operation provide additional perspectives on how these tasks execute and maintain their results. Figure 1 shows their relationship. The practical question is: *given a requested analysis, a workload and a required result, which computational strategies are plausible, and what evidence is needed to compare them?* The review answers it by explaining the work that methods change and the conditions under which their results and costs can be compared.

1.1. A computational perspective across process mining tasks

Reviews of process discovery, trace encoding and concept drift provide complementary accounts of particular tasks and technical concerns [3–5]. This review connects such perspectives through a common computational question: which work does a method change, under which workload conditions, and with what consequences for the analytical result and the cost of obtaining it? Table 1 makes this added perspective concrete against three purposefully selected, checked review designs. The comparison focuses on these three review designs and the computational perspective they provide.

1.2. Research questions for comparing scalable methods

RQ1 Which tasks, workload dimensions and computational bottlenecks do the methods address?

RQ2 Which mechanisms reduce, parallelize, relocate or reuse computational work, and which analytical properties do they preserve or change?

RQ3 Which workload dimensions, resource outcomes, measurement boundaries and implementation evidence do the assessed papers report?

RQ4 Which testable research priorities follow from the relationships between these mechanisms, costs and analytical properties?

The review contributes an integrated account of computational scalability that connects the work performed by a method with the result it provides and the costs of obtaining and maintaining that result. First, an *explanatory map* relates tasks, workload dimensions and bottlenecks to reduction, parallelization, relocation and reuse. Building on this organization, a *comparison framework* examines whether methods satisfy comparable outputs, assumptions, guarantees, diagnostic needs and historical scope, and whether measurements cover comparable work. These comparisons, in turn, support an *evaluation agenda* that identifies experiments needed to resolve questions about preparation and reuse, structural complexity, sustained operation and method composition. The map thus organizes the methods, the comparison framework identifies the conditions for choosing among them, and the agenda specifies what further evidence those choices require.

The map in Section 2 and thematic synthesis in Section 5 primarily address **RQ1** and the mechanism component of **RQ2**. The contract matrix in Section 5.3 and evidence atlas in Section 4 provide concrete dimensions for comparing analytical requirements and measured work, connecting **RQ2** and **RQ3**. The cross-task findings in Section 6 translate unresolved comparisons into the experiments in Section 6.7, addressing **RQ4**. These outputs support method selection under stated workload and answer requirements.

The *search map* is the classified collection of all 923 publications that met the initial eligibility criteria: a process mining task and a substantive computational contribution. It describes the literature identified by the searches, including publications beyond those cited in the narrative. Citations select sources that explain a mechanism, establish its analytical conditions or support a computational comparison. The *full text synthesis* examines 176 original studies from that citation selection in greater depth. Section 3.4 defines these collections and the selection decisions connecting them.

The rest of the paper is organized as follows. Section 2 introduces the framework for computational scalability and the dimensions used to compare methods. Section 3 describes the search strategy, screening and evidence extraction. Section 4 presents the evidence atlas, summarizing publication coverage, computational mechanisms and reporting patterns. Section 5 synthesizes the literature across process mining tasks, hardware acceleration and continuous operation. Section 6 connects the findings to method selection and future research priorities. Section 7 discusses threats to validity, and Section 8 concludes the paper.

2. A framework for computational scalability

2.1. Task and data structure determine computational cost

In a case-centric log, an event records an activity occurrence associated with a case, usually with a timestamp and further attributes. A trace $\sigma = \langle a_1, \ldots, a_k \rangle$ orders activities within one case. The log L is a multiset of traces, allowing repeated identical executions. Let C, V and A denote the numbers of cases, distinct trace variants and activity labels, respectively. These counts and the event count $N = \sum_{\sigma \in L} |\sigma|$ can grow at different rates. Grouping identical traces can be valuable when $V \ll C$, whereas candidate enumeration may remain difficult when A is large.

Process discovery constructs a model from recorded behavior. Conformance checking compares behavior with a reference model; alignments express this through matched and unmatched model and log moves. Predictive process monitoring estimates future outcomes from execution prefixes. Event preparation and querying provide the representations, correlations and subsets these analyses require. Each task has its own costs: filtering

Table 2. Workload dimensions and the costs they expose. Their relevance depends on the task and operating conditions.

Dimension	Computational concern
Event count N	Reading, filtering, sorting and storage
Cases C and variants V	Repeated computation and result reuse
Activity alphabet A	Candidate relations and process discovery structures
Reference model structure	Alignment state space and decomposition boundaries
Event and object relationships	Joins, traversal and correlation
Arrival rate and active cases	Throughput, latency and retained state

reduces data access work, while alignment search also depends on the trace and reference model.

Object-centric data allows an event to relate to several objects. OCEL motivates this representation through convergence and divergence problems caused by imposing a single case notion [6]. Object-centric analysis also depends on the numbers of objects and event-object relationships and their distributions. Its computational cost can therefore change even when event count remains fixed.

2.2. Parallelism, decomposition and incremental computation

High performance computing is used here in a task-oriented sense: parallel execution, specialized hardware, or algorithmic and memory improvements that make process mining computation feasible or faster at a stated scale. Big data methods also address high-volume storage, continuous arrival and heterogeneous event data. The scale at which a log becomes computationally demanding depends on the task, representation, model, resources and required result quality or latency.

Three distinctions guide classification. Data parallelism partitions data; task parallelism executes separate computations concurrently. Distributed execution introduces communication across workers or machines, whereas shared memory and GPU execution have different transfer and synchronization costs. Decomposition changes a problem into smaller subproblems and can reduce work on one machine; distributed deployment additionally requires allocating and coordinating those subproblems. The early distribution taxonomy motivates these distinctions for process mining [7].

Streaming methods must manage retained state under continuing input. Windows, summaries and incremental updates may bound memory while approximating historical behavior or limiting the answer's temporal scope. This differs from accelerating a computation that still requires the complete log. Representation also affects cost: event dataframes motivate redesigning algorithms around columnar operations [8], while database execution places work within storage and query operators.

2.3. Workload dimensions expose different bottlenecks

The overview in Figure 1 locates the tasks and operating perspectives. Table 2 summarizes the workload dimensions that can make their computations expensive.

2.4. Classifying computational work and required answers

Table 3. Classification axes used for full text extraction.

Axis	Question answered
Task or pipeline stage	What computation is performed?
Workload and bottleneck	What causes time, memory, communication or state growth?
Computational mechanism	What work is reduced, run concurrently, relocated or reused?
Operating regime	Does execution use a fixed log, incremental updates or an event stream, and where does it run?
Analytical contract	What output is promised, under which assumptions and scope?

The six themes form two visible groups. Event data engineering is a pipeline layer, followed by the process discovery, conformance checking and prediction tasks. Hardware acceleration and continuous operation are cross-cutting perspectives on all four. They provide overlapping reading paths through the same methods. Table 3 separates these roles. For example, a GPU implementation of streaming process discovery has a process discovery task, a hardware mechanism and a streaming regime. Its primary theme is only a convention for counting publications.

We use *analytical contract* to mean the output, supported inputs and assumptions, guarantee, diagnostic detail and historical scope that a method promises. This is an extraction framework proposed by this review. It supports comparison of methods against the required answer. An exact answer for a restricted model class, an exact result over a recent window and an estimate for the full log answer different questions. Fields requiring further evidence retain the value unspecified.

Four overlapping mechanism codes describe changes to work. Reduction removes or simplifies computation through pruning, sampling or compact representations. Parallelization executes work concurrently. Relocation moves computation into databases or nearer to event sources. Reuse avoids repeating earlier computation through indexes, summaries, cached results or shared models. A method may combine several codes.

What an experiment measures is different from which stages it includes. An experiment may focus on an operation, include its preparation, or vary the workload dimension that causes complexity. We record both the computational measure and the experimental change, while keeping theoretical bounds separate. This supports comparison of what each evaluation establishes under its stated conditions.

2.5. Separating concepts, counts and experimental results

Every figure identifies its evidence role. *Conceptual synthesis* shows mechanisms, comparisons or illustrative examples organized by the review. *Descriptive evidence* reports publication counts from an explicitly identified collection. *Reported experiment* reproduces numerical observations from one identified study and workload; any sum calculated by the review is marked separately. Conceptual node sizes, arrows and timelines explain relationships; measured costs appear on explicitly labeled axes.

For quantitative displays, captions state the counting unit, denominator, overlap rule and missing information policy. Blue intensity represents reporting share. “Unspecified” marks a field or mechanism whose details remain to be established; “not applicable” marks a field outside the task’s scope. These values describe evidence coverage. Blank illustrative cells indicate conceptual choices, while numerical zeros describe coded observations within the stated collection. The contract matrix and output illustration in Section 5.3 make the answer conditions concrete.

3. Review methodology

3.1. Review design, search cutoff and full text assessment

This systematic map uses the area classification approach of Petersen et al. [9]. PRISMA 2020 informs selection reporting and PRISMA-S informs search documentation [10,11]. A locally defined protocol guided conduct from before database screening. The purpose is to organize computational contributions and compare the conditions under which methods apply.

The database search cutoff was 11 September 2026 (UTC). It covered available publication history without language, subject, venue or citation count restrictions. Records available by that date could be eligible even if their issue year was later. Annual counts use adjudicated publication years, with verified first online years replacing issue years beyond the cutoff. The incomplete final year and indexing delays limit trend interpretation.

A protocol amendment on 14 September 2026 added full text assessment and expanded computational extraction. The review distinguishes the search map, the publications selected for citation and the original studies retained for full text synthesis. Section 3.4 defines membership in each collection and its role in the review.

3.2. Searching across tasks and computing mechanisms

The broad computational scope has a methodological consequence. Relevant work appears under process discovery, conformance checking, process querying, workflow terminology and streaming, as well as process mining. Computing terms may describe business process behavior or an application setting without identifying a computational contribution. The search therefore combines task vocabulary with mechanism vocabulary and checks their relationship during screening.

Four exploratory Elicit searches requested 20 records each, producing 80 ranked appearances and 71 distinct identifiers. They covered scalable process mining, streams and memory, hardware and decomposition, and storage and querying. These ranked results supplied vocabulary and known publications for checking query coverage.

Table 4. The seven search queries contributing candidates. Counts are distinct provider identifiers retrieved by each query. The same record can occur in several rows; eligibility is assessed after combining and reconciling the results.

Stage	Boolean expression	OpenAlex	Scopus
Q01	`"process mining" AND ("high performance computing" OR "big data")`	265	271
Q02	`"process mining" AND ("high performance computing" OR "big data" OR parallel OR distributed OR scalable OR scalability OR MapReduce OR Hadoop OR Spark OR GPU)`	1,242	924
Q03A	`("process mining" OR "process discovery" OR "conformance checking") AND ("high performance computing" OR "big data" OR parallel OR distributed OR scalable OR scalability OR MapReduce OR Hadoop OR Spark OR GPU OR CUDA OR multicore OR "multi-core" OR acceleration OR decomposition)`	1,698	1,163
Q03B	`("process mining" OR "process discovery" OR "conformance checking") AND (streaming OR "event stream" OR "event streams" OR "bounded memory" OR "out of core" OR "in memory" OR "large scale" OR "large event logs" OR "event log storage" OR "column store" OR "columnar" OR dataframe OR "data frame" OR "relational database" OR "graph database")`	828	469
Q04	`("process mining" OR "process discovery" OR "conformance checking" OR "predictive process monitoring" OR "process querying" OR "process query language") AND (SQL OR "column store" OR columnar OR dataframe OR "data frame" OR "in memory" OR "out of core" OR "bounded memory" OR "parallel computing" OR "distributed computing" OR "cloud computing" OR "event correlation" OR NoSQL OR MongoDB)`	735	192
Q05	`("workflow mining" OR "workflow discovery" OR "predictive process monitoring") AND ("big data" OR "high performance" OR parallel OR distributed OR scalable OR scalability OR streaming OR "event stream" OR "bounded memory" OR GPU OR Spark OR MapReduce OR cloud OR federated)`	106	100
Q06	`("process mining" OR "workflow mining" OR "process discovery" OR "conformance checking" OR "predictive process monitoring") AND (efficient OR efficiency OR online OR incremental OR approximation OR "computational complexity" OR "memory consumption" OR "execution time" OR "training time")`	2,593	2,108

Quotation marks delimit exact phrases in the generic queries. The candidate pool is the union of all seven queries across both databases.

Table 4 gives the seven Boolean queries used to retrieve candidates. OpenAlex and Scopus were searched through Scimesh 0.3.0. Scopus searched titles, abstracts and keywords, while OpenAlex used its combined title and abstract field. Each query combines task terms with computing terms. Their roles are:

Q01 The initial topic: process mining with high performance computing or big data.

Q02 Parallel and distributed execution, scalability and named computing platforms.

Q03A Process discovery and conformance checking, with hardware acceleration and decomposition terms.

Q03B Streaming, bounded memory, event storage and data representations for the same three tasks.

Q04 Process querying and predictive monitoring, with database, cloud and event correlation terms.

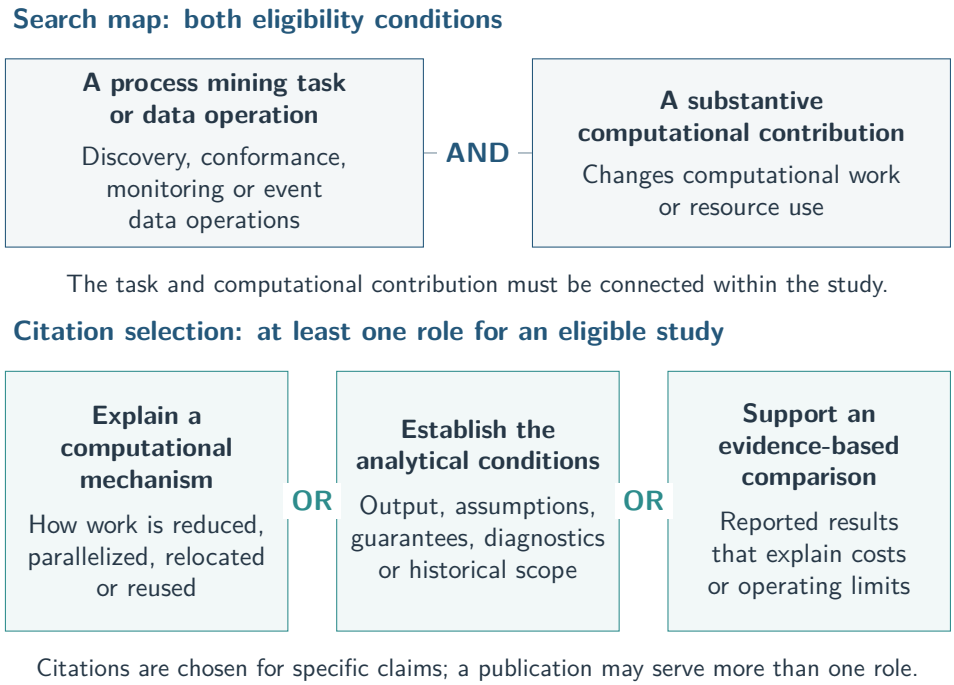

Figure 2. Screening criteria for original studies. Search map eligibility requires both a process mining task or data operation **AND** a substantive computational contribution, connected within the study. Among eligible mapped studies, a citation must explain a computational mechanism **OR** establish analytical conditions **OR** support an evidence-based comparison. Citation selection depends on the specific claim being supported; a publication may serve several roles.

Q05 Workflow mining and workflow discovery terminology, alongside predictive monitoring.

Q06 Efficient, online, incremental and approximate methods, including single-machine algorithms.

All seven queries contributed to one candidate pool: a record retrieved by any query entered that pool. We retrieved every result page, combined the records across queries and databases, reconciled repeated records and publication versions, and then applied the eligibility criteria below. Thus, the queries retrieve candidates and screening selects eligible publications. Checks against known publications helped expand the vocabulary. The search cutoff remained 11 September 2026 throughout the subsequent full text assessment.

3.3. Selecting computational contributions to process mining

Initial screening assessed titles, abstracts and bibliographic metadata. Figure 2 summarizes eligibility and the citation roles described below. An original study entered the search map when this evidence established both of the following:

1. **A process mining task or data operation.** The study addresses process discovery, conformance checking, predictive or prescriptive monitoring, or event preparation, storage or querying specifically for process mining. **AND**
2. **A substantive computational contribution.** The study develops or evaluates a method or infrastructure that changes computational work or resource use, such as parallel execution, specialized hardware, scale-oriented algorithms, compact state, approximation or process-specific data systems.

A single-machine algorithm, a formal computational construction or a comparative resource evaluation can satisfy the second criterion. Resource measurements are recorded separately

from eligibility. The task and the computational contribution must be connected in the study.

Screening requires a substantive computational contribution in addition to any general efficiency claim, predictive accuracy result, business-resource improvement or model level concurrency. Routine applications outside this scope are excluded. Reviews, books, tutorials and auxiliary evaluation resources serve as background unless they also contribute an original computational method or evaluation. Non-research containers, identified retractions and documented duplicate versions are excluded. Uncertain decisions identify the evidence needed to resolve eligibility.

Provider identifiers, normalized DOIs and title/year matches support identity reconciliation. Ambiguous versions and metadata conflicts were inspected individually. A metadata audit on 14 September 2026 corrected supported bibliographic discrepancies and linked exact publication duplicates. Related conference and journal contributions may remain separate. Counts therefore refer to publications, which can share implementations or experiments.

3.4. From the search map to citations and assessed studies

Search map: every initially eligible publication. Combining and reconciling the database results produced 5294 distinct candidate publications. The search map contains all 923 candidates assigned an include decision under the two criteria above. Each mapped publication has a bibliographic identity, an eligibility rationale and a classification of its task and computational approach. The map supplies the publication-year and thematic counts for the broader literature. Membership is independent of whether a publication is cited in the narrative. The map preserves the initial screening snapshot; subsequent full text decisions are reported separately.

Citation selection: sources for specific explanations and comparisons. The initial draft was developed through *purposive selection*: publications were chosen for the contribution they made to the discussion. For mapped original studies, citation serves at least one of three roles:

1. **Explain a computational mechanism.** Describe how a method reduces, parallelizes, relocates or reuses work for a process mining task, including a concrete algorithm, representation or implementation. **OR**
2. **Establish the analytical conditions.** Support a statement about the method's output, assumptions, guarantee, diagnostic detail or historical scope. **OR**
3. **Support an evidence-based comparison.** Supply a reported computational evaluation or a contrasting result needed to explain workload sensitivity, preparation, maintenance, resource cost or a method's operating limits.

These roles connect a citation to the particular claim it supports. Selection used judgment about which sources explained the approaches and contrasts discussed across the six themes. Related publications can serve complementary roles, such as introducing a method and evaluating an extension. A publication's eligibility for the map and its usefulness at a particular point in the narrative are therefore separate decisions. The resulting citation set is an explanatory selection, with its composition determined by the discussion. Background sources are cited for definitions, methodological guidance, supporting resources or comparison with earlier reviews, and remain outside original-study reporting counts.

From the initial bibliography to full text synthesis. The initial draft cited 188 mapped publications. It also used one process mining book for background and three methodology or reporting guides to establish the review approach. These sources formed the 192-reference bibliography from which full text assessment began. Assessment covered 186

references and retained 176 original studies for detailed synthesis, with 8 contextual or methodological sources and 2 exclusions. Six references remained covered by title and abstract screening. Access to full texts affected assessment coverage, while the two eligibility criteria determined inclusion in the synthesis.

All 176 original studies retained after full text assessment are cited in this paper. The 8 assessed contextual or methodological sources and three positioning reviews bring the final bibliography to 187 references. The three separately assessed reviews cover process discovery, trace encoding and concept drift. The historical count of 192 identifies the starting bibliography for assessment; the final citation set reflects the full text decisions and the positioning comparison. Publication coverage summaries describe the 923 mapped publications. Detailed computational reporting describes the 176 original studies, with subgroup denominators stated in each display.

3.5. Extracting computational evidence and its limits

Records were screened using the eligibility criteria above, with unclear cases reviewed separately. Full text assessment followed a common codebook and recorded decisions, reasons and supporting passages.

Elicit supported scoping and targeted recovery of bibliographic records and abstracts. OpenAI Codex assisted with screening, full text assessment, metadata reconciliation, extraction, synthesis and manuscript drafting. This assistance supported the review workflow; it does not constitute independent duplicate human screening.

We recorded the five classification axes, computational results, experimental settings, measurement limits and reported code releases. For streaming methods, we also recorded how information was updated and retained. Measurements, theoretical results and indirect resource indicators were kept separate; missing details were marked as unspecified. Central and borderline claims were checked against the full texts. Scripts organized the records and calculated descriptive counts.

3.6. Comparing mechanisms, guarantees and measured costs

The six themes organize the synthesis, with one primary theme per publication for reporting counts. Task and mechanism assignments can overlap. Comparisons consider what methods compute, their assumptions and guarantees, and the conditions under which costs were measured. The task-by-mechanism matrix counts publications assigned both codes.

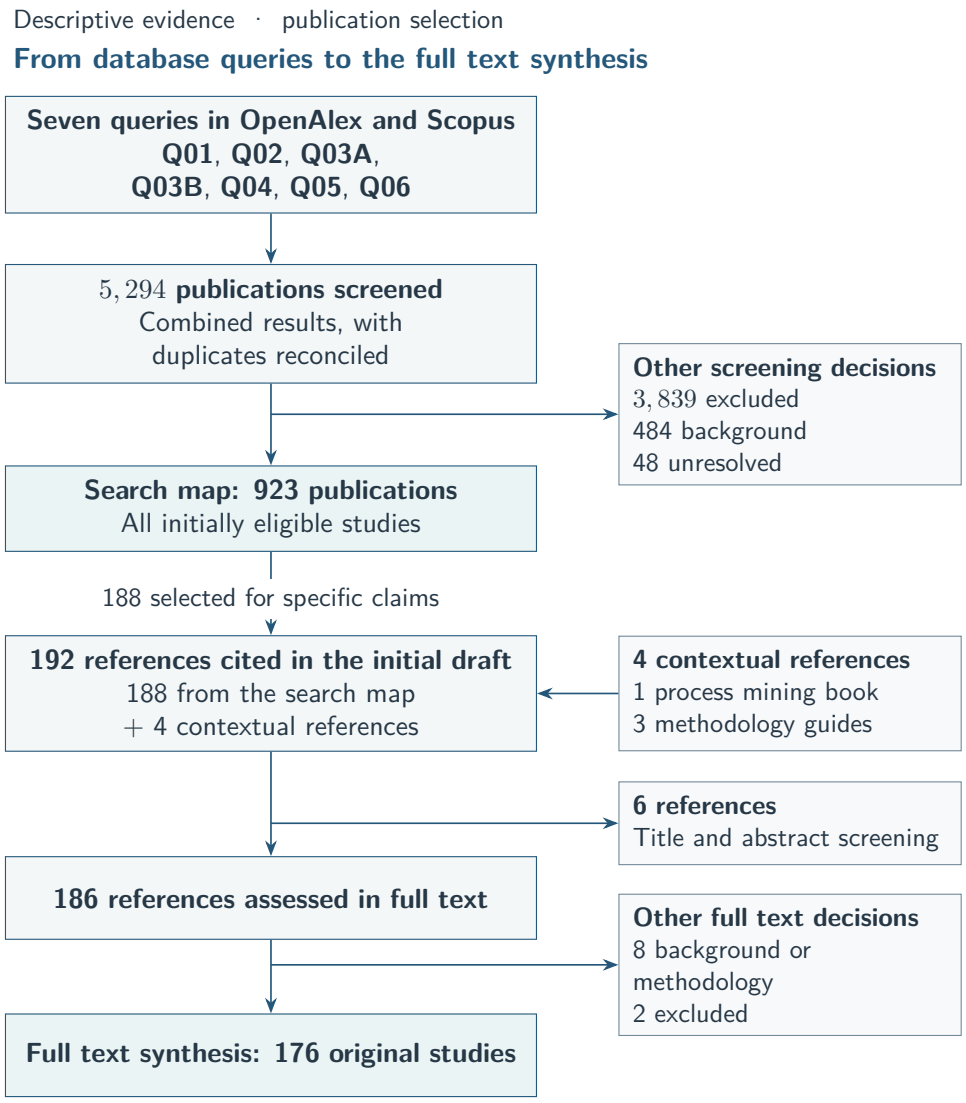

Figure 3. Descriptive evidence: publication selection from database searches to full text synthesis. The search map contains all 923 initially eligible publications. Of these, 188 were selected for citation alongside four contextual references. Full text assessment retained 176 original studies. Side branches show the other decisions and the six references covered by title and abstract screening.

Computational reporting uses assessed full texts; publication counts use the search map. Comparisons use the original themes, while the reporting heatmap uses updated classifications.

4. Evidence atlas

4.1. The synthesis examines a selected subset of the map

Figure 3 follows the selection from database queries to the full text synthesis. The seven queries produced 12704 retrieval appearances. Removing 4883 repeated appearances left 7821 provider records. Reconciling 2527 source and version duplicates yielded 5294 publication units. Initial screening mapped 923 original publications, excluded 3839, classified 484 as background and retained 48 with unresolved scope.

The initial draft cited 188 mapped publications and four contextual references. Section 3.4 explains the citation roles and their relationship to map membership. Full text assessment covered 186 of these 192 references. It retained 176 original studies, classified

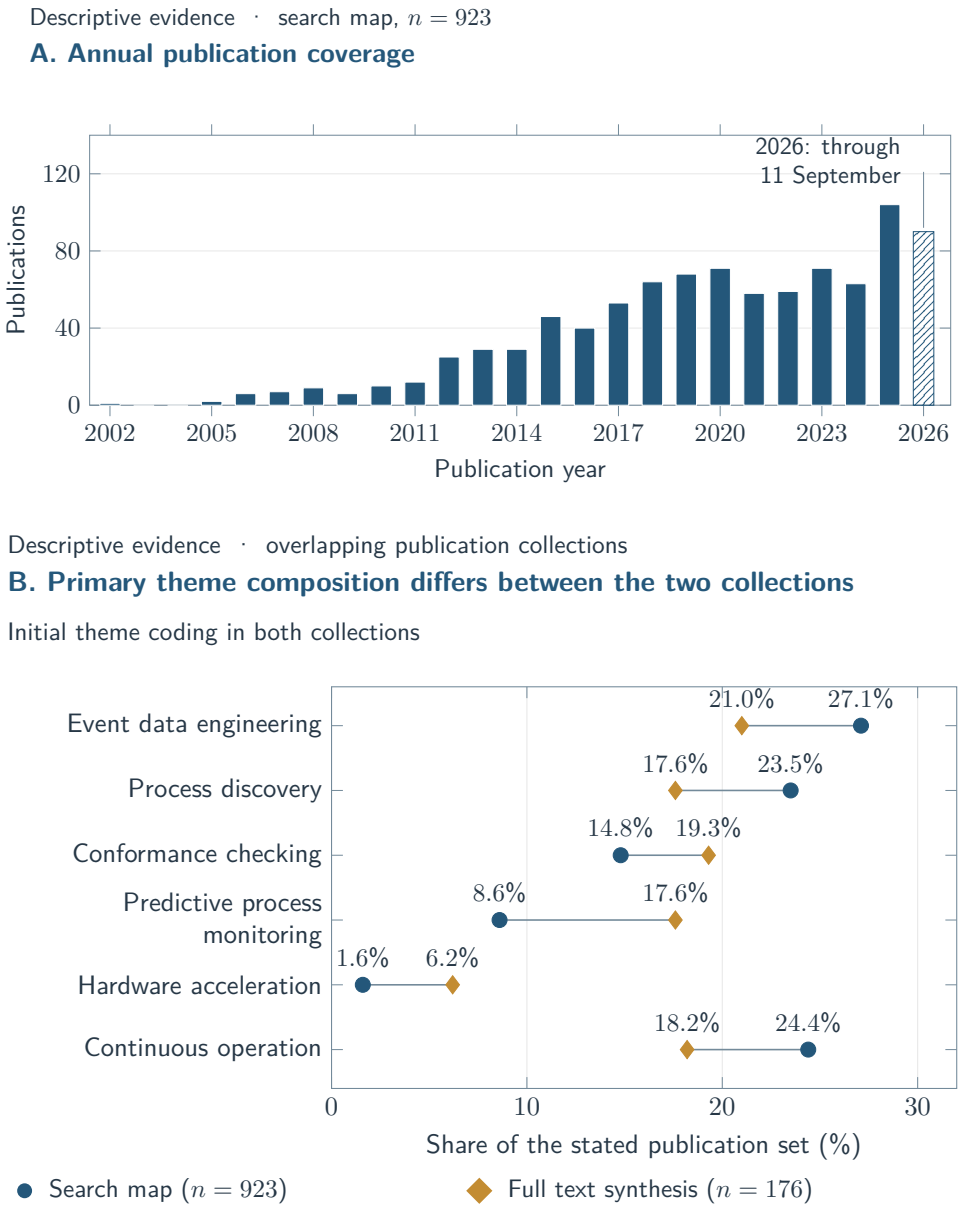

Figure 4. Descriptive evidence. A: publication years in the search map ($n = 923$); the 2026 window ends on 11 September. B: primary theme shares in the search map and full text synthesis ($n = 176$), using the initial assignment in both. Each publication occurs once per panel/series; all have adjudicated years and themes. One publication has a different current theme assignment (Section 3.6). The collections overlap, and related publications can share implementations and experiments.

8 as background or methodological guidance and excluded 2. All assessed references received a resolved decision, with 6 changes from initial eligibility judgments. The other six references remained covered by title and abstract screening. The search map retains its initial membership, while the full text synthesis uses the reassessed original studies.

The publication is the counting unit. Related conference and journal papers may describe shared implementations or experiments. Reporting percentages describe the 176-study full text synthesis; estimating prevalence across the broader search map would require a representative assessment of that population.

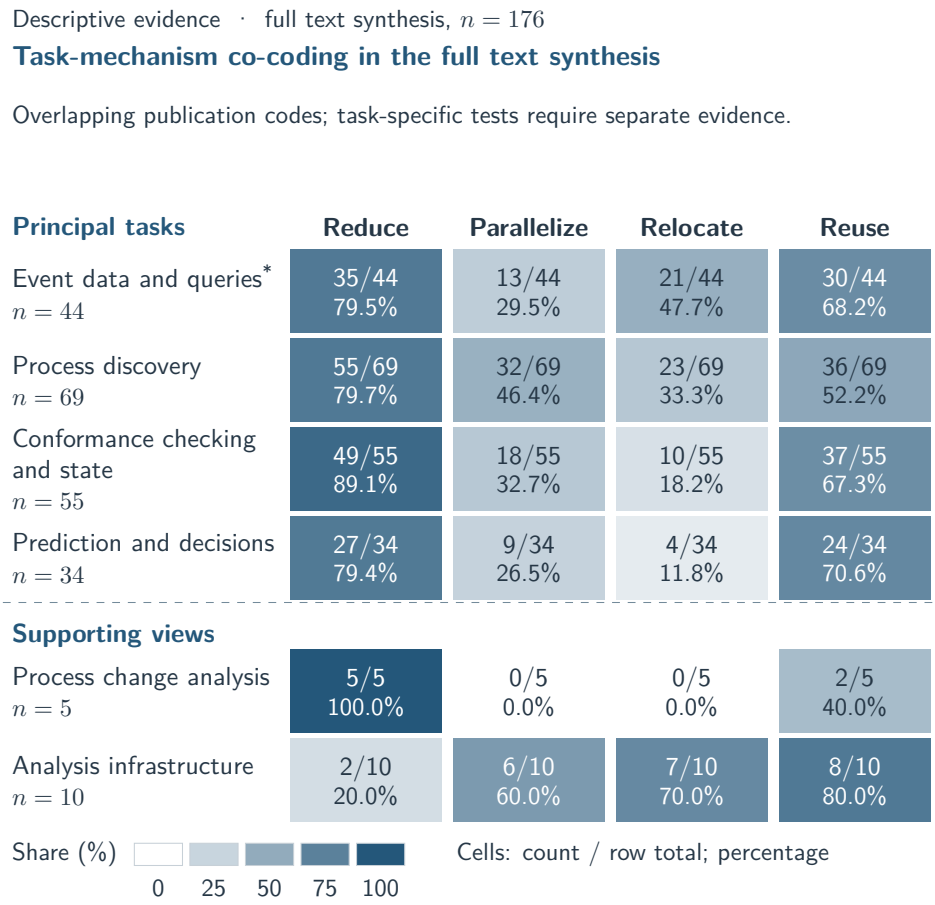

Figure 5. Descriptive evidence: task and mechanism assignments in the full text synthesis ($n = 176$). Publications count at most once per cell; task rows and mechanism columns overlap. Each row denominator counts publications with that task assignment. Every study has a task assignment; the one with unspecified mechanism coding remains in its task denominator (marked by an asterisk). Counts describe publication level co-coding, including formal constructions and infrastructure. Implementation and evaluation are assessed from study-specific evidence. Zero marks a cell with zero coded publications in this collection.

4.2. Publication selection changes the balance of themes

Figure 4 compares publication coverage and corpus composition. Both composition series use initial theme assignments. Prediction and hardware occupy larger shares of the full text synthesis than of the search map. These differences describe selection. The hardware category counts one primary reading path per publication; distributed process discovery can belong to the process discovery category.

Peer review status and the date of the first public version require publication-specific verification beyond provider categories and dates.

4.3. Reduction and reuse span process mining tasks

Figure 5 makes the four mechanisms visible across task boundaries. Each cell counts publications assigned both its task and mechanism codes. The rows include task-specific preparation and infrastructure only where recorded. Prediction and decisions include explanation and prescription; conformance checking and state include quality measures and state estimation. Process change analysis and general analysis infrastructure remain

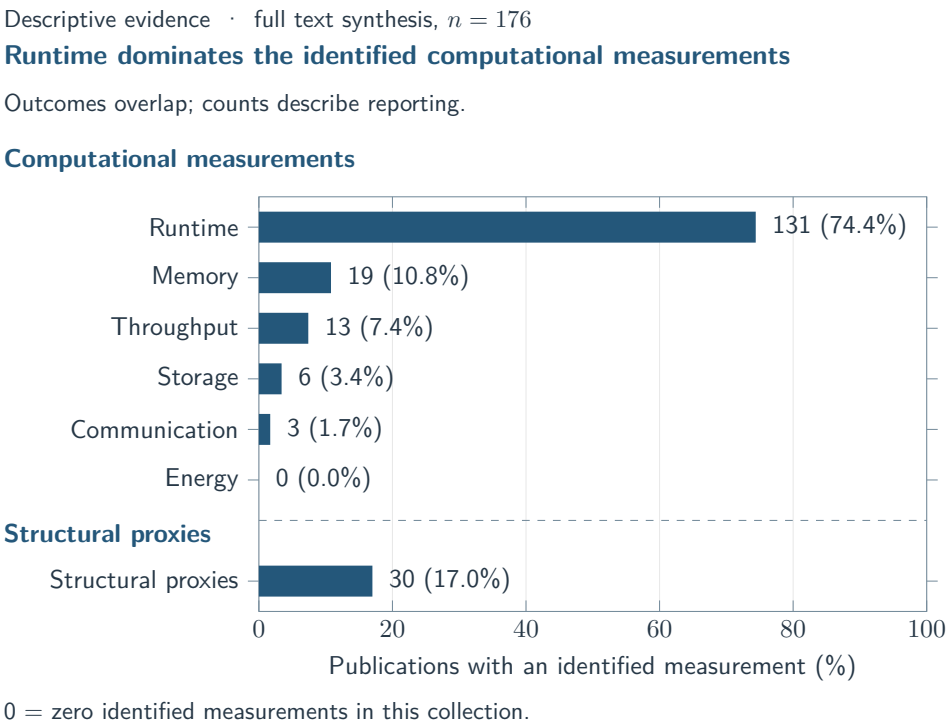

Figure 6. Descriptive evidence. Bars show the percentage of 176 original studies in the full text synthesis with each identified measurement; outcomes overlap. Structural proxies are separate from byte level memory measurements. A zero records zero identified measurements for that outcome. The chart summarizes reporting coverage within the assessed collection.

distinct supporting views. The row populations overlap, so one publication can contribute to several row totals.

Reduction and reuse recur across all four principal task rows. A publication can combine them with parallelization or relocation, as the thematic diagrams explain. Their frequency in the full text synthesis describes the mechanisms discussed, rather than their relative effectiveness.

4.4. Runtime dominates reporting of computational costs

The clearest reporting imbalance is between runtime and the other computational outcomes (Figure 6). This assessment identified runtime in 131/176 studies (74.4%), memory in 19/176 (10.8%), communication in 3/176 (1.7%) and energy in 0/176. Structural proxies such as parameter, model or retained state counts are displayed separately from byte level memory measurements. Computational outcomes quantify resource use; predictive quality and business performance describe separate dimensions of an evaluation.

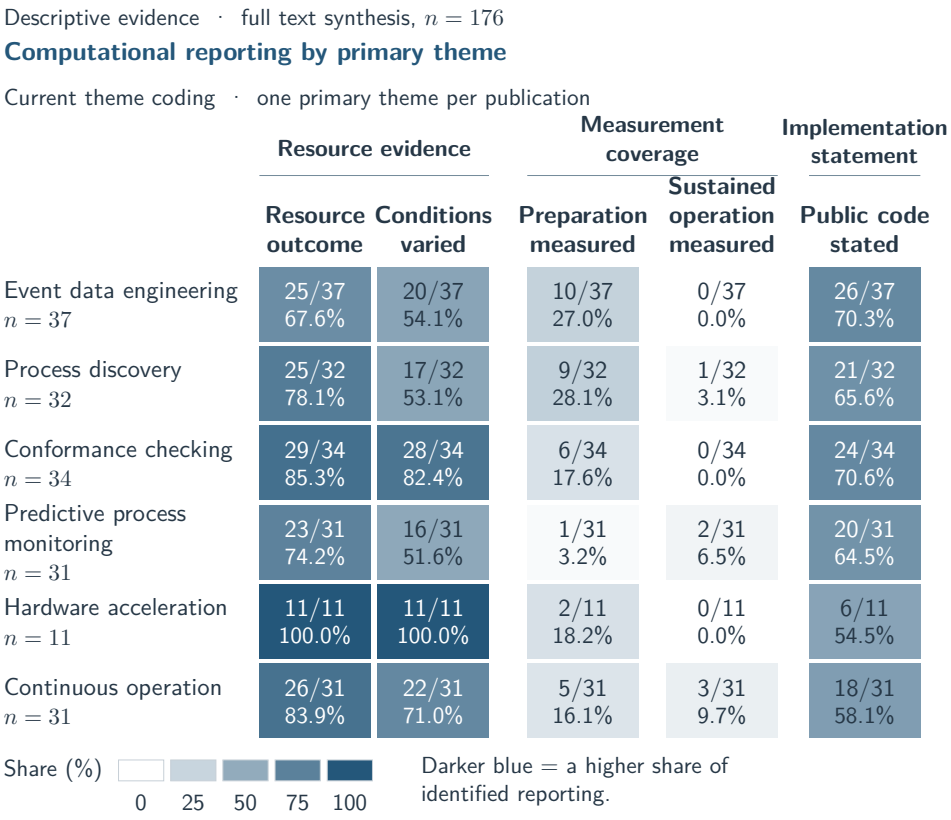

Figure 7. Descriptive evidence: computational reporting in the full text synthesis ($n = 176$). *Resource outcome* counts publications reporting at least one computational measurement or counted resource indicator. *Conditions varied* counts the subset comparing such outcomes across workloads, computing resources or method settings, including parameter comparisons. Each publication has one primary theme; columns overlap. Cells show the publication count, row total and percentage, with darker blue indicating a higher share. Current theme assignments differ from the initial coding for one publication (Section 3.6).

Figure 7 compares computational reporting across current primary themes. For example, 25 of the 37 event data engineering studies reported a resource outcome, and 20 also compared outcomes across conditions. Both percentages use all 37 studies as the denominator.

The assessment identified measurements including preparation costs in 33 of the 176 studies (18.8%) and measurements of sustained operation in 6 (3.4%). The preparation indicator records coverage of preparation stages; the particular stages included remain study-specific. Public code stated records an explicit code release statement in the assessed publication.

The measurement boundary changes the interpretation. Importing and preparing data add costs beyond the timed operation. Recurrent analyses can reveal whether maintained

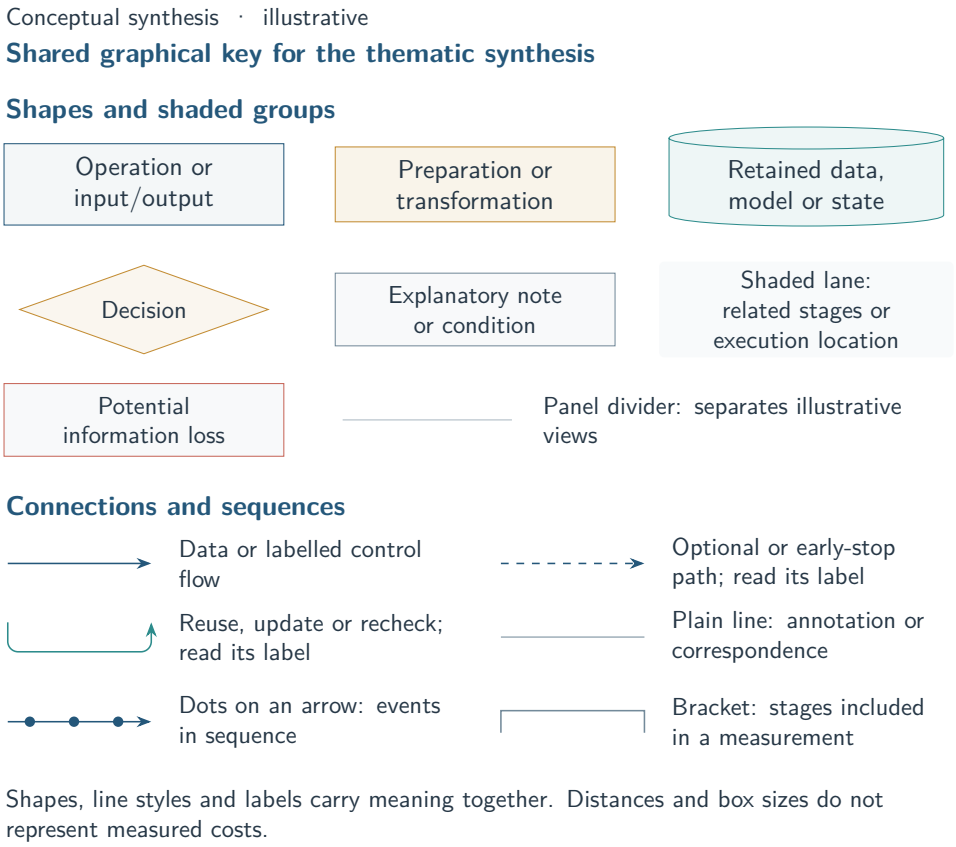

Figure 8. Conceptual synthesis: shared graphical key for the thematic synthesis. Shapes distinguish operations, preparation, retained information and decisions. Lines, arrows, event markers, shaded lanes and brackets describe relationships and measurement coverage. Labels specify the meaning of each route; the layout does not encode measured costs.

intermediates recover their construction cost. Continuous service also depends on retained cases, ordering and updates. These distinctions guide the following thematic comparisons of published experimental evidence.

5. Thematic synthesis

Figure 8 collects the graphical elements used in the thematic diagrams. Rectangles show operations or inputs and outputs, ochre boxes identify preparation, cylinders show retained information, and diamonds mark decisions. Callouts explain conditions or possible information loss, while shaded lanes group related stages or execution locations. Arrows show data or labelled control flow; return paths identify reuse, updates or rechecking. Dashed arrows mark optional or early-stop routes, while plain lines connect notes or corresponding events. Timeline dots represent events, and brackets identify the stages included in a measurement. Each subsection also opens with a compact overview of Reduce, Parallelize, Relocate and Reuse. Its terms identify complementary choices, not a required sequence of steps.

Preparing event data and scaling process mining tasks

Event data engineering prepares the inputs; process discovery, conformance checking and prediction perform different analyses on them. Each view follows the same questions: where work grows, how mechanisms change it, what answer is preserved, what the evaluation establishes and when the mechanism is useful.

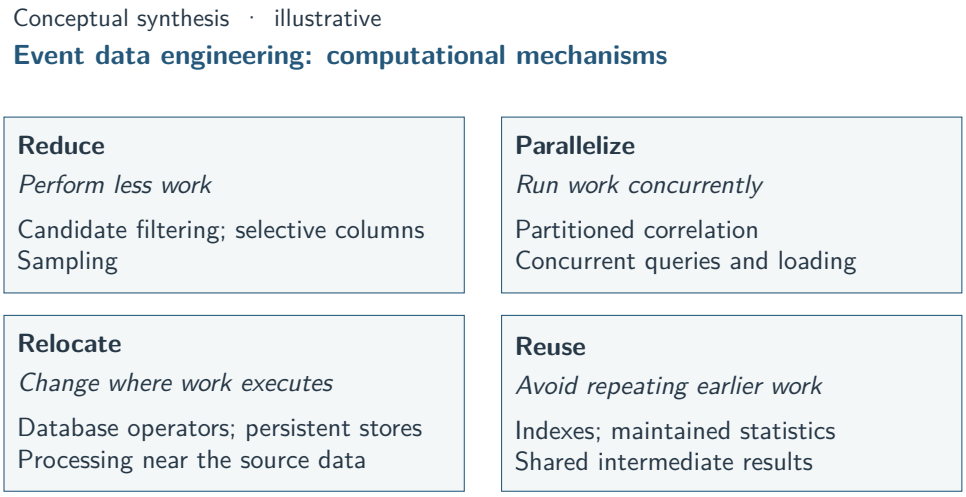

Figure 9. Conceptual synthesis: key terms for event data engineering. The four mechanisms connect candidate selection, concurrent processing, execution location and maintained data. These are complementary choices, with benefits and information requirements explained below.

5.1. Event data engineering: efficient access to usable inputs

Figure 9 groups the main computational choices: selecting less data, processing independent work concurrently, moving operations to stored data and reusing prepared results.

Event data engineering constructs usable event data from source records and provides access to those data for subsequent analysis. Records may need to be extracted, interpreted as activity occurrences, related to cases or objects, ordered and represented with suitable attributes. The result can be an event log, an object-centric collection or a queryable representation. Constructing and maintaining this input is distinct from discovering a model, performing conformance checking or making a prediction over it. The chosen representation determines what information those later computations can use.

5.1.1. Correlation and representation drive preparation costs

The thematic synthesis begins with data preparation because every task considered in this review depends on an event representation that must first be constructed and made accessible. A discovery or conformance checking experiment may begin with a prepared log, whereas an operational analysis must also obtain that log from source records. Locating these earlier costs establishes which part of the pipeline an acceleration claim covers and which workload dimensions should be examined. It also gives the later task sections a common starting point: their computational demands must be understood together with the cost of supplying the information they require.

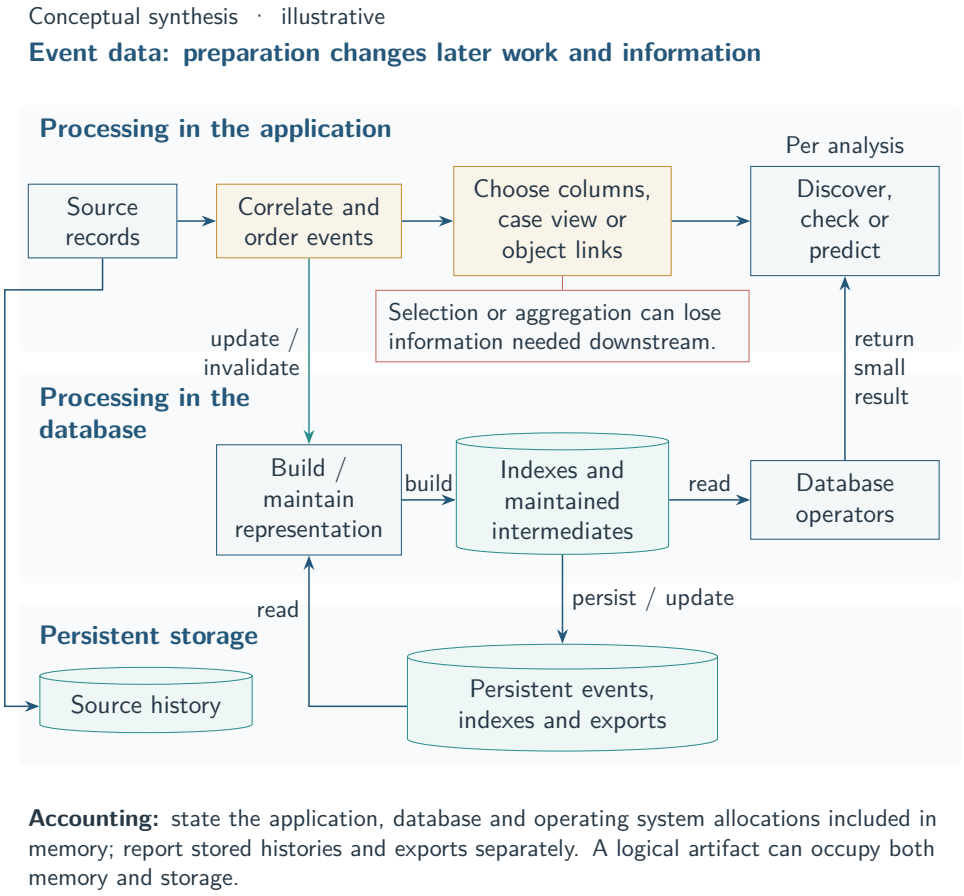

Figure 10. Conceptual synthesis. Event data engineering: correlation and representation feed repeated analyses across processing in the application, processing in the database and persistent storage. Lanes identify execution and storage roles; a logical artifact can have allocations in more than one accounting category. The diagram brings together mechanisms from the studies discussed in this section. Maintained intermediates and relocation are illustrated by DB-XES and relational XES, while exports can change information content [12–14].

Correlation can require searching many candidate record combinations; extraction and repeated scans increase data access work; and event-object relationships can enlarge joins and intermediate structures. Preparation cost depends both on the input volume and on the relationships, ordering and representation needed by subsequent analyses. Figure 10 separates preparation, repeated analysis and the resource boundaries.

5.1.2. Reducing preparation through filtering, storage and reuse

The preparation bottlenecks above lead to several possible interventions, each changing a different part of the work needed to supply an analysis. Correlation methods decide which record combinations to examine; storage designs determine how events are accessed; maintained intermediates carry earlier computation into later requests. Bringing these contributions together allows the review to compare reduction, parallelization, relocation and reuse at the same pipeline stage. The distinction matters when choosing between a single extraction and a sequence of analyses over growing data: a method that saves repeated scans may first require construction and continuing maintenance. The studies below are therefore organized around the work they change and the circumstances in which that work recurs.

Correlation search distributes tests of which source records belong together. The early MapReduce method searches for atomic and composite correlation conditions [15]. Its extended Event Correlation Analytics study compares sorting, hashing and an additional partitioning pass, then contrasts one-job and multipass searches through composite conditions [16]. The experiments vary message volume and worker count, assuming each reducer can hold its correlated-message buffer. Candidate counts, intermediate transfers and uneven work determine the benefit of distributing those tests.

Filtering and reuse can reduce correlation work before it is distributed. RF-GraP uses attribute statistics to filter candidates, a graph partitioner to group overlapping work, and reusable correlated-message structures to avoid repeated processing [17]. Its Spark evaluation measures runtime and remote shuffle bytes while varying events, attributes and workers. Relative to distributing candidate tests alone, these mechanisms also reduce their number and repeated inputs. The resulting correlations satisfy the selected predicates and thresholds; recovering the true business case assignment requires validating those choices against domain evidence.

Online correlation also limits candidate matches through time windows and domain rules. The Esper implementation measures latency, throughput and memory, but long-lived concurrent cases increase candidate state and the output can contain multiple probabilistic case assignments per event [18]. Its processing latency is separate from the waiting needed to close a correlation window.

Context aggregation reduces the representation searched during correlation. An Aggregated Correlation Graph groups service messages by context and indexes their attributes, allowing heuristic filtering of candidate correlations [19]. Repetition can keep the graph small, while new contexts and relationships can continue to enlarge it. Its compactness therefore depends on the diversity of the incoming records.

Domain rules can instead turn source observations into activity events directly. Generated IoT detection services reuse expert-selected sensor signatures as CEP rules [20]. Partial signatures permit earlier detection but can confuse activities with similar patterns; manufacturing and healthcare experiments establish detection quality, with systematic load and latency profiling deferred. A blockchain extraction layer applies configurable event abstractions to ledger state changes [21]. Its vehicle-manufacturing demonstration establishes the transformation, with latency measurement needed to evaluate its performance claim. These studies supply event construction mechanisms that can be evaluated further through load testing.

Sampling reduces input size according to a chosen preservation target. The statistical process discovery framework stops after sufficiently many sampled traces add no new information under a chosen abstraction [22]. Its probability statement concerns novelty of the next trace under the sampling assumptions. Representative-trace selection instead optimizes distributional fidelity using Earth Movers' Distance, including practical embeddings that avoid a full pairwise cost matrix [23]. Novelty and distributional fidelity preserve different aspects of a sample. Properties of the discovered model require their own preservation guarantees.

Privacy imposes another selection contract. TraVaS avoids constructing possible private variants by applying noisy partition selection directly to observed variant counts [24]. It guarantees differential privacy under its contribution assumptions and positive privacy slack, while potentially removing rare behavior. Its experiments measure utility; a computational comparison would additionally quantify the cost of the reduced representation. Here, the reduced representation must be judged against both privacy and the behavior retained for analysis.

Storage and maintained intermediates. Keeping event data outside application memory can increase the input that fits, while buffered access and retained intermediates change later query costs. Relational XES and XESLite implement alternatives to loading the complete log into application memory. Relational XES provides buffered access to database-resident events and measures both memory and access time, including database memory in its comparison [13]. XESLite offers several implementations behind the OpenXES interface, with different costs and capabilities: its compressed trace representation retains activity sequences and multiplicities, while its richer memory and external-storage variants support attributes. Their loading and scan benchmarks show different access costs and retained information behind the shared interface [25]. Memory boundaries also differ: JVM measurements for external storage cover a narrower boundary than the combined application and operating system footprint.

DB-XES adds maintained process discovery intermediates to persistent event storage. Insertion triggers update directly follows and supported MINERful statistics, allowing later process discovery to reuse them [12]. The benefit depends on repetition and maintenance. In the weekly experiment, the time for each report includes new insertion, updates, retrieval and process mining; it becomes lower than reloading the growing XES log and repeating process mining after about two months. This crossover compares individual reports in that workload; cumulative lifecycle break-even requires summing costs over the report sequence. The update rules require chronological append insertion, activity filtering can invalidate directly follows intermediates, and controller state remains necessary for open cases. Application memory savings also exclude the external database server.

Event dataframes reduce object overhead through columnar storage and selective loading. Their experiments measure loading, filtering, directly follows construction, memory and disk space, including synthetic logs with tens of millions of events [8]. The efficient shift-and-count construction assumes case and event order have been prepared; initial sorting is a separate cost. A native relational directly follows operator similarly avoids transferring raw events and repeated join-based construction, but its benchmark starts from populated storage or an already loaded log [26]. Both approaches change the access pattern and the algorithm together.

SQL-based declarative process mining shows why the execution plan matters as much as the database family. Indexes, integer encodings and multicore execution affect query time and storage, and parallel execution can be slower for a particular constraint and log [27]. Experiments with SQL-capable big data frameworks likewise distinguish cached execution from repeated loading and show workload-dependent benefits from additional workers [28]. These studies support choices tied to the query, data preparation and worker allocation.

Incremental view maintenance connects preparation to conformance checking. DBToaster maintains queries distinguishing satisfying, violating and pending states for instance-spanning constraints [29]. This avoids repeatedly evaluating the full query result, but the constraint queries are manually authored and assume an insert-only stream. The reported per-event update timings characterize processing over prepared data; a retained state bound requires accounting for the complete maintained representation. Its relevance to Section 5.3 is the maintenance of a specified compliance answer, rather than database storage alone.

5.1.3. Representations must retain the information an analysis needs

Event data engineering also establishes the information on which the rest of the review's tasks can operate. An efficient representation is suitable for a particular analysis only

when its events, attributes, ordering and relationships support the requested answer. This makes preservation a necessary link between the access mechanisms just discussed and the analytical contract introduced in Section 2. Query languages expose this issue through the patterns and match semantics they support; object-centric representations expose it through the relationships and case views they retain. Examining both together helps distinguish access choices that can serve the same downstream analysis from transformations whose computational benefit comes with a change in the available information.

Process query languages express behavioral patterns while their implementations determine execution costs. SIGNAL defines nested case and event columns, ordered data and specialized kernels as an execution design [30]. Celonis PQL combines compressed columnar storage, process-specific operators and cached queries on a multicore server, reporting production query volume and latency observations [31]. Those observations characterize operational use; attributing a speed advantage to a language feature requires controlled workload and hardware comparisons.

Formal expressiveness and execution reuse answer separate questions. The analysis of SIGNAL's conjunctive core establishes a polynomial data complexity upper bound for fixed queries, while leaving the matching hardness question open [32]. Translation to SQL with `MATCH_RECOGNIZE` demonstrates how selected behavioral patterns can use a standard query construct; broader language equivalence and complete translator performance remain questions for further evaluation [33]. Case partitioning, event order, match multiplicity and supported operators remain part of the contract.

Object-centric data add relationship structure to event volume. OCEL separates objects from events allowing associated events to share object information [6]. Property graphs make event-object and temporal relations explicit and support queries across interacting entities, at the cost of graph construction and additional storage [34]. The multiprocess graph study provides a particularly clear contrast: direct event pair aggregation is fastest for single case notions, but indexed trace construction is faster for the evaluated composite intersection, while composite union requires a different operator [35]. Thus, a change in the requested case view can reverse the preferred execution method even on the same stored data.

Persistent object-centric storage can increase capacity even when it adds runtime. MongoDB OCEL reports insertion, index construction, storage and process mining measurements from one million to one hundred million events [36]. It is slower than the compared in memory approach where both fit, while processing larger inputs after the latter exhausts memory. Its practical benefit is extending the feasible input range.

Query planning addresses the combinations examined within that stored data. OCPQ extends bindings through relationships and ordering predicates to avoid unnecessary Cartesian products [37]. Its fast example query results refer to companion work, and an early-stop safeguard can return explicitly incomplete results. Evaluating this mechanism requires query time and completeness in addition to storage capacity.

Extraction determines which relationships subsequent analyses can use and how much work constructs them. OnProm translates ontology-based access into source SQL and evaluates extraction over generated Dolibarr databases [38]. Its timing separates query and object construction from serialization; mapping design and indexing remain preparation costs. SAP extraction uses relationship graphs and domain blueprints, with query selection and joins depending on business knowledge [39]. Relation construction can dominate postprocessing. Both results make relationship preparation part of the input cost.

Reusing an intermediate format also requires checking what each export retains. Stackt provides a relational hub with evolving attributes and ingestion that appends groups of

records [14]. Its demonstrated pipeline establishes this integration mechanism, while export can lose some event-to-attribute-update and temporal object relationships. A downstream analysis requiring those relationships therefore needs a compatible export contract.

Overlapping loading and analysis can reduce waiting time despite contention. PM-Cube pushes multidimensional selection and aggregation into SQL and overlaps sublog loading with process mining [40]. Its experiments include query processing, process discovery and visualization: asynchronous execution increases isolated loading time but reduces total waiting time. The input transformation also matters, because event aggregation can remove repetitions and dependencies through intervening events.

Object-centric granularity operations preserve a different part of the input. Drill-down, roll-up, unfold and fold refine or coarsen type labels while retaining event-object connectivity [41]. The evaluation separates parsing from transformation and varies category counts; unfold time per call must be multiplied by the calls an exploration requires. Quality comparisons use transformed inputs, and conformance checking calculation fails for part of the educational sample. Object-centric filtering likewise preserves connected units of an already filtered log, using the relationships retained by the preceding filter [42]. Connectivity preservation therefore needs to be matched to the downstream property being analyzed.

5.1.4. Pipeline evaluation must include preparation and integration

The mechanisms and preservation conditions above establish what an event data system could contribute; evaluating that contribution requires following its costs into the analysis pipeline. Loading, index construction, data movement and maintenance may occur outside the operation whose runtime is reported, yet they affect the resources needed to obtain a result. This is the event data engineering contribution to the review's question about measurement boundaries and implementation evidence. Integrated platforms, distributed deployment and benchmark infrastructure provide different kinds of evidence, so their results must be interpreted at the boundary each actually evaluates. The following studies clarify when a reported improvement concerns an individual operation, a complete analysis or infrastructure for conducting further experiments.

Platform integration makes prepared data available to several analysis stages. The ProM and Hadoop connection demonstrates large log processing through HDFS references and specifically adapted process mining jobs [43]; compatibility and worker scaling for other ProM algorithms require separate adaptation and evaluation. A Dask pipeline combines case partitioning, cached graphs, process discovery and model evaluation [44]. Its alternatives produce differing quality values, and model changes can trigger reevaluation over all accumulated traces. Their reuse and partitioning benefits must therefore be assessed with the resulting model quality and repeated evaluation work.

Cross-organizational pipelines package algorithms and move them to data through distributed deployment. Their prototype and requirements survey establish a coordination mechanism, while access control and some interface checks remain incomplete [45]. ContinuumConductor's placement questions provide useful background for choosing where such stages run, but its scenario is methodological guidance rather than an implemented placement optimizer [46].

Benchmark infrastructure must also be interpreted at its own boundary. Distributed Event Factory implements configurable process-related streams and reports prototype generation throughput; this quantifies source generation capacity [47]. DPM-Bench varies computational topologies through simulated compute, storage and communication costs. Its outputs are modeled quantities rather than empirical hardware resource measure-

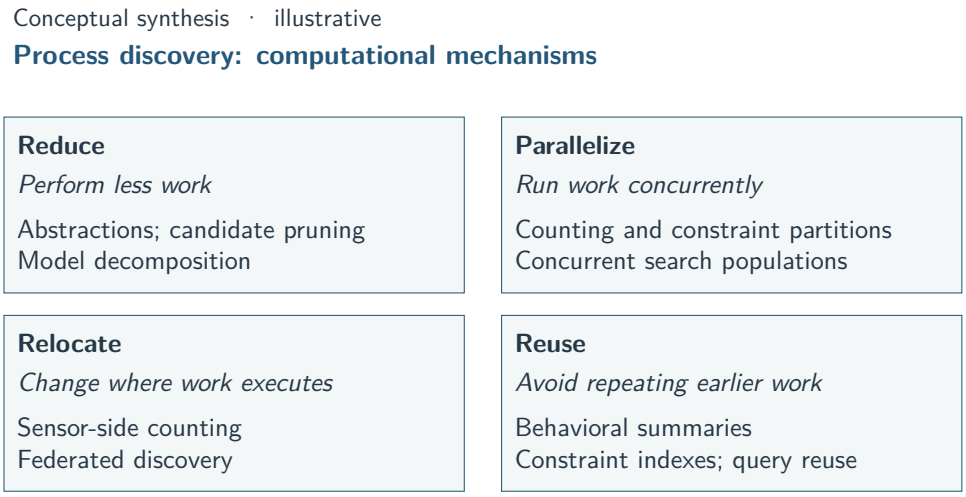

Figure 11. Conceptual synthesis: key terms for process discovery. The four mechanisms summarize ways to change discovery work. A smaller search, additional workers, local computation and retained evidence have different costs and model requirements.

ments [48]. These contributions support workload construction and comparison; deployed pipeline measurements establish the resulting analysis capacity.

5.1.5. Choose data systems by information needs and reuse

Choosing an event data infrastructure commits later analyses to particular access paths, retained information and maintenance requirements. The literature therefore needs to support a combined decision about which source relationships to construct, where to store them and which intermediates to keep for repeated use. Storage capacity alone cannot settle that decision when a required export loses relationships or an update invalidates a maintained statistic. This synthesis closes the preparation theme by connecting those dependencies to the discovery, conformance checking and prediction requirements considered next. It establishes which preparation choices can support a proposed pipeline and which construction, access and maintenance costs must be carried into the comparison of its tasks.

Selection should begin with the required events, attributes and relationships. Candidate filtering helps expensive correlation searches; persistent storage extends capacity; maintained indexes suit repeated queries over data whose update rules are known. For object-centric analysis, the required case view and relationship distribution guide the representation choice. Compare construction, access and maintenance over the intended analysis sequence, checking that each export retains the information the next operation needs.

5.2. Process discovery: scaling model construction

Figure 11 connects the four mechanisms to discovery concepts, from smaller candidate searches to local computation and reusable behavioral evidence.

Process discovery constructs a process model from recorded executions. Procedural models describe possible paths of execution, while declarative models state constraints on acceptable behavior. Different process discovery methods therefore produce different kinds of model even when they receive the same event log.

A process discovery procedure collects behavioral evidence from the log, such as activity order or repeated relations, then uses that evidence to construct and evaluate a model. Some methods test candidate model elements; others split the problem and

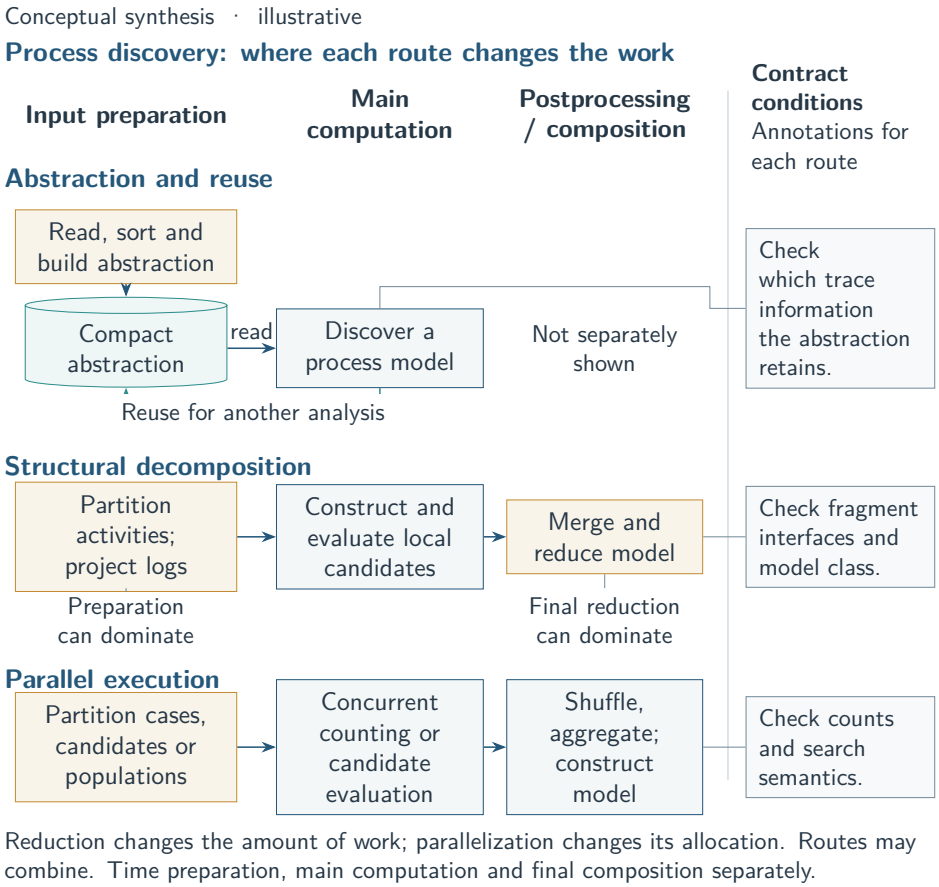

Figure 12. Conceptual synthesis. Process discovery routes are aligned by input preparation, main computation and postprocessing or composition. Abstraction construction prepares retained state for process mining and reuse; the top route focuses on preparation and process mining. Structural decomposition and parallel execution change different sources of work [2,49,50]. Contract conditions annotate each route rather than adding a processing stage. Each route has its own output conditions; combining routes requires evaluating their joint effect.

compose local results. Reading the observations and constructing the model are distinct sources of work, which the following mechanisms address in different ways.

5.2.1. Event volume and activity diversity create different costs

Once event data are available, process discovery introduces the additional problem of constructing a model from the behavior they record. This is the first task perspective in the review, and it makes the distinction between input volume and structural complexity especially consequential. A large log with repeated behavior and a smaller log with many interacting activities can place demands on different stages of the same miner. Identifying those stages gives the mechanism comparison that follows a basis in the work actually performed. It also explains why the review examines event counts, activity sets and composition requirements separately when assessing what a discovery method can scale to.

More events increase the work needed to collect behavioral evidence. A larger activity alphabet can expand the space of candidate relations, places or constraints. Complex behavior can make both candidate evaluation and model composition expensive. The general decomposition framework separates large case populations from large activity sets: reducing the latter can help even on one computer [51]. Adding workers and reducing the

search space therefore address different parts of the problem. Figure 12 separates scanning, candidate work and final composition.

5.2.2. Abstraction, pruning and decomposition reduce model search

The separation of scanning, candidate exploration and composition allows discovery methods to be compared by how they change model construction. Abstraction retains selected evidence for a miner, pruning avoids candidate evaluations, and decomposition creates smaller searches whose outputs must be combined. Parallel statistics and population search then offer ways of allocating work across concurrent computations. Examining these approaches together develops the review's mechanism perspective: methods with similar runtime aims can intervene at different stages and can introduce different preparation or composition costs. The comparison below follows those interventions through the discovery procedure, providing the basis for the subsequent assessment of model semantics and complete execution costs.

A compact abstraction can separate reading the log from constructing the model. Inductive process discovery from directly follows graphs scans the log once and then performs process discovery on the resulting graph [49]. The experiments include 100 million traces and, separately, a process with 10,000 activities. These maxima concern different workloads. Preparing the largest graph takes days while process discovery from it takes seconds. The soundness and conditional rediscovery results concern the process discovery procedure, while the abstraction can lose information needed to reproduce arbitrary observed behavior. Its practical value depends on whether the graph is already available or can serve several analyses.

Pruning reduces candidate enumeration before expensive evaluation. A matrix-based Alpha variant replaces exponential candidate construction with polynomial operations and illustrates the procedure in GPenSIM [52]. Petri net class metaproperties provide a complementary, formal route: they identify conditions under which whole families of expanded places can be pruned during eST process discovery [53]. Its closure conditions differ across structural classes, and membership checks can still require nonconstant work. The work establishes formal pruning conditions, with runtime evaluation planned as implementation develops. These approaches reduce candidate work through different assumptions about model structure.

Decomposition and composition. Structural decomposition makes process discovery operate on smaller activity sets. Passage-based decomposition partitions causal relations into local problems with controlled interfaces [54]. Decomposed ILP process discovery provides a computational comparison: in its case study, process discovery time falls from 2,245 to 97 seconds for one configuration [55]. The discovered model changes with the configuration, and the accompanying decomposed replay cost can be a lower bound on monolithic replay cost. The runtime benefit therefore concerns local model construction with those output conditions.

Choosing a partition trades smaller local searches against interactions between fragments. Activity clustering makes that choice an optimization problem using cohesion, coupling and balance criteria [56,57]. The partition's usefulness depends on both the work left within each cluster and the interfaces needed to compose the results. The two reports draw on the same case study and therefore share their experimental evidence.

Composition can cancel local savings. The Divide And Conquer framework preserves specified properties under its decomposition conditions, while allowing the discovered model to differ [2]. Its experiments show both gains and losses. In one Alpha example, local process discovery takes 87 seconds against about 800 seconds for monolithic process

discovery, but model reduction takes days. Other examples make Heuristics Miner and Inductive Miner slower after decomposition. RPSTHD similarly combines local ILP process discovery with heuristic routing and bottom-up composition [58]. Its storage comparison is revealing: preloading the log reduces one measured runtime from 80.17 to 1.58 seconds. This comparison shows the runtime effect of event access as well as partitioning. Assessing the memory tradeoff would require a corresponding resource measurement.

Domain-specific boundaries can define useful subproblems before optimization. TOM4L divides manufacturing message sequences into phases before local inference and concatenation [59]. Manual alignment of manufacturing routes participates in preparation. The application establishes how those phases support model construction; broader runtime comparisons would test transfer to other workloads.

Numerical decomposition serves a different modeling objective. Partitioned gamma mixture fitting uses component fits to initialize joint optimization of task duration distributions [60]. Its experiments measure time to likelihood targets alongside the frequency of reaching them. Runtime must be assessed with that success frequency. The result is a fitted duration distribution, rather than a partitioned control flow model or a statistical significance test for duration changes.

Several methods reduce work within a single machine. Split Miner filters a directly follows graph and constructs routing elements, with freedom from deadlock and a soundness result for acyclic models [61]. Its complexity depends linearly on event count, while activity count enters separately. Seven of the twelve benchmark logs are externally filtered before all miners run, and parameter selection also affects the comparison. Fast & Sound limits synthesis to affected subnets while preserving its targeted soundness and free-choice properties [62]. Its operation timings omit initial process discovery, so the result concerns subsequent modification work. Hybrid process discovery deliberately mixes formal places with informal causal relations [63]. The Spark implementation adds candidate pruning and parallel evaluation, but its safe pruning rule must be distinguished from an aggressive threshold that can remove results [64].

Compact representations reduce repeated evaluation by retaining the relations a particular miner needs. The eST^2 Miner works with partial order variants and uses inexpensive replay heuristics before exact max flow fallback [65]. Construction of a concurrency oracle precedes its main process mining comparison. Compaction helps on most tested fixed workloads, with benefits varying across logs, and the partial orders can admit behavior absent from the original total order traces. DisCoveR uses bitvector log abstractions to construct and simplify DCR graphs [66]. Its timings support interactive recommendations on the tested logs, with DCR semantics that differ from those of its procedural and declarative competitors. Both results depend on which behavioral information the representation retains.

Reusing previous candidate evaluations can further reduce search work. ACOBPAM grows alignments from already evaluated seed patterns [67]. Changed loop blocks and trees containing choice operators require classical alignment. Classical alignment is at least as fast on Traffic Fines, and both versions exceed 24 hours for depth three Sepsis patterns. Postprocessing and visualization are timed separately from process mining. This reuse helps when neighboring candidates share sufficient structure for their alignments to remain valid.

Parallel statistics and population search. Distributed statistics reduce the time spent collecting behavioral evidence before model construction. The early MapReduce Alpha implementation evaluates a fixed 80 GB workload on ten nodes [68]. The broader MapReduce abstraction study varies data volume, workers, trace length and activity count and reports

upload separately from job execution [50]. Its final aggregation still creates a central stage. Spark process discovery varies log and cluster sizes, but its volume experiment replicates a small set of traces over a fixed activity alphabet [69]. Together these studies cover fixed workload feasibility and sensitivity to selected workload dimensions. Assessing the complete miner also requires measuring subsequent model construction and aggregation.

Genetic process discovery distributes populations rather than only event statistics [70]. The sampling extension also reduces the traces used for intermediate fitness evaluation, enlarging samples when necessary and retaining full log validation steps [71]. Population size, migration and sample size alter both computational work and stochastic convergence. Their measured gains therefore combine concurrency with a changed search procedure.

5.2.3. Comparing discovery requires matching model semantics

The event data discussion examined whether a transformation supplies the information a later analysis needs. For discovery, the corresponding question concerns the model produced from that information: which behavior can it express, and under which assumptions does the method construct it? This question is central to comparing computational mechanisms because changing a constraint language, a counting convention or the retained history can change both the work and the answer. Declarative discovery and maintained constraint indexes make these dependencies explicit. Their treatment here develops the analytical contract for discovery and identifies the semantic conditions that must accompany a runtime comparison, including comparisons between an initial query and later queries over maintained state.

Declarative process discovery makes the requested constraint language part of the workload. Parallel Declare Miner partitions either candidate constraints or the event log, changing the allocation of testing work and intermediate results [72]. MP-Declare adds activation or target attributes to supported templates. Its Hadoop implementation creates large intermediate event pair collections, and the evaluated cluster runs take hours where local execution takes minutes [73]. The paper attributes this disadvantage to transfer and management costs. The full support extension measures shared memory Java execution on a quad-core computer [74]. It supports customizable templates, but correlation and temporal conditions remain future work. Its nonvacuous fulfillment convention and handling of equal activity pairs also differ from some competitors, making output semantics essential to interpreting runtime.

Persistent statistics can make repeated queries inexpensive after costly preparation. SIESTA separates event indexing, incremental constraint statistic maintenance and querying [75]. Its evaluated logs reach 96 million events through extension of public workloads. For one query, index preparation makes the SIESTA variants costly; cumulative comparisons over ten support thresholds show the benefit of reuse. Incremental maintenance still gets slower as retained event tables grow, and indexing slows when all traces continue across successive updates. The system assumes strict timestamp ordering and removes completed cases from its last checked state only when no further events will arrive. These exact updates rely on the stated ordering and completion assumptions; a global state bound would also need to cover retained event tables. Its optional temporal analysis also restricts the answer through a selected fraction of divergent patterns.

CBDeclare builds on this infrastructure by retaining the trace identifiers satisfying each constraint [76]. Set operations support OR, AND and XOR source or target branching, with greedy selection and explicit stopping conditions. It expands the supported semantics while restricting enumeration through the selection procedure. Its first query versus repeated query comparison includes index preparation, whereas later policy comparisons exclude

it. Its comparisons characterize three fixed logs and their query policies. These findings connect process discovery to event data engineering in Section 5.1 and to update semantics in Section 5.6.

5.2.4. Distributed discovery must account for communication

The preceding mechanisms show why the boundary of a discovery experiment affects the conclusion that can be drawn from it. Preparing an abstraction, searching for model elements and composing a result contribute to the requested analysis even when their costs are reported separately. Moving stages toward event sources adds communication and coordination to that accounting, while sampling changes which evidence reaches the miner. The studies below extend the review's comparison of reported costs to these distributed arrangements. They help establish whether an evaluation supports a claim about local computation, a complete discovery run or continuing operation, and which preservation assumptions must remain fixed when comparing those claims.

Moving process discovery toward data sources changes communication as well as model construction. EdgeMiner keeps direct succession evidence on sensor nodes and groups predecessor queries [77]. It reduces communication relative to all-node querying; central upload can use fewer messages. Exactness depends on ordering and retention assumptions. Federated Alpha+ supplies a complementary guarantee: equivalence with its centralized counterpart under the specified federation construction [78]. These results address data locality and answer preservation under different communication arrangements.

Sampling before distributed process discovery changes the evidence supplied to the miner. EdgeIM preserves selected structural features through sampling, local extraction and central aggregation [79]. Its fixed log server evaluation includes global sorting and sampling; sustained edge operation requires a further service experiment. The preserved start, end and directly follows sets omit frequencies and some trace information. Thus its computational comparison concerns process discovery from selected features, with a different preservation requirement from full log equivalence.

5.2.5. Match discovery to model requirements and bottlenecks

Selecting a discovery strategy requires more than comparing the largest logs or shortest runtimes reported by different studies. Those observations refer to particular model languages, behavioral information and stages of construction, which the preceding subsections have separated. Bringing these dimensions together allows the review to relate an analyst's model requirement to mechanisms that address the costly stage without losing the properties needed for use. It also identifies which preparation and composition costs must accompany the comparison. This closes the discovery perspective and prepares the transition to conformance checking, where the structure of a supplied reference model becomes a determinant of checking cost.

Repeated analyses of stable observations favor reusable abstractions and constraint indexes. Large activity sets motivate pruning or decomposition, provided the resulting model and composition costs meet the intended use. Distributed statistics suit expensive scans with manageable intermediate results. Selection should begin with the required model class and behavioral information, then compare preparation, search and composition as event volume and structural complexity vary separately.

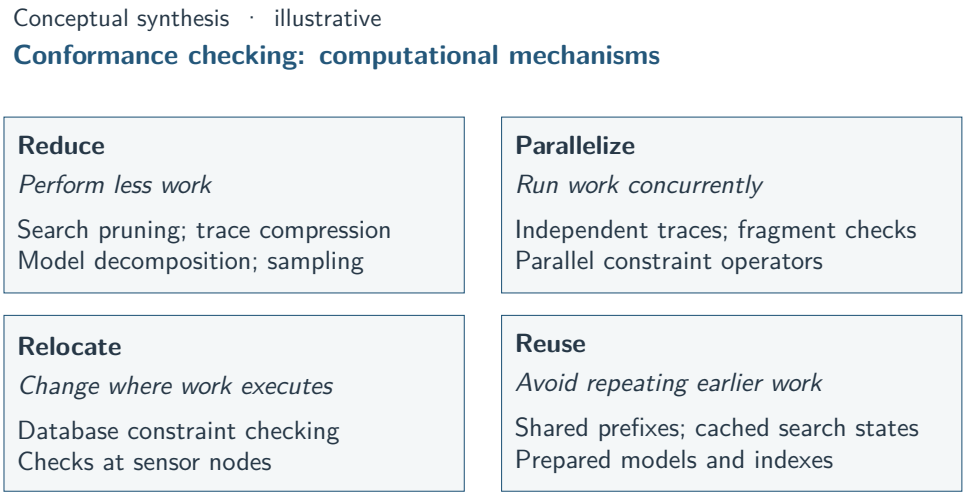

Figure 13. Conceptual synthesis: key terms for conformance checking. The four mechanisms act on checking effort, concurrent checks, execution location and repeated search. Their outputs range from detailed alignments to constraint judgments or estimates.

5.3. Conformance checking: scaling the required assessment

Figure 13 summarizes how checking methods reduce searches, run independent checks concurrently, move checks to databases or sensors, and reuse earlier work.

Conformance checking receives observed behavior and a reference model and assesses their agreement. Its output may explain individual deviations, judge whether constraints hold or provide an overall measure of agreement. The computation relates the observed events to behavior allowed by the reference and derives the requested assessment.

Alignment-based checking makes that relationship explicit. Matched moves advance the observed trace and the model together; unmatched moves advance only one side. An alignment explains correspondence through these moves, and an optimal alignment minimizes their specified costs. Constraint judgments and aggregate measures can require different computations and provide different detail. Accelerating conformance checking therefore first requires specifying which assessment must be produced.

5.3.1. Trace length and model structure drive alignment search

Conformance checking changes the computational question from constructing a model to relating observed behavior to a supplied reference. The reference introduces a source of complexity that cannot be characterized by log volume alone: a small number of traces may still require difficult searches through model behavior. Within the review's account of workload dimensions, this distinction explains why increasing the number of checks and increasing the difficulty of an individual check require separate attention. Establishing where the work grows also prepares the comparison of reuse, parallel checking and search reduction below. Each becomes relevant under different combinations of trace repetition, trace length, model structure and requested diagnostic output.

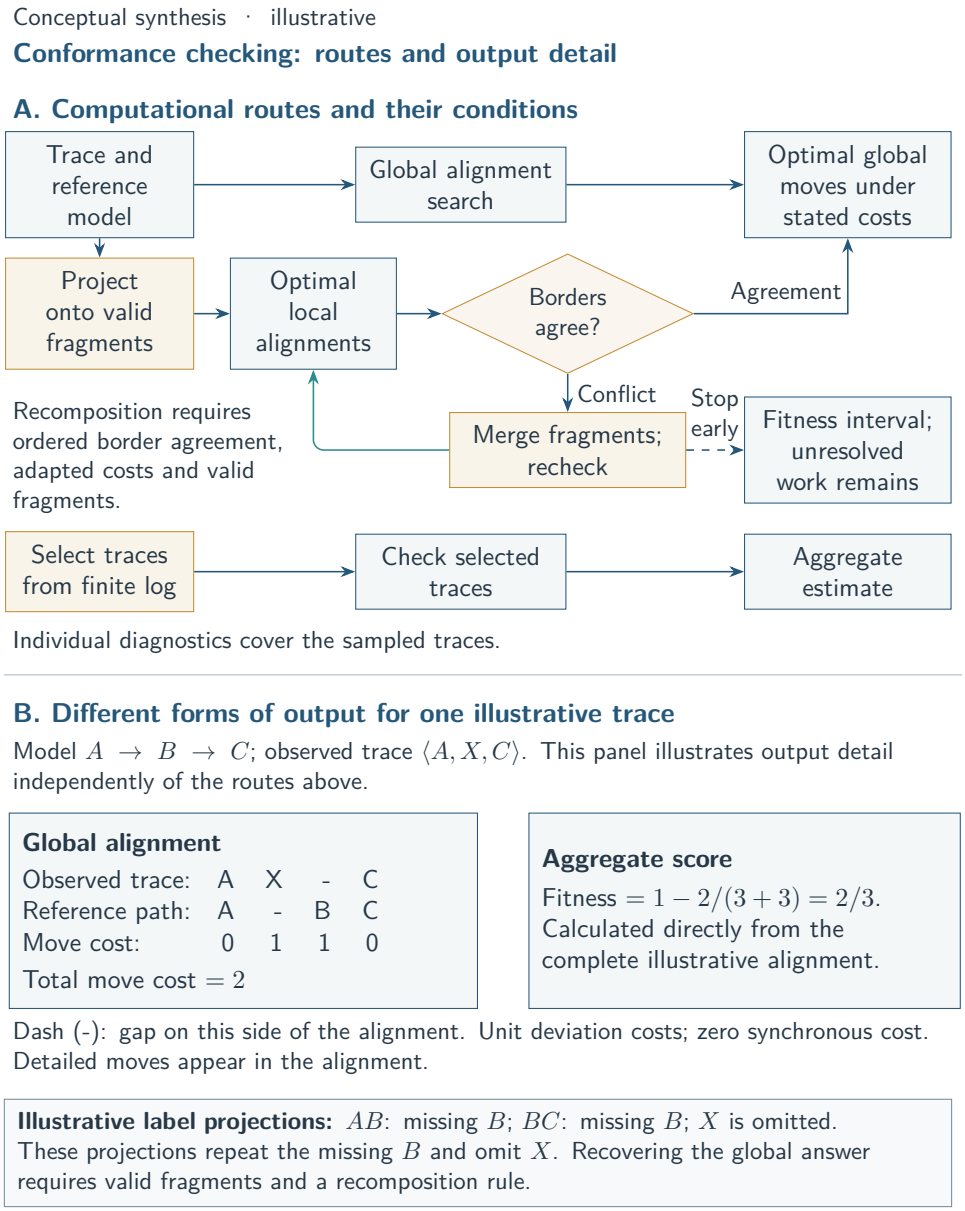

Figure 14. Conceptual synthesis. A: computational routes and their conditions. Recomposition can recover an optimal global alignment given valid fragments, optimal local alignments, adapted costs and ordered border agreement; conflicts require merging and rechecking, and early stopping can leave a fitness interval [80]. Sampling has an aggregate contract [81]. B: independent illustrations of output detail for an invented trace and model, with unit log/model move costs and zero synchronous cost. The label projections illustrate local indications; valid decomposition and recomposition conditions are required to derive a global result. The normalized score is calculated directly from the illustrative alignment.

Long traces enlarge alignment search, concurrency enlarges the reachable model state space, and data conditions introduce additional dependencies. Repeated traces and fragments can also cause the same work to be performed many times. A useful comparison therefore specifies both the bottleneck and the answer that an acceleration method preserves. Figure 14 contrasts the routes and illustrates their different outputs.

5.3.2. Smaller alignment searches require careful reconstruction

When an individual alignment search is the limiting computation, changing its representation or dividing it into smaller problems offers a route to feasibility. These mechanisms connect conformance checking to the decomposition strategies examined for discovery, but the result that must be recovered is different: local checks must support the required judgment about a trace and its reference model. Compression and decomposition therefore provide a useful setting for examining reduction together with answer preservation. The discussion follows the work from problem preparation through local checking to reconstruction, making explicit when smaller searches yield a global result and when they leave bounds, conflicts or additional checking work.

Indication based reduction compresses model and trace fragments, computes a macro alignment, and expands it through local sequence alignment [82]. Local expansion is optimal, while the resulting global alignment may be suboptimal. Its runtime and sampled resident memory measurements therefore characterize a checker with that reconstruction contract.

Decomposition and interface conflicts. Decomposition replaces one large check with smaller checks, whose interfaces constrain how far the model can be divided. Generic Petri net decomposition, SESE decomposition and partition topology organize these subproblems through model structure [83–85]. SESE bridging preserves place context and shares only unique visible transitions. Silent transitions and repeated labels constrain the attainable fragment size [84].

The useful global result depends on what the decomposition preserves. Generic decomposition preserves perfect fitting and supplies alignment cost lower bounds [83]; passage decomposition preserves perfect fitting under its labeling and boundary conditions [7]. Partition topology targets binary fitting, and discarding small components can lose its partition guarantee [85]. Data aware decomposition requires independent, disjoint variable sets and suitable guard fragments for perfect-fitting equivalence [86]. These guarantees support fitting decisions and local diagnosis. Reconstructing globally optimal data alignments requires additional guarantees for the variable and fragment interactions.

A fragment can be harder to replay than its size suggests. Removing its surroundings may introduce source transitions and permit many unhelpful executions. Hiding surrounding labels and then applying reductions retains more control context and produces tighter alignment cost lower bounds [87]. This improvement moves cost into preparation: reported reduction times of 25 to 167 seconds occur where subsequent replay takes at most five seconds. Its usefulness therefore depends on how often the prepared representation can be reused.

Recomposition makes the connection between local and global answers explicit. Local alignments that agree on their ordered shared borders can be combined into an optimal global alignment under appropriately divided move costs. Where agreement fails, components are merged and realigned [80]. Completion yields the exact result; interruption can leave a guaranteed interval for fitness. The experiments run on a single node and vary noise and time budgets, so this is evidence for reducing search through decomposition, rather than for adding workers. A companion tool exposes the configuration and replay decisions [88]. Improved merging uses conflicts and component sizes to reduce repeated recomposition while retaining the inherited agreement conditions [89].

Table 5. Analytical contract matrix for selected conformance checking methods. Exactness is interpreted relative to each method's output and assumptions. Fields requiring further evidence are marked unspecified.

Method	Required output	Supported inputs	Guarantee	Diagnostic detail	History
Data decomposition	Local data/control checks	Data nets: independent variables, suitable guards	Perfect-fitting equivalence; global optimal reconstruction unproved	Fragment deviations	Completed logs
Recomposition	Global alignment or fitness interval	Valid fragments, optimal local solutions, adapted costs	Optimal with ordered border agreement; containing interval if stopped	Full moves for resolved traces	Completed logs
Indication reduction	Expanded macro alignment	Supported flow indication reductions	Local expansion optimal; global alignment may be suboptimal	Expanded local moves	Finite traces
Tree dynamic programming	Optimal alignment	Process trees; unique visible labels for polynomial bound	Minimum cost under stated costs and structure	Trace moves	Completed traces
Markovian tree evaluation	Substring fitness / precision	Process trees; fixed substring length	Exact substring metric	Aggregate metric	Finite log, bounded contexts
Statistical sampling	Aggregate fitness / deviation estimate	Random traces; stated stopping assumptions	Probability bound on additional information	Checked variants, aggregate hotspots	Finite log

Sources in row order: de Leoni et al. [86]; Lee et al. [80]; Taymouri and Carmona [82]; Schwanen et al. [90]; Goulart Rocha and van der Aalst [91]; Bauer et al. [81].

Table 5 compares the required outputs, guarantees and diagnostic detail of selected conformance checking methods.

5.3.3. Exactness depends on the output and method assumptions

The decomposition results make clear why a common task label is insufficient for comparing conformance checking methods. An analyst seeking an explanation with minimum alignment cost for a particular trace requires a different answer from one estimating agreement across a log. This is where the review's analytical contract becomes a concrete comparison tool: Table 5 separates the output, supported inputs, guarantee and diagnostic detail. The following methods extend that comparison to shared representations, restricted model classes and approximate search. Examining their conditions explains which reductions preserve the requested assessment and which require the user to accept a different level of optimality or diagnostic coverage.

Shared representations avoid repeated alignment work, with guarantees that depend on how reuse is controlled. Shared automata process common log prefixes and suffixes

through a synchronized product supporting optimal alignments for one-safe sound models [92]. A complementary S-component method requires unique visible labels and free-choice structure and produces valid, potentially suboptimal alignments. Retained silent-transition identities expose conflicts that trigger full model fallback. Split point caching instead reuses search states: reopening a closed state when a cheaper route is found retains optimality; the faster naive cache can return suboptimal results. Both cache variants can increase measured memory [93].

Model concurrency creates another source of repeated search: alternative interleavings of independent events. Petri net unfoldings represent partial orders under one-safe, easy-sound model assumptions [94]. Their strongest results concern substantial parallelism and difficult deviations; classical alignment performs better for some simpler, less parallel cases. Thus repeated prefixes, repeated search states and concurrent interleavings motivate different representations and different workload tests.

Model restrictions can make particular conformance checking computations polynomial. Dynamic programming computes optimal alignments for process trees with unique visible labels [90]; repeated labels fall outside that bound. A separate tree algorithm computes Markovian substring fitness and precision compositionally, avoiding construction of the full model automaton [91]. Its polynomial result fixes the substring length. The first result preserves a minimum-cost move sequence, whereas the second computes a specified substring measure. The required diagnostic output determines which bound is relevant.

Long traces create another reduction opportunity. ConLES searches sliding subtraces while using model structure to discourage locally attractive continuations that cannot lead to a suitable global answer [95]. Its global heuristic requires the remaining trace, and its bounded candidate search is approximate. The windows therefore define a search mechanism within a complete finite trace. Tandem repeat compression reduces repeated portions of a trace and reconstructs valid approximate alignments with costs at least as large as the optimum [96]. The trace is reconstructed losslessly, while alignment reconstruction retains its approximation contract. Its core uses restricted concurrency-free models, while S-components extend it to concurrency with further approximation. Benefits depend on useful repetition and can disappear for small inputs.

5.3.4. Preparation and coordination can offset faster checking

Once the required assessment and its guarantees are fixed, the remaining comparison concerns the work needed to obtain it. Conformance checking makes this measurement problem particularly visible because preparing a model, selecting traces or constructing a surrogate can precede a fast checking operation, while reconstruction may follow it. Distributed implementations add import and coordination costs and may use resources that differ from those of a local baseline. Reading these evaluations together connects the task's algorithmic literature to the review's broader question about what reported resource outcomes establish. The evidence below is interpreted in terms of the included stages, the assessed workload and the diagnostic or approximation conditions of the resulting answer.

Distribution is useful when independent trace checks outweigh import and coordination costs. The Hadoop implementation maps traces to alignments and combines aggregate fitness contributions [97]. Its fixed-cluster experiments vary log volume and fitness; overhead makes small logs slower, and machines differ from the local baseline. CC4Spark integrates distributed import and preparation with configurable checking and decomposition, also reporting cases where standalone execution is faster [98]. These comparisons establish workload-dependent gains while leaving an isolated worker scaling curve unresolved.

Distributing model fragments adds a composition requirement to parallel checking. Horizontal parallel alignment combines decomposition and Spark execution for sound, safe, block-structured workflow nets [99]. Exact cost requires preserving all full firing sequences and taking the minimum across all submodels; its distributed pseudocode collects costs. Experiments vary log size and noise on four fixed workers, and overlapping submodels can make decomposition slower. The cost of repeated checks across overlaps belongs alongside the benefit of concurrent execution.

Other engines accelerate different semantics. DECLAREALIGNER uses an optimistic repair heuristic to guide alignment search for declarative models [100]. KnoBAB evaluates data aware temporal conditions through indexed tables, shared query work and parallel operators [101]. Its counting shortcuts preserve satisfaction while potentially discarding event witnesses needed for detailed diagnosis. Operator timings exclude operand construction and database access. Porifera evaluates object centric behavioral constraints directly in relational databases [102]. Historical constraints can require evidence beyond a final snapshot, and statement triggers may report temporary violations that disappear after transaction completion. These database checkers return constraint judgments, which differ from DECLAREALIGNER's alignment output.

The placement of a check can also serve communication or trust requirements. DisCC computes footprint based checks at sensor nodes, corrects event ordering and shares aggregates. Its evaluation focuses on fitness; assessing deployment efficiency would additionally require computational and communication measurements [103]. BlockCheck compiles multi-perspective BPMN rules for per-instance smart contracts in a proposed consortium architecture [104]. Its local Ganache experiments compare detected violations. Runtime, throughput and multi-node experiments would extend that evaluation to deployment costs. Their deployment objectives should remain visible when they are compared with a centralized alignment engine.

Sampling and approximation costs. Sampling reduces the number of checked traces, but selection rules preserve different information. Statistical sampling bounds the probability that another trace adds information beyond a chosen tolerance; the bound quantifies information novelty, while absolute full log fitness error requires a separate assessment [81]. Dispersion based selection uses a sample size formula and similarity filtering, with the accepted sample's fitness error assessed empirically [105]. Both supply aggregate assessments with less diagnostic coverage than checking every trace.

Sample preparation can consume the checking time saved. For Hospital Billing, adding the separately reported preparation stages to sampled checking gives about 1192 seconds, compared with 1095 seconds for checking the complete log [105]. This review-calculated sum includes transformation, dispersion, sample construction and checking; the study does not report it as a measured end-to-end total. Section 6.3 discusses why preparation matters. In the extended sampling study, quality checks sometimes increase runtime, and larger samples can accumulate more approximation error [106]. Selection, checking and quality control therefore need a joint cost and error comparison.

Prepared surrogates reduce a different part of the computation: evaluating a trace against model behavior. A proxy trie substitutes finite model behavior coverage for exhaustive search; its errors also depend on budgeted heuristic search, and its preparation measurements exclude model and proxy generation [107]. Learned approximation predicts fitness from labeled alignment examples [108]. That evaluation establishes prediction quality; a timing comparison would quantify the runtime effect. In both cases, the cost and coverage of the prepared surrogate determine the meaning of a fast subsequent estimate.

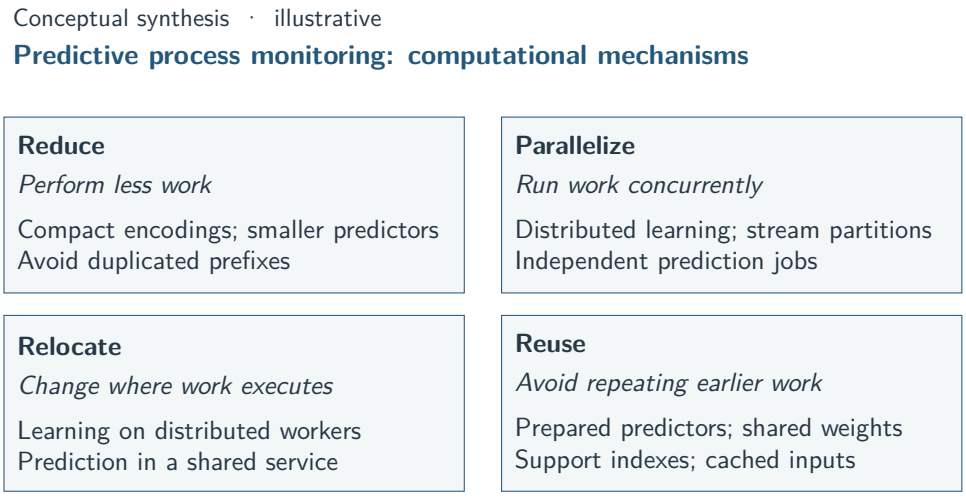

Figure 15. Conceptual synthesis: key terms for predictive process monitoring. The four mechanisms apply to preparing predictors, serving requests and maintaining them. Shared or distributed execution still requires accounting for communication, retained inputs and updates.

Changing the measure used for conformance checking can reduce the behavioral detail evaluated. Improved token replay limits token flooding and caches fragments, while its replay can differ from a valid complete model path [109]. Markovian abstractions compare finite behavioral contexts for fitness and precision; short contexts approximate behavior, while larger contexts increase preparation cost and can exhaust memory [110]. These computations offer different summaries of agreement from an alignment's complete move sequence.

Changing the cost objective is separate from approximating its solution. Exact discounted alignment retains both marking and length under safe, easy-sound model assumptions; dropping length adds an approximation [111]. Discounted anti-alignments add a visit cap that can also remove optimality, while the selected execution provides an upper bound for classical anti-alignment precision [112]. Comparisons must state both the objective and any search restriction.

5.3.5. Choose checking methods by guarantees and diagnostic needs

Choosing a conformance checking method determines what evidence will be available to explain or assess process behavior. The distinction between a globally optimal alignment, a local violation, a containing interval and an estimated score therefore has practical consequences when computation must fit a time or memory budget. The preceding literature provides the guarantees and cost boundaries needed to make that choice explicit. Synthesizing them here turns the review's analytical contract into a selection rule: first establish the evidence the decision requires, then identify mechanisms that can supply it under the workload and model conditions. This reasoning also supports the later comparison of approximation and reuse across tasks.

Exact reconstruction favors methods whose model restrictions and border conditions can be met. Aggregate monitoring can use sampling or surrogates when their error and diagnostic coverage are acceptable. Repeated traces favor reuse, while sufficiently expensive independent checks can justify distribution. Each comparison should include preparation, reconstruction and any unresolved checks needed for the required decision.

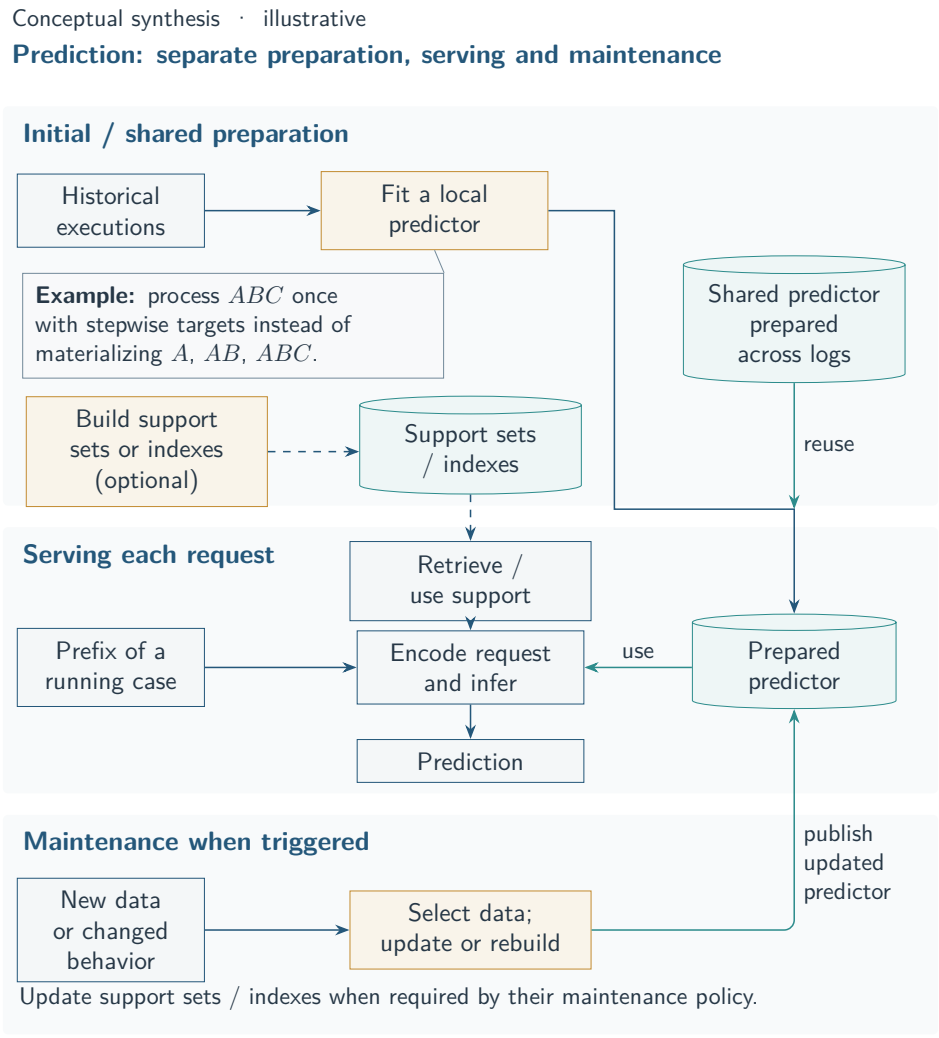

Figure 16. Conceptual synthesis. Preparation, per-request serving and triggered maintenance are separate lanes. A prepared predictor may come from local fitting or reuse across logs; serving can reuse it unchanged. The optional dashed support branch distinguishes building and retaining support sets or indexes from retrieving or using them during a request. The sequence example, shared prediction and drift-triggered updates show different opportunities [113–115]. Compare the same targets and available information, accounting separately for preparation, requests and maintenance, including support costs when weights are frozen.

5.4. Predictive process monitoring: managing lifecycle costs

Figure 15 groups the main opportunities across learning and prediction. Moving work to a shared service can combine with concurrent jobs and reuse of prepared inputs or models.

Predictive process monitoring (PPM) uses information available from an ongoing execution to estimate its future behavior or outcome. The observed part of the execution is its *prefix*. Targets considered here include the next activity, a future activity sequence, remaining execution time and the eventual case outcome. PPM estimates what happens after the observation point, whereas process discovery constructs a model of recorded behavior and conformance checking assesses agreement with a reference.

Historical executions are used to learn or prepare a predictor; some methods reuse a predictor prepared across other logs. Serving a prediction, also called *inference*, applies the predictor to the information available from a running case. New events may prompt further requests for that case. Maintaining the predictor is a separate choice: a method may update its parameters or supporting data as observations or behavior change, while predictions for running cases can also use a fixed predictor. Preparation, serving and maintenance therefore form distinct parts of the computational lifecycle.

5.4.1. Preparation, requests and updates create distinct costs

Predictive monitoring extends the review from analyzing recorded behavior to supplying answers repeatedly while executions are still in progress. Its computational demands therefore depend on how historical preparation, individual requests and later updates are distributed over the period of use. A training improvement may matter most when predictors are rebuilt frequently, whereas request latency becomes central when many running cases need timely estimates. Separating these demands establishes the workload perspective for this theme and prevents a result about one lifecycle stage from standing in for the whole service. It also connects prediction to the preceding discussion of reusable preparation and to the later analysis of continuous operation.

Historical preparation may repeat work across overlapping prefixes or candidate models; serving costs depend on the representation and computation required for each request; maintenance adds update work and retained state. The following methods address different parts of this lifecycle. The relevant analytical contract includes the prediction target, available prefix and attributes, supported labels, update delay and retained history. Accuracy and computational cost must be interpreted together at that boundary. Figure 16 separates initial learning, repeated requests and maintenance.

5.4.2. Compact inputs and reused predictors reduce repeated work

The lifecycle distinction above opens several routes to reducing prediction costs: changing how examples are prepared, simplifying the computation of a predictor, reusing learned structures and allocating work across workers. Their inclusion in one comparison is important to the review because the source of a computational saving can lie outside the learning architecture itself. Repeated prefixes, repeated fits and repeated requests each offer a different opportunity to avoid work. The following studies are organized around those opportunities, from distributed preparation to compact representations and serving decisions. This makes it possible to relate each reported benefit to the stage it changes before examining whether the target and available information remain comparable.

Distributed training divides learning work among workers, adding communication and aggregation to obtain a final predictor. The Spark nested-predictor approach partitions the extraction of frequent sequences, relaxes local support thresholds with a statistical bound and trains classifiers associated with a sequence tree [116]. The bound concerns supported-pattern recovery, rather than prediction accuracy. Its one-node and four-node comparison measures training, with tree depth and classifier count remaining sources of cost. Spark failure detection studies provide complementary evidence of distributed hidden Markov model implementations and predictive evaluation, with worker scaling curves needed to quantify allocation sensitivity [117,118]. Together, these studies separate an implemented execution arrangement from measured sensitivity to worker allocation.

Partitioned stream processing assigns incoming learning and prediction work to workers, creating opportunities for concurrent updates and requests. Flink-based online prediction implements this arrangement [119,120]. The former measures processing of a fixed Kafka backlog while varying workers. This demonstrates parallel execution on a fixed

backlog; sustained arrivals and independently varied active case counts would test service capacity. The latter updates models and predictions incrementally, with an evaluation focused on prediction error. Resource scaling requires additional measurement. In both cases, the explicit bound concerns a transition's delay samples. Bounding total state also requires limits for stored cases, paths and model components.

Compact inputs and predictors. Avoiding duplicated examples can matter before choosing a learning architecture. DA-LSTM+ encodes complete traces for sequence training, removing repeated prefix materialization and avoiding redundant case attribute handling [113]. Its study times the search across eighteen configurations, so the evidence concerns more than one selected model fit. Loading training data from HDF5 in groups supports memory-conscious training; byte level measurements would be needed to quantify its memory savings and capacity. Its bounded output activation also restricts predicted durations to the training maximum. Hybrid trace decomposition instead divides data into sublogs for end state prediction; the evaluated prefix threshold changes runtime, and inconsistent timing units in the text and table prevent a reliable numerical speedup claim [121].

Architecture simplification reduces the size or number of model operations. Comparisons of smaller LSTM and Transformer configurations report parameter counts, training epochs and predictive quality [122]. They characterize model size and learning behavior; runtime and RAM measurements would quantify the corresponding resource savings. The quasi-recurrent predictor measures lower training time while reusing weights across prefix length models, under the evaluated device allocation and historical workloads [123]. Comparing these mechanisms requires the actual training workload, including prefix specific fits and filtered long traces.

Compact incremental encodings update a bounded representation of the observed prefix instead of reconstructing the complete sequence for each request. Hyperdimensional prediction incrementally combines an encoding of the latest activity trigram with a fixed-size vector, then compares it with outcome prototypes [124]. This reduces representation and update work, while the randomized additive encoding can lose details of complete event order. Its timing combines query encoding with inference, and prediction earliness measures the number of events observed before a decision. RAM use requires a separate measurement.

Counts of local activity contexts can replace iterative representation learning and provide reusable predictor inputs. Activity context embeddings derive expected local counts for uncertain events and reuse the resulting vectors in a predictor [125]. The timing covers embedding construction and similarity operations; downstream neural training adds a separate cost. Local realization enumeration relies on independence assumptions and the evaluation truncates uncertain events to a few possible realizations. Wider contexts can still produce very large matrices. Representation quality can differ between intrinsic activity similarity and next activity prediction.

Pattern-based predictors reuse structures learned from observed subsequences, replacing some repeated model evaluation with indexed search. BEST replaces neural suffix prediction with reusable trees of observed bilateral subtrace patterns [126]. Matching stops along incompatible branches, and a global probability cutoff prunes rare patterns before completing the tree. The experiments measure training time, suffix inference time, pattern count and inference throughput. It also reports memory failures for large logs with insufficient pruning. Stronger pruning can remove long predictive chunks and increase the number of inference iterations, so tree size and inference work must be assessed together. Each chosen local pattern has appeared in training, while stitching patterns can produce a new complete suffix. A trained tree can serve both next activity and suffix queries through

different search depth limits, providing a concrete reuse opportunity with task-dependent predictive behavior.

Inference work depends on both model selection and the outputs requested. Promotion selects between two active lightweight predictors after all candidates have been trained; later online learning remains future work [127]. Its latency measurements support the selection mechanism, while quantifying the claimed memory benefit requires a separate resource measurement. ParallAct predicts several next activities and avoids online alignment preprocessing, but its richer output takes longer than the single-target comparator [128]. Thus selecting a cheaper predictor and requesting more predictions change different parts of serving work.

Predicting time distributions introduces further input and output conditions. Pro3Log uses fixed-size case encodings and clustered experts as an alternative to a global probabilistic model [129]. Lifecycle timestamps permit separate processing and waiting times; completion-only records yield combined inter-completion durations that concurrency can confound. The evaluation focuses on predictive distributions and quality. Unseen categories are routed to a random expert, and several large log quality comparisons use unequal sampling budgets. Its comparison therefore requires matching timestamp information, predictive distributions and training budgets.

5.4.3. Shared predictors shift costs into preparation and support

Prediction adds a distinctive condition to the review's analytical contract: the answer must be produced from information legitimately available at the observation point. The target, supported labels and access to completed examples therefore determine both what can be predicted and what preparation or retrieval work is required. This becomes especially consequential when a shared predictor is reused across logs or adapted to a new setting, because some learning has already occurred elsewhere and supporting examples may still be needed. Examining those conditions positions shared models and services within the same computational comparison as locally prepared predictors. It establishes what must be held constant before claims about reduced fitting or adaptation work can be interpreted.

Shared-model prediction reuses a predictor across later applications, moving some fitting work into shared preparation. This can avoid a separate target-specific model fit when the predictor and its adaptation procedure support that setting. The in-context foundation predictor trains a common encoder across logs, then uses a support set to predict within a new log with the encoder's parameters held fixed [114]. Its predicted labels are restricted to those represented in the supplied support. Support construction also uses completion and temporal restrictions to keep the demonstration compatible with the intended prediction setting. The retrieval-augmented extension studies support selection and compares the fuller prediction head with a frozen nearest-neighbor alternative [130]. The simpler alternative can match or exceed the head in the reported settings.

These shared-predictor evaluations establish prediction quality under their support selection rules. Quantifying total resource savings requires a lifecycle comparison. A lifecycle comparison would include shared pretraining, initial embeddings, support retrieval, repeated support encoding, storage and support updates. Frozen parameters eliminate parameter fitting during adaptation while leaving those operations. The number of later applications and the frequency of changing support determine whether shared preparation is recovered.

Parameter-efficient adaptation fits only part of a prepared model or a smaller attached component, reducing the work that must be repeated for a new setting. Direct event token encoding with a pretrained language model reduces selected training and validation

times compared with narrative input, while remaining slower than a compact recurrent comparator [131]. The measured fits cover adaptation; the pretrained model's original training and later deployment add lifecycle costs. ATLAS keeps a pretrained language model frozen and updates an LSTM on arriving events, resetting it after drift detection [132]. Its total replay timing is generally higher than that of several simpler stream learners. Completing replay faster than the historical duration of a log is useful feasibility evidence, while bursts and concurrent cases require a service capacity experiment.

Shared prediction services can retain prepared inputs and models for several analysts and coordinate independent jobs. Nirdizati implements this form of reuse. Its backend retains prepared data, models and status information, passes database keys between workers and coordinates independent jobs [133,134]. The later paper measures throughput for API workflows involving uploads, job requests and result browsing on one fixed host. These measurements count API requests, with model training completion and sustained workflow rates requiring separate accounting. The cache and queue architecture creates opportunities for sharing preparation across analysts, with cache savings and concurrent training capacity requiring targeted experiments.

5.4.4. Keeping predictions current adds update and retention costs

The preceding comparisons concern how a predictor is prepared and used; a service must also decide how to respond when its observations and predictive relationships change. This gives maintenance a central place in the review's assessment of evidence, since update frequency, the history retained for learning and the delay before an updated model becomes available all contribute to continued operation. A resource measurement for one fit or request leaves these recurring costs unresolved. The studies below therefore connect drift handling, new categories and retention policies to the computation they trigger. Their evaluation also provides the bridge to Section 5.6, where sustained service and changing retained state are examined across process mining tasks.

Drift handling detects changes in predictive behavior and selects when and how to update a predictor. Reusing selected history or parameters may reduce update work; the trigger and the cost must both be assessed. DARWIN maintains shared prefixes for incomplete cases, detects changes in delayed prediction errors through ADWIN and fine-tunes Word2Vec and LSTM parameters on the selected window [115]. Completed case entries are removed and the previous predictor remains available while adaptation finishes. The runtime comparison between fine-tuning and rebuilding varies across its experiments. The reported small memory figures cover the prefix synopsis, outstanding prediction table and ADWIN window, excluding the complete neural model, training process and device allocations. Ordered append events and explicit completion are required; late arrivals and corrections remain future work.

Handling a new category can involve a reserved encoding, an update or a reconstruction of the predictor. These operations differ in whether they merely accept the input or learn its predictive meaning. The unseen-variability study makes this distinction explicit. Unknown attribute values map to a void code, whereas prediction for an unseen current activity becomes available after an update [135]. Its evaluated model is rebuilt on an expanding history; proposed reserve capacity for preserving learned structure is deferred. Cumulative training times show that novel-sequence triggers can approach daily update accuracy with fewer rebuilds. A constrained transition system predictor instead introduces possible unseen successors from compliance knowledge and reuses the structure for ordinary probability updates [136]. New labels still require reconstruction, and supported

constraints restrict the added behavior. Its faster alternatives also have different accuracy, so predictive quality remains part of the comparison between mechanisms.

Update scheduling changes how often training is repeated. DynaTrainCDD selects when to retrain and how much recent data to use [137]. Its update runtime excludes initial training and can exceed a simple fixed window strategy while improving accuracy. Variant coverage and label distribution triggers report update counts and average per-update times, with the more accurate label trigger sometimes updating more often [138]. These measurements characterize update frequency and fitting cost; total monitoring cost additionally includes detection and initial preparation.

Retention determines how much data or how many models remain available for each update. An adaptive memory mechanism samples a fixed number of historical prefixes per earlier task and caches relation information, retaining one buffer per task [139]. Total memory can therefore grow with task count. GRAHOF measures simulation runtime and model counts while varying the number of cases processed together and the update thresholds. It allows the retained model set to grow and waits for all cases in each group to finish [140]. Both approaches reuse history, with storage growth and delayed completion of case groups affecting their operating cost.

TSUNAMI illustrates these issues in customer churn prediction. It predicts and adapts daily, retains customer receipt histories within a configured horizon and can restart a returning customer after the previous history expires [141]. Training blocks selected through ADWIN reside on disk and are reloaded for drift adaptation. Its reported daily runtime and selected state memory reveal operational costs, within the selected measurement boundary. The complete TensorFlow footprint and sensitivity to arrival rate require additional measurements. A time horizon limits history age, while the number of customers and receipts within that horizon can still vary.

Neighboring explanation and prescription workloads require their own contracts. Control flow segmentation reduces the coalition features used for SHAP explanations, changing the explanatory units from individual events to segments; its evaluation focuses on the resulting explanations, with resource savings requiring measurement [142]. In prescription, a comparison of Markov decision processes and offline deep reinforcement learning examines computational cost alongside business KPI outcomes [143]. The Markov approach abstracts event histories and measures preparation, model construction and dynamic programming; the deep learning comparison times training under a fixed step budget. Nearly constant epoch time characterizes the fixed step budget; sensitivity to log size requires varying the training workload. The abstraction and different timing boundaries must accompany the reported computational advantage.

5.4.5. Choose predictors for their request and update workloads

Two applications seeking the same prediction can face different computational decisions when one reuses a stable predictor and the other repeatedly adapts to new behavior or labels. The implications of the prediction literature must therefore be expressed in terms of the intended request and update pattern as well as predictive quality. The preparation, serving and maintenance evidence reviewed above provides the components of that comparison, including costs that remain when model parameters are shared or frozen. Bringing them together places predictive monitoring within the review's selection framework and motivates the subsequent operating perspective: a chosen predictor must remain affordable and informative throughout the service for which it is used.

Avoiding duplicated prefixes targets historical preparation; compact encodings and model selection target requests; shared predictors can spread fitting costs across appli-

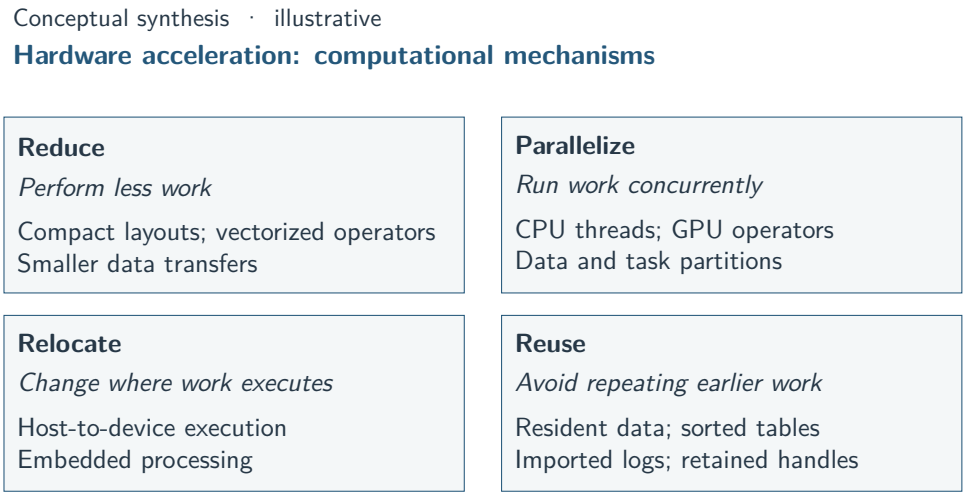

Figure 17. Conceptual synthesis: key terms for hardware acceleration. The four mechanisms connect representation choices to device execution and reuse. Data movement, device capacity and preparation determine whether faster operations improve the complete analysis.

cations; drift triggers target update frequency. Hold the prediction target and available information constant, count retained models and examples, and measure the cost of staying current. A service for concurrent running cases also needs arrival rate and update latency tests (Section 5.6).

Executing and maintaining analyses across tasks

Hardware acceleration concerns how work executes; continuous operation concerns how inputs, retained state and answers evolve. These views revisit task methods only to examine the additional execution or operating conditions.

5.5. Hardware acceleration: gains, overhead and reuse

Figure 17 relates compact representations and concurrent operators to the placement of work and reuse of resident data. These choices often contribute together to a hardware result.

Hardware acceleration revisits the preceding tasks from the perspective of where and how their computations execute. It is an execution perspective rather than an additional analysis task. An implementation identifies work units that can run on CPU cores, a GPU or separate workers, prepares suitable data representations and makes inputs available at those execution locations. Partial results may then need synchronization or combination. The useful execution choice depends on the available concurrency and the data movement needed to exploit it.

5.5.1. Data movement, memory and coordination limit parallel gains

The task sections have shown where process mining work arises and how algorithms can reduce or reuse it. The hardware perspective asks a complementary question: how much of the remaining work can be executed efficiently on the available processors? Answering it requires examining the preparation, transfer, memory and coordination demands that accompany concurrency. These demands explain why a computationally expensive task does not automatically benefit from an additional device or more workers. Locating them gives the review a basis for comparing execution choices across tasks and for separating a faster central operation from an improvement in the complete analysis supplied to the user.

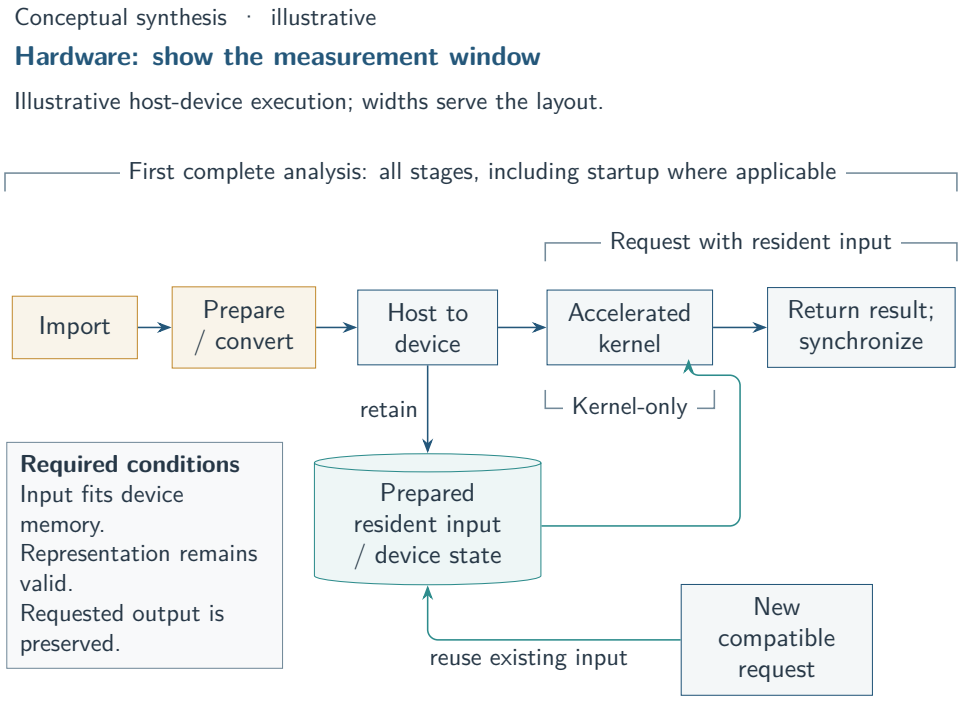

Figure 18. Conceptual synthesis. Illustrative host-device execution with three measurement brackets: kernel-only, a request with resident input, and the first complete analysis. Resident-input requests include result transfer and synchronization for the illustrated implementation; a new request reuses retained input rather than returning results into it. Stage widths are schematic. Transfer-sensitive transition counting and reusable GPU dataframes motivate these boundaries [144,145]; startup, CPU assistance, conversions and synchronization depend on the implementation.

Transition counts, candidate constraints and genetic populations expose different kinds of concurrency and impose different memory requirements. Preparing their representations, distributing the inputs and combining results can become substantial parts of the cost. A large event count can make parallel execution worthwhile, but activity diversity, repeated conversions and irregular dependencies can determine whether that opportunity is realized. Figure 18 makes preparation, transfers and resident-data reuse explicit.

5.5.2. Partitions, data layouts and reuse shape hardware gains

Partitions and data layouts connect the abstract opportunity for concurrency to an executable process mining method. They determine which operations can proceed independently, which information must be moved and which prepared structures remain available for later requests. Revisiting discovery and data access methods from this perspective serves a specific purpose in the review: it shows how representation, parallelization and reuse contribute together to an observed hardware result. The studies below follow these choices from counting and candidate evaluation to libraries that retain data across operations. This organization makes the frequency of analysis part of the comparison, because preparation and transfer costs are recovered over the requests that actually reuse their results.

In transition counting, including data transfer reduces the GPU advantage to approximately CPU performance in the reported setting [144]. PM4Py-GPU likewise reports slower

import on every tested log, despite improvements in subsequent operations [145]. Together, these results motivate separate comparisons of a one-off analysis and repeated operations on prepared, resident data.

Preparing regular data structures exposes operations that a device can execute concurrently. The transition counting implementations accelerate activity or user relations used by several analyses: CPU workers maintain local counts and combine them, while GPU variants reorganize counting, sorting and reduction [144]. Their main comparison assumes GPU-resident data. Representation size also matters: retaining all activity pairs exhausts memory for the 1,381-label workload. Device suitability therefore depends on activity diversity as well as event volume.

Partitions and device layouts. Alpha process discovery exposes substantial work in relation construction. Vidushi combines MATLAB vectorization with multicore or GPU execution [146]. Its comparisons include worker and device transfers but exclude worker pool startup. GPU memory capacity motivates splitting the input, and small activity alphabets favor the tested implementation. The OpenMP Alpha and Flexible Heuristic Miner studies identify task and data parallelism and evaluate synthetic workloads with varying activity counts [147,148]. Although the experimental environment includes a cluster, the reported OpenMP computations run within one node. Their thread counts vary with the chosen activity configurations, so the results reflect a joint change in processor allocation and workload structure.

Declarative process discovery makes alternative partitions explicit. Parallel Declare Miner assigns either groups of candidate constraints or chunks of the log to workers [72]. The former separates what is tested; the latter separates observations whose partial results must be combined. This contrast connects shared memory design to distributed process discovery in Section 5.2. The chosen partition must preserve the required counts and constraint semantics, and its scheduling and aggregation costs remain part of the computation.

GPU genetic process discovery changes the representation together with its operators. A causal matrix is encoded through three arrays, with corresponding crossover, mutation and parallel fitness operations [149]. The evaluation concerns both runtime and convergence, so the observed gain accompanies changes to the search procedure. A deterministic process discovery implementation instead computes CPU footprints for event windows and sends regularized work to a GPU through Aparapi [150]. Its flags reduce divergent control flow, but initialization is substantial relative to very small CPU jobs. A global state bound would additionally need to cover retained cases and history.

Reuse across operations and tools. Libraries and language bridges can retain prepared logs across several operations. PM4Py-GPU uses RAPIDS dataframes for case tables, variants and directly follows computations, reusing sorting and other prepared structures [145]. Its experiments increase event volume through replication while keeping alphabet and variant diversity largely fixed. They demonstrate the value of this representation for suitable operations on those workloads; sensitivity to independently growing diversity requires further evaluation.

A Rust core offers another way to expose efficient operations to existing tools [151]. The paper reports benchmarks and public implementations, including a bridge to Java and Python and an XES importer. The bridge retains an imported Rust log and returns small derived results when possible. Its comparison includes conversion costs and shows why repeatedly transferring a whole log differs from projecting or aggregating it through a retained handle. The five fixed log benchmarks support this integration design, while differences in baseline optimization limit broad language level conclusions.

5.5.3. Embedded response times apply to a restricted checking task

A restricted deployment adds an answer-preservation question to the execution choices just examined. Fitting a computation to limited device resources can depend on assumptions about the model or on limiting the diagnostic response, so response time must be interpreted together with those choices. This connects hardware acceleration directly to the analytical contracts developed in the task sections. The embedded conformance checking prototype below provides a concrete case for examining that connection: its supported model class, stopping rule and preparation location define what its measured response can establish. Including this case helps the review distinguish the feasibility of a particular deployed check from the broader requirements of conformance analysis.

A ground-based prototype for on-orbit conformance checking partitions a workflow net through spectral clustering and prepares compact representations for CUDA execution on a Jetson TX2i [152]. It uses one GPU with one CUDA stream. The measured event-based checks take roughly 3.3 to 3.6 milliseconds in the reported scenarios, including CPU assistance and device interaction but excluding ground-side model preparation. This establishes prototype response times, with flight deployment and energy savings requiring further evaluation.

The embedded checker's result is deliberately restricted. Supported nets exclude invisible transitions and duplicate labels, and checking stops at the first discrepancy. Its response times therefore apply to that diagnostic task. A complete optimal alignment or a model with unsupported transition types requires a different checker and a separate computational comparison.

5.5.4. Hardware results depend on workloads and timing scope

Hardware results bring together properties of an algorithm, its implementation, its workload and the resources assigned to it. Assessing what the literature establishes therefore requires identifying which of these factors an experiment varies and which stages its timing includes. This is the role of the evidence comparison here: it connects reported operation times and device feasibility to the review's question about comparable computational measurements. It also distinguishes measurements on executing hardware from simulation and integration demonstrations, which support different conclusions. These distinctions are needed before extending an observed result to a different activity alphabet, processor allocation, sequence of requests or deployment platform.

The CPU and GPU experiments cover operation timings, selected conversion costs and feasibility within device memory. Their differing import and initialization boundaries explain why a fast operator can suit repeated analysis while offering little one-off benefit. Replicated logs and thread counts tied to activity configurations leave the effects of independently varying workload structure and processor allocation unresolved.

Quantum-oriented evaluation currently provides a different kind of evidence. The inter-case predictive process monitoring study evaluates quantum learning models through classical simulation with PennyLane and sampled circuit measurements [153]. It also demonstrates a Camunda connection to an IBM quantum service. The experiments characterize predictive quality and simulated runtime, including training data sampling costs. Establishing hardware quantum advantage requires a matched comparison on quantum hardware beyond that integration demonstration.

5.5.5. Choose hardware for the complete analysis and expected reuse

A decision to use additional processors or a specialized device commits the analysis to a particular representation, memory capacity and pattern of data movement. The operation

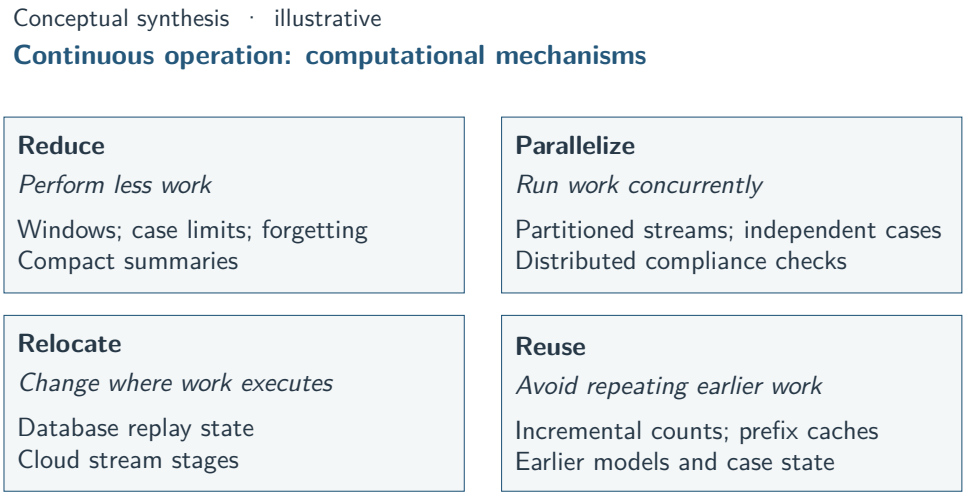

Figure 19. Conceptual synthesis: key terms for continuous operation. The four mechanisms change how continuing input is processed and remembered. Windows, forgetting and retained summaries each require explicit policies for answer scope, updates and returning cases.

timings reviewed above become useful for that decision when connected to the task's required output and expected sequence of requests. This synthesis places hardware choice within the review's broader method selection framework, including the possibility that an otherwise suitable operator cannot accommodate the relevant activity diversity or recover its preparation cost. It also prepares the later discussion of mechanism composition: data retained efficiently for one device operation must still support the inputs, outputs and conversions required by adjacent stages.

For repeated analysis, retaining sorted tables or imported logs can recover transfer and conversion costs. For a one-off run, those costs belong beside operator time. Embedded deployment additionally requires matching the supported model and diagnostic result to the application. These choices should be tested against the same CPU workload while varying activity diversity, representation size and processor allocation independently.

5.6. Continuous operation: managing state, history and change

Figure 19 links the four mechanisms to continuing arrivals: limiting retained history, partitioning work, moving processing or state, and updating earlier results.

Continuous operation maintains analyses while events keep arriving, rather than assuming a complete fixed log is available. Events extend unfinished cases and may require revising an answer or updating the information from which it is computed. The operating regime applies to data preparation, process discovery, conformance checking and prediction.

Four properties describe different aspects of this setting. *Online availability* makes an answer accessible during ongoing execution; *incremental computation* reuses earlier work when observations change; *bounded retained state* requires an explicit size bound on stored information; and *adaptation* adjusts an analysis to changes in process behavior. Each property requires its own guarantee and supporting evidence. Section 5.4 examines predictor preparation, requests and maintenance. This section generalizes the arrival, case history, retention and update questions across tasks.

5.6.1. Arrival rates and unfinished cases drive accumulating work

Continuous operation extends the workload perspective of the earlier themes from the size of an available input to the way computation accumulates over time. An analysis must

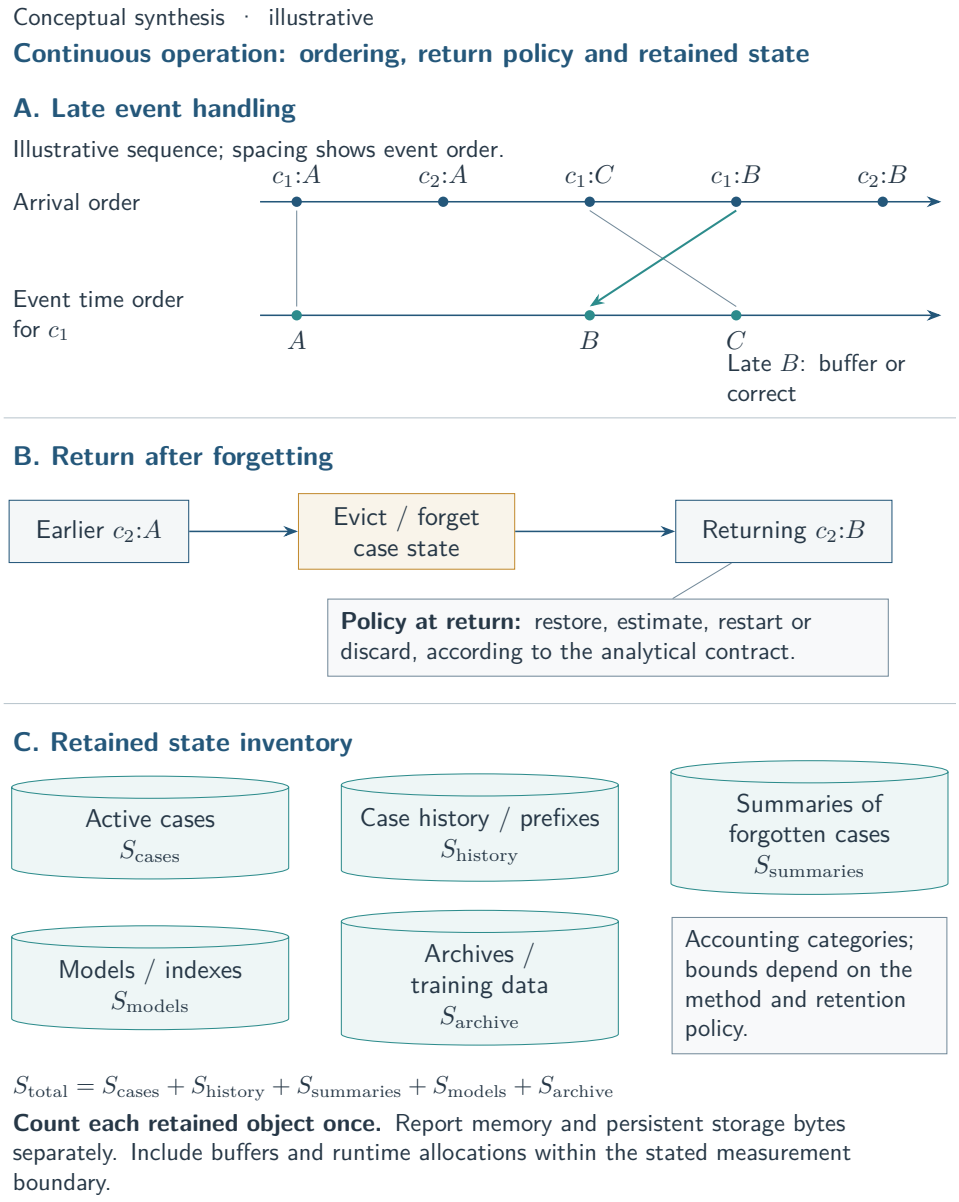

Figure 20. Conceptual synthesis. A: arrival order and event time order are linked explicitly; late c_1:B belongs between A and C. B: eviction precedes the return of case c_2, where the stated restoration, estimation, restart or discard policy takes effect. C: retained state categories form an inventory of stored information. Arrivals illustrate ordering; their spacing serves the layout. Retention-constrained conformance checking and unordered directly follows maintenance motivate the return and late event policies [154,155]. Count each retained object once; report memory and persistent storage bytes separately under the stated boundary.

process arrivals, maintain information about unfinished cases and supply answers while earlier observations may still need to be revisited. This makes arrival rate and retained history central to the review's account of scalability, alongside event volume and structural complexity. Identifying these sources of work prepares the comparison of incremental methods below and clarifies why online availability, bounded state and timely adaptation require separate evidence. It also links the prediction lifecycle to operating conditions that matter for discovery, conformance checking and data preparation alike.

Arrival rate, the number of unfinished cases and the extent of retained history create different sources of work. Reordered events may require repairs to earlier state, and

behavior changes may trigger model updates. A miner may process each event quickly while accumulated case histories continue growing. It may also maintain a compact summary while rebuilding a model remains expensive [156,157]. Figure 20 connects event and case histories to the retained state inventory.

5.6.2. Reusing summaries and case state requires retention policies

Successive answers in a continuous analysis often depend on much of the same earlier evidence. Summaries and retained case state offer ways to reuse that evidence while incorporating new observations, connecting this theme to the indexes and prepared representations discussed throughout the review. Their distinctive requirement is that the retained information must remain adequate as input continues and cases remain unfinished. Comparing maintained discovery evidence with conformance checking state reveals what each task needs to remember, when a final answer must be extracted and which costs persist between updates. This comparison establishes the role of retention policies before the following subsection examines the additional demands of disorder and process change.

For process discovery, a compact summary separates event updates from later model construction. The stream based abstract representation architecture implements this separation [156]. Its resource experiments measure updates to directly follows representations; downstream model extraction adds a separate cost. This architecture is useful when several model requests can share continuously maintained evidence.

The retention policy determines how much evidence remains in the summary. The early Heuristics Miner study compares bounded queues, aging and ordinary Lossy Counting [158]. Queue capacity limits retained entries, whereas the Lossy Counting parameter controls approximation error. A hard memory cap requires a separate retention limit. Its statistical guarantees require stationarity and sufficient retention of relevant activities, relations and concurrent cases. A policy therefore needs both an error interpretation and a storage interpretation.

Maintained process discovery evidence. Replacement and aging control historical influence in different ways. A method that constructs process maps treats activity relations and case predecessors as cache contents, comparing replacement policies through frequency loss and update time [159]. Its memory estimates count storage words and assume known case endings. Dynamic Constructs Competition Miner instead ages observations by occurrence or time [160]. Its preliminary throughput concerns footprint updates; model interpretation takes additional seconds, and most adaptation delays are measured in simulated time or trace counts. Removing an entry releases its storage, whereas reducing its weight changes influence and can leave the entry stored.

Update intervals determine when maintained process discovery evidence is consumed. Mini-Process Cubes partitions stream data and reuses succession and short loop matrices, combining historical and current evidence through a manually selected weight [161]. Its dump and reset policy writes accumulated events to disk and clears the working set, while historical storage can continue growing. Measuring memory use and responsiveness remains a proposed extension. A Moodle application maintains daily and cumulative learning paths from xAPI statements [162]. Grouping and mapping dominate its measured computation, while model construction is brief. These designs make the chosen update interval and the cost of constructing the summary central to the comparison.

Incremental model reuse also supports interactive and application-specific updates. Cortado extends selected model subtrees using observed variants and analyst guidance on a finite log [163]. Its guarantee concerns algorithmic extension and selected behavior; manual changes need separate checking, and the demonstrated setting uses finite logs.

SOFiA uses incrementally learned workflows for smart-office activity recognition [164]. Its evaluation establishes recognition performance; latency testing would characterize computational response. Both illustrate incremental use beyond a throughput benchmark. *Conformance checking state for unfinished cases.* Incremental prefix alignment reuses an earlier search and can preserve optimality with its exact pruning and reversion conditions [157]. Restricting reversion reduces retained search but changes the guarantee. The paper measures stored search states as a structural resource proxy. A distributed implementation adds synchronous move shortcuts, retained search structures and a prefix cache [165]. Its fixed deployment experiment examines event processing and backlog. Case deletion remains future work, so retained history can continue growing across the deployment.

Explicit retention policies expose the distinction between a case limit and a history limit. Capping alignment states per case reduces each retained prefix, but total state still depends on the number of cases [166]. The retention constrained framework adds case forgetting, resumable summaries and learned marking prediction [154]. Its combined case and prefix limits bound the main alignment store, while its repository of forgotten case summaries can continue growing. The learned alternative replaces some stored evidence with prediction and needs classifier preparation. The resulting guarantees describe the accuracy and conditions of approximate resumed checking.

Alternative representations move or replace this state. GO-TR uses Neo4j replay images and Cypher operations [167]. Its reported memory measurement covers the Python client, excluding database state; preparation is also outside the online timing. C-3PA searches a prepared proxy trie and attaches confidence and completeness information to approximate prefix alignments [168]. Its experiments include trie preparation and show memory growth over a long event sequence. HMMConf uses fixed model matrices and per-case state distributions, with a configured case limit [169]. Its evaluation measures both offline training and online updates, with the stress run remaining within the configured case cap.

An n-gram index makes state estimation a bounded suffix lookup [170]. For a fixed suffix length, query time is independent of accumulated trace length. This benefit applies to successfully prepared indexes: construction requires model reachability analysis, and preparation fails for some evaluated models. A service using the index must account for that construction cost and feasibility before relying on its lookup rate.

5.6.3. Late events and drift need different update policies

Reusing state over time requires deciding what an answer should mean when later information challenges the assumptions under which it was produced. Arrival disorder can reveal that an earlier event sequence was incomplete, while process drift can make an established description less representative of current behavior. These are different reasons for revisiting an answer, with different correction, waiting and adaptation costs. Their joint treatment here develops the temporal part of the review's analytical contract: which history an answer covers, when it is available and under what conditions it is revised. The following studies connect those requirements to ordering policies, event preparation, change detection and model updates across the task themes.

Arrival disorder can alter observed directly follows relations even when the underlying process remains stable. Buffering waits for ordering evidence and may discard events outside its tolerance. Speculative processing emits an early result but must revise relations after a late event [155]. The unordered DFG study evaluates resulting quality. A resource evaluation would additionally need to measure correction costs and bound the histories retained for repair.

Event time aware conformance checking retains timestamps and replays an affected suffix when late information arrives, with adaptive decay controlling the search [171]. Back to Order further handles equal timestamps through partially ordered groups and candidate reorderings [172]. Exact substring shortcuts reduce work in favorable cases, but the general reordering space can grow factorially. The experiments characterize runtime and alignment quality on sampled traces; a service-wide bound would also need to cover the active case population.

Preparation shares the same response budget. An Esper based correlator maps unlabeled events to candidate case identifiers and accepts changing domain constraints [18]. SwiftMend maintains approximate control flow context and clusters activity labels incrementally [173]. The correlator's response also depends on how probability calculations are grouped and how long unresolved events wait in time windows. SwiftMend reports mean context and clustering time per event. Establishing a hard state bound would additionally require accounting for all retained labels and clusters. Their relevance to Section 5.1 is that fast checking depends on both usable case and activity information and an efficient downstream solver.

Detection, characterization and adaptation. Drift detection identifies change; characterization explains it; adaptation updates a model or its retained evidence. An adaptive window detector reuses counts already maintained by a stream miner [174]. Its fixed complexity argument assumes a bounded summary window and fixed activity alphabet. Its evaluation measures detection quality and delay in pruning periods, rather than CPU time. Species based window selection instead adjusts windows using estimated coverage of activities, relations or variants [175]. The prototype measures latency as window size increases and throughput on one real log. The representativeness criterion applies to the selected abstraction; full behavioral coverage and a hard memory bound require separate guarantees.

Model libraries let drift detectors reuse earlier descriptions of behavior. A relation matrix detector maintains current, candidate and previous models, with a FIFO limit on the previous model library [176]. Its input units are complete traces, and its reported time is per change pattern. Those units define the response measured by the experiment.

Explaining a detected change adds model comparison to detection. Structural drift characterization compares process trees and expresses sudden changes as model operations [177]. Exact A* minimizes its defined tree edit mapping cost; a faster greedy method can return a less concise explanation. Its measured time includes constructing two process trees and searching for edits, while detection and waiting for a stable post-drift window are excluded. The tradeoff therefore concerns the cost and concision of characterizing an already identified change.

Predictive updates expose the cost of retaining and revisiting history. The remaining time framework compares period, case count and drift triggers with training history selection [178]. Average update times range from seconds to more than a thousand seconds across models and logs; feature enrichment can increase training time. Historical logs remain stored even when only new traces are used for fitting, and changed important features trigger reconstruction from all history. Update timings therefore describe individual fits, while storage and the number of fits determine lifetime cost.

Retaining task-specific parameters offers another way to recover earlier predictive behavior. Continual next activity prediction with prompts receives external drift locations, retains trees and prompts for each task, and updates backbone weights as well as prompts [179]. Its normalized training and inference times characterize processing over the stream,

while individual event response requires finer-grained measurement. The comparison must account for both predictive forgetting and growth in the retained task representations.

5.6.4. Service capacity must be tested under sustained arrivals

The preceding mechanisms specify how answers and state can be updated, but sustained operation is a property that must be established over continuing input. Event processing times become evidence of service capacity when related to arrival rates, backlog, the active case population and the work of adaptation. This makes continuous operation a particularly demanding instance of the review's question about evaluation boundaries: an experiment must account for work that accumulates or is triggered after ordinary updates. The studies below provide arrival-driven experiments, worker comparisons and supporting evaluation infrastructure. Reading them together identifies which operating conditions have been measured and which further experiments are needed to assess the promised response and retention behavior.

The Kinesis Flexible Heuristics Miner partitions cloud stream stages and incrementally updates the dependency graph [180]. Its three-hour experiment increases arrival rates on fixed provisioned infrastructure. The implementation retains retired case identifiers, discards later events for those cases under a statistical retirement policy for a fixed process. Inconsistent shard capacity and per-minute record figures prevent a reliable numerical capacity bound, while the experiment still identifies the tested arrival and retention regime.

Worker variation helps separate service capacity from a fixed deployment's behavior. Distributed SCIFF varies worker count when measuring the arrival rate that avoids growing backlog [181]. It partitions traces and compliance models; model partitioning requires unchained rules, and fine partitions can increase event replication. Timing semantics also affect the answer: omitting current time accommodates unordered arrival but postpones missing event decisions until closure; using current time enables prompt deadline checks under ordered arrival. The resulting capacity applies to that rule and timing configuration.

Reusable infrastructure supports implementing these services. Beamline supplies Java/Flink and Python/ReactiveX implementations with a common event representation [182]. Its tool paper establishes platform availability, leaving computational benchmarking to later evaluation. That implementation evidence serves a different purpose from an arrival rate or retained state experiment.

Comparative drift evaluations reveal how implementation and measurement choices affect conclusions. One study evaluates accuracy, memory, runtime and other goals, using different memory definitions across heterogeneous implementations [183]. Another unifies implementations and parameters for offline sudden drift detection [184]. Its average processing time per event characterizes offline detection, with sustained stream response requiring an arrival driven experiment.

Controlled workloads and reusable implementations support such comparisons. CPN and PLG2 generators provide source models and changes [185,186]. AVOCADO supplies challenges, implementations and latency definitions for constructing resource benchmarks [187]. These resources make later experiments reproducible while leaving service measurements to the deployed analysis.

5.6.5. Choose history and update policies for the operating profile

Selecting a continuously operating method also selects policies for what happens when history is forgotten, an old case returns or a late event changes an earlier answer. The literature's implications therefore depend on an operating profile that describes these situations alongside ordinary arrivals and update requests. Bringing the mechanisms and evidence together at this point completes the review's progression from identifying

expensive work to judging whether a method can remain useful over time. It provides the transition to Section 6, where preparation, answer preservation and measured costs are compared across tasks. The recommendations below identify the histories and service outcomes needed to make that continuing-use requirement testable.

Specify an operating profile: updates for individual events or groups of events, unfinished case population, retention limits, return after forgetting, arrival disorder and adaptation delay. Maintained summaries suit frequent requests when periodic extraction remains affordable. Prefix reuse needs a policy for long-lived and returning cases; disorder handling needs time for buffering or repair. Evaluate the chosen policy over sustained arrivals, including all retained stores, backlog and answer error, so that its service capacity matches the intended operating conditions.

6. Discussion: comparing and choosing methods

The studies reviewed here show that **useful scalability** takes several forms. A method may make a larger analysis possible, shorten the wait for an answer, reduce the cost of repeated requests or support a service that runs continuously. These benefits depend on the task and the intended use. Choosing a method therefore starts with the result that is needed and the work required to obtain it. The following discussion brings together the main lessons across tasks and explains how they can guide comparisons and further research.

6.1. Match computational mechanisms to the main bottleneck

The four mechanisms provide a simple way to explain how a method changes computation. **Reduction** performs less work, for example by filtering candidates, compressing repeated behavior or dividing a difficult search into smaller problems. **Parallelization** runs independent work at the same time. **Relocation** changes where work takes place, such as moving a query into a database or checking events near their source. **Reuse** keeps prepared information or earlier results so that later requests do not have to start from the beginning.

These choices often work together. Distributed genetic discovery both samples traces during parts of its search and spreads work across populations [71]. PM4Py-GPU combines concurrent operations with prepared data structures that can be reused [145]. Their benefits cannot be explained by additional processors alone. The practical starting point is the **main bottleneck**: the work that consumes the time or memory needed for the intended analysis. Preparing data, moving it between workers and combining results may become bottlenecks themselves. These costs help explain why adding workers can have little benefit for a small or poorly partitioned task.

Capacity is also a useful outcome. MongoDB OCEL handles larger inputs after the compared in-memory approach exhausts memory, even though it is slower where both approaches fit [36]. For an analyst whose data cannot otherwise be processed, making the analysis possible may matter more than reducing runtime on a smaller log. A comparison should state which of these benefits it is intended to establish.

6.2. Align answer requirements before comparing costs

A meaningful comparison begins with the **required answer**. Conformance checking may provide a detailed explanation of deviations, a judgment about a particular constraint or an estimated score for a log. These outputs support different decisions. An overall score may be sufficient for routine monitoring, while investigating a particular case may require the individual deviations. Similarly, a model discovered from a recent window describes a

different history from one based on all recorded executions. A runtime difference is difficult to interpret without knowing which information each method supplies.

The same issue appears in prediction. ParallAct predicts several possible next activities and takes longer than its single-target comparator in the reported evaluation [128]. The extra time accompanies a richer output. Comparisons should therefore align the requested targets, available information and level of detail before attributing a difference to computational efficiency. Where outputs differ, the evaluation should explain whether the additional information is useful for the intended decision.

Acceptable approximation should also be specified in advance. A faster estimate may be suitable when some error is tolerable, but the amount and kind of error still matter. A method that omits rare behavior or covers only selected traces needs to be judged against the use of that result. The review's analytical contract makes these questions explicit: what is returned, which inputs are supported, what is guaranteed and which history is covered. Stating these conditions in plain terms makes it easier to identify methods that can meet the same need.

6.3. Reuse must recover preparation and maintenance costs

The **complete cost** of obtaining an answer includes more than the central computation. Loading and sorting events, constructing indexes, preparing a model and transferring data can all take substantial time. PM4Py-GPU reports slower import despite faster subsequent operations, while prepared state-estimation indexes can offer inexpensive lookups only after successful construction [145,170]. A measurement that begins after these stages answers a useful but narrower question: how expensive is the operation once its input is ready?

Repeated use changes this comparison. An index or prepared representation may be expensive to build but serve many later requests. DB-XES illustrates this by maintaining statistics that later discovery reports can reuse [12]. The value of preparation therefore depends on how often the information is used and how much work each use avoids. For a single analysis, preparation may consume the time saved by faster checking, as the reported Hospital Billing stages illustrate [105].

The **useful lifetime** of prepared information is equally important. New events, changed queries or different model requirements may require an update or a rebuild. Frequent rebuilding can prevent later savings from recovering the initial cost. Evaluation should therefore distinguish a first run, repeated requests over stable data and requests made while the data change. Each comparison should include the maintenance needed to keep answers valid, along with failures to construct or store the prepared representation. This explains when reuse is a practical advantage and when its preparation is too costly for the intended use.

6.4. Sustained operation depends on state and history policies

Continuous operation introduces costs that may grow after an ordinary event has been processed. **Retained state** is the information kept for future work: unfinished case histories, cached results, summaries, models and supporting data. Limiting one store does not necessarily limit the whole system. The reviewed retention framework can bound its main alignment store while retaining a growing repository of forgotten-case summaries, and GRAHOF can retain an increasing collection of predictive models [140,154]. Moving replay state into a database also leaves costs outside the application process being measured [167].

Sustained operation requires examining what happens over continuing arrivals and updates. A system may handle ordinary events quickly but accumulate waiting work

during bursts, while many cases remain unfinished or when a model must be updated. A useful evaluation therefore varies arrival rate and the number of active cases, measures response times and waiting work, and tracks all retained information over time. Short event timings alone cannot establish that the service will remain responsive within its available resources.

A clear **history policy** connects resource limits to the answers supplied. If an old case returns after its history has been removed, the method may restore information, estimate the missing state, restart the case or discard the event. Late events can also require correcting an earlier answer. These choices affect both cost and meaning. Reporting them allows a user to decide whether the service's memory savings and response times are compatible with the histories that occur in practice.

6.5. Workload structure determines which mechanisms help

Workload structure helps explain why methods behave differently on logs of similar size. A log with many repeated traces offers opportunities to reuse results. A log with the same event count but many different traces offers less repetition. More activity labels can enlarge discovery searches, while a more complex reference model can make checking each trace harder. Object-centric analysis adds another source of work through the relationships between events and objects. Event count is therefore a useful description of scale, but it does not capture all the work a method must perform.

The **requested analysis** matters as well. Graph representations incur construction and storage costs, and changing the requested case view can change which query method is faster or supported [34,35]. A result obtained for one query should therefore be interpreted with that query's information requirements. Likewise, repeating existing traces creates a larger dataset while leaving much of its behavioral diversity unchanged. Such an experiment tests volume growth under that structure; it provides limited evidence about more varied behavior.

A clearer comparison varies relevant features separately where possible. For example, event count can be held fixed while the number of distinct traces or activities changes. Worker allocation can then be varied on the same inputs. Reporting these choices helps readers understand whether a benefit comes from handling more data, exploiting repetition, reducing a difficult search or using additional resources. It also makes the result easier to relate to a new workload.

6.6. Assess information loss and cost when combining methods

Combining mechanisms is promising because each can address a different part of an analysis. A database may prepare a smaller input for an accelerated operation, or a retained summary may support repeated checks. The first requirement is **information compatibility**: the output of one stage must retain what the next stage needs. Stackt illustrates why this needs attention, since an export can omit temporal or event-to-attribute relationships required downstream [14]. A transformation can be computationally convenient while making a later question impossible to answer in the intended way.

The second requirement is to evaluate the **combined cost and result**. Conversion, transfer and coordination may consume savings achieved inside individual components. Sampling and approximation can also change the final answer together: the reviewed combination reduces checking time in the reported setting while increasing fitness error [106]. The relevant question is whether the completed analysis still gives a sufficiently useful answer at an acceptable cost.

Comparing each component separately with the complete combination helps explain these effects. The inputs, requested outputs and available resources should remain compara-

ble, and the evaluation should count work at the interfaces as well as inside the components. This makes it possible to identify where savings arise and where they disappear. It also distinguishes a combination that has been evaluated from one that remains a proposal for further investigation.

6.7. Testing reuse, structural diversity and sustained operation

The findings point to three practical priorities. The first is **reuse under change**: measure how repeated requests recover preparation costs as updates and rebuilds become more frequent. The second is **structural diversity**: test how methods respond to different traces, activities, models and object relationships at comparable event volumes. The third is **long-running service**: include unfinished cases, late arrivals, returning cases and model updates while tracking response, waiting work and retained information. Together, these experiments would clarify when a reported improvement remains useful beyond the original workload.

Further comparisons should separate the effects of smaller problems from those of additional workers and test whether benefits survive when methods are combined. Shared predictors also need comparisons that include initial preparation, supporting examples, repeated requests and upkeep. Across these settings, **comparable evidence** requires matching the answer being requested and making the measured costs explicit. Unsuccessful runs and resource exhaustion are informative outcomes, because they show where a method stops being a practical option.

Finally, the review distinguishes a **missing method**, a **missing evaluation** and **limited review coverage**. A promising method may already exist while its performance under the relevant conditions remains unmeasured. Conversely, a measurement absent from this selected collection may be available elsewhere. The absence of identified energy measurements in the full text synthesis therefore supports a call for further evidence; it does not establish their absence across the wider literature. Keeping these distinctions clear directs future work toward the question that actually remains unresolved.

7. Threats to validity

OpenAlex and Scopus differ in coverage and indexing, and relevant work may use terms outside the queries. Checks against known publications helped refine the search but cannot establish complete coverage. Indexing delays and the partial 2026 window also limit comparisons over time.

The full text synthesis uses a purposive selection of papers from the search map (Section 3.4). Citation roles and access to full texts can favor particular methods, venues or versions. Reporting counts therefore describe the assessed collection and should not be treated as estimates for the entire field. Related publications may also share methods or experiments despite the checks for duplicate records.

Eligibility, classification and extraction require judgment, especially when a paper combines several tasks or mechanisms. A common codebook and targeted checks support consistency, but independent duplicate assessment could reveal different decisions. The review synthesizes published evidence without replicating experiments or checking software correctness. Differences in workloads, hardware, baselines and measurement coverage limit comparisons, and unspecified details remain unresolved. The conclusions therefore apply under the conditions reported by the assessed studies.

8. Conclusion

Computational scalability in process mining depends on the task, its representation and the required analytical result. For **RQ1**, the assessed papers expose costs in event preparation, process discovery, conformance checking, continuous state management and prediction. These workloads depend on event count together with variants, activities, model structure, object relationships and active cases.

For **RQ2**, methods reduce, parallelize, relocate and reuse work, often in combination. Their analytical contracts determine whether they are alternatives: supported models, output detail, approximation and historical scope must be aligned before comparing runtime. Preparation and reuse costs further determine whether a faster operation improves the intended workload.

For **RQ3**, the evidence atlas separates measured resources, varied conditions, evaluated stages and implementation statements within the 176-study full text synthesis. The assessment identified runtime measurements in 131 studies, measurements including preparation costs in 33 and sustained operation measurements in six. The evidence spans narrow operation timings and broader lifecycle studies. These counts characterize reporting within the selected full text synthesis; broader prevalence and performance comparisons require further evaluation.

For **RQ4**, useful experiments test break-even reuse, structural workload changes, sustained state growth and composition of methods. The practical procedure is to specify the required result, characterize the workload and intended scalability objective, identify the costly stage, shortlist compatible mechanisms, and compare complete relevant costs under aligned conditions. The framework and contract matrix organize these comparisons and identify the evidence needed to evaluate them. The central question is which method provides the required result at an acceptable cost across preparation, execution and maintenance.

Author Contributions: A.B. carried out all aspects of the work: conceptualization, methodology, software, validation, formal analysis, investigation, resources, data curation, writing—original draft preparation, writing—review and editing, visualization, supervision, project administration and funding acquisition.

Funding: Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and from the German Federal Ministry of Research, Technology and Space (BMFTR), the Ministry of Culture and Science of North Rhine-Westphalia (MKW NRW), and the Hessian Ministry of Science and Research, Arts and Culture (HMWK) under grant agreement No 101250682.

Institutional Review Board Statement: Not applicable. This review synthesizes published literature and involves no recruitment of human participants or experiments on animals.

Informed Consent Statement: Not applicable.

Data Availability Statement: This review is based on the cited literature. Search queries and aggregate results are included in the article. The accompanying document `search-map-uncited-references.pdf` provides the eligibility criteria, references, inclusion rationales and classifications for the 743 mapped publications not cited in the review.

Acknowledgments: Elicit supported scoping and targeted bibliographic and abstract recovery. OpenAI Codex assisted with screening, full text assessment, metadata reconciliation, extraction, synthesis and manuscript drafting. The methodology describes these uses.

Conflicts of Interest: The author declares no competing interests.

References

1. van der Aalst, W. *Process Mining: Data Science in Action*, 2 ed.; Springer Berlin Heidelberg, 2016; pp. 1–467. https://doi.org/10.1007/978-3-662-49851-4.
2. Verbeek, H.M.W.; van der Aalst, W.M.P.; Munoz-Gama, J. Divide and Conquer: A Tool Framework for Supporting Decomposed Discovery in Process Mining. *The Computer Journal* **2017**, *60*, 1649–1674. https://doi.org/10.1093/comjnl/bxx040.
3. Augusto, A.; Conforti, R.; Dumas, M.; La Rosa, M.; Maggi, F.M.; Marrella, A.; Mecella, M.; Soo, A. Automated Discovery of Process Models from Event Logs: Review and Benchmark. *IEEE Transactions on Knowledge and Data Engineering* **2019**, *31*, 686–705. https://doi.org/10.1109/tkde.2018.2841877.
4. Tavares, G.M.; Oyamada, R.S.; Barbon, S.; Ceravolo, P. Trace encoding in process mining: A survey and benchmarking. *Engineering Applications of Artificial Intelligence* **2023**, *126*, 107028. https://doi.org/10.1016/j.engappai.2023.107028.
5. Sato, D.M.V.; De Freitas, S.C.; Barddal, J.P.; Scalabrin, E.E. A Survey on Concept Drift in Process Mining. *ACM Computing Surveys* **2022**, *54*, 1–38. https://doi.org/10.1145/3472752.
6. Ghahfarokhi, A.F.; Park, G.; Berti, A.; van der Aalst, W.M.P. OCEL: A Standard for Object-Centric Event Logs. In Proceedings of the New Trends in Database and Information Systems. Springer International Publishing, 2021, Vol. 1450, *Communications in Computer and Information Science*, pp. 169–175. https://doi.org/10.1007/978-3-030-85082-1_16.
7. van der Aalst, W.M.P. Distributed Process Discovery and Conformance Checking. In Proceedings of the Fundamental Approaches to Software Engineering. Springer Berlin Heidelberg, 2012, Vol. 7212, *Lecture Notes in Computer Science*, pp. 1–25. https://doi.org/10.1007/978-3-642-28872-2_1.
8. Berti, A. Increasing Scalability of Process Mining using Event Dataframes: How Data Structure Matters. arXiv (Cornell University), 2019. https://doi.org/10.48550/arxiv.1907.12817.
9. Petersen, K.; Vakkalanka, S.; Kuzniarz, L. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology* **2015**, *64*, 1–18. https://doi.org/10.1016/j.infsof.2015.03.007.
10. Page, M.J.; McKenzie, J.E.; Bossuyt, P.M.; Boutron, I.; Hoffmann, T.C.; Mulrow, C.D.; Shamseer, L.; Tetzlaff, J.M.; Akl, E.A.; Brennan, S.E.; et al. The PRISMA 2020 statement: an updated guideline for reporting systematic reviews. *BMJ* **2021**, *372*, n71. https://doi.org/10.1136/bmj.n71.
11. Rethlefsen, M.L.; Kirtley, S.; Waffenschmidt, S.; Ayala, A.P.; Moher, D.; Page, M.J.; Koffel, J.B.; PRISMA-S Group. PRISMA-S: an extension to the PRISMA Statement for Reporting Literature Searches in Systematic Reviews. *Systematic Reviews* **2021**, *10*, 39. https://doi.org/10.1186/s13643-020-01542-z.
12. Syamsiyah, A.; van Dongen, B.F.; van der Aalst, W.M.P. DB-XES: Enabling Process Discovery in the Large. In Proceedings of the Data-Driven Process Discovery and Analysis. Springer International Publishing, 2018, Vol. 307, *Lecture Notes in Business Information Processing*, pp. 53–77. https://doi.org/10.1007/978-3-319-74161-1_4.
13. van Dongen, B.F.; Shabani, S. Relational XES : data management for process mining. In Proceedings of the Proceedings of the CAiSE 2015 Forum at the 27th International Conference on Advanced Information Systems Engineering. CEUR-WS.org, 2015, Vol. 1367, *CEUR Workshop Proceedings*, pp. 169–176.
14. Bosmans, L.; Peeperkorn, J.; Goossens, A.; Lugaresi, G.; De Smedt, J.; De Weerdt, J. Dynamic and Scalable Data Preparation for Object-Centric Process Mining. arXiv (Cornell University), 2024. https://doi.org/10.48550/arxiv.2410.00596.
15. Reguieg, H.; Toumani, F.; Motahari-Nezhad, H.R.; Benatallah, B. Using Mapreduce to Scale Events Correlation Discovery for Business Processes Mining. In Proceedings of the Business Process Management. Springer Berlin Heidelberg, 2012, Vol. 7481, *Lecture Notes in Computer Science*, pp. 279–284. https://doi.org/10.1007/978-3-642-32885-5_22.
16. Reguieg, H.; Benatallah, B.; Nezhad, H.R.M.; Toumani, F. Event Correlation Analytics: Scaling Process Mining Using Mapreduce-Aware Event Correlation Discovery Techniques. *IEEE Transactions on Services Computing* **2015**, *8*, 847–860. https://doi.org/10.1109/tsc.2015.2476463.
17. Cheng, L.; Van Dongen, B.F.; Van Der Aalst, W.M.P. Efficient Event Correlation over Distributed Systems. In Proceedings of the 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID). IEEE, 2017, pp. 1–10. https://doi.org/10.1109/ccgrid.2017.94.
18. Helal, I.M.; Awad, A. Online correlation for unlabeled process events: A flexible CEP-based approach. *Information Systems* **2022**, *108*, 102031. https://doi.org/10.1016/j.is.2022.102031.
19. Guabtni, A.; Motahari-Nezhad, H.R.; Benatallah, B. Using Graph Aggregation for Service Interaction Message Correlation. In Proceedings of the Advanced Information Systems Engineering. Springer Berlin Heidelberg, 2011, Vol. 6741, *Lecture Notes in Computer Science*, pp. 642–656. https://doi.org/10.1007/978-3-642-21640-4_47.
20. Seiger, R.; Kurz, A.F.; Franceschetti, M. Online detection of process activity executions from IoT sensors using generated event processing services. *Future Generation Computer Systems* **2025**, *174*, 107987. https://doi.org/10.1016/j.future.2025.107987.
21. Koschmider, A.; Duchmann, F. Extraction of Meaningful Events for Process Mining from Blockchain. In *Blockchain and Robotic Process Automation*; Springer International Publishing, 2021; pp. 13–29. https://doi.org/10.1007/978-3-030-81409-0_2.

22. Bauer, M.; Senderovich, A.; Gal, A.; Grunske, L.; Weidlich, M. How Much Event Data Is Enough? A Statistical Framework for Process Discovery. In Proceedings of the Advanced Information Systems Engineering. Springer International Publishing, 2018, Vol. 10816, *Lecture Notes in Computer Science*, pp. 239–256. https://doi.org/10.1007/978-3-319-91563-0_15.
23. Bernard, G.; Andritsos, P. Selecting Representative Sample Traces from Large Event Logs. In Proceedings of the 2021 3rd International Conference on Process Mining (ICPM). IEEE, 2021, pp. 56–63. https://doi.org/10.1109/icpm53251.2021.9576679.
24. Rafiei, M.; Wangelik, F.; van der Aalst, W.M.P. TraVaS: Differentially Private Trace Variant Selection for Process Mining. In Proceedings of the Process Mining Workshops. Springer Nature Switzerland, 2023, Vol. 468, *Lecture Notes in Business Information Processing*, pp. 114–126. https://doi.org/10.1007/978-3-031-27815-0_9.
25. Mannhardt, F. XESLite - managing large XES event logs in ProM. Technical Report BPM-16-04, BPMcenter.org, 2016.
26. Syamsiyah, A.; van Dongen, B.F.; Dijkman, R.M. A Native Operator for Process Discovery. In Proceedings of the Database and Expert Systems Applications. Springer International Publishing, 2018, Vol. 11030, *Lecture Notes in Computer Science*, pp. 292–300. https://doi.org/10.1007/978-3-319-98812-2_25.
27. Schönig, S.; Di Ciccio, C.; Mendling, J. Configuring SQL-based process mining for performance and storage optimisation. In Proceedings of the Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing. ACM, 2019, pp. 94–97. https://doi.org/10.1145/3297280.3297532.
28. Hinkka, M.; Lehto, T.; Heljanko, K. Assessing Big Data SQL Frameworks for Analyzing Event Logs. In Proceedings of the 2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP). IEEE, 2016, pp. 101–108. https://doi.org/10.1109/pdp.2016.26.
29. Aamer, H.; Montali, M.; Van den Bussche, J. What Can Database Query Processing Do for Instance-Spanning Constraints? In Proceedings of the Business Process Management Workshops. Springer International Publishing, 2023, Vol. 460, *Lecture Notes in Business Information Processing*, pp. 132–144. https://doi.org/10.1007/978-3-031-25383-6_11.
30. Kampik, T.; Lücke, A.; Horstmann, J.; Wheeler, M.; Eickhoff, D. SIGNAL - The SAP Signavio Analytics Query Language. arXiv (Cornell University), 2023. https://doi.org/10.48550/arxiv.2304.06811.
31. Vogelgesang, T.; Ambrosy, J.; Becher, D.; Seilbeck, R.; Geyer-Klingeberg, J.; Klenk, M. Celonis PQL: A Query Language for Process Mining. In *Process Querying Methods*; Springer International Publishing, 2022; pp. 377–408. https://doi.org/10.1007/978-3-030-92875-9_13.
32. Kampik, T.; Okulmus, C. Expressive Power and Complexity Results for SIGNAL, an Industry-Scale Process Query Language. In Proceedings of the Business Process Management Forum. Springer Nature Switzerland, 2024, Vol. 526, *Lecture Notes in Business Information Processing*, pp. 3–19. https://doi.org/10.1007/978-3-031-70418-5_1.
33. Brand, J.; Kampik, T.; Okulmus, C.; Weidlich, M. One Language to Rule Them All: Behavioural Querying of Process Data Using SQL. In Proceedings of the Process Mining Workshops. Springer Nature Switzerland, 2025, Vol. 533, *Lecture Notes in Business Information Processing*, pp. 18–30. https://doi.org/10.1007/978-3-031-82225-4_2.
34. Esser, S.; Fahland, D. Multi-Dimensional Event Data in Graph Databases. *Journal on Data Semantics* **2021**, *10*, 109–141. https://doi.org/10.1007/s13740-021-00122-1.
35. Nour Eldin, A.; Assy, N.; Kobeissi, M.; Baudot, J.; Gaaloul, W. Enabling Multi-process Discovery on Graph Databases. In Proceedings of the Cooperative Information Systems. Springer International Publishing, 2022, Vol. 13591, *Lecture Notes in Computer Science*, pp. 112–130. https://doi.org/10.1007/978-3-031-17834-4_7.
36. Berti, A.; Ghahfarokhi, A.F.; Park, G.; van der Aalst, W. A Scalable Database for the Storage of Object-Centric Event Logs (Extended Abstract). In Proceedings of the Proceedings of the ICPM Doctoral Consortium and Demo Track 2021. CEUR-WS.org, 2021, Vol. 3098, *CEUR Workshop Proceedings*, pp. 19–20.
37. Küsters, A.; van der Aalst, W. Easy and Efficient Object-Centric Process Querying with the OCPQ Tool. In Proceedings of the Joint Proceedings of the Best Dissertation Award, Doctoral Consortium, and Demonstration & Resources Forum at BPM 2025. CEUR-WS.org, 2025, Vol. 4032, *CEUR Workshop Proceedings*, pp. 184–191. https://doi.org/10.18154/rwth-2025-09779.
38. Xiong, J.; Xiao, G.; Kalayci, T.E.; Montali, M.; Gu, Z.; Calvanese, D. A Virtual Knowledge Graph Based Approach for Object-Centric Event Logs Extraction. In Proceedings of the Process Mining Workshops. Springer Nature Switzerland, 2023, Vol. 468, *Lecture Notes in Business Information Processing*, pp. 466–478. https://doi.org/10.1007/978-3-031-27815-0_34.
39. Berti, A.; Park, G.; Rafiei, M.; van der Aalst, W.M.P. A generic approach to extract object-centric event data from databases supporting SAP ERP. *Journal of Intelligent Information Systems* **2023**, *61*, 835–857. https://doi.org/10.1007/s10844-023-00799-9.
40. Vogelgesang, T.; Appelrath, H.J. A Relational Data Warehouse for Multidimensional Process Mining. In Proceedings of the Data-Driven Process Discovery and Analysis. Springer International Publishing, 2017, Vol. 244, *Lecture Notes in Business Information Processing*, pp. 155–184. https://doi.org/10.1007/978-3-319-53435-0_8.
41. Khayatbashi, S.; Miri, N.; Jalali, A. Advancing Object-Centric Process Mining with Multi-Dimensional Data Operations. arXiv (Cornell University), 2024. https://doi.org/10.48550/arxiv.2412.00393.

42. Berti, A. Filtering and Sampling Object-Centric Event Logs. arXiv (Cornell University), 2022. https://doi.org/10.48550/arxiv.2205.01428.
43. Hernandez, S.; van Zelst, S.J.; Ezpeleta, J.; van der Aalst, W.M. Handling big(ger) logs : connecting ProM 6 to Apache Hadoop. In Proceedings of the Proceedings of the BPM Demo Session 2015. CEUR-WS.org, 2015, Vol. 1418, *CEUR Workshop Proceedings*, pp. 80–84.
44. Mohamed, B.; ElHelw, M.; Awad, A. Towards Scalable Process Mining Pipelines. In Proceedings of the 2023 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech). IEEE, 2023, pp. 0175–0182. https://doi.org/10.1109/dasc/picom/cbdcom/cy59711.2023.10361330.
45. Burattin, A.; Kindler, E.; Dyhre, N.; Vestrup, S.; Zerbato, F.; Weber, B. Process Mining Pipelines with Controlled Sharing of Data and Algorithms. In Proceedings of the Business Process Management Workshops. Springer Nature Switzerland, 2025, Vol. 534, *Lecture Notes in Business Information Processing*, pp. 357–369. https://doi.org/10.1007/978-3-031-78666-2_27.
46. Reiter, H.; Edinger, J.; Kabierski, M.; Koschmider, A.; Landsiedel, O.; Lepsien, A.; Lu, X.; Marrella, A.; Serral, E.; Schulte, S.; et al. ContinuumConductor : Decentralized Process Mining on the Edge-Cloud Continuum. arXiv, 2025. https://doi.org/10.48550/arxiv.2512.07280.
47. Reiter, H.; Imenkamp, C.; Koschmider, A.; Hasselbring, W. Distributed Event Factory: A Tool for Generating Event Streams on Distributed Data Sources. In Proceedings of the Doctoral Consortium and Demo Track 2024 at the International Conference on Process Mining 2024. CEUR-WS.org, 2024, Vol. 3783, *CEUR Workshop Proceedings*.
48. Reiter, H.; Rathje, P.; Landsiedel, O.; Hasselbring, W. DPM-Bench: Benchmark for Distributed Process Mining Algorithms on Cyber-Physical Systems. arXiv (Cornell University), 2025. https://doi.org/10.48550/arxiv.2502.09975.
49. Leemans, S.J.J.; Fahland, D.; van der Aalst, W.M.P. Scalable Process Discovery with Guarantees. In Proceedings of the Enterprise, Business-Process and Information Systems Modeling. Springer International Publishing, 2015, Vol. 214, *Lecture Notes in Business Information Processing*, pp. 85–101. https://doi.org/10.1007/978-3-319-19237-6_6.
50. Hernandez, S.; Ezpeleta, J.; Zelst, S.v.; Aalst, W.M.P.v.d. Assessing Process Discovery Scalability in Data Intensive Environments. In Proceedings of the 2015 IEEE/ACM 2nd International Symposium on Big Data Computing (BDC). IEEE, 2015, pp. 99–104. https://doi.org/10.1109/bdc.2015.31.
51. van der Aalst, W.M.P. A general divide and conquer approach for process mining. In Proceedings of the Proceedings of the 2013 Federated Conference on Computer Science and Information Systems. IEEE, 2013, pp. 1–10.
52. Roci, A.; Davidrajuh, R. A Polynomial-time Alpha-Algorithm for Process Mining. *International journal of simulation: systems, science & technology* **2018**, *19*, 12.1–12.7. https://doi.org/10.5013/ijssst.a.19.05.12.
53. Pegoraro, M.; Mannel, L.L. Leveraging Meta-Properties of Petri Net Classes to Efficiently Enforce Formal Guarantees in Bottom-Up Process Discovery. In Proceedings of the Mining a Scientist's Process. Springer Nature Switzerland, 2026, Vol. 16480, *Lecture Notes in Computer Science*, pp. 506–524. https://doi.org/10.1007/978-3-032-17618-9_34.
54. van der Aalst, W.M.P. Decomposing Process Mining Problems Using Passages. In Proceedings of the Application and Theory of Petri Nets. Springer Berlin Heidelberg, 2012, Vol. 7347, *Lecture Notes in Computer Science*, pp. 72–91. https://doi.org/10.1007/978-3-642-31131-4_5.
55. Verbeek, H.M.W.; van der Aalst, W.M.P. Decomposed Process Mining: The ILP Case. In Proceedings of the Business Process Management Workshops. Springer International Publishing, 2015, Vol. 202, *Lecture Notes in Business Information Processing*, pp. 264–276. https://doi.org/10.1007/978-3-319-15895-2_23.
56. Hompes, B.; Verbeek, E.; van der Aalst, W. Finding suitable activity clusters for decomposed process discovery. In Proceedings of the Proceedings of the 4th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2014). CEUR-WS.org, 2014, Vol. 1293, *CEUR Workshop Proceedings*, pp. 16–30.
57. Hompes, B.F.A.; Verbeek, H.M.W.; van der Aalst, W.M.P. Finding Suitable Activity Clusters for Decomposed Process Discovery. In Proceedings of the Data-Driven Process Discovery and Analysis. Springer International Publishing, 2015, Vol. 237, *Lecture Notes in Business Information Processing*, pp. 32–57. https://doi.org/10.1007/978-3-319-27243-6_2.
58. Yan, Z.; Sun, B.; Chen, Y.; Wen, L.; Hu, L.; Wang, J.; Yang, M.; Wang, L. Decomposed and parallel process discovery: A framework and application. *Future Generation Computer Systems* **2019**, *98*, 392–405. https://doi.org/10.1016/j.future.2019.03.048.
59. Viale, P.; Benayadi, N.; Le Goc, M.; Pinaton, J. DISCOVERING LARGE SCALE MANUFACTURING PROCESS MODELS FROM TIMED DATA - Application to STMicroelectronics' Production Processes. In Proceedings of the Proceedings of the 5th International Conference on Software and Data Technologies, Volume 1: ICSOFT. SciTePress, 2010, pp. 227–235. https://doi.org/10.5220/0002930302270235.
60. Yang, L.; McClean, S.; Faddy, M.; Donnelly, M.; Khan, K.; Burke, K. Automating mixture model fitting of task durations for process conformance checking. *Data Mining and Knowledge Discovery* **2025**, *39*, 53. https://doi.org/10.1007/s10618-025-01131-5.

61. Augusto, A.; Conforti, R.; Dumas, M.; La Rosa, M.; Polyvyanyy, A. Split miner: automated discovery of accurate and simple business process models from event logs. *Knowledge and Information Systems* **2019**, *59*, 251–284. https://doi.org/10.1007/s10115-018-1214-x.
62. Huang, T.H.; Schneider, E.; Pegoraro, M.; van der Aalst, W.M.P. Fast & Sound: Accelerating Synthesis-Rules-Based Process Discovery. In Proceedings of the Enterprise, Business-Process and Information Systems Modeling. Springer Nature Switzerland, 2024, Vol. 511, *Lecture Notes in Business Information Processing*, pp. 259–274. https://doi.org/10.1007/978-3-031-61007-3_20.
63. van der Aalst, W.M.P.; De Masellis, R.; Di Francescomarino, C.; Ghidini, C. Learning Hybrid Process Models from Events: Process Discovery Without Faking Confidence. In Proceedings of the Business Process Management. Springer International Publishing, 2017, Vol. 10445, *Lecture Notes in Computer Science*, pp. 59–76. https://doi.org/10.1007/978-3-319-65000-5_4.
64. Cheng, L.; van Dongen, B.F.; van der Aalst, W.M. Scalable Discovery of Hybrid Process Models in a Cloud Computing Environment. *IEEE Transactions on Services Computing* **2020**, *13*, 368–380. https://doi.org/10.1109/tsc.2019.2906203.
65. Folz-Weinstein, S.; Rennert, C.; Mannel, L.L.; Bergenthum, R.; van der Aalst, W. eST² Miner - Process Discovery Based on Firing Partial Orders. In Proceedings of the Advanced Information Systems Engineering. Springer Nature Switzerland, 2025, Vol. 15702, *Lecture Notes in Computer Science*, pp. 59–75. https://doi.org/10.1007/978-3-031-94571-7_4.
66. Back, C.O.; Slaats, T.; Hildebrandt, T.T.; Marquard, M. DisCoveR: accurate and efficient discovery of declarative process models. *International Journal on Software Tools for Technology Transfer* **2022**, *24*, 563–587. https://doi.org/10.1007/s10009-021-00616-0.
67. Acheli, M.; Grigori, D.; Weidlich, M. Mining a Minimal Set of Behavioral Patterns using Incremental Evaluation. arXiv (Cornell University), 2024. https://doi.org/10.48550/arxiv.2402.02921.
68. Evermann, J.; Assadipour, G. Big data meets process mining: implementing the alpha algorithm with map-reduce. In Proceedings of the Proceedings of the 29th Annual ACM Symposium on Applied Computing. ACM, 2014, Vol. 2, pp. 1414–1416. https://doi.org/10.1145/2554850.2555076.
69. Hicham, R.; Anis, B.M. Processes meet Big Data: Scaling process discovery algorithms in Big Data environment. *Journal of King Saud University - Computer and Information Sciences* **2022**, *34*, 8478–8489. https://doi.org/10.1016/j.jksuci.2021.02.008.
70. Bratosin, C.; Sidorova, N.; van der Aalst, W. Distributed genetic process mining. In Proceedings of the IEEE Congress on Evolutionary Computation. IEEE, 2010, pp. 1–8. https://doi.org/10.1109/cec.2010.5586250.
71. Bratosin, C.; Sidorova, N.; van der Aalst, W. Distributed Genetic Process Mining Using Sampling. In Proceedings of the Parallel Computing Technologies. Springer Berlin Heidelberg, 2011, Vol. 6873, *Lecture Notes in Computer Science*, pp. 224–237. https://doi.org/10.1007/978-3-642-23178-0_20.
72. Maggi, F.M.; Di Ciccio, C.; Di Francescomarino, C.; Kala, T. Parallel algorithms for the automated discovery of declarative process models. *Information Systems* **2018**, *74*, 136–152. https://doi.org/10.1016/j.is.2017.12.002.
73. Sturm, C.; Schönig, S. Big Data Meets Process Science: Distributed Mining of MP-Declare Process Models. In Proceedings of the Enterprise Information Systems. Springer International Publishing, 2019, Vol. 363, *Lecture Notes in Business Information Processing*, pp. 396–423. https://doi.org/10.1007/978-3-030-26169-6_19.
74. Sturm, C.; Fichtner, M.; Schönig, S. Full Support for Efficiently Mining Multi-Perspective Declarative Constraints from Process Logs. *Information* **2019**, *10*, 29. https://doi.org/10.3390/info10010029.
75. Mavroudopoulos, I.; Balaktsis, C.; Varvoutas, K.; Kougka, G.; Gounaris, A.; Comuzzi, M. Declarative process mining in big data scenarios using an application-agnostic framework. *Process Science* **2025**, *2*, 6. https://doi.org/10.1007/s44311-025-00013-9.
76. Balaktsis, C.; Mavroudopoulos, I.; Comuzzi, M.; Gounaris, A.; Maggi, F.M. Discovering Comprehensive Branched Declarative Process Constraints. In Proceedings of the Business Process Management Forum. Springer Nature Switzerland, 2026, Vol. 564, *Lecture Notes in Business Information Processing*, pp. 147–164. https://doi.org/10.1007/978-3-032-02929-4_9.
77. Andersen, J.; Rathje, P.; Imenkamp, C.; Koschmider, A.; Landsiedel, O. EdgeMiner: Distributed Process Mining at the Data Sources. In Proceedings of the Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing. ACM, 2025, pp. 705–713. https://doi.org/10.1145/3672608.3707873.
78. Nucciarelli, L.; Gatta, R.; Tudor, A.M.; Tavazzi, E.; Arcuri, G.; Vallati, M.; Ibanez-Sanchez, G.; Valero-Ramon, Z.; Llatas, C.F.; Damiani, A. Towards Distributed Process Discovery in Healthcare: Testing and Proving the Feasibility of the Federated Alpha+ Algorithm. In Proceedings of the Artificial Intelligence in Medicine. Springer Nature Switzerland, 2025, Vol. 15735, *Lecture Notes in Computer Science*, pp. 294–299. https://doi.org/10.1007/978-3-031-95841-0_55.
79. Su, X.; Liu, C.; Lu, F.; Cheng, L.; Zeng, Q.; Zhang, S. EdgeIM: An Efficient Edge-Based Process Model Discovery Technique. In Proceedings of the 2025 IEEE International Conference on Web Services. IEEE, 2025, pp. 404–410. https://doi.org/10.1109/ICWS67624.2025.00057.
80. Lee, W.L.J.; Verbeek, H.; Munoz-Gama, J.; van der Aalst, W.M.; Sepúlveda, M. Recomposing conformance: Closing the circle on decomposed alignment-based conformance checking in process mining. *Information Sciences* **2018**, *466*, 55–91. https://doi.org/10.1016/j.ins.2018.07.026.

81. Bauer, M.; van der Aa, H.; Weidlich, M. Estimating Process Conformance by Trace Sampling and Result Approximation. In Proceedings of the Business Process Management. Springer International Publishing, 2019, Vol. 11675, *Lecture Notes in Computer Science*, pp. 179–197. https://doi.org/10.1007/978-3-030-26619-6_13.
82. Taymouri, F.; Carmona, J. Model and Event Log Reductions to Boost the Computation of Alignments. In Proceedings of the Data-Driven Process Discovery and Analysis. Springer International Publishing, 2018, Vol. 307, *Lecture Notes in Business Information Processing*, pp. 1–21. https://doi.org/10.1007/978-3-319-74161-1_1.
83. van der Aalst, W.M.P. Decomposing Petri nets for process mining: A generic approach. *Distributed and Parallel Databases* **2013**, *31*, 471–507. https://doi.org/10.1007/s10619-013-7127-5.
84. Munoz-Gama, J.; Carmona, J.; van der Aalst, W.M. Single-Entry Single-Exit decomposed conformance checking. *Information Systems* **2014**, *46*, 102–122. https://doi.org/10.1016/j.is.2014.04.003.
85. Munoz-Gama, J.; Carmona, J.; van der Aalst, W.M.P. Conformance Checking in the Large: Partitioning and Topology. In Proceedings of the Business Process Management. Springer Berlin Heidelberg, 2013, Vol. 8094, *Lecture Notes in Computer Science*, pp. 130–145. https://doi.org/10.1007/978-3-642-40176-3_11.
86. de Leoni, M.; Muñoz-Gama, J.; Carmona, J.; van der Aalst, W.M.P. Decomposing conformance checking on Petri nets with data. Technical Report BPM-14-06, BPMcenter.org, 2014.
87. Verbeek, H.M.W. Decomposed Replay Using Hiding and Reduction as Abstraction. In Proceedings of the Transactions on Petri Nets and Other Models of Concurrency XII. Springer Berlin Heidelberg, 2017, Vol. 10470, *Lecture Notes in Computer Science*, pp. 166–186. https://doi.org/10.1007/978-3-662-55862-1_8.
88. Lee, W.L.J.; Verbeek, H.; Munoz-Gama, J.; van der Aalst, W.M.; Sepúlveda, M. Replay using recomposition: Alignment-based conformance checking in the large. In Proceedings of the Proceedings of the BPM Demo Track and BPM Dissertation Award. CEUR-WS.org, 2017, Vol. 1920, *CEUR Workshop Proceedings*.
89. Lee, W.L.J.; Munoz-Gama, J.; Verbeek, H.M.W.; van der Aalst, W.M.P.; Sepúlveda, M. Improving Merging Conditions for Recomposing Conformance Checking. In Proceedings of the Business Process Management Workshops. Springer International Publishing, 2019, Vol. 342, *Lecture Notes in Business Information Processing*, pp. 31–43. https://doi.org/10.1007/978-3-030-11641-5_3.
90. Schwanen, C.T.; Pakusa, W.; van der Aalst, W.M.P. A Dynamic Programming Approach for Alignments on Process Trees. In Proceedings of the Process Mining Workshops. Springer Nature Switzerland, 2025, Vol. 533, *Lecture Notes in Business Information Processing*, pp. 84–97. https://doi.org/10.1007/978-3-031-82225-4_7.
91. Goulart Rocha, E.; van der Aalst, W.M.P. Polynomial-Time Conformance Checking for Process Trees. In Proceedings of the Business Process Management. Springer Nature Switzerland, 2023, Vol. 14159, *Lecture Notes in Computer Science*, pp. 109–125. https://doi.org/10.1007/978-3-031-41620-0_7.
92. Reißner, D.; Armas-Cervantes, A.; Conforti, R.; Dumas, M.; Fahland, D.; La Rosa, M. Scalable alignment of process models and event logs: An approach based on automata and S-components. *Information Systems* **2020**, *94*, 101561. https://doi.org/10.1016/j.is.2020.101561.
93. Li, T.; van Zelst, S.J. Cache Enhanced Split-Point-Based Alignment Calculation. In Proceedings of the 2022 4th International Conference on Process Mining (ICPM). IEEE, 2022, pp. 120–127. https://doi.org/10.1109/icpm57379.2022.9980633.
94. Siddiqui, A.; van der Aalst, W.M.P.; Schuster, D. Computing Alignments for Partially-Ordered Traces Through Petri Net Unfoldings. In Proceedings of the Application and Theory of Petri Nets and Concurrency. Springer Nature Switzerland, 2025, Vol. 15714, *Lecture Notes in Computer Science*, pp. 411–432. https://doi.org/10.1007/978-3-031-94634-9_20.
95. Bogdanov, E.; Cohen, I.; Gal, A. Conformance Checking for Less: Efficient Conformance Checking for Long Event Sequences. arXiv (Cornell University), 2025. https://doi.org/10.48550/arxiv.2505.21506.
96. Reißner, D.; Armas-Cervantes, A.; La Rosa, M. Efficient Conformance Checking using Alignment Computation with Tandem Repeats. arXiv, 2020. https://doi.org/10.48550/arXiv.2004.01781.
97. Shugurov, I.; Mitsyuk, A. Applying MapReduce to Conformance Checking. *Proceedings of the Institute for System Programming of the RAS* **2016**, *28*, 103–122. https://doi.org/10.15514/ispras-2016-28(3)-7.
98. Valencia-Parra, Á.; Varela-Vaca, Á.J.; Gómez-López, M.T.; Carmona, J. CC4Spark: Distributing event logs and big complex conformance checking problems. In Proceedings of the Proceedings of the Best Dissertation Award, Doctoral Consortium, and Demonstration & Resources Track at BPM 2021. CEUR-WS.org, 2021, Vol. 2973, *CEUR Workshop Proceedings*, pp. 136–140.
99. Cheng, L.; Liu, C.; Zeng, Q. Optimal alignments between large event logs and process models over distributed systems: An approach based on Petri nets. *Information Sciences* **2023**, *619*, 406–420. https://doi.org/10.1016/j.ins.2022.11.052.
100. Casas-Ramos, J.; Lama, M.; Mucientes, M. DeclareAligner: A leap towards efficient optimal alignments for declarative process model conformance checking. *Engineering Applications of Artificial Intelligence* **2025**, *160*, 111683. https://doi.org/10.1016/j.engappai.2025.111683.

101. Bergami, G.; Appleby, S.; Morgan, G. Quickening Data-Aware Conformance Checking through Temporal Algebras. *Information* **2023**, *14*, 173. https://doi.org/10.3390/info14030173.
102. Basmer, M.; Fălcuşan, R.D.; Montali, M.; Weidlich, M. Porifera: A Relational Approach to Conformance Checking for Object-Centric Behavioral Constraints. In Proceedings of the Advanced Information Systems Engineering. Springer Nature Switzerland, 2026, Vol. 16559, *Lecture Notes in Computer Science*, pp. 366–384. https://doi.org/10.1007/978-3-032-28117-3_21.
103. Andersen, J.; Rathje, P.; Landsiedel, O. Check My Flow: Distributed Conformance Checking at the Source. In Proceedings of the Business Process Management Workshops. Springer Nature Switzerland, 2025, Vol. 534, *Lecture Notes in Business Information Processing*, pp. 101–112. https://doi.org/10.1007/978-3-031-78666-2_8.
104. Suliman, A.T.; Kadadha, M.; Mizouni, R.; Otrok, H.; Damiani, E.; Al-Qutayri, M. Blockcheck: A consortium blockchain-based conformance checking framework for business processes. *Internet of Things* **2023**, *21*, 100652. https://doi.org/10.1016/j.iot.2022.100652.
105. Marin-Castro, H.M.; Morales-Sandoval, M.; González-Compean, J.L.; Hernandez, J. A novel trace-based sampling method for conformance checking. *PeerJ Computer Science* **2024**, *10*, e2601. https://doi.org/10.7717/peerj-cs.2601.
106. Bauer, M.; van der Aa, H.; Weidlich, M. Sampling and approximation techniques for efficient process conformance checking. *Information Systems* **2022**, *104*, 101666. https://doi.org/10.1016/j.is.2020.101666.
107. Awad, A.; Raun, K.; Weidlich, M. Efficient Approximate Conformance Checking Using Trie Data Structures. In Proceedings of the 2021 3rd International Conference on Process Mining (ICPM). IEEE, 2021, pp. 1–8. https://doi.org/10.1109/icpm53251.2021.9576845.
108. Fang, H.; Zhang, S.; Mei, Z. Log-Driven Conformance Checking Approximation Method Based on Machine Learning Model. *International Journal of Advanced Computer Science and Applications* **2024**, *15*. https://doi.org/10.14569/ijacsa.2024.01507138.
109. Berti, A.; van der Aalst, W. Reviving token-based replay: Increasing speed while improving diagnostics. In Proceedings of the Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2019. CEUR-WS.org, 2019, Vol. 2371, *CEUR Workshop Proceedings*, pp. 87–103.
110. Augusto, A.; Armas-Cervantes, A.; Conforti, R.; Dumas, M.; La Rosa, M. Measuring Fitness and Precision of Automatically Discovered Process Models: A Principled and Scalable Approach. *IEEE Transactions on Knowledge and Data Engineering* **2022**, *34*, 1870–1888. https://doi.org/10.1109/tkde.2020.3003258.
111. Boltenhagen, M.; Chatain, T.; Carmona, J. A Discounted Cost Function for Fast Alignments of Business Processes. In Proceedings of the Business Process Management. Springer International Publishing, 2021, Vol. 12875, *Lecture Notes in Computer Science*, pp. 252–269. https://doi.org/10.1007/978-3-030-85469-0_17.
112. Boltenhagen, M.; Chatain, T.; Carmona, J. An A*-Algorithm for Computing Discounted Anti-Alignments in Process Mining. In Proceedings of the 2021 3rd International Conference on Process Mining (ICPM). IEEE, 2021, pp. 25–31. https://doi.org/10.1109/icpm53251.2021.9576887.
113. Roider, J.; Zanca, D.; Eskofier, B.M. Efficient Training of Recurrent Neural Networks for Remaining Time Prediction in Predictive Process Monitoring. In Proceedings of the Business Process Management. Springer Nature Switzerland, 2024, Vol. 14940, *Lecture Notes in Computer Science*, pp. 238–255. https://doi.org/10.1007/978-3-031-70396-6_14.
114. Berti, A.; Aalst, W.M.P.V.D. An In-Context Foundation Model for Predictive Process Monitoring on Event Logs. *IEEE Access* **2026**, *14*, 16959–16983. https://doi.org/10.1109/access.2026.3658877.
115. Pasquadibisceglie, V.; Appice, A.; Castellano, G.; Malerba, D. DARWIN: An online deep learning approach to handle concept drifts in predictive process monitoring. *Engineering Applications of Artificial Intelligence* **2023**, *123*, 106461. https://doi.org/10.1016/j.engappai.2023.106461.
116. Ceci, M.; Spagnoletta, M.; Lanotte, P.F.; Malerba, D. Distributed Learning of Process Models for Next Activity Prediction. In Proceedings of the Proceedings of the 22nd International Database Engineering & Applications Symposium on - IDEAS 2018. ACM Press, 2018, pp. 278–282. https://doi.org/10.1145/3216122.3216125.
117. Jaramillo, J.; Arias, J. Automatic classification of event logs sequences for failure detection in WfM/BPM systems. In Proceedings of the 2019 IEEE Colombian Conference on Applications in Computational Intelligence (ColCACI). IEEE, 2019, pp. 1–6. https://doi.org/10.1109/colcaci.2019.8781973.
118. Jaramillo, J.; Arias-Londoño, J.D. Fail Detection in WfM/BPM Systems from Event Log Sequences Using HMM-Type Models. In Proceedings of the Applications of Computational Intelligence. Springer International Publishing, 2019, Vol. 1096, *Communications in Computer and Information Science*, pp. 223–234. https://doi.org/10.1007/978-3-030-36211-9_18.
119. Rico, P.; Cuadrado, F.; Duenas, J.C.; Andion, J.; Parada G., H.A. Business Process Event Prediction Through Scalable Online Learning. *IEEE Access* **2021**, *9*, 136313–136333. https://doi.org/10.1109/access.2021.3117147.
120. Rico, P.; Cuadrado, F.; Duenas, J.C. Online Model Generation for Scalable Predictive Process Monitoring. In Proceedings of the 2021 International Young Engineers Forum (YEF-ECE). IEEE, 2021, pp. 7–12. https://doi.org/10.1109/yef-ece52297.2021.9505122.

121. Tariq, Z.; Charles, D.; McClean, S.; McChesney, I.; Taylor, P. Time Efficient End-State Prediction Through Hybrid Trace Decomposition Using Process Mining. In Proceedings of the 2022 14th International Conference on Computational Intelligence and Communication Networks (CICN). IEEE, 2022, pp. 34–40. https://doi.org/10.1109/cicn56167.2022.10008305.
122. Ansari, A.; Kirchdorfer, L.; Hadian, R. On the Simplification of Neural Network Architectures for Predictive Process Monitoring. arXiv (Cornell University), 2025. https://doi.org/10.48550/arxiv.2509.17145.
123. Guo, N.; Lu, T.; Liu, C.; Xu, X.; Zeng, Q. Business Process Remaining Time Prediction Based on Bidirectional QRNN with Attention Mechanism. *Emerging Science Journal* **2026**, *10*, 627–637. https://doi.org/10.28991/esj-2026-010-02-01.
124. Asgarinejad, F.; Thomas, A.; Hildebrant, R.; Zhang, Z.; Ren, S.; Rosing, T.; Aksanli, B. Optimized Early Prediction of Business Processes with Hyperdimensional Computing. *Information* **2024**, *15*, 490. https://doi.org/10.3390/info15080490.
125. Kirchmann, H.; Fahrenkrog-Petersen, S.A.; Lu, X.; Weidlich, M. Distributional similarity between activities in certain and uncertain event data. *Process Science* **2026**, *3*, 21. https://doi.org/10.1007/s44311-026-00055-7.
126. Rauch, S.; Frey, C.M.M.; Maldonado, A.; Schuster, D.; Tavares, G.; Seidl, T. Hierarchical structuring of bilaterally expanding subtrace patterns for efficient tree-based activity suffix prediction. *Process Science* **2026**, *3*, 16. https://doi.org/10.1007/s44311-026-00050-y.
127. Bollig, B.; Függer, M.; Nowak, T.; Zeinaty, P. Promoting Simple Agents: Ensemble Methods for Event-Log Prediction. arXiv (Cornell University), 2026. https://doi.org/10.48550/arxiv.2604.21629.
128. Ceci, M.; Impedovo, A.; Pellicani, A. Leveraging Multi-target Regression for Predicting the Next Parallel Activities in Event Logs. In Proceedings of the ECML PKDD 2020 Workshops. Springer International Publishing, 2020, Vol. 1323, *Communications in Computer and Information Science*, pp. 237–248. https://doi.org/10.1007/978-3-030-65965-3_15.
129. Kunkler, M.; Rinderle-Ma, S. Probabilistic learning of processing and waiting times: a scalable approach for process event log data. *Process Science* **2026**, *3*, 3. https://doi.org/10.1007/s44311-025-00036-2.
130. Berti, A.; van der Aalst, W.M. Retrieval-Augmented In-Context Foundation Model for Predictive Process Monitoring. Preprints.org, 2026. https://doi.org/10.20944/preprints202607.0705.v1.
131. Oyamada, R.S.; Peeperkorn, J.; De Weerdt, J.; De Smedt, J. Domain Adaptation of LLMs for Process Data. arXiv (Cornell University), 2025. https://doi.org/10.48550/arxiv.2509.03161.
132. Pasquadibisceglie, V.; Capone, S.; Appice, A.; Malerba, D. Predictive Process Monitoring Through the Lens of Deep Online Learning. In Proceedings of the Foundations of Intelligent Systems. Springer Nature Switzerland, 2026, Vol. 16685, *Lecture Notes in Computer Science*, pp. 236–246. https://doi.org/10.1007/978-3-032-32643-0_20.
133. Rizzi, W.; Simonetto, L.; Di Francescomarino, C.; Ghidini, C.; Kasekamp, T.; Maggi, F.M. Nirdizati 2.0: New Features and Redesigned Backend. In Proceedings of the Proceedings of the Dissertation Award, Doctoral Consortium, and Demonstration Track at BPM 2019. CEUR-WS.org, 2019, Vol. 2420, *CEUR Workshop Proceedings*, pp. 154–158.
134. Rizzi, W.; Di Francescomarino, C.; Ghidini, C.; Maggi, F.M. Nirdizati: an advanced predictive process monitoring toolkit. *Journal of Intelligent Information Systems* **2025**, *63*, 259–291. https://doi.org/10.1007/s10844-024-00890-9.
135. Mangat, A.S.; Rinderle-Ma, S. Next-Activity Prediction for Non-stationary Processes with Unseen Data Variability. In Proceedings of the Enterprise Design, Operations, and Computing. Springer International Publishing, 2022, Vol. 13585, *Lecture Notes in Computer Science*, pp. 145–161. https://doi.org/10.1007/978-3-031-17604-3_9.
136. Chen, Q.; Winter, K.; Rinderle-Ma, S. Predicting Unseen Process Behavior Based on Context Information from Compliance Constraints. In Proceedings of the Business Process Management Forum. Springer Nature Switzerland, 2023, Vol. 490, *Lecture Notes in Business Information Processing*, pp. 127–144. https://doi.org/10.1007/978-3-031-41623-1_8.
137. Kosciuszek, T.; Hassani, M. Online Next Activity Prediction Under Concept Drifts. In Proceedings of the Advanced Information Systems Engineering Workshops. Springer Nature Switzerland, 2024, Vol. 521, *Lecture Notes in Business Information Processing*, pp. 335–346. https://doi.org/10.1007/978-3-031-61003-5_28.
138. Lee, S.; Lu, X.; Reijers, H.A. Investigating Tailored Retraining for Online Process Predictions Using Log Features. In Proceedings of the Research Challenges in Information Science. Springer Nature Switzerland, 2025, Vol. 547, *Lecture Notes in Business Information Processing*, pp. 401–417. https://doi.org/10.1007/978-3-031-92474-3_24.
139. Nur Iman, A.; Mustafa Kamal, I.; Kim, D.; Bae, H. Continual Learning Approach With Adaptive Memory Mechanism for Predictive Process Monitoring in Dynamic Business Environments. *IEEE Access* **2025**, *13*, 206767–206783. https://doi.org/10.1109/access.2025.3636194.
140. Spenrath, Y.; Hassani, M. Predicting Business Process Bottlenecks In Online Events Streams Under Concept Drifts. In Proceedings of the 34th International ECMS Conference on Modelling and Simulation, ECMS 2020; Steglich, M.; Mueller, C.; Neumann, G.; Walther, M., Eds. ECMS, 2020, Vol. 34, *Communications of the ECMS*, pp. 190–196. https://doi.org/10.7148/2020-0190.
141. Pasquadibisceglie, V.; Appice, A.; Ieva, G.; Malerba, D. TSUNAMI - an explainable PPM approach for customer churn prediction in evolving retail data environments. *Journal of Intelligent Information Systems* **2024**, *62*, 705–733. https://doi.org/10.1007/s10844-023-00838-5.

142. Sahatova, K.; Oyamada, R.S.; Lu, X.; De Smedt, J. Feature Attribution-Based Explainability Analysis of Deep Learning Models in Predictive Process Monitoring. arXiv (Cornell University), 2026. https://doi.org/10.48550/arxiv.2607.17783.
143. Branchi, S.; Buliga, A.; Di Francescomarino, C.; Ghidini, C.; Graziosi, R.; Meneghello, F.; Ronzani, M. Learning optimal policies from event logs through reinforcement learning: A comparison of deep and MDP-based approaches. *Information Systems* **2026**, *141*, 102763. https://doi.org/10.1016/j.is.2026.102763.
144. Ferreira, D.R.; Santos, R.M. Parallelization of Transition Counting for Process Mining on Multi-core CPUs and GPUs. In Proceedings of the Business Process Management Workshops. Springer International Publishing, 2017, Vol. 281, *Lecture Notes in Business Information Processing*, pp. 36–48. https://doi.org/10.1007/978-3-319-58457-7_3.
145. Berti, A.; Nghia, M.P.; van der Aalst, W.M.P. PM4Py-GPU: A High-Performance General-Purpose Library for Process Mining. In Proceedings of the Research Challenges in Information Science. Springer International Publishing, 2022, Vol. 446, *Lecture Notes in Business Information Processing*, pp. 727–734. https://doi.org/10.1007/978-3-031-05760-1_49.
146. Kundra, D.; Juneja, P.; Sureka, A. Vidushi: Parallel Implementation of Alpha Miner Algorithm and Performance Analysis on CPU and GPU Architecture. In Proceedings of the Business Process Management Workshops. Springer International Publishing, 2016, Vol. 256, *Lecture Notes in Business Information Processing*, pp. 230–241. https://doi.org/10.1007/978-3-319-42887-1_19.
147. Sahu, M.; Nayak, G.K. An efficient parallel framework for process discovery using OpenMP. *International Journal of Advanced Computer Research* **2019**, *9*, 112–123. https://doi.org/10.19101/ijacr.2018.839057.
148. Sahu, M.; Lunia, P.; Mohanty, S.N. Parallel Flexible Heuristic Miner for Process Discovery. *SN Computer Science* **2023**, *4*, 524. https://doi.org/10.1007/s42979-023-01948-1.
149. Li, G.L.; Jing, S.Y.; Guo, B.; Shen, Y. Parallel Genetic Process Mining Based on Graphics Processing Unit. *Journal of the University of Electronic Science and Technology of China* **2019**, *48*, 918–924.
150. Fauzan, H.; Sarno, R.; Saikhu, A. Accelerating real-time deterministic discovery through single instruction multiple data graphical processor unit for executing distributed event logs. *International Journal of Electrical and Computer Engineering (IJECE)* **2024**, *14*, 4214–4227. https://doi.org/10.11591/ijece.v14i4.pp4214-4227.
151. Küsters, A.; van der Aalst, W.M.P. Developing a High-Performance Process Mining Library with Java and Python Bindings in Rust. arXiv, 2024. https://doi.org/10.18154/rwth-2025-00398.
152. Li, N. On-Orbit Event-Based Conformance Checking Using GPU. *IEEE Access* **2019**, *7*, 52689–52711. https://doi.org/10.1109/access.2019.2911125.
153. Hill, S.; Fitzek, D.; Delfmann, P.; Corea, C. Inter-case Predictive Process Monitoring: A candidate for Quantum Machine Learning? arXiv (Cornell University), 2023. https://doi.org/10.48550/arxiv.2307.00080.
154. Zaman, R.; Hassani, M.; van Dongen, B.F. Conformance checking of process event streams with constraints on data retention. *Information Systems* **2023**, *117*, 102228. https://doi.org/10.1016/j.is.2023.102228.
155. Awad, A.; Weidlich, M.; Sakr, S. Process Mining over Unordered Event Streams. In Proceedings of the 2020 2nd International Conference on Process Mining (ICPM). IEEE, 2020, pp. 81–88. https://doi.org/10.1109/icpm49681.2020.00022.
156. van Zelst, S.J.; van Dongen, B.F.; van der Aalst, W.M.P. Event stream-based process discovery using abstract representations. *Knowledge and Information Systems* **2018**, *54*, 407–435. https://doi.org/10.1007/s10115-017-1060-2.
157. van Zelst, S.J.; Bolt, A.; Hassani, M.; van Dongen, B.F.; van der Aalst, W.M.P. Online conformance checking: relating event streams to process models using prefix-alignments. *International Journal of Data Science and Analytics* **2019**, *8*, 269–284. https://doi.org/10.1007/s41060-017-0078-6.
158. Burattin, A.; Sperduti, A.; van der Aalst, W.M.P. Heuristics Miners for Streaming Event Data. *arXiv (Cornell University)* **2012**. https://doi.org/10.48550/arxiv.1212.6383.
159. Leno, V.; Armas-Cervantes, A.; Dumas, M.; La Rosa, M.; Maggi, F.M. Discovering process maps from event streams. In Proceedings of the Proceedings of the 2018 International Conference on Software and System Process. ACM, 2018, pp. 86–95. https://doi.org/10.1145/3202710.3203154.
160. Redlich, D.; Molka, T.; Gilani, W.; Blair, G.; Rashid, A. Dynamic Constructs Competition Miner - Occurrence- vs. Time-Based Ageing. In Proceedings of the Data-Driven Process Discovery and Analysis. Springer International Publishing, 2015, Vol. 237, *Lecture Notes in Business Information Processing*, pp. 79–106. https://doi.org/10.1007/978-3-319-27243-6_4.
161. Ahammad, A.; Bouchal, A.; Radgui, M. Mini-Process Cubes: Enabling Real-Time Analysis and Process Discovery in Online Process Mining. In Proceedings of the Progress in Intelligent Computing and Secure Communication Systems. Springer Nature Switzerland, 2025, Vol. 1555, *Lecture Notes in Networks and Systems*, pp. 80–89. https://doi.org/10.1007/978-3-031-99997-0_8.
162. Quakulinski, L.; Judel, S.; Wagner, M.; Schroeder, U. Anwendung von Process Mining zur kontinuierlichen Lernpfadidentifikation in Lernmanagementsystemen. In Proceedings of the 21. Fachtagung Bildungstechnologien (DELFI). Gesellschaft für Informatik e.V., 2023, Vol. 338, *Lecture Notes in Informatics*, pp. 213–218. https://doi.org/10.18420/delfi2023-34.

163. Schuster, D.; van Zelst, S.J.; van der Aalst, W.M.P. Cortado-An Interactive Tool for Data-Driven Process Discovery and Modeling. In Proceedings of the Application and Theory of Petri Nets and Concurrency. Springer International Publishing, 2021, Vol. 12734, *Lecture Notes in Computer Science*, pp. 465–475. https://doi.org/10.1007/978-3-030-76983-3_23.
164. De Carolis, B.; Ferilli, S.; Mallardi, G. Learning and Recognizing Routines and Activities in SOFiA. In Proceedings of the Ambient Intelligence. Springer International Publishing, 2014, Vol. 8850, *Lecture Notes in Computer Science*, pp. 191–204. https://doi.org/10.1007/978-3-319-14112-1_16.
165. Schuster, D.; Kolhof, G.J. Scalable Online Conformance Checking Using Incremental Prefix-Alignment Computation. In Proceedings of the Service-Oriented Computing - ICSOC 2020 Workshops. Springer International Publishing, 2021, Vol. 12632, *Lecture Notes in Computer Science*, pp. 379–394. https://doi.org/10.1007/978-3-030-76352-7_36.
166. Zaman, R.; Hassani, M.; van Dongen, B.F. Efficient memory utilization in conformance checking of process event streams. In Proceedings of the Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing. ACM, 2022, pp. 437–440. https://doi.org/10.1145/3477314.3507217.
167. Waspada, I.; Sarno, R.; Astuti, E.S.; Prasetyo, H.N.; Budiraharjo, R. Graph-Based Token Replay for Online Conformance Checking. *IEEE Access* **2022**, *10*, 102737–102752. https://doi.org/10.1109/access.2022.3208098.
168. Raun, K.; Nielsen, M.; Burattin, A.; Awad, A. C-3PA: Streaming Conformance, Confidence and Completeness in Prefix-Alignments. In Proceedings of the Advanced Information Systems Engineering. Springer Nature Switzerland, 2023, Vol. 13901, *Lecture Notes in Computer Science*, pp. 437–453. https://doi.org/10.1007/978-3-031-34560-9_26.
169. Lee, W.L.J.; Burattin, A.; Munoz-Gama, J.; Sepúlveda, M. Orientation and conformance: A HMM-based approach to online conformance checking. *Information Systems* **2021**, *102*, 101674. https://doi.org/10.1016/j.is.2020.101674.
170. Chapela-Campa, D.; Dumas, M. Efficient Online Computation of Business Process State From Trace Prefixes via N-Gram Indexing. *IEEE Transactions on Services Computing* **2025**, *18*, 770–783. https://doi.org/10.1109/tsc.2025.3547235.
171. Raun, K.; Tommasini, R.; Awad, A. Adaptive Handling of Out-of-order Streams in Conformance Checking. In Proceedings of the Proceedings of the 26th International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data (DOLAP 2024). CEUR-WS.org, 2024, Vol. 3653, *CEUR Workshop Proceedings*, pp. 9–17.
172. Raun, K.; Tommasini, R.; Awad, A. Back to the Order: Partial orders in streaming conformance checking. *Information Systems* **2025**, *133*, 102566. https://doi.org/10.1016/j.is.2025.102566.
173. Kalukapuge, S.; ter Hofstede, A.H.M.; Wynn, M.T. SwiftMend: An Approach to Detect and Repair Activity Label Quality Issues in Process Event Streams. In Proceedings of the Cooperative Information Systems. Springer Nature Switzerland, 2025, Vol. 15506, *Lecture Notes in Computer Science*, pp. 131–149. https://doi.org/10.1007/978-3-031-81375-7_8.
174. Hassani, M. Concept Drift Detection Of Event Streams Using An Adaptive Window. In Proceedings of the 33rd International ECMS Conference on Modelling and Simulation, ECMS 2019; Iacono, M.; Palmieri, F.; Gribaudo, M.; Ficco, M., Eds. ECMS, 2019, Vol. 33, *Communications of the ECMS*, pp. 230–239. https://doi.org/10.7148/2019-0230.
175. Imenkamp, C.; Kabierski, M.; Reiter, H.; Weidlich, M.; Hasselbring, W.; Koschmider, A. Determining Window Sizes Using Species Estimation for Accurate Process Mining over Streams. In Proceedings of the Advanced Information Systems Engineering. Springer Nature Switzerland, 2025, Vol. 15701, *Lecture Notes in Computer Science*, pp. 109–124. https://doi.org/10.1007/978-3-031-94569-4_7.
176. Liu, N.; Huang, J.; Cui, L. A Framework for Online Process Concept Drift Detection from Event Streams. In Proceedings of the 2018 IEEE International Conference on Services Computing (SCC). IEEE, 2018, pp. 105–112. https://doi.org/10.1109/scc.2018.00021.
177. Ostovar, A.; Leemans, S.J.J.; Rosa, M.L. Robust Drift Characterization from Event Streams of Business Processes. *ACM Transactions on Knowledge Discovery from Data* **2020**, *14*, 1–57. https://doi.org/10.1145/3375398.
178. Guo, N.; Liu, C.; Mo, Q.; Cao, J.; Ouyang, C.; Lu, X.; Zeng, Q. Business Process Remaining Time Prediction Based on Incremental Event Logs. *IEEE Transactions on Services Computing* **2025**, *18*, 1308–1320. https://doi.org/10.1109/tsc.2025.3562338.
179. Hassani, M.; Verbeek, T.; van Straten, S. Chameleons Do Not Forget: Prompt-Based Online Continual Learning for Next Activity Prediction. *International Journal of Cooperative Information Systems* **2026**, *35*, 2650003. https://doi.org/10.1142/s0218843026500036.
180. Evermann, J.; Rehse, J.R.; Fettke, P. Process Discovery from Event Stream Data in the Cloud - A Scalable, Distributed Implementation of the Flexible Heuristics Miner on the Amazon Kinesis Cloud Infrastructure. In Proceedings of the 2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). IEEE, 2016, pp. 645–652. https://doi.org/10.1109/cloudcom.2016.0111.
181. Loreti, D.; Chesani, F.; Ciampolini, A.; Mello, P. A distributed approach to compliance monitoring of business process event streams. *Future Generation Computer Systems* **2018**, *82*, 104–118. https://doi.org/10.1016/j.future.2017.12.043.
182. Burattin, A. Beamline: A comprehensive toolkit for research and development of streaming process mining. *Software Impacts* **2023**, *17*, 100551. https://doi.org/10.1016/j.simpa.2023.100551.
183. Ceravolo, P.; Tavares, G.M.; Junior, S.B.; Damiani, E. Evaluation Goals for Online Process Mining: A Concept Drift Perspective. *IEEE Transactions on Services Computing* **2022**, *15*, 2473–2489. https://doi.org/10.1109/tsc.2020.3004532.

184. Adams, J.N.; Pitsch, C.; Brockhoff, T.; van der Aalst, W.M.P. An Experimental Evaluation of Process Concept Drift Detection. *Proceedings of the VLDB Endowment* **2023**, *16*, 1856–1869. https://doi.org/10.14778/3594512.3594517.
185. van Zelst, S.J.; van Dongen, B.; van der Aalst, W. Know what you stream : generating event streams from CPN Models in ProM 6. In Proceedings of the Proceedings of the BPM Demo Session 2015. CEUR-WS.org, 2015, Vol. 1418, *CEUR Workshop Proceedings*, pp. 85–89.
186. Burattin, A. PLG2: Multiperspective process randomization with online and offline simulations. In Proceedings of the Proceedings of the BPM Demo Track 2016. CEUR-WS.org, 2016, Vol. 1789, *CEUR Workshop Proceedings*, pp. 1–6.
187. Imenkamp, C.; Maldonado, A.; Reiter, H.; Werner, M.; Hasselbring, W.; Koschmider, A.; Burattin, A. AVOCADO: The Streaming Process Mining Challenge. In Proceedings of the Proceedings of Modellierung 2026. Gesellschaft für Informatik e.V., 2026, Vol. P-378, *Lecture Notes in Informatics*, pp. 107–121. https://doi.org/10.18420/modellierung2026-07.