TOWARDS AUTOMATIC VERIFICATION OF SEMANTIC BUSINESS PROCESS MODEL CONSISTENCY: LITERATURE REVIEW AND HYBRID NEURO-SYMBOLIC APPROACH

Oleksandr Rudskyi[1]

DOI: https://doi.org/10.30525/978-9934-26-715-4-52

Abstract. The subject of this research is the automated verification of consistency between natural-language descriptions of business processes and their formal models expressed in the Business Process Model and Notation (BPMN 2.0). Organizations routinely maintain both representations of every documented process: a formal model with precise execution semantics, and a narrative embedded in regulations, standard operating procedures and technical specifications. Because the two artifacts are authored by different people at different times and revised independently, they diverge structurally rather than accidentally. In regulated sectors such as banking, insurance, healthcare and public administration, such divergence carries direct regulatory consequence, yet reconciliation remains a manual and unreliable activity. *The purpose* of the research is to determine whether large language models, conformance checking and neuro-symbolic architectures can be composed into a method that verifies the consistency of a textual process description with its BPMN model while retaining formal guarantees, and to establish by implementation whether such a composition is practically realizable. *Methodology* of the study combines a structured literature review with design-science construction and empirical evaluation. Fifty-eight publications, retrieved through keyword searches across five bibliographic databases covering January 2021 to July 2026 and extended to foundational earlier works, were organized into nine thematic areas spanning BPMN formalization, declarative constraint languages, text-to-model extraction, language models in business process management, conformance checking, neuro-symbolic artificial intelligence, multi-agent

[1] Postgraduate Student at the Department of Software Engineering and Management Intelligent Technologies,
National Technical University "Kharkiv Polytechnic Institute", Ukraine

© Oleksandr Rudskyi

orchestration, inconsistency detection, and evaluation resources. *Results* of the study include a four-module pipeline, derived from the identified gaps and implemented so as to separate semantic interpretation performed by language-model agents from verification performed by deterministic checkers over execution traces and lane assignments. Evaluation used annotated reference data for component accuracy and mutation-based testing for end-to-end detection, compared against a baseline omitting formal verification. *Practical implications.* It is concluded that the components required for automated text-model verification are individually mature but have not been combined, and that the literature has bifurcated into generation of models from text and generation of verification properties from models, leaving their composition unaddressed. Five research gaps were identified, three of which the proposed approach addresses, one partially, and one of which remains open. *Value/originality.* The prototype demonstrates that the composition is realizable, achieving F1-scores of 85.7% for semantic mapping, 83.3% for constraint extraction and 92.3% for resource extraction on a real process. A single unimplemented mechanism – the evaluation of guard conditions against the branch conditions of the model – rather than the architecture, was established as the factor limiting end-to-end performance, which directs subsequent development.

INTRODUCTION

Organizations that operate documented processes maintain two descriptions of each of them. One is a formal model in the Business Process Model and Notation, with execution semantics precise enough to support simulation and automated analysis. The other is a narrative in a regulation, a standard operating procedure, or a technical specification, written for people rather than for machines. The two are assumed to describe the same process. They are produced by different people at different times, revised independently, and reconciled – when they are reconciled at all – by manual review. In regulated sectors this reconciliation is not optional: auditors require demonstrable agreement between what is documented and what is executed, and discrepancies carry regulatory consequence. Yet no automated method establishes, on parsed BPMN 2.0 and with formal guarantees, whether a given narrative and a given model describe the same behavior.

Novelty and relevance. The absence is not for want of relevant technology. Three research communities have independently produced the components such a method would require: natural language processing, which interprets process narratives and has been transformed since 2023 by large language models; process mining, which decides whether observed behavior conforms to a model and returns a concrete witness when it does not; and formal methods, which decide temporal properties over a model's execution semantics. Each line has reached the maturity required for integration; none has been integrated with the others for this purpose. The scientific interest of the problem lies precisely there: it is not a question of inventing a missing capability but of composing existing ones across a boundary that has not been crossed, and of determining whether the interface between a component that interprets and a component that proves can be made narrow enough to be trustworthy.

Aim and tasks. The aim of this chapter is to determine whether large language models, conformance checking, and neuro-symbolic architectures can be composed into a method that verifies the consistency of a textual process description with its BPMN 2.0 model while retaining formal guarantees, and to establish by implementation whether such a composition is practically realizable. Six tasks follow from this aim:

1) to review the state of the art across the research areas bearing on the problem, and to characterize what each contributes and what each lacks;

2) to identify the research gaps that separate the existing literature from a working verification method;

3) to formulate the principle governing the division of labor between neural and symbolic components, and to identify a formalism suitable as the interface between them;

4) to design a pipeline architecture realizing that division of labor;

5) to evaluate the resulting prototype on a real business process using mutation-based testing against a baseline that omits formal verification;

6) to determine which component limits end-to-end performance, and thereby to direct subsequent work.

Methodology. The literature was collected through structured keyword searches in Google Scholar, Semantic Scholar, DBLP, ACL Anthology and arXiv, covering January 2021 to July 2026 and extended to earlier foundational works where conceptual coherence required it. Query families

combined the terms “BPMN”, “business process”, “natural language”, “consistency”, “conformance”, “large language model”, “neuro-symbolic”, “DECLARE” and “multi-agent” with related variants. Priority was given to Q1 and Q2 journals, to the principal conferences in business process management and conceptual modeling, and to leading venues in artificial intelligence and computational linguistics; workshop papers and preprints were admitted where they introduced influential methods or constituted the only accessible form of recent work. Fifty-eight publications passed the relevance filter and are organized below into nine thematic areas, presented in seven sections since several areas were merged to avoid repetition. The constructive part of the work follows a design-science logic: an architecture is derived from the gaps identified in the review, implemented, and evaluated empirically. Evaluation combines annotated reference data for component accuracy with mutation-based testing for end-to-end detection, compared against a baseline that submits the same inputs to a language model without formal verification.

Chapter structure. Sections 1 and 2 state the problem and review the formal foundations on which any solution rests. Sections 3 to 7 survey the five bodies of work bearing on it: extraction of models from text, conformance checking and the formal verification of BPMN, neuro-symbolic integration and multi-agent orchestration, prior work on text-model inconsistency, and the datasets and evaluation methodologies available. Section 8 consolidates the review into five research gaps, and Section 9 describes the prototype architecture and reports its evaluation. The chapter closes with conclusions and directions for further work.

1. Text-Model Divergence Problem

1.1. Two representations, one process

Contemporary organizations document their business processes through two complementary but heterogeneous means. On one side, business analysts produce formal graphical models in the Business Process Model and Notation, an Object Management Group standard with precisely defined execution semantics covering activities, gateways, parallel flows, events, and actor roles [1]. On the other, the very same processes are described in natural-language narratives embedded in internal regulations, standard

operating procedures, user manuals, and technical specifications. The two artifacts are expected to agree.

In practice such agreement is rarely guaranteed. The two representations are authored by different people at different times; updates propagate asymmetrically; and informal human review is unreliable at catching subtle divergence. The drift is therefore structural rather than accidental – a consequence of how process documentation is produced, not of carelessness in any particular organization. In regulated domains such as banking, insurance, healthcare, and public administration, discrepancies between documented and executable processes translate directly into regulatory risk. In software engineering, divergence between a process specification and its implementation model produces defects that are expensive to trace to their source. Auditors and regulators increasingly demand demonstrable consistency between textual documentation and process models, but supply neither methodology nor tooling for establishing it automatically.

The difficulty is not a shortage of techniques for either representation considered separately. What is absent is any method that treats the two representations as two views of one process which must be checked against each other. Closing that absence requires a component capable of reading a narrative and a component capable of proving a property, and the central question of this chapter is whether the two can be composed.

1.2. Key Concepts and Terminology

BPMN 2.0 provides a graphical language for representing business processes. Its token-flow execution semantics covers activities, exclusive, parallel, inclusive and event-based gateways, catching and throwing events, pools and swimlanes, and message flows. Formal analysis of BPMN typically proceeds by mapping it into Petri nets [2] or labeled transition systems, which makes classical model-checking techniques applicable.

DECLARE is a declarative process modeling language introduced by Pesic and van der Aalst [3] that characterizes a process through temporal constraints rather than an explicit control flow. Each constraint instantiates a template with concrete activities. Response(A, B) requires that every occurrence of A be followed eventually by B. NotCoExistence (A, B) requires that A and B never occur in the same execution. The semantics of DECLARE is grounded in linear temporal logic over finite traces [4], which

renders constraints directly verifiable by finite-state methods. Writing the Response template explicitly, for a trace σ:

$$\forall\ i\colon \sigma_i = A \rightarrow \exists\ j > i\colon \sigma_j = B. \quad (1)$$

This declarative character is what makes DECLARE suitable as a target formalism for information extracted from text. A narrative rarely specifies a complete control flow; it states obligations, orderings and exclusions, which is precisely what a constraint template expresses.

Conformance checking verifies whether behavior recorded in an event log conforms to behavior prescribed by a model [5], [6]. It generalizes to the present setting with one substitution: the reference behavior is derived not from a log of past executions but from a textual narrative.

Large language models are transformer-based generative models that exhibit strong performance on natural-language understanding. Their characteristic weaknesses – hallucination, output inconsistency, and the absence of formal guarantees – motivate the neuro-symbolic paradigm [7], [8], which pairs neural components for the perception of unstructured input with symbolic components for guaranteed reasoning. The problem stated above has the shape this paradigm was designed for: one half of the task is irreducibly semantic and tolerant of approximation, the other is formal and admits none.

2. Formal Foundations: BPMN and DECLARE

Any approach to text-BPMN verification rests on two pillars: a formal semantics for BPMN, and a verifiable constraint language into which meaning extracted from text can be cast. This section reviews the works that established both, since the feasibility of the entire enterprise depends on them.

2.1. Declarative Constraints and DECLARE

Pesic and van der Aalst [3] introduced the ConDec language in 2006 to accommodate loosely structured, knowledge-intensive processes for which prescribing a complete control flow is neither possible nor desirable. Subsequent work delivered the first complete DECLARE engine [9] and an efficient automata-based mechanism for evaluating constraints over finite traces [4], the latter a cornerstone for any verifier that must check derived constraints against enumerated executions.

Di Ciccio and Mecella [10] approached the same formalism from the opposite direction with MINERful, a scalable declarative process miner that discovers constraints from event logs. Its role here is complementary: constraints discovered from logs provide an independent reference against which constraints extracted from text could in principle be cross-validated, a possibility examined as Gap 2 in Section 8.

Two properties of DECLARE matter for what follows. First, the template inventory is finite and small. Second, inventories differ across the literature: Fionda et al. [11] enumerate roughly twenty templates organized into four classes – existence, choice, relation, and negation – while other treatments use narrower sets. The prototype in Section 9 employs a subset of ten, selected so that each violation admits a finite witnessing trace.

The alternative – having a language model emit temporal-logic formulae directly – is expressively more general but brittle: a formula is syntactically valid under a wide range of operator substitutions, so a single misplaced operator yields a well-formed constraint with entirely different meaning and no local signal that anything is wrong. Selection from a catalog converts an unbounded generation problem into a bounded classification one, at the cost of expressiveness. This trade-off underlies the design decision described in Section 9.

2.2. BPMN Semantics and Formalization

On the model side, Dijkman, Dumas and Ouyang [2] provided the canonical mapping from BPMN to Petri nets, establishing that BPMN admits classical reachability and liveness analysis. Their contribution is what licenses treating a BPMN diagram as a formal object rather than as documentation: once mapped, a model has a well-defined set of executions, and any property expressible over those executions becomes decidable. The textbook by Weske [12] fixed the terminology of the business process management life cycle used throughout the community and adopted here.

Taken together, these contributions establish the precondition for everything that follows: a BPMN model determines a set of admissible executions, and a DECLARE constraint determines a predicate over them. What remains missing is the translation from a narrative into such predicates – a linguistic problem rather than a formal one, and the subject of Sections 3 and 6.

3. LLM-Based Process Model Extraction

3.1. From Rule-Based to Neural Extraction

The earliest systematic attempt to derive process models from textual descriptions was that of Friedrich, Mendling and Puhlmann [13], who assembled a deterministic linguistic pipeline combining anaphora resolution, role extraction and heuristic gateway insertion. Evaluated on 47 text-model pairs drawn from industry and textbooks, their technique generated on average 77% of the models correctly. That result established the feasibility of the task, and the pipeline remained the dominant paradigm for close to a decade; it is still used as a baseline. The inverse problem – generating natural-language descriptions from models [14] – delimits, by specifying what a faithful verbalization must preserve, the semantic correspondence a consistency check should test.

A wave of learning-based extractors followed, culminating in demonstrations that in-context learning with general-purpose language models matches or surpasses rule-based extractors on annotated benchmarks [15], and in a universal prompting strategy achieving state-of-the-art results on process-model information extraction [16]. The systematic review by Van Woensel and Motie [17] provides a meta-level taxonomy of extraction methods and confirms the field's decisive shift toward learning-based approaches, while documenting a scarcity of gold-standard annotated datasets that constrains objective evaluation – a limitation revisited in Section 7.

3.2. Large Language Models in BPM

The adoption of large language models across business process management began with the position paper of Vidgof, Bachhofner and Mendling [18], which mapped opportunities and challenges across the BPM life cycle. Grohs et al. [19] supplied the first substantial empirical demonstration that a general-purpose language model can match or exceed hand-engineered baselines on several BPM tasks, including the extraction of declarative process models from textual descriptions – an important precedent for the constraint-extraction problem treated in this chapter, and the source of annotated data later reused in benchmark work. Busch et al. [20] studied prompt-engineering strategies in BPM systematically, providing a methodological foundation for later multi-agent systems.

Kourani et al. [21] introduced sandboxed code generation for the synthesis of partially ordered workflow models and benchmarked sixteen language models across twenty real-world processes, with evaluation grounded in process mining through simulated event logs and conformance measures; the PM-LLM-Benchmark [22] and a life-cycle survey [24] extend evaluation across the spectrum of process-mining tasks. Rebmann et al. [23] evaluated language models on semantics-aware mining tasks such as semantic anomaly detection – the nearest neighbor to inconsistency detection among those tasks – supplying fine-tuned open-weight baselines.

The most recent contributions concentrate on generation from text at scale. Matei et al. [25] present a multi-stage pipeline that both constructs ground truth and reconstructs models: multilingual BPMN files are translated, validated through execution-oriented compliance checks, and iteratively repaired, after which descriptions are generated from validated models and used to reconstruct executable BPMN. From 750 public diagrams they obtained 387 validated ground-truth models with average reconstruction similarity above 0.75.

A consistent message emerges from this cluster. Language models are effective at the semantic interpretation of process-related text, but they provide no formal guarantees. Several authors state explicitly that hallucination and output variability make standalone deployment unsuitable in compliance-critical settings. This observation is what motivates coupling such models with symbolic verification rather than replacing one with the other, and it is examined in Section 5.

3.3. Generation Versus Verification

A structural feature of this body of work deserves emphasis, because the argument of the chapter depends on it. Nearly every contribution surveyed above addresses the same direction of travel: given a text, produce a model. The inverse situation – a model already exists, a description already exists, and the question is whether they agree – receives almost no attention.

The asymmetry is not merely one of volume. Generation and verification differ in what counts as success. A generator is evaluated by similarity to a reference model, and partial credit is meaningful: a diagram that captures most of a process is useful. A verifier must return a verdict, and a verdict that is usually right is of limited value in a compliance setting, where the

purpose of the exercise is precisely to catch the discrepancy that human review missed. Metrics developed for generation – graph edit distance, embedding similarity, conformance-based fitness – do not transfer to a task whose output is a claim about agreement together with evidence for it.

Nor is the asymmetry closing. Recent work has extended in a second direction rather than toward verification of consistency: the derivation of verification properties from BPMN models, in which a language model produces temporal-logic formulae from a diagram for subsequent model checking. That line, examined in Section 5, shares with the present work an interest in formal guarantees, but it takes the model as its source of truth and never consults a narrative.

The literature has therefore bifurcated. One branch translates text into models; the other translates models into verification properties. Both are active, both are maturing, and both have produced results that a consistency verifier would need. What neither branch addresses is their composition – extracting behavioral constraints from a narrative and checking them against a model that was authored independently. This framing is more precise than the claim that no work exists on text-model consistency, which would be inaccurate: the rule-based approaches examined in Section 6 addressed exactly that problem, but on simplified representations and before language models were available. The accurate statement is that the two capabilities required for a modern solution now exist separately and have not been joined.

4. Conformance Checking and Formal Verification

4.1. Conformance Checking Foundations

Conformance checking is the sub-discipline of process mining that quantifies the discrepancy between modelled and observed behavior. The foundational treatment by van der Aalst [5] and the dedicated monograph by Carmona et al. [6] set out the field comprehensively. Adriansyah, van Dongen and van der Aalst [26] introduced cost-optimal alignment as the canonical algorithmic primitive for comparing a log with a model: rather than reporting a binary verdict, an alignment identifies the cheapest sequence of moves reconciling an observed trace with an executable path, so that the location and magnitude of a deviation are recovered along with its existence. Dunzer et al. [27] surveyed 37 conformance-checking methods

and organized them along algorithmic, model-language and quality-metric axes, providing the taxonomy within which any new contribution in the area must situate itself.

Two features of this literature bear directly on the problem of this chapter. The first is the notion of a counter-example. Alignment-based conformance checking does not merely assert that a log fails to fit a model; it produces a concrete object exhibiting the failure. This is what distinguishes formal verification from statistical assessment, and it is the property a text-model verifier must preserve if its output is to be actionable. An analyst who is told only that a description and a model disagree can do nothing; an analyst shown the execution path on which they disagree can decide whether to correct the model, correct the text, or record the divergence as intentional.

The second is the nature of the reference. Across the whole of this literature the reference behavior arrives as an event log – a record of what actually happened. The model is checked against evidence of execution. This assumption is so pervasive that it is rarely stated, yet it is exactly the assumption that fails in the present setting, where no executions exist and the reference is a narrative describing what should happen. Extending the paradigm therefore requires a translation layer that converts free-form text into artifacts a conformance checker can consume. That requirement defines the role a language model can play, and the constraint templates of Section 2 define the target formalism into which the translation must land.

4.2. Formal Verification of BPMN Models

Alongside log-based conformance checking, a parallel line applies model checking to BPMN directly. Corradini et al. [28] developed the BProVe framework, which supplies an operational semantics for BPMN 2.0 in terms of labeled transition systems – bypassing the Petri-net intermediate step discussed in Section 2 – and integrates with rewriting-logic and statistical model-checking tools to support verification of temporal properties over collaboration models. Their work establishes that properties of realistic BPMN collaborations, including those involving multiple pools and message exchange, are mechanically decidable.

A recent contribution begins to soften the boundary between the two paradigms. Donadello et al. [29] introduced conformance checking of fuzzy logs against declarative temporal specifications, implementing the

constraint semantics as a differentiable soft-logic layer. Their construction makes symbolic constraints usable within a neural training loop, and it is a conceptual precedent for neuro-symbolic integration in this domain. It remains, however, a log-based technique: the uncertainty it accommodates is uncertainty about what was executed, not uncertainty about what a document means.

Taken together, the two lines reviewed here supply the second of the two components a text-BPMN verifier requires. Section 3 established that language models can interpret process narratives but offer no guarantees; this section establishes that symbolic methods offer guarantees but cannot read. Neither deficiency is remediable within its own paradigm – a more capable language model does not acquire the ability to prove, and a more capable model checker does not acquire the ability to read. The remaining question is architectural: whether the two can be composed such that each operates only where it is reliable, and whether the interface between them can be made narrow enough to be trustworthy. That question is the subject of Section 5, and the answer proposed in this chapter is that a finite catalog of constraint templates, instantiated over a vocabulary fixed by the model itself, is such an interface.

5. Neuro-Symbolic and Multi-Agent Approaches

5.1. Neuro-symbolic Integration in BPM

The integration of neural perception with symbolic reasoning has matured from a programmatic ambition into a working engineering practice. Manhaeve et al. [7] demonstrated with DeepProbLog that neural predicates can be embedded directly within probabilistic logic programming, so that a learned component supplies uncertain evidence while a logical engine retains responsibility for inference. The position survey by d'Avila Garcez and Lamb [8] framed this convergence as the "third wave" of artificial intelligence. The unifying principle is a division of labor rather than a blending of mechanisms: neural components interpret unstructured inputs; symbolic components guarantee the correctness of what follows from those interpretations.

Within business process management this principle has been taken up along several independent lines. Duranti et al. [30] proposed a framework for trustworthy knowledge extraction that couples large language models with

symbolic verification modules. In their process-mining setting, a language model generates linear temporal logic specifications from sets of finite traces representing event logs, and the resulting specifications are passed to model checkers for linear temporal logic on finite traces; in a parallel knowledge-representation setting, the same architecture extracts description-logic concepts separating positively and negatively labeled instances. Their work is the closest methodological precedent for the architecture developed in Section 9, since it establishes that a language model can be confined to the role of proposing formal artifacts while a deterministic component retains the authority to accept or reject them.

Related work applies soft LTLf formulations and constraint-guided counterfactuals to predictive monitoring [31; 32], demonstrating that declarative constraints can act as an effective inductive bias on neural models, but remains oriented toward predicting or explaining future behavior rather than verifying a specification that already exists.

A more recent line addresses formal verification of BPMN directly. Tang et al. [33] present LLaMA-PG, a two-phase framework for generating verification properties from BPMN models. In the first phase, model-checking knowledge is injected into a base language model through continued pre-training on a curated formal-verification corpus; in the second, step-by-step instruction fine-tuning decomposes the translation into attentional filtering, causal inference, property formulation, and symbolic translation, producing explicit intermediate reasoning traces rather than an opaque single-step mapping. The authors motivate this decomposition by the specific difficulty of BPMN, which mixes executable control-flow semantics with diagrammatic markup and encodes business constraints only incompletely in its XML structure. A related contribution from the same research group [34] organizes BPMN modeling as a hierarchy of agents executing a "generate-check-correct" cycle with retrieval from a knowledge base and invocation of external checking tools.

LLaMA-PG is the nearest neighbor to the approach developed here, and the distinction is worth stating precisely. It derives temporal-logic properties from the BPMN model itself: no natural-language description appears among its inputs, and its output is a property set for subsequent model checking rather than a verdict on the agreement between two representations of the same process. The direction of translation is model-to-property; the

direction pursued here is text-to-constraints, checked against a model that already exists. The two are complementary rather than competing, and together they instantiate the bifurcation identified in Section 3.

5.2. Multi-agent LLM Orchestration

The second relevant development is the operationalization of language models as cooperating agents. Yao et al. [35] introduced ReAct, which interleaves reasoning traces with tool-calling actions, grounding a claim against an external artifact rather than judging it from the model's own parameters. Shinn et al. [36] extended this with Reflexion, adding a self-evaluation loop that transfers directly to a verifier capable of revising its own extracted constraints.

System-level frameworks followed. AutoGen [37] provides an open platform for composing conversable agents, and MetaGPT [38] encodes standard operating procedures into a multi-agent assembly line with cross-agent validation. ChatDev [39] contributes the pattern of communicative dehallucination, in which paired agents cross-check one another's outputs to reduce error rates – an idea of evident relevance to verification tasks, where false positives and false negatives are both costly. The survey by Guo et al. [40] catalogs the rapid expansion of the field.

The transfer of this paradigm into business process management is recent and narrow. MAO [41] represents the first sustained attempt to orchestrate multiple language-model agents for BPMN generation, structuring the task into generation, refinement, reviewing, and testing phases, with the reviewing phase explicitly targeting semantic hallucination and the testing phase targeting format errors through external tools. Its architecture – typed communication between agents combined with tool integration – is a direct prior for any multi-agent text-BPMN verifier.

Two observations follow from this cluster. First, multi-agent decomposition has been empirically validated as a means of reducing language-model error rates, and the mechanism responsible – subjecting each agent's output to an independent check – is precisely the mechanism a verification pipeline requires. Second, within business process management the paradigm has so far been applied only to generation. No published system combines typed multi-agent orchestration with symbolic verification for the purpose of establishing whether a process model agrees with its textual description.

6. Detecting Text-Model Inconsistencies

6.1. Rule-Based Inconsistency Detection

The problem addressed by this chapter was posed, and partially solved, a decade ago. Van der Aa, Leopold and Reijers [42] presented the first approach to identifying inconsistencies between a process model and a corresponding textual description automatically. Their technique aligns activities in the model with sentences in the narrative using linguistic analysis, and detects two classes of divergence: activities described in an order that the model does not permit, and model activities absent from the text altogether. The journal extension [43] refined the technique and established the evaluation setting that subsequent work has largely inherited. The doctoral thesis of van der Aa [44] consolidates five alignment and comparison techniques into a single treatment and remains the most systematic account of the problem available.

Two follow-up contributions extended the scope in directions that remain relevant. The first introduced behavioral spaces [45] to accommodate the ambiguity inherent in natural language: rather than committing to one reading of an underspecified sentence, the technique represents the set of behaviors compatible with all admissible readings, and reports an inconsistency only when no reading reconciles text with model. This distinguishes genuine divergence from an artifact of interpretation. The same difficulty arises in a language-model-based setting in a different form, and Section 9 returns to it: a system that commits to whichever reading the model produced may report a divergence that is an artifact of interpretation rather than a defect of either artifact. The second contribution applied uniform representations to align textual and model-based process descriptions [46].

The most direct precursor of the present work is van der Aa, Di Ciccio, Leopold and Reijers [47], which presented the first automated extractor of DECLARE constraints from natural-language text. On a curated corpus the approach achieved a precision of 0.77 and a recall of 0.72, for an F1-score of 0.74. This result serves two purposes here. It establishes that the translation from narrative to declarative constraints is feasible at useful accuracy without any learned language component, and it supplies the reference point against which any language-model-based extractor should be assessed.

The limitations of this cluster are, however, systematic. All of these techniques operate on simplified representations of the model side – activity lists, control-flow graphs, behavioral spaces – rather than on parsed BPMN 2.0 with its full inventory of gateway types, events, pools and message flows. All rely on hand-engineered linguistic rules whose coverage is bounded by the phenomena their authors anticipated, and all predate general-purpose language models by several years. The problem was therefore not abandoned as solved: work on it lapsed where the available linguistic technology had been exhausted, and the technology that would have relaxed that bound arrived afterwards. Recent work on ambiguity detection and elimination in automated executable process modeling [48] represents the beginning of a resumption on the new technological basis.

6.2. Matching Text to Model Elements

Any verifier must first establish which fragment of text corresponds to which element of the model, and this sub-problem has its own literature. The arrival of sentence embeddings, particularly Sentence-BERT [49], raised the baseline expectation for semantic matching across the field by making cross-domain and cross-lingual comparison practical without task-specific training.

The corresponding caution is important. A mapping component alone cannot establish consistency. Knowing that a fragment corresponds to a node establishes surface-level alignment: it says that the text mentions the activity. It says nothing about whether the ordering, the exclusivity, or the obligation expressed in the narrative is respected by the model's execution semantics. A description and a model may share every activity and still permit entirely different behaviors. Alignment is a prerequisite for verification, not a substitute for it, and this distinction motivates the separation of mapping from constraint extraction described in Section 9, where the two are distinct concerns with distinct failure modes.

7. Datasets, Benchmarks and Mutation Testing

7.1. Datasets and Missing Benchmarks

Empirical progress in this area is constrained by the resources available for evaluation. The PET dataset [50] has become the de facto benchmark for text-to-model research, supplying annotated documents with entity- and relation-level ground truth.

A specific limitation affects verification research. These resources provide ground truth for extraction – what a correct model derived from a text looks like – not for verification, which requires knowing which inconsistencies exist between a given text and a given model. No public benchmark of triples comprising a text, a BPMN model and ground-truth inconsistencies exists. The automated ground-truth construction pipeline of Matei et al. [25] narrows the gap without closing it: their 387 validated models are reference targets for reconstruction, not annotated cases of disagreement.

The benchmark of Busch and Leopold [51] is closer in one respect. It defines mining of declarative process models from text as one of four BPM tasks, with 104 test samples and published code, and reports per-constraint-type precision, recall and F1 across seven language models. Their best reported F1 on that task is 0.211. The figure is instructive about task difficulty rather than directly comparable with results obtained under different conditions, and Section 9 returns to it. Their process question-answering dataset is built on the same publicly available process collection [52] used for evaluation in Section 9.

7.2. Mutation Testing and Explanation

In the absence of annotated inconsistencies, deliberate seeding provides a workable substitute. Mutation-based evaluation, long established in software engineering, perturbs a correct artifact to produce test cases whose expected outcome is known by construction. Tadeesom and Suwannasart [53] defined 25 BPMN-specific mutation operators; Gómez-Abajo et al. [54] demonstrated automated mutation campaigns through seed-model synthesis; and Lopes and Guerreiro [55] surveyed techniques for BPMN testing and formal verification, providing a methodological map for experimental design. The attraction of this approach is that it manufactures exactly the ground truth the field lacks: a mutated model differs from its description in a known and enumerable way. Its limitation is equally clear – seeded defects reflect the imagination of whoever seeded them, and detection rates on synthetic mutations bound what can be claimed about naturally occurring divergence.

Finally, explanation. Explainable artificial intelligence is well developed in process analytics, with decision-support systems grounded in feature

attribution [56] and a recent systematic review of interpretable methods for predictive process monitoring [57]. This literature is directed at explaining predictions. Explaining a verification failure is a different task with a different input: the object to be communicated is not a distribution over outcomes but a concrete execution path that violates a stated constraint. Rendering such a path into prose an analyst can act on is a constrained generation problem that the explainability literature in this domain has not addressed systematically.

8. Identified Research Gaps

Cross-cluster synthesis of the reviewed literature uncovers five interrelated gaps. They are stated here with their current status rather than as an unqualified list of openings, since two of the five have been partially occupied by work appearing during the period of this review.

Gap 1. No end-to-end verification of text against BPMN 2.0. The pioneering work of van der Aa and colleagues [42; 43; 44; 47] operates on simplified representations, and recent language-model contributions concentrate on generation. No publication integrates parsing of a real BPMN model, semantic analysis of a full description, and symbolic verification of their joint consistency.

Gap 2. No cross-validation between text-derived and model-derived constraint sets. Constraints can be extracted from text [47] and mined from logs [10]; deriving both and checking their mutual consistency would provide an independent correctness signal, and the infrastructure for comparing constraint sets is mature. The design has not been pursued, and is identified here as an open direction rather than one addressed by this work.

Gap 3. Multi-agent orchestration in BPM is confined to generation. Frameworks validated in software engineering [37; 38; 39] transfer to BPMN only for generation [34; 41]. No published system combines typed multi-agent orchestration with symbolic verification for checking a model against a narrative.

Gap 4. Neuro-symbolic work in BPM addresses prediction and property generation, not consistency. Predictive monitoring [31; 32] and specification verification against logs [30] leave the input side untouched, and LLaMA-PG [33] now occupies the verification-property corner. The remaining gap is narrower than it was and is defined by that input side:

no neuro-symbolic method takes a natural-language description as the source of the properties to be verified.

Gap 5. Natural-language explanation of verification violations is underserved. Explainable artificial intelligence in BPM targets predictions [56; 57]; counter-example-driven explanation, phrased for an analyst without formal-methods training, is not systematically addressed.

Three further deficiencies impede empirical progress. First, no public benchmark of text, model and ground-truth inconsistency triples exists; resources such as PET [50] provide ground truth for extraction only. Second, mutation-testing methodology for BPMN [53; 54], mature in itself, has not been applied in conjunction with language-model-based verifiers. Third, obtaining formal guarantees on language-model outputs in this domain remains an emerging concern, addressed seriously only by Duranti et al. [30] and, from the evaluation side, by Fionda et al. [11].

Table 1 maps each gap to the contribution or status of the approach developed in Section 9.

Table 1

Research gaps and their status in the present work

Gap	Description	Status in this work
1	No end-to-end text ↔ BPMN 2.0 verification	Addressed. Integrated pipeline: BPMN parser, LLM agents, symbolic verifier. Coverage limited to single-pool models; pools, message flows and cycles remain outside scope
2	No cross-validation of text- and model-derived constraint sets	Open. Constraints extracted from text only; log-side extraction not implemented
3	Multi-agent LLM in BPM limited to generation	Addressed. Typed orchestration of Mapper, Formalizer, Resource and Explainer agents for a verification task
4	Neuro-symbolic BPM limited to prediction and property generation	Partially addressed. Hybrid verification with formal guarantees from the symbolic module; input side is natural language rather than the model
5	NL explanation of verification violations underserved	Addressed architecturally. Dedicated Explainer agent producing analyst-readable text from counter-examples; explanation quality not yet evaluated

9. Prototype Architecture and Findings

The architecture and the results summarized in this section were first reported in [58], a conference paper co-authored with A. M. Kopp; the present section restates them in the context of the gaps identified in Section 8 and extends the discussion of their limitations. To establish whether those gaps are addressable in practice rather than only in principle, a prototype pipeline was implemented and evaluated on a single real-world process. The evidence presented below is accordingly evidence of feasibility, not a statistically grounded performance claim.

9.1. Pipeline Architecture and Modules

The system accepts two inputs – a BPMN 2.0 XML model and a plain-text process description – and produces a structured conformance report enumerating detected inconsistencies together with natural-language explanations. It comprises four modules orchestrated by LangGraph, a framework for stateful agent graphs. All inter-module data flows are governed by typed data contracts, which permits each component to be tested independently and prevents malformed intermediate artifacts from propagating downstream. The architecture is shown in Fig. 1.

Module A performs data ingestion: a parser reads the BPMN XML, discards diagrammatic markup, and constructs a process graph of nodes, edges and lane assignments, while a text loader segments the description into semantic fragments corresponding to individual actions or conditions.

Module B is the neuro-translation layer and carries the principal novelty. It employs three language-model agents. The Mapper receives the list of BPMN nodes – with identifiers, names, types and lane assignments – together with the list of text fragments, and produces a correspondence from each fragment to the most semantically similar model element, accompanied by a confidence score. The Formalizer reads the full description and extracts behavioral constraints expressed as DECLARE templates [3], selecting from ten: Existence, Absence, Response, Precedence, Succession, Chain Response, Chain Precedence, Not Co-Existence, Choice and Exclusive Choice. Table 2 shows representative constraints extracted from the evaluated process description alongside the sentences that produced them.

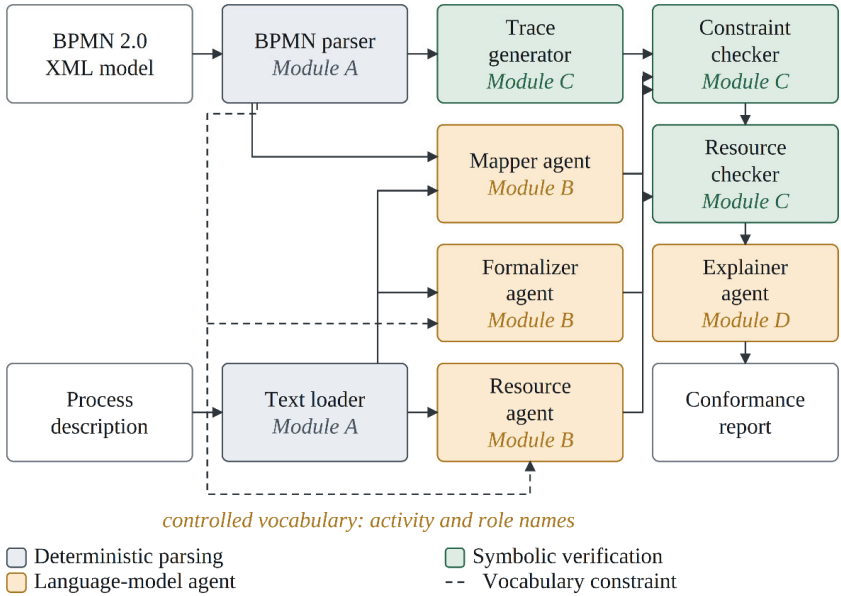


Figure 1. Architecture of the neuro-symbolic pipeline for text-BPMN conformance checking

Table 2

Examples of DECLARE constraints extracted from the process description

Template	Extracted constraint	Source text
Existence	Existence(“Clarify shipment method”)	the secretary clarifies who will do the shipping
Response	Response(“Get 3 offers”, “Select company”)	invites three companies… and selects one
Precedence	Precedence(“Package goods”, “Prepare pickup”)	if ready, goods are prepared for pickup
Not Co-Existence	NotCoExistence(“Get offers”, “Write label”)	large amounts… special, against small… normal post

The third agent, the Resource Agent, extracts assertions about which organizational role performs which activity, expressed as Performer(activity,

role). From the sentence stating that the logistics department insures the parcel, it derives Performer("Insure parcel", "Logistics"). This axis is orthogonal to the behavioral one: a model may admit exactly the execution sequences the description prescribes while assigning those activities to the wrong roles, and no constraint over traces can detect the difference. Resource extraction is a separate agent invocation rather than an extension of the Formalizer, because the two tasks differ in what they read for and fail in different ways.

The critical design decision in this module is the use of a controlled vocabulary: the agents may reference only names that occur in the parsed model. The restriction is enforced in two independent layers – the prompt admits only names drawn from the graph, and a deterministic post-validation step discards any artifact referencing a name absent from it, so that hallucinated activities cannot reach the verifier even when the model produces them. For resource assertions the vocabulary is two-sided, covering both activity names and lane names, which is what binds a phrase such as "the head of the logistics department" to the coarser lane label the model actually carries. Post-validation is restricted to task-typed nodes, so that event labels are no longer admissible as constraint arguments. The motivation for the controlled-vocabulary design is supported by independent evidence: Fionda et al. [11] evaluated ten language models on bidirectional translation between DECLARE specifications and natural language, and found that translation quality degrades consistently as the number of distinct activities grows, more sharply than it degrades with the number of constraints. Binding the extractor to an alphabet fixed by the model is therefore a mitigation of an empirically documented failure mode.

Module C performs symbolic verification and involves no language model, ensuring that results are deterministic and reproducible. It first enumerates the execution traces admitted by the process graph: exclusive gateways yield one trace per branch, parallel gateways yield branch permutations, and inclusive gateways yield all non-empty subsets of branches. Branch collection terminates at the merge point of the split that originated it, without which activities following the join are duplicated across branches; the evaluated process yields ten traces. Traces are generated by depth-first traversal of the BPMN graph directly, without an intermediate Petri net representation. Each extracted constraint is then checked against

every generated trace, and any trace violating a constraint is retained as a counter-example. Resource assertions are checked separately against the lane assignments recorded in the graph, a check that reads the model structure directly and does not depend on trace enumeration; an assertion naming an activity absent from the model is treated as vacuously satisfied, so that a deleted activity is not reported twice. If no path to a final state exists, the model is reported as structurally unsound before behavioral verification proceeds.

Module D translates formal counter-examples into prose. A violation of Existence("Write package label") with a counter-example trace in which the parcel is insured and prepared for dispatch is rendered as an explanation stating that the description requires a package label to be written, whereas the model admits a path on which no label is ever produced. This bridges the gap between verification output and the comprehension of an analyst without training in formal methods. The quality of the generated explanations is not evaluated below; the module is reported here as an architectural component rather than as a validated contribution.

9.2. Preliminary Evaluation and Findings

Evaluation used the "Dispatch of Goods" process from a publicly available collection of BPMN models employed in academic research [52], comprising seven tasks, six gateways of three kinds, and three swim lanes. The same collection supplies the process descriptions used by Busch and Leopold [51], which supports the choice as an established rather than an arbitrary one. Seven text-to-model mappings, thirteen DECLARE constraints and six resource assertions were annotated manually as a reference standard. Detection capability was assessed through mutation testing, following the methodology established for BPMN by Tadeesom and Suwannasart [53]: five mutations were seeded into the model, each introducing one known inconsistency. A pure-LLM baseline, which submits the model structure and description in a single prompt and asks for inconsistencies directly, was run on every mutation for comparison.

Two features of the protocol require statement, since they determine what the reported rates mean. First, the extracted artifacts are frozen: constraints, mappings and resource assertions are extracted once from the unmutated model and reused unchanged across all mutants, so that the

model under verification is the only variable. Re-invoking the extractor for each mutant, as an earlier protocol did, conflates structural change with extraction variance. Second, detection is assessed per assertion rather than by counting reported violations: a mutation counts as detected only if the specific assertion encoding the violated statement is itself newly reported. A change in the total number of violations is not accepted as evidence, because it can arise from paths appearing or disappearing for reasons unrelated to the seeded defect.

One seeded mutation was excluded as invalid. It deletes an activity the description marks as conditional, so the mutated model remains consistent with the text and there is no inconsistency for either method to detect; the denominator is therefore four rather than five. Results are summarized in Table 3.

Table 3

Preliminary evaluation results

Aspect	Measure	Result
Semantic mapping	Precision / Recall / F1	85.7% / 85.7% / 85.7%
Constraint extraction	Precision / Recall / F1	90.9% / 76.9% / 83.3%
Resource extraction	Precision / Recall / F1	85.7% / 100% / 92.3%
Mutation detection, hybrid	per-assertion criterion	2 of 4, precision 1.00
Mutation detection, baseline	defect explicitly named	3 of 4

The mapping agent correctly linked six of seven fragments. The single error assigned a fragment to a node identifier absent from the model, and notably received a confidence score of zero, suggesting that thresholding on confidence would suppress this class of error at little cost.

The Formalizer produced eleven constraints, ten of which matched the reference set. The three false negatives were all of Precedence type: the underlying ordering relation was identified correctly but expressed only as Response, indicating a systematic preference for one template where two apply. The false positives that referenced event labels rather than task labels in the earlier version of this evaluation no longer occur, since post-validation now admits only task-typed nodes as constraint arguments. The Resource Agent extracted seven assertions, six of which matched the reference set, with complete recall.

On the unmutated model the pipeline reports four spurious constraint violations and no spurious resource violations. Each is attributable to a named mechanism: three arise from constraints carrying a guard condition, which is recorded in a dedicated field but never evaluated against the branch conditions in the model, and one from a constraint that should not have been extracted. Two mechanisms that produced false violations in the earlier version have been removed, with measurable effect: constraint extraction F1 rose from 0.769 to 0.833 and the number of false violations on the correct model fell from seven to four.

The detection rates are more informative when separated by the dimension of the inconsistency. On the resource dimension both methods identify both seeded defects. On model coverage – an activity present in the model but absent from the description – the hybrid pipeline fails while the baseline succeeds, and the reason is structural rather than incidental: assertions derived from a description cannot constrain what the description does not mention. On the behavioral dimension, where a gateway branch is removed, both methods fail. The hybrid pipeline reported no false detections across the four mutants, giving a detection precision of 1.00 against a recall of 0.50; the baseline attains a higher recall of 0.75 without an equivalent guarantee, since its output is an unverified claim rather than a counter-example.

Five independent runs of the complete pipeline on the unmutated model produced identical mappings, constraints, resource assertions and violation sets, with a median end-to-end latency of 17.0 seconds and a range of 15.4 to 17.3 seconds. The determinism follows from decoding at temperature zero under a constrained output schema and should not be assumed to persist at higher temperatures or with larger activity vocabularies. Runtime is dominated by language-model latency rather than by trace enumeration, since the number of traces admitted by models of this size remains small; scalability to larger models is therefore bounded by trace growth rather than by inference cost.

9.3. Limitations and Open Problems

Four limitations are material rather than incidental. First, guard conditions are extracted into a dedicated field but are never compared with the branch conditions present in the parsed model, although those

conditions are available. This single unimplemented mechanism accounts for three of the four false violations on the correct model and for the failure on the behavioral mutation, and it is therefore the most economical target for further work.

Second, the Resource Agent emitted one assertion the description does not support, attributing the preparation of goods to the warehouse where the text states only who collects them. The prompt contained an explicit exemplar for precisely this distinction, which bounds what further prompt engineering can be expected to achieve for this error class.

Third, inconsistencies of model coverage lie outside the reach of the present method by construction. Assertions derived from a description cannot constrain what the description omits, so an activity added to the model without a counterpart in the text produces no violation. Detecting this class requires a check over the mapping rather than over the constraint set – reporting model nodes to which no text fragment was mapped – which the present system does not implement.

Fourth, evaluation rests on one process and a reference standard authored by a single annotator, who is also the author of the method; no inter-annotator agreement was measured and no external validation has been performed. A second process, exhibiting event-based gateways, multiple end states and temporal conditions, has been prepared but not yet evaluated; it exercises control-flow constructs absent from the first and is expected to test the trace generator more severely than the extraction agents. The declarative-mining task defined by Busch and Leopold [51], which comprises 104 samples with published code, offers a route to independent measurement, although its five-template inventory omits several templates central to the present work. That benchmark is instructive in another respect: across seven language models, the best reported F1 on unconstrained extraction of declarative constraints from text is 0.211, which indicates how difficult the task is when the activity vocabulary is not constrained. The figure is not directly comparable with the results reported above – the datasets, template inventories and matching criteria all differ, and the present sample is both smaller and easier – but it establishes that the problem addressed by the controlled-vocabulary design is a real and unsolved one.

CONCLUSIONS

This chapter examined the verification of consistency between natural-language descriptions of business processes and their BPMN 2.0 models, reviewing fifty-eight publications across nine thematic areas and reporting the preliminary evaluation of a prototype constructed on the basis of that review. The following conclusions are drawn.

1. The components required for automated text-model verification exist individually and have not been combined. The bodies of work reviewed above are each independently mature. No published method integrates them for the purpose of deciding whether a narrative and a model describe the same process.

2. The literature has bifurcated, and the unaddressed direction is their composition. One branch generates models from text; a second, more recent, generates verification properties from models. Both pursue formal rigor and both have produced results a verifier would need. Neither extracts behavioral constraints from a narrative in order to check them against a model authored independently.

3. Five research gaps were identified, of which three are addressed by the present work, one partially, and one remains open. End-to-end verification and multi-agent orchestration applied to verification rather than generation are addressed, as is counter-example-driven explanation at the level of architecture. Neuro-symbolic verification with a natural-language source is partially addressed, the corresponding territory having been narrowed by recent work on property generation from models. Cross-validation between text-derived and model-derived constraint sets remains open and is not claimed.

4. The composition is practically realizable. A four-module pipeline separating semantic interpretation from deterministic verification was implemented and evaluated on a real process. Semantic mapping achieved an F1-score of 85.7%, constraint extraction 83.3% and resource extraction 92.3%; under a criterion that counts a mutation as detected only when the assertion encoding the violated statement is itself newly reported, the pipeline detected two of four valid seeded inconsistencies with a detection precision of 1.00, against three of four for a baseline that offers no equivalent guarantee. These figures establish feasibility on one process; they do not establish performance.

5. One unimplemented verification mechanism, not the architecture, is the limiting factor. The pipeline reports four spurious violations on the unmutated model, three of which arise because guard conditions are extracted into a dedicated field but never compared with the branch conditions present in the parsed model. The same omission accounts for the failure on the behavioral mutation. Removing two further mechanisms that produced false violations raised constraint-extraction F1 from 0.769 to 0.833 and reduced the false violations from seven to four, which localizes the remaining bottleneck precisely rather than attributing it to extraction quality in general.

6. A restricted formalism is a viable interface between neural and symbolic components. Confining the extractor to a finite catalog of constraint templates instantiated over a vocabulary fixed by the parsed model constrains the language model to a formally verifiable output space. Independent evaluation showing that translation quality degrades as the activity alphabet grows supports this design.

Directions for further work. Four follow directly. Condition-aware verification, so that guarded constraints are evaluated only on traces satisfying their guards, addresses the identified bottleneck most directly. External validation against the published declarative-extraction benchmark would replace a self-authored reference standard on one process with an independent measurement on 104 samples. Evaluating extracted constraint sets by their behavior over traces, rather than by set overlap with a reference, would measure what the verifier actually depends on. A check over the mapping rather than over the constraint set – reporting model nodes to which no text fragment was mapped – would reach the class of inconsistency in which the model contains an activity the description omits, which assertions derived from the description cannot constrain by construction.

Beyond these, two longer-term directions warrant attention: constructing and publishing a benchmark of text, BPMN model and ground-truth inconsistency triples, whose absence constrains every empirical contribution in this area; and adapting the treatment of linguistic ambiguity developed for rule-based methods, under which a violation is reported only when no admissible reading of the text reconciles it with the model.

References:

1. Object Management Group. (2013). Business process model and notation (BPMN) version 2.0.2 (OMG Standard). https://www.omg.org/spec/BPMN/2.0.2/About-BPMN

2. Dijkman, R. M., Dumas, M., & Ouyang, C. (2008). Semantics and analysis of business process models in BPMN. *Information and Software Technology*, 50(12), 1281–1294. https://doi.org/10.1016/j.infsof.2008.02.006

3. Pesic, M., & van der Aalst, W. M. P. (2006). A declarative approach for flexible business processes management. *Business Process Management Workshops*, LNCS 4103, 169–180. https://doi.org/10.1007/11837862_18

4. Maggi, F. M., Montali, M., Westergaard, M., & van der Aalst, W. M. P. (2011). Monitoring business constraints with linear temporal logic: An approach based on colored automata. BPM 2011, LNCS 6896, 132–147. https://doi.org/10.1007/978-3-642-23059-2_13

5. van der Aalst, W. M. P. (2016). Process mining: Data science in action (2nd ed.). Springer. https://doi.org/10.1007/978-3-662-49851-4

6. Carmona, J., van Dongen, B. F., Solti, A., & Weidlich, M. (2018). Conformance checking: Relating processes and models. Springer. https://doi.org/10.1007/978-3-319-99414-7

7. Manhaeve, R., Dumančić, S., Kimmig, A., Demeester, T., & De Raedt, L. (2021). Neural probabilistic logic programming in DeepProbLog. *Artificial Intelligence,* 298, 103504. https://doi.org/10.1016/j.artint.2021.103504

8. d'Avila Garcez, A., & Lamb, L. C. (2023). Neurosymbolic AI: The 3rd wave. *Artificial Intelligence Review*, 56(11), 12387–12406. https://doi.org/10.1007/s10462-023-10448-w

9. Pesic, M., Schönenberg, H., & van der Aalst, W. M. P. (2007). DECLARE: Full support for loosely-structured processes. IEEE EDOC 2007, 287. https://doi.org/10.1109/EDOC.2007.14

10. Di Ciccio, C., & Mecella, M. (2015). On the discovery of declarative control flows for artful processes. *ACM Transactions on Management Information Systems,* 5(4), Article 24. https://doi.org/10.1145/2629447

11. Fionda, V., Ielo, A., & Ricca, F. (2025). Are large language models fluent in declarative process mining? IJCAI-25, 4500–4508. https://doi.org/10.24963/ijcai.2025/501

12. Weske, M. (2019). Business process management: Concepts, languages, architectures (3rd ed.). Springer. https://doi.org/10.1007/978-3-662-59432-2

13. Friedrich, F., Mendling, J., & Puhlmann, F. (2011). Process model generation from natural language text. CAiSE 2011, LNCS 6741, 482–496. https://doi.org/10.1007/978-3-642-21640-4_36

14. Leopold, H., Mendling, J., & Polyvyanyy, A. (2012). Generating natural language texts from business process models. CAiSE 2012, LNCS 7328, 64–79. https://doi.org/10.1007/978-3-642-31095-9_5

15. Bellan, P., Dragoni, M., & Ghidini, C. (2022). Extracting business process entities and relations from text using pre-trained language models and in-context

learning. EDOC 2022, LNCS 13585, 182–199. https://doi.org/10.1007/978-3-031-17604-3_11

16. Neuberger, J., Ackermann, L., van der Aa, H., & Jablonski, S. (2025). A universal prompting strategy for extracting process model information from natural language text using large language models. ER 2024, LNCS 15238, 38–55. https://doi.org/10.1007/978-3-031-75872-0_3

17. Van Woensel, W., & Motie, S. (2024). NLP4PBM: A systematic review on process extraction using natural language processing with rule-based, machine and deep learning methods. *Enterprise Information Systems*, 18(11). https://doi.org/10.1080/17517575.2024.2417404

18. Vidgof, M., Bachhofner, S., & Mendling, J. (2023). Large language models for business process management: Opportunities and challenges. BPM Forum 2023, LNBIP 490, 107–123. https://doi.org/10.1007/978-3-031-41623-1_7

19. Grohs, M., Abb, L., Elsayed, N., & Rehse, J.-R. (2023). Large language models can accomplish business process management tasks. BPM Workshops 2023, LNBIP 492, 453–465. https://doi.org/10.48550/arXiv.2307.09923

20. Busch, K., Rochlitzer, A., Sola, D., & Leopold, H. (2023). Just tell me: Prompt engineering in business process management. BPMDS 2023, LNBIP 479, 3–11. https://doi.org/10.1007/978-3-031-34241-7_1

21. Kourani, H., Berti, A., Schuster, D., & van der Aalst, W. M. P. (2025). Process modeling with large language models. Software and Systems Modeling. https://doi.org/10.1007/s10270-025-01318-w

22. Berti, A., Kourani, H., & van der Aalst, W. M. P. (2024). PM-LLM-Benchmark: Evaluating large language models on process mining tasks. ICPM 2024 Workshops, LNBIP 533. https://doi.org/10.48550/arXiv.2407.13244

23. Rebmann, A., Schmidt, F. D., Glavaš, G., & van der Aa, H. (2025). Evaluating the ability of LLMs to solve semantics-aware process mining tasks. Process Science, 2, Article 10. https://doi.org/10.1007/s44311-025-00019-3

24. Estrada-Torres, B., del Río-Ortega, A., & Resinas, M. (2024). Mapping the landscape: Exploring large language model applications in business process management. BPMDS 2024, LNBIP, 22–31. https://doi.org/10.1007/978-3-031-61007-3_3

25. Matei, I., Zhenirovskyy, M., Menaka Sekar, P. K., & Wong, H. Y. (2026). Automated BPMN model generation from textual process descriptions: A multi-stage LLM-driven approach. arXiv. https://doi.org/10.48550/arXiv.2604.12105

26. Adriansyah, A., van Dongen, B. F., & van der Aalst, W. M. P. (2011). Conformance checking using cost-based fitness analysis. IEEE EDOC 2011, 55–64. https://doi.org/10.1109/EDOC.2011.12

27. Dunzer, S., Stierle, M., Matzner, M., & Baier, S. (2019). Conformance checking: A state-of-the-art literature review. S-BPM ONE 2019. ACM. https://doi.org/10.48550/arXiv.2007.10903

28. Corradini, F., Fornari, F., Polini, A., Re, B., Tiezzi, F., & Vandin, A. (2021). A formal approach for the analysis of BPMN collaboration models. *Journal of Systems and Software,* 180, 111007. https://doi.org/10.1016/j.jss.2021.111007

29. Donadello, I., Felli, P., Innes, C., Maggi, F. M., & Montali, M. (2024). Conformance checking of fuzzy logs against declarative temporal specifications. BPM 2024, LNCS 14940, 39–56. https://doi.org/10.1007/978-3-031-70396-6_3

30. Duranti, D., Giorgini, P., Mazzullo, A., Robol, M., & Roveri, M. (2025). LLM-driven knowledge extraction in temporal and description logics. EKAW 2024, LNCS 15370, 190–208. https://doi.org/10.1007/978-3-031-77792-9_12.

31. Mezini, A., Umili, E., Donadello, I., Maggi, F. M., Mancanelli, M., & Patrizi, F. (2025). Neuro-symbolic predictive process monitoring. arXiv:2509.00834.

32. Buliga, A., Di Francescomarino, C., Ghidini, C., Montali, M., & Ronzani, M. (2025). Generating counterfactual explanations under temporal constraints. AAAI 2025, 15622–15631.

33. Tang, H., Wang, X., Zhang, Y., Ran, Q., & Gao, H. (2026). LLaMA-PG: A domain-knowledge-enhanced framework for verification property generation from BPMN models. *International Journal of Cognitive Computing in Engineering*, 7, 515–537. https://doi.org/10.1016/j.ijcce.2026.05.003

34. Xu, Q., Zhang, Y., Tang, H., Ran, Q., Gao, H., & Xu, Y. (2026). H2A-BPMN: A hierarchical and hybrid agent framework for industrial BPMN automated modeling. CollaborateCom 2025, LNICST 681. https://doi.org/10.1007/978-3-032-21171-2_10

35. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. ICLR 2023. arXiv:2210.03629

36. Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., & Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. NeurIPS 2023. arXiv:2303.11366

37. Wu, Q., Bansal, G., Zhang, J., et al. (2023). AutoGen: Enabling next-gen LLM applications via multi-agent conversation. COLM 2024. arXiv:2308.08155

38. Hong, S., Zhuge, M., Chen, J., et al. (2024). MetaGPT: Meta programming for a multi-agent collaborative framework. ICLR 2024. arXiv:2308.00352

39. Qian, C., Liu, W., Liu, H., et al. (2024). ChatDev: Communicative agents for software development. ACL 2024, 15174–15186. arXiv:2307.07924

40. Guo, T., Chen, X., Wang, Y., et al. (2024). Large language model based multi-agents: A survey of progress and challenges. IJCAI 2024, 8048–8057. https://doi.org/10.24963/ijcai.2024/890

41. Lin, L., Jin, Y., Zhou, Y., Chen, W., & Qian, C. (2025). MAO: a framework for process model generation with Multi-Agent Orchestration. *IEEE Transactions on Services Computing*, 18(6), 3987–4000. https://doi.org/10.1109/tsc.2025.3611816

42. van der Aa, H., Leopold, H., & Reijers, H. A. (2015). Detecting inconsistencies between process models and textual descriptions. BPM 2015, LNCS 9253, 90–105. https://doi.org/10.1007/978-3-319-23063-4_6

43. van der Aa, H., Leopold, H., & Reijers, H. A. (2017). Comparing textual descriptions to process models — The automatic detection of inconsistencies. *Information Systems*, 64, 447–460. https://doi.org/10.1016/j.is.2016.07.010

44. van der Aa, H. (2018). Comparing and aligning process representations: Foundations and technical solutions. LNBIP 323. Springer. https://doi.org/10.1007/978-3-319-94634-4

45. van der Aa, H., Leopold, H., & Reijers, H. A. (2016). Dealing with behavioral ambiguity in textual process descriptions. BPM 2016, LNCS 9850, 271–288. https://doi.org/10.1007/978-3-319-45348-4_16

46. Sànchez-Ferreres, J., van der Aa, H., Carmona, J., & Padró, L. (2018). Aligning textual and model-based process descriptions. *Data & Knowledge Engineering*, 118, 24–40. https://doi.org/10.1016/j.datak.2018.09.001

47. van der Aa, H., Di Ciccio, C., Leopold, H., & Reijers, H. A. (2019). Extracting declarative process models from natural language. CAiSE 2019, LNCS 11483, 365–382. https://doi.org/10.1007/978-3-030-21290-2_23

48. Matei, I., Menaka Sekar, P. K., Zhenirovskyy, M., Wong, H. Y., Kohmura, S., Hotta, S., & Inomata, A. (2026). Ambiguity detection and elimination in automated executable process modeling. arXiv. https://doi.org/10.48550/arXiv.2604.10884

49. Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. EMNLP-IJCNLP 2019. arXiv:1908.10084

50. Bellan, P., van der Aa, H., Dragoni, M., Ghidini, C., & Ponzetto, S. P. (2022). PET: An annotated dataset for process extraction from natural language text. BPM Workshops 2022, LNBIP 460, 315–321. https://doi.org/10.48550/arXiv.2203.04860

51. Busch, K., & Leopold, H. (2024). Towards a benchmark for large language models for business process management tasks. arXiv:2410.03255

52. Camunda. (2023). BPMN for research [Computer software]. GitHub. https://github.com/camunda/bpmn-for-research

53. Tadeesom, P. P., & Suwannasart, T. (2017). Mutation operators in BPMN model. ICIDE 2017: Proceedings of the 2017 International Conference on Industrial Design Engineering, 45–48. https://doi.org/10.1145/3178264.3178286

54. Gómez-Abajo, P., Guerra, E., de Lara, J., & Merayo, M. G. (2020). Seed model synthesis for testing model-based mutation operators. EMMSAD 2020, LNBIP 386, 83–98. https://doi.org/10.1007/978-3-030-58135-0_6

55. Lopes, T., & Guerreiro, S. (2023). Assessing business process models: A literature review on techniques for BPMN testing and formal verification. *Business Process Management Journal,* 29(8), 133–162. https://doi.org/10.1108/BPMJ-11-2022-0557

56. Galanti, R., de Leoni, M., Monaro, M., Navarin, N., Marazzi, A., Di Stasi, B., & Maldera, S. (2023). An explainable decision support system for predictive process analytics. *Engineering Applications of Artificial Intelligence*, 120, 105904. https://doi.org/10.1016/j.engappai.2023.105904

57. Mehdiyev, N., Majlatow, M., & Fettke, P. (2025). Interpretable and explainable machine learning methods for predictive process monitoring: A systematic literature review. *Artificial Intelligence Review*, 58(12). https://doi.org/10.1007/s10462-025-11399-0

58. Rudskyi, O., & Kopp, A. (2026). Implementation and preliminary evaluation of a neuro-symbolic pipeline for text-BPMN conformance checking. 2026 IEEE 7th KhPI Week on Advanced Technology (KhPIWeek).