Software and Systems Modeling
https://doi.org/10.1007/s10270-025-01318-w

SPECIAL SECTION PAPER

# Evaluating large language models on business process modeling: framework, benchmark, and self-improvement analysis

Humam Kourani[1,2] · Alessandro Berti[2] · Daniel Schuster[2,3] · Wil M. P. van der Aalst[1,2]

Received: 15 November 2024 / Revised: 26 July 2025 / Accepted: 4 August 2025

**Abstract**

Large language models (LLMs) are rapidly transforming various fields, including the field of business process management (BPM). LLMs provide new ways for analyzing and improving operational processes. This paper assesses the capabilities of LLMs on business process modeling using a framework for automating this task and a robust evaluation approach. We design a comprehensive benchmark, consisting of 20 diverse business processes, and we demonstrate our evaluation approach by assessing 16 current state-of-the-art LLMs from major AI vendors. Our analysis highlights significant performance variations across LLMs and reveals a positive correlation between efficient error handling and the quality of generated models. It also shows consistent performance trends within similar LLM groups. Furthermore, we use our evaluation approach to investigate LLM self-improvement techniques, encompassing self-evaluation, input optimization, and output optimization. Our findings indicate that output optimization, in particular, offers promising potential for enhancing quality, especially in models with initially lower performance. Our contributions provide insights for leveraging LLMs in BPM, paving the way for more advanced and automated process modeling techniques.

**Keywords** Business process modeling · Large language models · Generative AI · Benchmarking · Process mining

## 1 Introduction

Process modeling is a crucial element of business process management (BPM), acting as a comprehensive toolkit for understanding, documenting, analyzing, and optimizing intricate business operations. It encompasses various forms—from textual descriptions to visual diagrams and executable models—thereby providing a multi-dimensional approach to capturing the nuances of organizational processes.

Business process modeling integrates several key perspectives, each focusing on distinct aspects of processes. These include the *control-flow perspective*, which maps out the sequence of activities and their interdependencies; the *data perspective*, which deals with the creation, manipulation, and usage of data throughout the process; the *resource perspective*, which identifies the human and system resources required for process execution; and the *operational perspective*, which outlines the rules and execution semantics governing the process. Our focus in this paper is primarily on enhancing the control-flow perspective because it forms the foundational structure that supports the integration of data, resources, and operational aspects into a process model.

Traditionally, business process modeling requires considerable manual effort and a deep understanding of sophisticated process modeling languages such as BPMN (Business Process Model and Notation) [1] and Petri nets [2]. Moreover, maintaining these process models to reflect changes in business operations is an ongoing challenge, presenting significant obstacles for individuals lacking expertise in these

Communicated by Dominik Bork and Arnon Sturm.

✉ Humam Kourani
humam.kourani@fit.fraunhofer.de

Alessandro Berti
a.berti@pads.rwth-aachen.de

Daniel Schuster
schuster@pads.rwth-aachen.de

Wil M. P. van der Aalst
wvdaalst@pads.rwth-aachen.de

[1] Fraunhofer Institute for Applied Information Technology FIT, Schloss Birlinghoven, 53757 Sankt Augustin, Germany

[2] RWTH Aachen University, Ahornstraße 55, 52074 Aachen, Germany

[3] Process Intelligence Solutions, Kurfürstenstraße 5, 52066 Aachen, Germany

Published online: 02 September 2025

languages, thus highlighting the need for more streamlined methodologies in process modeling.

The emergence of large language models (LLMs) such as GPT-4 [3] and Gemini [4] offers a promising avenue for enhancing the efficiency and accessibility of process modeling. Trained on vast and varied datasets, these models are skilled in a range of tasks from generating coherent and contextually relevant text to solving complex problems and producing executable code [5–7]. Their capability to process and interpret complex textual inputs in natural language positions LLMs as particularly suitable for tasks like process modeling that require the generation and refinement of structured outputs from textual descriptions.

Our previously introduced framework [8] leverages LLMs to automate the generation and refinement of process models from textual descriptions. It employs sophisticated techniques in prompt engineering, error handling, and code generation, transforming detailed process descriptions into process models. This framework utilizes the Partially Ordered Workflow Language (POWL) [9] as an intermediate representation due to the quality guarantees it provides, particularly in ensuring *soundness* [9], and the possibility to export POWL models in standard notations such as BPMN and Petri nets. Preliminary results in [8] demonstrated the practicality and effectiveness of this framework, showcasing its clear advantage over alternative solutions due to the usage of POWL to ensure soundness.

This paper extends the work presented in [8] by providing a robust and repeatable evaluation approach for comparative assessments of LLM capabilities in business process modeling. This approach incorporates a comprehensive benchmark consisting of 20 diverse business processes, each paired with a ground-truth model and a simulated event log. This setup enables automated quality assessment of generated process models via conformance checking [10]. Our evaluation targets diverse LLM capabilities such as natural language understanding, code generation, adherence to instructions, and error correction. We demonstrate this evaluation approach by assessing 16 state-of-the-art LLMs from various AI vendors, such as Google (https://ai.google.dev/), OpenAI (https://openai.com/), Anthropic (https://www.anthropic.com/), Meta (https://ai.meta.com/), and Mistral AI (https://mistral.ai/).

Beyond benchmarking, we use our evaluation approach to explore the potential of LLM *self-improvement* techniques for enhancing the quality of generated process models. By self-improvement, we refer to the ability of LLMs to autonomously refine their performance by leveraging their reasoning capabilities to detect and correct issues or improve clarity. We investigate three self-improvement strategies: *self-evaluation*, *input optimization*, and *output optimization*. In self-evaluation, the LLM generates multiple responses, evaluates them based on predefined criteria, and selects the most accurate or appropriate one. Output optimization involves the LLM iteratively revising its own response to better fulfill the task requirements. In input optimization, the LLM reformulates or enriches the input prompt to guide itself toward a better output. These techniques are inspired by recent advances in AI self-refinement, where models like GPT-4 have demonstrated the ability to improve code quality through iterative feedback or enhance text coherence by rephrasing ambiguous statements [11]. By applying these strategies, we aim to evaluate whether LLMs can autonomously refine their performance within our process modeling framework, resulting in more accurate and reliable process models.

The structure of this paper is as follows. First, Sect. 2 provides foundational background knowledge on related concepts, and Sect. 3 discusses related work. Section 4 provides a detailed overview of our LLM-based process modeling framework. Section 5 introduces the ProMoAI tool, which supports our framework. Section 6 presents the benchmarking analysis, including the experimental setup, evaluation metrics, and a discussion of the results. Section 7 investigates the LLM self-improvement strategies and analyzes their impact on model quality. Finally, Sect. 8 discusses implications, limitations, and future directions, while Sect. 9 concludes the paper.

## 2 Background

This section provides an overview of the foundational concepts underpinning our work, including the general principles of process modeling, an examination of relevant modeling languages, the emerging role of large language models in process modeling, and the essential concepts of event data, process simulation, and conformance checking. This background lays the groundwork for understanding our proposed process modeling framework and evaluation methodology. While the focus in this section is on introducing the necessary conceptual foundations, the subsequent section (Sect. 3) shifts to a review of related research efforts.

### 2.1 Process modeling

Business process modeling is the act of representing the steps, decisions, and flow of a business process in a formal way [12]. It is a cornerstone of BPM because a clear process model enables organizations to understand and analyze how work is done, identify inefficiencies, and communicate process knowledge. By creating a visual or formal model of a process, stakeholders can discuss improvements and perform analysis (e.g., simulation or verification) before implementing changes in the real world. In essence, a process model serves as a blueprint of organizational workflows, supporting their documentation, analysis, and improvement.

Organizations typically capture their business processes by creating process models that document how work is performed. These models are usually developed by business analysts in close collaboration with domain experts to ensure accuracy and shared understanding [13]. In practice, process modeling often starts with informal techniques (e.g., interviews, workshops, simple flow sketches) and then progresses to formal modeling using standardized notations and tools. Organizations employ various modeling tools, ranging from simple diagramming software to specialized business process management platforms. A widely used notation is BPMN [1], which has become an industry standard for process diagrams due to its expressiveness and relative ease of understanding by business users.

Traditionally, creating business process models demands substantial manual work and a strong grasp of complex process modeling languages. Before the advent of LLMs, automated extraction of process models from textual descriptions primarily relied on traditional Natural Language Processing (NLP) and rule-based techniques. Modeling approaches exploited methods such as dependency parsing, part-of-speech tagging, and semantic role labeling to identify verbs corresponding to activities and to infer relationships between process elements [14, 15]. These techniques were often combined with pattern matching and keyword extraction to map unstructured text to formal process constructs (e.g., mapping verbs to activities and conjunctions like “then” or “or” to BPMN sequence flows or gateways). However, challenges such as natural language ambiguity and variability in textual descriptions often necessitated significant human involvement and hindered the full automation of process modeling with NLP [16].

## 2.2 Process modeling languages: BPMN, Petri net, POWL

Over the years, numerous process modeling languages have been developed, each with different levels of formality and expressiveness. BPMN is one of the most widely used standards for modeling business processes [1]. BPMN provides a graphical notation (flowchart-like diagrams) that is accessible to business users, yet capable of representing complex control-flow patterns (e.g., parallel branches, decisions, loops, message flows). It defines various elements such as events (signaling timers, messages, or errors), activities (tasks or subprocesses), gateways (for branching and merging control flow like XOR or AND gateways), and artifacts (data objects, annotations, etc.). BPMN’s popularity in industry comes from its intuitive visual nature and its coverage of complex business scenarios.

*Petri nets* offer a more formal foundation for modeling the control-flow perspective of processes [17]. A Petri net is a bipartite graph consisting of places (conditions or states) and transitions (activities or events), connected by arcs. Firing of transitions produces movement of tokens between places, capturing the state changes in a process. Petri nets have a solid mathematical underpinning, enabling formal analysis of process properties (e.g., deadlock detection [18]). In the context of business processes, a specific class called *Workflow-nets (WF-net)* (a connected Petri net with unique start and end places) is often used to model processes. Petri nets are very expressive in modeling concurrency, synchronization, and resource sharing [19, 20]. The trade-off, however, is that Petri net diagrams are less intuitive to non-experts, especially for large processes, and they are limited to the control-flow perspective of the process (e.g., no distinction between different types of events and no explicit notion of roles). Nonetheless, Petri nets serve as a common formal denominator for many process analysis and process mining techniques.

Process modeling with BPMN and Petri nets can lead to quality issues, as it is possible to generate models with unreachable parts, for instance. To address this, the concept of *soundness* is introduced, and many automated process model discovery methods rely on languages that ensure soundness (e.g., process trees [21] and POWL [22]). A process tree is a hierarchical model where leaves represent single tasks and inner nodes are control-flow operators used to combine submodels, forming larger ones. POWL is a hybrid process modeling notation that blends the hierarchical structure of process trees with the flexibility of partial orders. It addresses limitations in process trees by supporting complex concurrency patterns while preserving the soundness guarantee, making it a valuable tool for process modeling and discovery.

POWL models are defined recursively, starting with atomic activities—single tasks like “Submit order”—as the base case. These are combined using three control-flow constructs: partial orders, exclusive choices (XOR), and loops. A partial order $\prec$ is an irreflexive, transitive, and asymmetric relation that specifies dependencies among sub-processes. For example, if $\psi_1 \prec \psi_2$, all activities in the sub-model $\psi_1$ must precede those in the sub-model $\psi_2$, while unordered sub-models run concurrently. The XOR operator, written as $\times(\psi_1, \dots, \psi_n)$ models mutually exclusive choices, allowing only one sub-model per execution. The loop operator, $\circlearrowleft(\psi_{\text{do}}, \psi_{\text{redo}})$, defines a cycle where $\psi_{\text{do}}$ executes at least once, followed by optional alternations of $\psi_{\text{redo}}$ and $\psi_{\text{do}}$. These constructs are nested hierarchically, enabling complex structures like a partial order containing an XOR sub-model.

POWL models can be transformed into widely recognized process modeling languages, including Workflow Nets and BPMN. In [23], an algorithm is proposed for converting POWL models into sound Workflow Nets, which can subsequently be translated into BPMN diagrams. For example, partial orders are mapped to parallel gateways in BPMN, while XORs and loops are represented using exclusive gateways. This convertibility enhances POWL's utility as a robust intermediate representation for modeling business processes, facilitating its integration with BPM tools.

## 2.3 Process modeling with large language models

LLMs are a class of artificial intelligence models that have been trained on vast amounts of text data to learn the patterns of human language. Technically, most state-of-the-art LLMs are based on the *transformer architecture* [24], which enables efficient learning from sequences. By training on billions of words (for instance, internet text, books, articles), these models develop a statistical understanding of how words and sentences are formed. Modern LLMs like GPT-3 are extremely large neural networks with hundreds of billions of parameters, trained in a self-supervised manner to predict the next word in a sentence [25]. Through this training, they capture not just grammar and syntax, but also a significant amount of world knowledge and facts that appear in text.

The general capabilities of LLMs are broad: they can generate human-like coherent text, translate between languages, summarize documents, answer questions, and even write code or structured data when prompted correctly. The key feature is that these models can generalize to a wide range of language tasks without task-specific programming—much of the "skill" is implicit in the model after reading so much text. They are called "large" because their size (and the size of their training corpora) is what differentiates them from earlier generations of language models, granting them emergent abilities in understanding and generating text that smaller models could not achieve.

In the context of BPM, and specifically process modeling, LLMs have recently emerged as a very promising tool. Traditionally, creating a process model from a textual description (like a paragraph describing how an order is handled) was a manual task requiring a skilled analyst. Analysts would interpret the natural language, which could be ambiguous or incomplete, and translate it into formal elements (events, tasks, gateways, etc.). This translation is challenging and time-consuming. The idea of leveraging LLMs is that these models have absorbed a lot of procedural knowledge and patterns from textual descriptions of processes (among the vast training data, there are likely documents like manuals, guidelines, etc., which describe processes). Therefore, an LLM can potentially take a natural language description of a process and automatically produce a candidate process model, or at least assist in doing so [8]. In essence, the LLM acts as a bridge between unstructured descriptions and structured process representations.

## 2.4 Event data and process simulation

When processes are executed in real life or in information systems, they leave behind traces of what happened. These traces are recorded as event data (often organized as *event logs*). An event log is a collection of records, where each record (event) corresponds to an occurrence of an activity within a specific process instance (often called a *case*). At minimum, each event in a log contains: a *case identifier* (tying the event to a particular process instance or workflow execution), an *activity name* (what step was executed), and a *timestamp* (when it happened). Often, additional attributes are recorded as well, such as the resource or person who performed the activity.

Event data is essential for process mining techniques. Given an event log, one can discover a process model that reflects the observed behavior (*process discovery*), check conformance of the log against an existing model (*conformance checking*), or enhance a model with additional perspectives (*process enhancement*) [26]. Event logs thus provide an objective view of how processes actually execute, which is often more complex or varied than what the official documentation suggests. By analyzing event logs, organizations can find bottlenecks, deviations, and opportunities for improvement grounded in real data.

*Process simulation* is a key component of BPM. Process simulation means using a process model to generate artificial execution data [27]. If we have a process model, we can simulate the process by virtually "executing" it many times, following the different paths allowed by the model, to produce synthetic event logs. By simulating a modeled process, analysts can obtain metrics like cycle times, waiting times, throughput, and resource utilization, and use these to identify potential bottlenecks or inefficiencies. Simulation allows asking "what-if" questions—for example, what if the arrival rate of orders doubles, or what if an extra employee is added to a particular task—and observing the impact on process performance without any real-world risk [28]. This capability makes simulation a powerful tool for process improvement and decision support.

## 2.5 Conformance checking

*Conformance checking* is a key discipline in process mining that focuses on comparing a process model with an event log of the same process to see how well the model and the reality align [10]. Given a process model (the expected or normative behavior) and an event log (the observed behavior from executions), conformance checking techniques evaluate to what extent the log fits the model and vice versa. In practice, this helps identify deviations: points where the real process deviates from the prescribed model (e.g., steps skipped or done in the wrong order, or extra steps performed that are not in the model).

Several *metrics* have been defined to quantify conformance. Two of the core metrics are *fitness* and *precision*. Fitness (sometimes also called *recall*) measures how much of the observed behavior in the log is captured by the model. A model has perfect fitness (1.0 or 100%) if all traces in the event log can be completely replayed by the model from start to end (i.e., the model can generate every sequence of activities that actually happened). Low fitness indicates the model is missing behavior that actually occurs in reality. Precision, on the other hand, measures the opposite: how much behavior the model allows that was not seen in the log. A model with perfect precision does not permit any behavior that was not observed in reality. In other words, precision penalizes models that overgeneralize beyond the recorded behavior.

Conformance checking is typically performed using algorithms that try to replay the event log on the model [29]. One classical approach for Petri nets and similar models is *token-based replay*, where one *fires* the transitions of the Petri net according to the sequence of events in a trace and tracks where tokens get consumed or produced. More advanced conformance techniques use *alignments*. The idea of alignments is to find the best possible correspondence between a trace from the log and an execution of the model by allowing certain minimal corrections. We can imagine aligning a trace and a model run as two sequences and possibly inserting "gaps" or "moves" where they don't agree: a *log move* (meaning the log had an event that the model didn't expect at that point) or a *model move* (meaning the model would do something that the log didn't have). By finding an alignment with minimal moves, one can pinpoint exactly where the deviations are. From alignments, metrics like fitness can be computed in a robust way (taking into account how many model/log moves were needed). Alignments also facilitate diagnosing where in the process the differences occur, not just aggregate scores.

## 3 Related work

Building upon the foundational concepts presented in Sect. 2, this section reviews related research, including traditional NLP methods for process modeling, applications of LLMs in BPM, approaches to LLM self-improvement, and relevant evaluation methods and benchmarks.

An overview of various methods for extracting process information from textual content is presented in [30]. The study in [31] utilizes Natural Language Processing (NLP) and text mining techniques to derive process models directly from text, while [32] combines NLP with computational linguistics to generate BPMN models. The approach in [33] applies NLP to extract structured relationship representations, referred to as *fact types*, from textual data, which are then converted into BPMN components. The BPMN Sketch Miner, as detailed in [34], uses process mining [35] to produce BPMN models from text described in a *domain-specific language*. Commercial solutions are also adopting AI for process modeling; for example, Process Talks (https://processtalks.com) offers an AI-driven platform for generating process models from textual descriptions.

Recent studies have explored the integration of LLMs in BPM, investigating their potential applications and challenges. Several works [36, 37] delve into how LLMs can be employed for BPM and process mining tasks. The papers [38] and [39] explore the application of LLMs in process discovery and process querying, respectively. Research has also explored on leveraging LLMs for knowledge extraction from textual sources. For instance, [40] compared NLP and LLM-based approaches for constructing a Business Process Knowledge Base from interview texts, finding that LLMs with enriched prompts outperformed traditional NLP. Similarly, [41] proposed a prompt-based in-context learning strategy to extract process knowledge from descriptions and construct knowledge graphs, validating its feasibility in low-resource scenarios. Beyond knowledge extraction, the research in [42] employs BERT [43] to classify and analyze process execution logs, aiming to enhance process monitoring and anomaly detection. Finally, [44] investigates the broader impacts of LLMs in conceptual modeling, proposing potential applications that extend beyond traditional BPM tasks, and [45] discusses the limitations of using GPT-4 for conceptual modeling.

A significant line of research investigates the use of LLMs for generating process models directly from various inputs. Studies like those in [8, 46] applied LLMs to generate process models directly from unstructured text. In [47–49], methods for generating process models through dialogue-based interactions and chatbots are proposed. The paper [50] showcases the ability of LLMs to translate textual descriptions into procedural and declarative process model constraints. Addressing practical aspects, the paper [51] explores using LLMs to generate BPMN models adhering to company-specific style guides. To improve generation accuracy, the method proposed in [52] employs a multi-step approach, breaking the text-to-model task into smaller, sequential subtasks for the LLM. Furthermore, [53] utilized ChatGPT with prompt engineering techniques to enhance the semantic quality of business process models by identifying missing concepts, demonstrating the potential of LLMs to improve model completeness beyond visual generation.

The challenge of *hallucination* in LLMs, where models generate plausible but factually incorrect outputs, has been extensively surveyed, for example, in [54]. To mitigate this, various *self-improvement strategies* enabling LLMs to enhance their own reliability have been explored; a survey of automatic correction techniques is provided in [55], while [56] investigates enabling LLMs to self-correct their reasoning processes. A key underlying concept, termed the *Generation-Verification Gap (GV-Gap)* [57], highlights that LLMs are often more accurate at *verifying* a given answer's correctness than generating a correct answer initially. This principle motivates techniques where multiple candidate outputs are generated, and the LLM itself is used to evaluate and select the best one. For instance, [58] uses AI feedback for improving safety alignment (Constitutional AI), while [59] developed benchmarks to assess LLMs acting as judges. Another prominent paradigm is *iterative self-improvement*. In [11], the Self-Refine framework was proposed, allowing models to iteratively refine their outputs based on self-generated feedback. Similarly, [60] introduced the Reflexion technique, where agents learn to improve through verbal reinforcement derived from self-reflection. In [61], the authors propose the concept of *sharpening*, where a pre-trained LLM refines its outputs by using itself as a verifier during post-training. The effectiveness of such self-improvement capabilities is often linked to model scale; foundational *scaling laws* were established in [62], and later refined (e.g., in [63]), suggesting that larger models generally possess enhanced potential for self-correction and refinement.

Benchmarks like the BPI Challenge datasets have long served as standards for evaluating process mining algorithms [64]. However, these benchmarks primarily target traditional algorithmic approaches rather than LLM-based methods, limiting their applicability to our context. Several benchmarks for evaluating LLMs on process mining and BPM tasks are proposed. The benchmark in [65] assesses LLMs across a spectrum of process mining tasks, utilizing self-evaluation by LLMs to judge the quality of results. This approach contrasts with the benchmark proposed in this paper, which uses process descriptions aligned with ground truth models, allowing for a more informed and objective assessment of model quality. Further studies such as [66, 67] propose benchmarks that focus on causal reasoning and the explanation of decision points within business processes. Additionally, [68] introduces benchmarks for semantic-aware process mining tasks like semantic anomaly detection and next activity prediction.

This paper extends our previous work [8] by incorporating the following contributions. The evaluation of the proposed LLM-based process modeling framework has been significantly broadened to include a larger and more diverse set of processes, as well as a wider range of state-of-the-art LLMs. Additionally, a robust qualitative assessment methodology has been implemented, employing conformance checking against ground truth event logs to enable a more rigorous evaluation of the generated process models. Furthermore, we explore LLM self-improvement techniques to assess whether LLMs can autonomously refine their performance within our framework.

## 4 LLM-based process modeling framework

In this section, we present a detailed overview of our framework, which harnesses the capabilities of LLMs to generate and refine process models based on natural language process descriptions.

### 4.1 Framework overview

Figure 1 offers a high-level view of our proposed framework.

The framework begins by having users provide a textual description of a process in natural language. After receiving the process description, additional information is integrated to create a comprehensive prompt (the prompt engineering strategies are discussed in Sect. 4.3). This prompt is carefully crafted to instruct the LLM in generating executable code that can then be used to create process models (the selection of the modeling language is discussed in Sect. 4.2). A set of functions designed specifically for process model creation aids in this code generation. Once the prompt is prepared, it is sent to the LLM. Our framework is not dependent on any specific LLM and can function with any advanced LLM that supports a large context window and code generation. After receiving the LLM's response, we extract the generated code and attempt to execute it (details in Sect. 4.4).

In case errors are encountered during code extraction or execution, an error-handling mechanism is activated, sending a refined prompt back to the LLM to address the issue (discussed in Sect. 4.5). Upon successful execution and process model creation, users can view or export the model using established process modeling notations such as BPMN and Petri nets. Furthermore, the framework allows users to provide feedback on the generated model, which can then be incorporated to further refine the model, enabling continuous improvement.

### 4.2 Process representation

To explain the various stages of our framework, we implement an instance that uses the Partially Ordered Workflow Language (POWL) [9] for intermediate process representation. The framework's core principles allow for integration with other modeling languages depending on process modeling requirements. This section highlights the reasons behind our choice of POWL.

Our objective is to generate process models using common notations that professionals in the business process management and process mining fields are familiar with, such as BPMN and Petri nets. POWL represents a subclass of both BPMN and Petri nets, i.e., POWL models can be automatically transformed into Petri nets or BPMN models (cf. Sect. 2.2). POWL was selected as the intermediate representation for the following reasons:

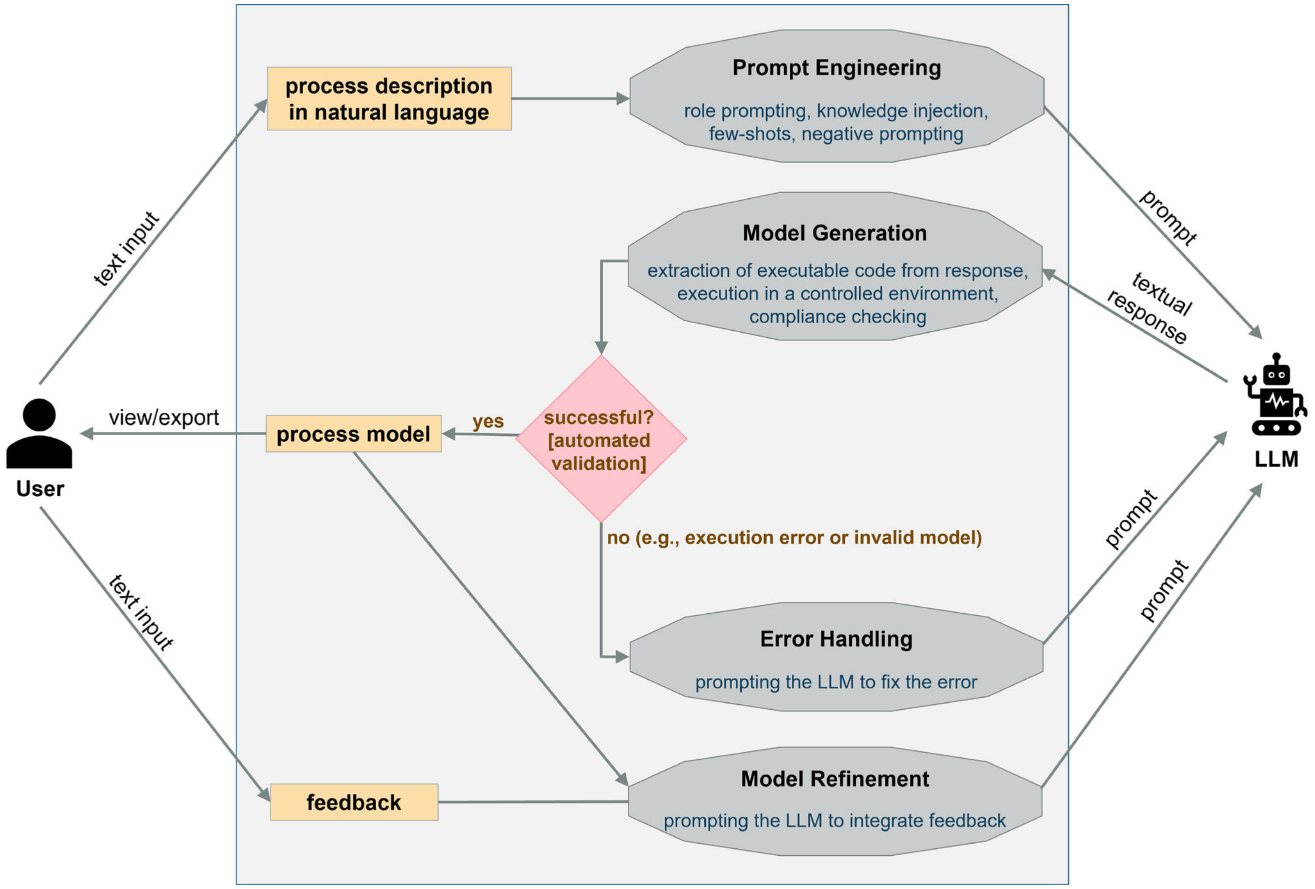

process description in natural language
Prompt Engineering
role prompting, knowledge injection, few-shots, negative prompting
text input
prompt
Model Generation
extraction of executable code from response, execution in a controlled environment, compliance checking
textual response
User
view/export
process model
yes
successful? [automated validation]
LLM
no (e.g., execution error or invalid model)
prompt
prompt
text input
Error Handling
prompting the LLM to fix the error
Model Refinement
prompting the LLM to integrate feedback
feedback

**Fig. 1** LLM-based process modeling framework

- *Soundness Guarantees:* Unlike BPMN and Petri nets, POWL inherently guarantees soundness.
- *Simplicity:* POWL's hierarchical structure simplifies model generation by enabling the recursive creation of submodels, which are then combined into larger models. It also assumes tasks are parallel unless otherwise specified, reflecting the concurrent nature of many real-world processes. This assumption simplifies model generation since task order does not always need to be explicitly defined.
- *Expressive Power:* While both POWL and process trees [21] ensure soundness, POWL supports a wider range of process structures [9]. It allows for the modeling of more complex dependencies while retaining the quality guarantees of hierarchical process modeling languages.

### 4.3 Prompt engineering

This section outlines the prompt engineering strategies we use to guide the LLM in generating process models from natural language descriptions.

The key strategies we implemented within our framework are:

- *Role Prompting:* This approach involves assigning a specific role to the LLM to shape its behavior [69]. We instruct the LLM to act as a process modeling expert, who is familiar with common process constructs. Additionally, we ask the LLM to act as a process owner, capable of filling in gaps in the process descriptions based on its expertise.
- *Knowledge Injection:* This strategy refers to injecting specific knowledge that the LLM may not have encountered during its training [70]. We provide comprehensive knowledge about POWL, offering detailed insights into its hierarchical structure and the semantics of the different POWL components. Moreover, our framework leverages LLM capabilities in generating executable code [6] by instructing the LLM to generate Python code that utilizes a predefined set of functions we designed for the safe generation of POWL models. We provide a detailed explanation of these predefined methods and how they can be used to generate POWL models. Listing 1 illustrates the knowledge injected about POWL.

- *Few-Shots Learning:* This technique involves providing the LLM with multiple example input–output pairs to train it on the task [25]. For instance, Listing 2 shows

**Listing 1** Knowledge injection on generating POWL models. Lines extending beyond the displayed text are abbreviated with "..." for compactness.

```
Use the following knowledge about the POWL modeling language:
  A POWL model is a hierarchical model. POWL models are recursively generated ...
  We define three types of POWL models. The first type is the base case ...
  For the second type of POWL models, we use an operator (xor or loop) to ...
  The third type of POWL models is defined as a partial order over n>=2 ...
Provide the Python code that recursively generates a POWL model. Save the final ...
Assume the class ModelGenerator is properly implemented and can be imported ...
ModelGenerator provides the functions described below:
 - activity(label) generates an activity. It takes 1 string argument, which is ...
 - xor(*args) takes n >= 2 arguments, which are the submodels. Use it to model ...
 - loop(do, redo) takes 2 arguments, which are the do and redo parts. Use it to ...
 - partial_order(dependencies) takes 1 argument, which is a list of tuples of ...
Note: for any powl model, you can call powl.copy() to create another instance ...
```

one of the few-shots examples we use for training the LLM to generate POWL models starting from process descriptions. Some of the process descriptions we use for implementing few-shots learning are adapted from [71].

- *Negative Prompting:* Negative prompting involves specifying what the LLM should avoid in its response [72]. We apply this strategy by instructing the LLM to avoid common mistakes that can occur during the generation of POWL models, such as generating partial orders that violate irreflexivity. Moreover, we extend our few-shots demonstrations with common mistakes that should be avoided during the construction of each process. For example, a typical mistake in modeling the bicycle manufacturing process (cf. Listing 2) is incorrectly representing the choice between accepting and rejecting the order as a simple XOR between the two activities "Reject order" and "Accept order". Instead, the choice should be modeled as an XOR between two paths—one path for rejecting the order and another path starting with accepting the order and including all the steps that follow acceptance. In other words, the steps that follow order acceptance (e.g., preparing, assembling, and shipping the bicycle) must be included within the acceptance path inside the XOR construct, so they are not reachable if the order is rejected.

## 4.4 Model generation and refinement

After receiving the LLM's response, the Python code snippet is extracted from the response, which might also include additional text (e.g., intermediate reasoning steps). If the code extraction is successful, then the extracted code is executed to generate the model. Executing code generated by an LLM involves multiple considerations to handle security risks and invalid results. The following strategies are implemented to ensure a safe environment for producing valid process models:

- In order to eliminate the risk of executing unsafe code, we restrict the LLM to use the predefined functions we designed for the generation of POWL models. We employ a strict process to verify that the code strictly complies with the prompted coding guidelines, explicitly excluding the use of external libraries or constructs that may pose security threats.
- Validation rules are in place to verify that the generated model conforms to the POWL specifications, such as the requirement that all partial orders respect transitivity and irreflexivity.

The framework supports displaying and exporting the generated POWL models as Petri nets or BPMN models for broader use within the business process management and process mining communities.

### Refinement Loop

The framework also includes a refinement loop, allowing users to provide textual feedback on the generated models. Based on this feedback, the LLM is prompted to revise the model, ensuring continual improvement.

## 4.5 Error handling

Despite their advanced capabilities, LLMs are not immune to generating faulty outputs. We employ a robust error-handling mechanism tailored to mitigate potential inaccuracies and ensure the reliability of the generated process models.

Recognizing the variability in the severity and implications of errors, we categorize them into two distinct groups:

- **Critical Errors:** These are severe issues that affect functionality or pose security risks, such as execution failures or major model validation violations. Critical errors halt the process and must be resolved before a valid model can be produced. Their presence typically indicates a major

**Listing 2** One of the examples used for few-shots learning and negative prompting. Lines extending beyond the displayed text are abbreviated with “...” for compactness.

~~~
Process description for example 1:
A small company manufactures customized bicycles. Whenever the sales department ...

Process model for example 1:
```python
from utils.model_generation import ModelGenerator
gen = ModelGenerator()
create_process = gen.activity(’Create process instance’)
reject_order = gen.activity(’Reject order’)
accept_order = gen.activity(’Accept order’)
inform = gen.activity(’Inform storehouse and engineering department’)
process_part_list = gen.activity  (’Process part list’)
check_part = gen.activity(’Check required quantity of the part’)
reserve = gen.activity(’Reserve part’)
back_order = gen.activity(’Back-order part’)
prepare_assembly = gen.activity(’Prepare bicycle assembly’)
assemble_bicycle = gen.activity(’Assemble bicycle’)
ship_bicycle = gen.activity(’Ship bicycle’)
finish_process = gen.activity(’Finish process instance’)

check_reserve = gen.xor(reserve, back_order)

single_part = gen.partial_order(dependencies=[(check_part, check_reserve)])
part_loop = gen.loop(do=single_part, redo=None)
accept_poset = gen.partial_order(
    dependencies=[(accept_order, inform),
                  (inform, process_part_list),
                  (inform, prepare_assembly),
                  (process_part_list, part_loop),
                  (part_loop, assemble_bicycle),
                  (prepare_assembly,  assemble_bicycle),
                 (assemble_bicycle, ship_bicycle)])

choice_accept_reject = gen.xor(accept_poset, reject_order)

final_model = gen.partial_order(
    dependencies=[(create_process,  choice_accept_reject),
                  (choice_accept_reject, finish_process)])
```

Common errors to avoid for example 1:
creating a local choice between ’reject_order’ and ’accept_order’ instead of ...
~~~

misunderstanding by the LLM of the task requirements. Example critical errors include:

- Failure to extract a code snippet from the response, indicating that the LLM either did not generate code as requested or used incorrect formatting.
- Syntax errors in the generated code that prevent its execution.
- Inclusion of unsafe constructs, such as attempts to import unauthorized external libraries.
- Attempts to use non-existent constructs or parameters due to LLM hallucinations.
- Violations of POWL’s structural constraints, for example, generating partial orders with cyclic dependencies.

- **Adjustable Errors:** These are less severe issues that affect the quality or optimality of the process model but do not necessarily prevent its successful generation. These errors can be adjusted automatically, allowing for a degree of flexibility in their resolution. However, such intervention is approached with caution to prevent significant deviations from the behavior of the intended process. An example of an adjustable error is as follows:

  - **Error:** In an order handling process, the LLM attempts to reuse the activity “Pay” twice within the process—once to model a choice between paying or canceling the order (i.e., $\times$(Cancel, Pay)) and once in a partial order to specify that the payment occurs after item selection and before delivery (i.e., a sequence $\rightarrow$ (Select, Pay, Deliver)).

**Listing 3** Example illustrating an error-handling prompt sent back to the LLM after a validation failure.

```
Executing your code led to an error! Please update the model to fix the error.
Make sure to save the updated final model in the variable 'final_model'.
Error occurred at line 12: 'initial_order = gen.partial_order(dependencies)'
Exception: The transitive closure of the provided relation violates irreflexivity!
```

  - **Auto-adjustment:** This error can be automatically resolved by creating two distinct copies of the activity "Pay".
  - **Implication:** The generated process model after auto-adjustment deviates from the intended behavior since there should only be one payment activity in the process. Rather than duplicating the activity, the two submodels should be combined to incorporate the delivery activity within the choice (i.e., → (Select, ×(Cancel, → (Pay, Deliver)))).

Our framework uses an iterative error-handling loop, engaging the LLM to address the identified issues. A new prompt that details the error and requests the LLM to address it, along with the conversation history, are submitted to the LLM. An example of such a prompt is shown in Listing 3. This iterative cycle facilitates dynamic correction, leveraging the LLM's capabilities to refine and improve the generated code.

Critical errors are handled by prompting the LLM repeatedly until a solution is found or the maximum allowed attempts are reached. If the LLM fails to fix the error after the allowed number of attempts, the system terminates the process and marks the model generation as unsuccessful. Adjustable errors are resolved automatically if the LLM fails to address them within a few iterations.

## 5 Tool support

In this section, we present the ProMoAI tool [73], which supports our LLM-driven process modeling framework. ProMoAI is implemented as a standalone Python library (https://pypi.org/project/promoai/) and as a web application deployed using Streamlit (https://promoai.streamlit.app/). In a forthcoming release, ProMoAI will also be available as a data application on the KNIME Analytics Platform [74], expanding its reach to a broader audience of data analysts.

### 5.1 Backend: the python library

The core of ProMoAI is implemented as a modular Python library. This backend layer encapsulates the process modeling logic and serves as the foundation for all functionalities. Key components include:

- **Core Engine:** Including prompt engineering, code extraction, model generation and validation, error handling, and iterative model re-design.
- **LLM Integration:** Providing seamless support for several LLM providers. While originally demonstrated with OpenAI's models [73], the architecture supports integration with other providers with minimal adaptation, enhancing flexibility and future-proofing the system against advances in AI technology. Currently, the following providers are supported:
  - **Google:** Providing the Gemini models (https://ai.google.dev/).
  - **OpenAI:** Providing GPT-4 and O1 models and their variants (https://openai.com/).
  - **DeepSeek:** Providing advanced chatting and reasoning DeepSeek models (https://www.deepseek.com/).
  - **Anthropic:** Providing Claude models (https://www.anthropic.com/).
  - **DeepInfra:** Supporting popular open-source large language models like Meta's LLaMa and also enabling custom model deployment (https://deepinfra.com/).
  - **Mistral AI:** Providing the Mistral and Mixtral models (https://mistral.ai/).
- **Input Handling:** Accepting three types of inputs:
  - **Text:** A natural language description of a process.
  - **Process Model:** An existing semi-block-structured Petri net or BPMN [75]. This additional feature was recently added to ProMoAI and is beyond the scope of this paper.
  - **Event Data:** An event log in the XES format. When an event log is uploaded as the initial input, the process discovery algorithm from [22] is used to generate the initial process model. Support for this input type was also recently added as a new feature in ProMoAI and is beyond the scope of this paper.
- **Model Export and Visualization:** Conversion of the generated POWL models into BPMN and Petri nets, exporting them in the corresponding formats, and visualization of the models. These features are powered by algorithms implemented in the Python library PM4Py [76].

The backend library can be installed in Python with the command `pip install promoai`. This design ensures that ProMoAI can be easily integrated into custom Python workflows independent of its user interface.

## 5.2 Frontend: streamlit deployment

The current user interface for ProMoAI is built using Streamlit (https://streamlit.io/), providing a web-based environment for users who may not wish to interact directly with the Python library. A screenshot of the tool is shown in Fig. 2. Key aspects of the Streamlit deployment include:

- **LLM Configuration:** Users can select their desired AI provider, specify the LLM name, and enter the required API key through an intuitive configuration panel.
- **Multi-Input Interface:** The frontend supports all three input types through intuitive text and upload widgets. Users can easily enter the process description as plain text or drag and drop files (existing process models or event logs) to move directly into the refinement step.
- **Visualization and Export Options:** Users can select between three view options (POWL, BPMN, and Petri net); the corresponding visualization will be rendered as an SVG image. Users can also use the two export buttons to download the model in the BPMN and PNML formats.
- **Interactive Feedback Loop:** After generating an initial process model, users can provide textual feedback to refine the model. After clicking on the "Update Model" button, the feedback comment is sent to the LLM to refine the model, and the model will be updated accordingly. Additionally, the full feedback history is displayed to the user.

The Streamlit application is hosted on the cloud and is accessible online at https://promoai.streamlit.app/. It can also be hosted locally by cloning the public GitHub repository available at https://github.com/humam-kourani/ProMoAI.

# 6 Benchmarking state-of-the-art large language models

In this section, we design a robust evaluation approach for benchmarking LLMs on process modeling and we apply it to assess a diverse set of state-of-the-art LLMs. The objective is to assess our framework's capability to effectively generate high-quality business process models from natural language descriptions. Additionally, our evaluation serves as a benchmark for assessing the capabilities of state-of-the-art LLMs in a task that involves (1) modeling business processes starting from natural language descriptions, (2) generating executable code, (3) following instructions embedded in the input prompts, and (4) incorporating feedback to iteratively resolve errors and improve the quality of the outputs.

We start by detailing the evaluation design in Sect. 6.1. Afterwards, we address the selection of LLMs in Sect. 6.2 and we discuss the obtained results in Sect. 6.3. Finally, in Sect. 6.4, we conduct two supplementary analyses to explore specific factors that may influence LLM behavior within our framework. Note that all data and results are available at https://github.com/humam-kourani/EvaluatingLLMsProcessModeling
.

## 6.1 Comparative evaluation approach

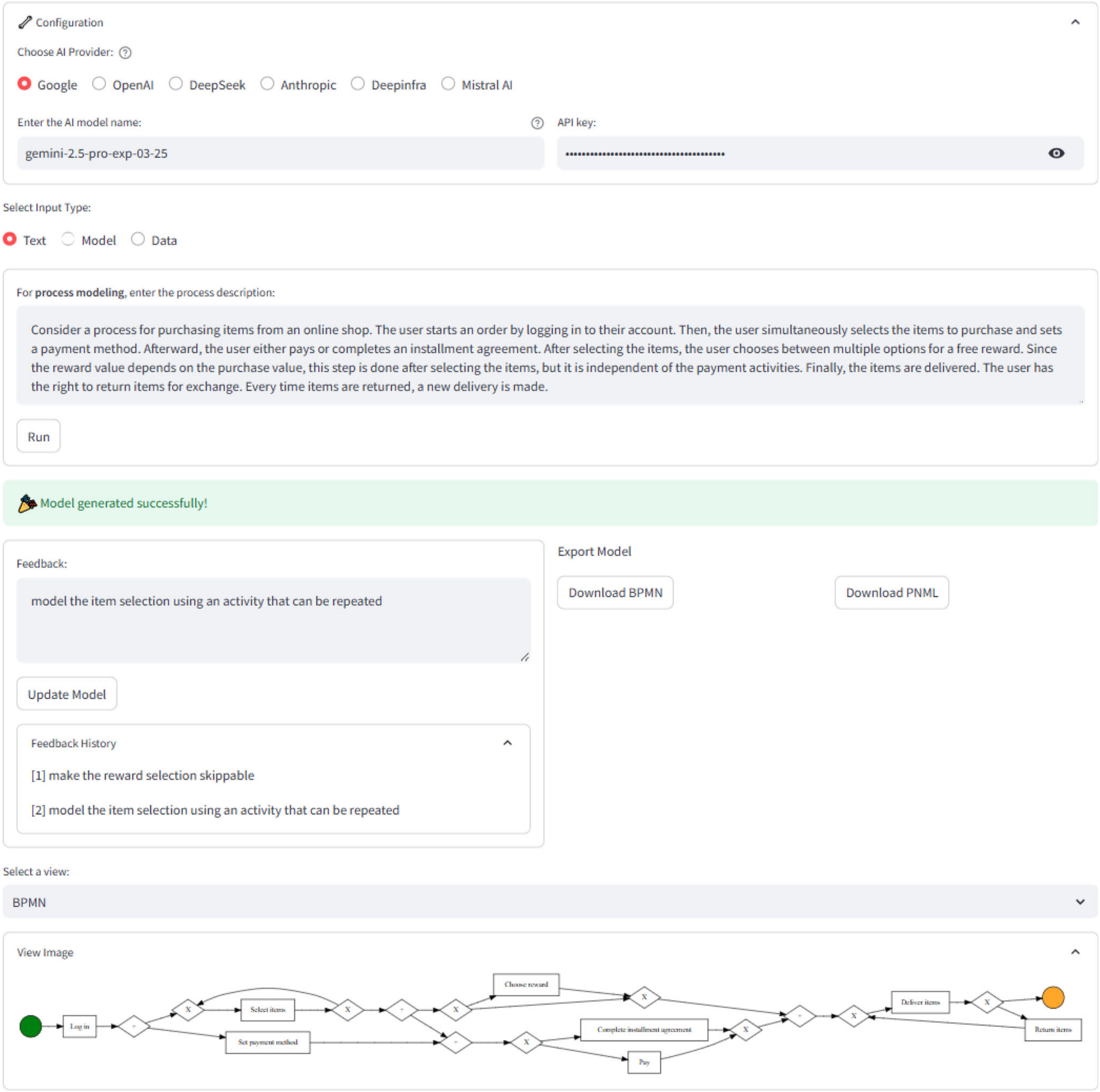
While evaluating specific LLMs may provide useful insights at a given point in time, such assessments quickly become obsolete as the technology continues to evolve. Therefore, rather than focusing solely on individual model evaluations, we propose a comparative approach that offers a more lasting contribution. This approach is designed to facilitate the comparison of future LLMs on business process modeling, including the evaluation of their self-improvement capabilities (cf. Sect. 7). Our evaluation approach not only supports the assessment of LLMs as they are developed but also ensures that comparisons remain relevant and adaptable to the rapidly changing landscape of LLM technology.

In this section, we describe the components of our comparative evaluation approach. Specifically, we outline the design of the processes we use in our evaluation in (Sect. 6.1.1). We then explain our method for generating ground truth event logs via simulation (Sect. 6.1.2). Afterwards, we present our approach for assessing the quality of LLM-generated process models using conformance checking (Sect. 6.1.3). Finally, we describe the configuration for handling errors during model generation (Sect. 6.1.4).

### 6.1.1 Design of processes

To evaluate the LLMs within our framework, we designed a set of 20 distinct pairs of process descriptions and their corresponding ground truth POWL model. We refer to these processes as p1, ..., p20 throughout this paper. Two processes were adapted from our previous work [8]: an order handling process [9] and a hotel service process [71]. The remaining 18 processes were created to represent diverse business domains, including manufacturing, healthcare, finance, logistics, customer service, and more. This diversity ensures that the evaluation is not biased toward a specific industry or process type.

Both the process descriptions and the ground truth process models were manually designed. The POWL models were not discovered from existing descriptions but rather constructed alongside them to ensure alignment. That is,



Configuration

Choose AI Provider:

● Google ○ OpenAI ○ DeepSeek ○ Anthropic ○ Deepinfra ○ Mistral AI

Enter the AI model name: gemini-2.5-pro-exp-03-25

API key: ••••••••••••••••••••••••••••••••••••••••

Select Input Type:

● Text ○ Model ○ Data

For **process modeling**, enter the process description:

Consider a process for purchasing items from an online shop. The user starts an order by logging in to their account. Then, the user simultaneously selects the items to purchase and sets a payment method. Afterward, the user either pays or completes an installment agreement. After selecting the items, the user chooses between multiple options for a free reward. Since the reward value depends on the purchase value, this step is done after selecting the items, but it is independent of the payment activities. Finally, the items are delivered. The user has the right to return items for exchange. Every time items are returned, a new delivery is made.

Run

Model generated successfully!

Feedback: model the item selection using an activity that can be repeated

Update Model

Feedback History

[1] make the reward selection skippable

[2] model the item selection using an activity that can be repeated

Export Model

Download BPMN

Download PNML

Select a view: BPMN

View Image

**Fig. 2** A screenshot of ProMoAI

the processes are constrained to the structures expressible in POWL. This intentional design avoids the inclusion of process elements that lack clear semantics or that our framework cannot model due to language limitations. This approach allows us to assess the LLM's performance realistically without introducing challenges that fall outside the scope of our framework.

The ground truth POWL models were converted into Petri nets (using the algorithm from [23]) and saved in the PNML format. The process descriptions are available at https://github.com/humam-kourani/EvaluatingLLMsProcessModeling/tree/main/ground_truth/ground_truth_process_descriptions, while the ground truth process models are available at https://github.com/humam-kourani/EvaluatingLLMsProcessModeling/tree/main/ground_truth/ground_truth_pn.

The ground truth processes were intentionally designed to vary along several dimensions. This includes:

- **Process Description Length**: Ranged from 79 to 230 words and from 525 to 1,567 characters.
- **Level of Detail**: Process descriptions were crafted to include essential elements such as key activities and indi-

**Table 1** Characteristics of the 20 processes used in the evaluation

| Process | Process Description | | Ground Truth Process Model | | | | |
|---|---|---|---|---|---|---|---|
| | #Words | #Chars | #Activities (non-silent) | #Petri Net Nodes | #Choices | #Loops | #Partial Orders |
| p1 | 88 | 578 | 8 | 26 | 2 | 0 | 3 |
| p2 | 97 | 629 | 16 | 43 | 2 | 2 | 1 |
| p3 | 94 | 601 | 11 | 27 | 2 | 1 | 1 |
| p4 | 98 | 659 | 10 | 28 | 1 | 0 | 1 |
| p5 | 90 | 573 | 11 | 26 | 1 | 1 | 2 |
| p6 | 92 | 580 | 10 | 27 | 2 | 1 | 1 |
| p7 | 85 | 565 | 13 | 49 | 4 | 2 | 5 |
| p8 | 92 | 593 | 12 | 24 | 1 | 0 | 1 |
| p9 | 97 | 644 | 13 | 34 | 1 | 1 | 2 |
| p10 | 80 | 525 | 9 | 35 | 3 | 1 | 2 |
| p11 | 87 | 605 | 12 | 43 | 2 | 1 | 4 |
| p12 | 93 | 654 | 10 | 28 | 1 | 2 | 2 |
| p13 | 99 | 637 | 9 | 21 | 2 | 0 | 2 |
| p14 | 126 | 837 | 10 | 20 | 0 | 1 | 3 |
| p15 | 79 | 556 | 10 | 26 | 1 | 0 | 2 |
| p16 | 219 | 1457 | 24 | 63 | 3 | 1 | 7 |
| p17 | 196 | 1327 | 25 | 62 | 6 | 0 | 4 |
| p18 | 230 | 1567 | 26 | 78 | 8 | 2 | 7 |
| p19 | 109 | 653 | 8 | 23 | 1 | 1 | 1 |
| p20 | 162 | 959 | 13 | 32 | 2 | 0 | 2 |

cations of decisions (e.g., describing choices or repeated behavior), ensuring sufficient information for control-flow modeling. However, the descriptions varied in how explicitly they specified structural elements. While none of the descriptions were truly vague, certain structural details were deliberately left ambiguous to investigate the LLM's ability to reason about process structure. For example, some descriptions did not specify whether tasks should be executed concurrently or sequentially. Similarly, we did not explicitly state that a task or a path is skippable in cases where this could be easily inferred from the context. This approach reflects real-world ambiguity and challenges the LLM to infer appropriate behavior based on the process context.

- **Process Size**: The number of activities in the ground truth models ranged from 8 to 26, providing a mix of simple and complex processes.
- **Structural Complexity**: The processes were designed to cover different levels of structural complexity for the three process constructs supported by POWL:
  - *Choices*: Included processes with skippable activities, simple choices between single activities, and choices involving complex sub-processes.
  - *Loops*: Incorporated processes with repeatable activities, simple loops over single activities, and loops involving complex sub-processes.
  - *Partial Orders*: Varied from highly sequential processes to those with high concurrency, as well as complex partial orders that contain non-hierarchical dependencies.

To provide further insight into the characteristics of our process descriptions and their corresponding ground truth models, Table 1 summarizes key dimensions. For each process description, we report the number of words and characters. For each ground truth Petri net, we report the number of non-silent activities, the total number of nodes (transitions and places), and metrics reflecting its structural complexity: the number of choice, loop, and partial order operators used in the original POWL model before its conversion to a Petri net. Listing 4 and Listing 5 show two example textual descriptions, illustrating variations in length and complexity. The corresponding ground truth models are shown as BPMN diagrams in Fig. 3, providing a visual reference for their structure.

### 6.1.2 Event log generation

Assessing the quality of LLM-generated process models directly against the input natural language description poses significant challenges for objective, automated, and reproducible evaluation. To overcome this, we adopted a quantitative evaluation approach leveraging conformance checking

**Listing 4** Textual description for process p9 (644 characters, 97 words).

```
The process starts with identifying an idea for a new product or improvement to an
existing one. The R\&D team conducts initial research and feasibility studies,
followed by drafting design concepts. After selecting a promising design, a
prototype is built using available materials and resources. The prototype
undergoes various tests to assess its functionality, safety, and market potential.
Feedback from the testing phase is collected, and the prototype may be refined
accordingly. If a refinement is needed, then the testing phase is reinitiated. The
process ends when the prototype is either approved for further development or discarded.
```

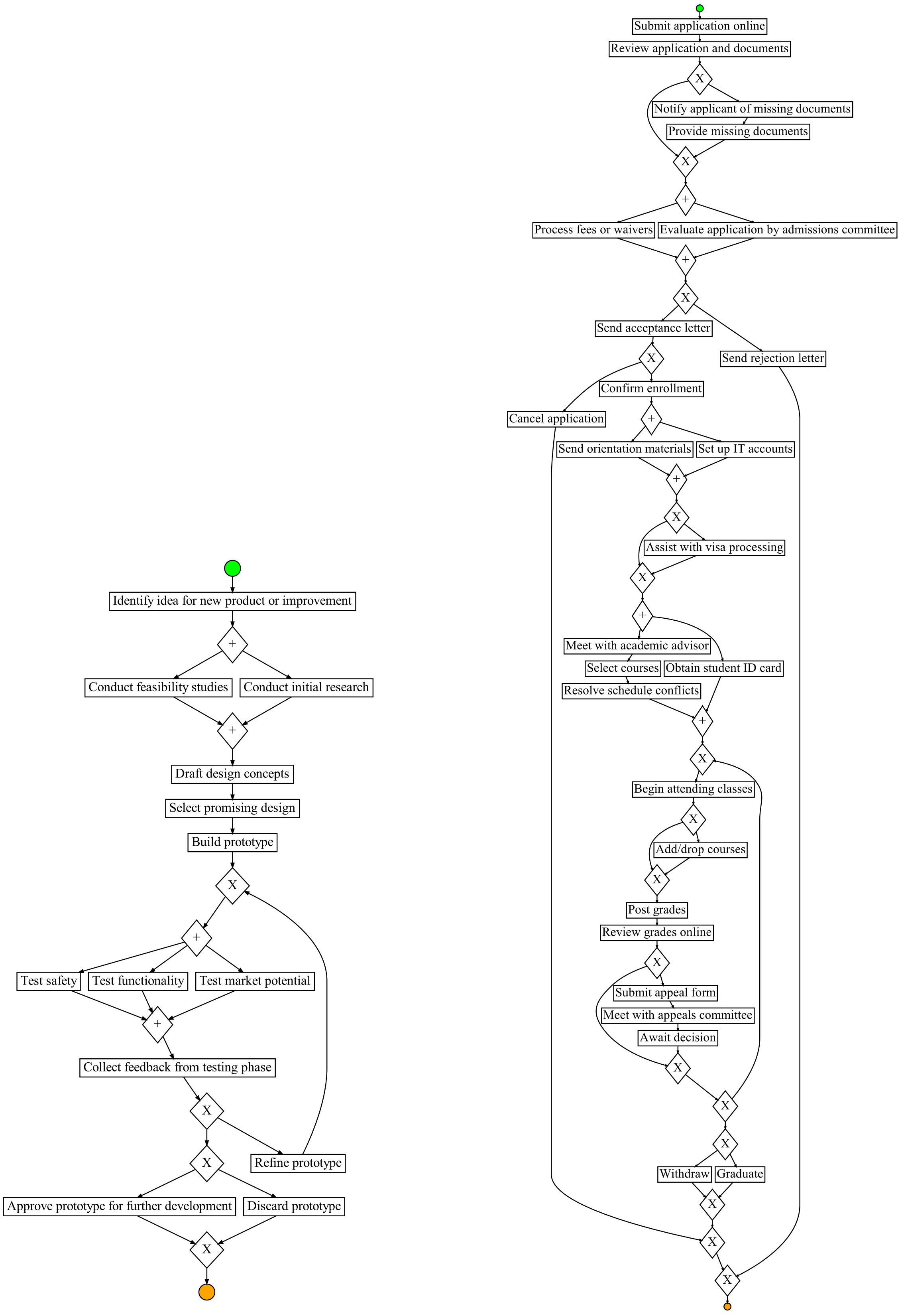
**Listing 5** Textual description for process p18 (1567 characters, 230 words).

```
A university enrollment system involves the following steps:
Prospective students submit an application online.
The admissions office reviews the application and supporting documents.
If documents are missing, the applicant is notified to provide the missing items.
Upon receiving all documents, the application is evaluated by the admissions
committee.
Concurrently, the finance department processes any application fees or waivers.
If the application is accepted, an acceptance letter is sent. Otherwise, a
rejection letter is sent and the process ends.
After being accepted, the student must then confirm enrollment by a specified
deadline; otherwise the application will be canceled.
If the student confirms, they receive orientation materials and the IT department
sets up student accounts for email, online portals, and library access.
If the student is international, the international student office assists with visa
 processing.
The student obtains a student ID card and starts creating their study plan, which
includes:
Meeting with an academic advisor.
Selecting courses.
Resolving any schedule conflicts.
The student begins attending classes.
Throughout each semester, the student may add or drop courses within the add/drop
period.
At the end of the semester, grades are posted, and the student can review them online.
If the student has any grievances, they can file an appeal, which includes:
Submitting an appeal form.
Meeting with the appeals committee.
Awaiting a decision.
The process repeats each semester until the student graduates or withdraws.
```

techniques (cf. Sect. 2.5). To facilitate quantitative evaluation through conformance checking, we simulated event logs from the ground truth POWL models. These logs serve as the basis for assessing the quality of the models generated by the LLMs using our framework.

The simulation was meticulously conducted to produce comprehensive and representative event logs from the ground truth models. This process is based on two key assumptions to ensure both feasibility and thorough coverage of possible process behaviors. First, we assume that all decision points within the process follow an equal distribution, meaning each possible choice at a decision point has an equal probability of being selected. This assumption simplifies the simulation by treating all paths through decision points as equally likely, thereby ensuring unbiased exploration of all possible *trace variants* (i.e., distinct activity sequences). Second, we limit loops to a maximum length of two iterations, allowing only up to two executions of the loop's do-part. This constraint is necessary to prevent the generation of infinitely long event logs. By capping the loop iterations, we maintain a manageable size for the event logs while still capturing the essential repetitive behavior of the process.

We performed the simulation using both process tree and Petri net playout techniques available in PM4Py [76]. For models that can be translated into process trees, we used the process tree playout method because it is highly scalable and allows us to set explicit limits on loop depth to match our assumption. For models involving complex partial order constructs that cannot be captured with process trees, we used the extensive Petri net playout algorithm. In these cases, we applied a hybrid approach that uses the Petri net playout method only for the specific submodels requiring it


Identify idea for new product or improvement
Conduct feasibility studies
Conduct initial research
Draft design concepts
Select promising design
Build prototype
Test safety
Test functionality
Test market potential
Collect feedback from testing phase
Refine prototype
Approve prototype for further development
Discard prototype
Submit application online
Review application and documents
Notify applicant of missing documents
Provide missing documents
Process fees or waivers
Evaluate application by admissions committee
Send acceptance letter
Send rejection letter
Confirm enrollment
Cancel application
Send orientation materials
Set up IT accounts
Assist with visa processing
Meet with academic advisor
Select courses
Obtain student ID card
Resolve schedule conflicts
Begin attending classes
Add/drop courses
Post grades
Review grades online
Submit appeal form
Meet with appeals committee
Await decision
Withdraw
Graduate

(a) R&D process (p9).

(b) University enrollment process (p18).

**Fig. 3** Ground truth models for processes p9 and p18

and the process tree playout method for simpler submodels. This hybrid strategy ensured that our final event logs accurately reflect the intended process behavior without solely relying on the Petri net playout approach, which suffers from scalability issues.

Our simulation method is designed to achieve complete coverage of all traces, given our assumption on loop depth. Each simulated event log includes one instance of every unique trace variant possible within the ground truth model, thereby ensuring comprehensive coverage of the process behavior. The simulated event logs are available at https://github.com/humam-kourani/EvaluatingLLMsProcessModeling/tree/main/ground_truth/ground_truth_xes_one_trace_per_variant.

#### 6.1.3 Quality assessment

With the simulated ground truth event logs, we assess the quality of the LLM-generated models using conformance checking techniques. We assess the quality of the LLM-generated models in relation to the ground truth event logs by calculating fitness and precision metrics. Specifically, we employ the token-based fitness metric [77] and the ET-Conformance precision metric [78], both implemented in the PM4Py library [76]. The overall quality score for each generated model is determined as the harmonic mean of the fitness and precision values. High scores (approaching 1.0) indicate a high level of conformance with the ground truth event logs, whereas lower scores (closer to 0.0) reflect poor model quality. This quality assessment is performed exclusively on the final model that is successfully generated after the error-handling loop concludes; intermediate outputs with errors are not evaluated.

***Standardizing Activity Labels***

To ensure accurate conformance checking between LLM-generated process models and the ground truth event logs, it is essential to standardize activity labels. Without this standardization, inconsistencies in activity naming can result in misleading quality scores, as LLM-generated models may assign different labels to equivalent activities, introduce new activities, or combine multiple activities into a single one. To address this, we extend the initial prompt with the list of activity labels of the ground truth log, presented in a random order to prevent the LLM from inferring sequential dependencies based on label ordering. We instruct the LLM to generate a process model using the same activity labels. While this may limit the LLM's flexibility in naturally expressing process steps, it mirrors real-world scenarios where key process steps are known, even if the overall process model is not formally defined. Moreover, this standardizing is crucial for enabling conformance checking and automated quantitative assessment.

#### 6.1.4 Error handling settings

Our framework incorporates an error-handling loop. When an error is detected, a new prompt is sent to the LLM containing details of the error and the relevant context from previous iterations, instructing it to correct the issue (cf. Sect. 4.5). We use a threshold of ten iterations for addressing adjustable errors by the LLM. If errors persist beyond these ten iterations, the framework automatically attempts to fix them, which may impact the overall quality of the generated models. For example, if the LLM reuses a sub-model within the same POWL model twice, our system automatically applies a predefined correction rule that creates a copy of the reused sub-model to resolve the issue.

After auto-resolving adjustable errors, if critical errors remain (e.g., a syntax error), an additional five iterations for handling the errors by the LLM are permitted. By setting high iteration thresholds, we aim to assess the LLMs' proficiency in understanding and correcting their outputs based on feedback and to ensure fairness by providing ample opportunity for each model to succeed.

### 6.2 Selection of large language models

To ensure a comprehensive and representative evaluation of our process modeling framework, we selected LLMs based on a diverse set of criteria. Our evaluation methodology is designed to be flexible and easily extendable, allowing new models to be seamlessly integrated as advancements in LLM technology emerge. Below, we list the key criteria that guided our selection process:

- **State-of-the-Art Performance:** At the time of conducting the experiments, we aimed at including models that were considered cutting edge.
- **Diverse Vendor Coverage:** We included models from major AI providers (e.g., OpenAI, and Google, and Anthropic) to avoid potential vendor bias and capture a broad market perspective.
- **Inclusion of Open-Source Models:** To evaluate freely available alternatives, we ensured that prominent open-source models (e.g., LLaMa and Mistral) were part of the selection.
- **Community Adoption:** We included models that have substantial community usage and were heavily featured in recent studies (e.g., GPT-4 and Gemini), ensuring that our evaluation reflects popular, real-world applications.
- **Varied Architectures:** We selected models with varied architectures, including transformer-based and mixture-of-experts models.
- **Context Window Capacity:** We only selected models with a context window of at least 16k tokens, ensuring compatibility with our framework.

- **Varied Sizes:** We chose models ranging from smaller, faster variants to very large models (e.g., GPT-4o vs. GPT-4o-Mini).
- **Varied Specializations:** Our selection covers models with different strengths, including those optimized for speed (e.g., Gemini-1.5-Flash-002), reasoning (O1 models), explicit prompt following (e.g., LLaMa instruct models), and code generation (Codestral).
- **Accessibility and Reproducibility:** We selected models that are readily accessible through public APIs or open-source deployments, facilitating the reproducibility of our experiments.

Based on aforementioned criteria, we selected 16 state-of-the-art LLMs. The selected models are:

- **GPT-4**, **GPT-4o**, and **GPT-4o-Mini**: Developed by OpenAI (https://openai.com/), GPT-4 is renowned for its advanced reasoning abilities, extensive knowledge base, and large context window. It excels in tasks requiring deep understanding and generation of coherent, contextually relevant text. GPT-4o, and GPT-4o-Mini are optimized versions of GPT-4, offering faster performance and reduced computational requirements while maintaining high-quality outputs.
- **O1-Preview** and **O1-Mini**: The O1 series contains the latest advancements from OpenAI, outperforming previous models like GPT-4o across competitive benchmarks. O1-Preview is designed for deep reasoning, excelling in complex tasks like math, coding, and science. O1-Mini is a smaller, faster, and more cost-effective version.
- **Gemini-1.5-Pro-002** and **Gemini-1.5-Flash-002**: Developed by Google (https://ai.google/), the Gemini models are trained on diverse datasets. The Pro version is designed for enhanced reasoning capabilities, while the Flash variant emphasizes speed and efficiency.
- **Claude-3.5-Sonnet**: From Anthropic (https://www.anthropic.com/), Claude-3.5-Sonnet is designed for high performance with a focus on safety and reliability. It operates with a 200k token context window and is optimized for code generation and complex reasoning tasks.
- **Mistral-Large-2**, **Codestral**, and **Mixtral-8x22B**: These models from Mistral AI (https://mistral.ai/) are designed for efficient training and inference with support for multilingual tasks. Mistral-Large-2 is top-tier reasoning model provided by Mistral AI for high-complexity tasks. Mixtral-8x22B is particularly noteworthy due to its mixture-of-experts architecture, which allows it to activate only a subset of its parameters during each inference, making it highly efficient. Codestral is optimized for generating and understanding code in a wide array of programming languages.
- **Llama-3.1-405B-Instruct** and **Llama-3.2-90B-Vision-Instruct**: These open-source models from Meta (https://ai.meta.com/) are trained on extensive corpora and designed for instruction following and complex reasoning tasks. Llama-3.1-405B-Instruct is notable for its large parameter size, enhancing its capability to handle intricate tasks.
- **Llama-3.1-Nemotron-70B-Instruct**: Developed by Nvidia (https://www.nvidia.com/) as an advanced version of Meta's Llama-3.1-70B, Nemotron leverages Nvidia's cutting-edge hardware and fine-tuning techniques to offer high-performance capabilities.
- **Qwen2.5-72B-Instruct**: Developed by Alibaba Cloud (https://www.alibabacloud.com/), Qwen2.5-72B-Instruct is a powerful open-source model known for its multilingual support and proficiency in handling complex instructions.
- **WizardLM-2-8x22B**: This is an advanced open-source model designed by Microsoft (https://www.microsoft.com/).

Note that for the open-source models Llama-3.1-405B-Instruct, Llama-3.2-90B-Vision-Instruct, Llama-3.1-Nemotron-70B-Instruct, Qwen2.5-72B-Instruct, and WizardLM-2-8x22B, we used the instances hosted by Deep Infra (https://deepinfra.com/).

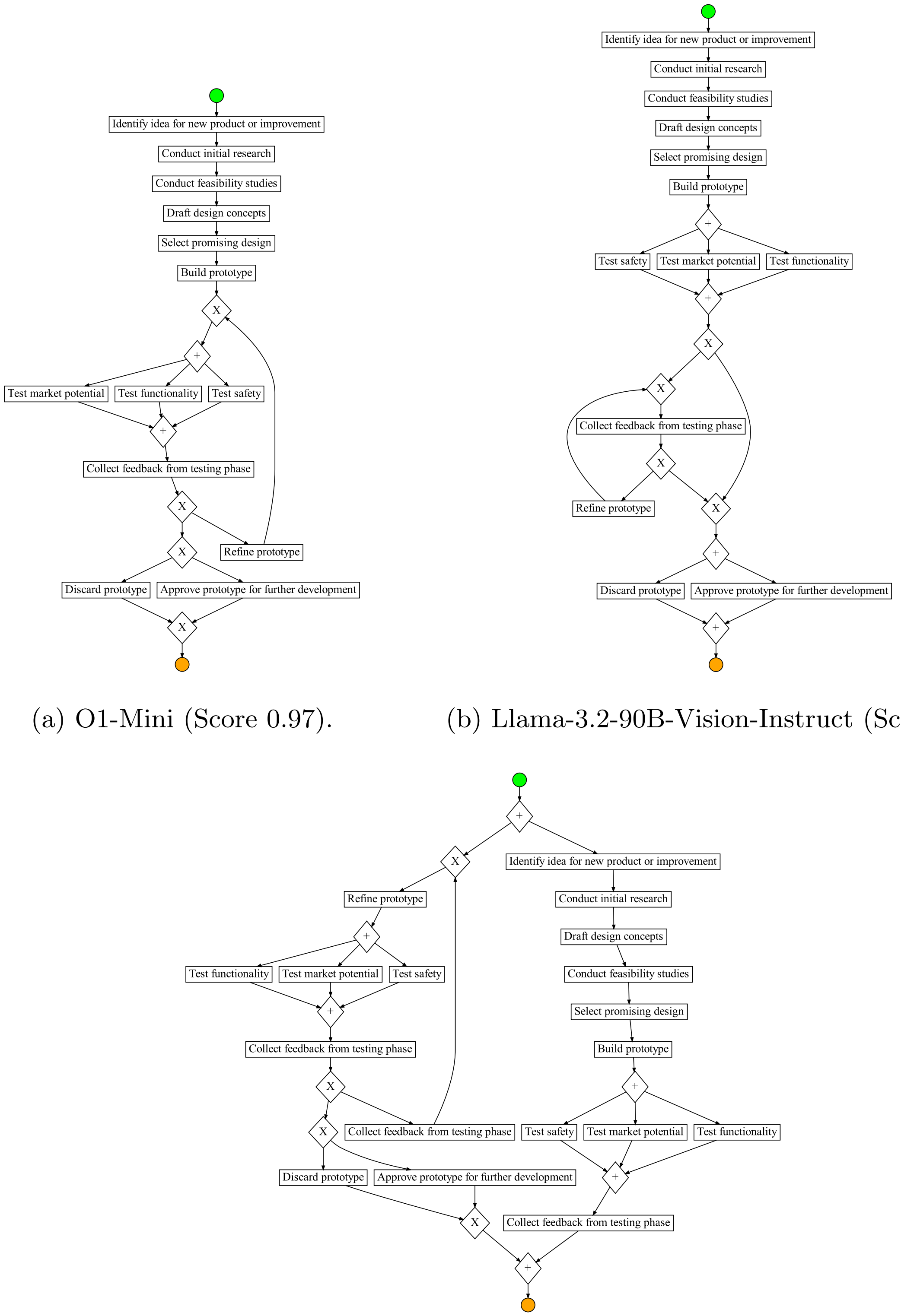
## 6.3 Evaluation results

In this section, we present the results of our evaluation, focusing on the error-handling performance (Sect. 6.3.1), quality of the generated models (Sect. 6.3.2), time efficiency (Sect. 6.3.3), and overall observations (Sect. 6.3.4).

Figure 4 shows three example models generated for the R&D process (p9). The high-quality model generated with O1-Mini (Score: 0.97) closely matches the ground truth model (Fig. 3a). In contrast, the model from Llama-3.2-90B-Vision-Instruct (Score: 0.83) contains a major flaw: while it correctly identifies the loop for refining the prototype, it incorrectly places the testing activities outside the loop body. The model generated by Codestral (Score: 0.56) exhibits even more critical issues, introducing two incorrect concurrent branches and duplicating activities within them.

### 6.3.1 Error handling performance

We analyzed how each LLM performed in terms of the number of iterations required to generate a valid process model without errors. In Table 2, we report the average number of iterations required to produce a valid model by each LLM, the number of cases where only one iteration was needed (indicating no errors), the number of instances where more than ten iterations were necessary (indicating



(a) O1-Mini (Score 0.97).

(b) Llama-3.2-90B-Vision-Instruct (Score 0.83).

(c) Codestral (Score 0.56).

**Fig. 4** LLM-generated models for the R&D process (p9)

**Table 2** Error handling performance metrics

| Model | Avg. Num. Iterations | Num. Cases without Errors | Num. Cases with Auto-Adjustment | Num. Cases with Failures |
|---|---|---|---|---|
| Claude-3.5-Sonnet | 1.35 | 16 | 0 | 0 |
| O1-Mini | 1.4 | 14 | 0 | 0 |
| O1-Preview | 1.5 | 14 | 0 | 0 |
| Gemini-1.5-Pro-002 | 1.95 | 13 | 0 | 0 |
| GPT-4o | 2.25 | 9 | 0 | 0 |
| Llama-3.1-405B-Instruct | 2.55 | 9 | 0 | 0 |
| Mistral-Large-2 | 2.6 | 10 | 0 | 0 |
| Llama-3.2-90B-Vision-Instruct | 2.95 | 8 | 0 | 0 |
| Gemini-1.5-Flash-002 | 3.3 | 4 | 0 | 0 |
| Mixtral-8x22B | 3.6 | 10 | 3 | 1 |
| GPT-4 | 3.9 | 2 | 0 | 0 |
| Codestral | 3.9 | 7 | 2 | 1 |
| GPT-4o-Mini | 4.05 | 6 | 3 | 0 |
| Llama-3.1-Nemotron-70B-Instruct | 4.05 | 3 | 1 | 0 |
| Qwen2.5-72B-Instruct | 4.65 | 4 | 2 | 0 |
| WizardLM-2-8x22B | 5.2 | 8 | 5 | 0 |

failure to resolve adjustable errors), and the number of cases where no valid model was generated after 15 iterations (indicating failure to resolve critical errors). In addition to the metrics reported in this section, Sect. 6.3.3 analyzes time performance and the impact of errors, comparing the average time per iteration and the average total generation time for each LLM.

We do not undertake a detailed error-type analysis in this evaluation because we observed that the vast majority of errors were validation-related. These errors typically stemmed from misinterpretations of POWL semantics (e.g., generating partial orders with cycles), particularly in processes with higher structural complexity. This suggests that the LLMs generally succeeded in following our prompt instructions to generate syntactically valid code, and that most issues arose from the inherent complexity of the modeled behavior rather than from basic generation failures.

Claude-3.5-Sonnet demonstrated exceptional performance, with an average of 1.35 iterations and 16 out of 20 instances where only one iteration was sufficient to produce a valid model. This performance was closely mirrored by the O1 models and Gemini-1.5-Pro-002, which averaged less than two iterations and managed to produce models without any errors in 14 and 13 instances, respectively. These results suggest that these models possess robust reasoning capabilities and an ability to generate high-quality results on the first attempt.

Among the models evaluated, WizardLM-2-8x22B stood out with the highest average number of iterations (5.2) and five instances where more than 10 iterations were needed. However, it is notable that despite these higher iteration counts, WizardLM-2-8x22B did not experience any failures. Models such as GPT-4 and Gemini-1.5-Flash-002 demonstrated low single-iteration success rates (2 and 4 cases, respectively); however, these models were able to achieve a moderate average number of iterations and managed to resolve both adjustable and critical errors. This underscores their effectiveness in incorporating feedback to iteratively improve the quality of their outputs. In contrast, other models showed a higher tendency for requiring additional iterations and occasional failures. Mixtral-8x22B necessitated manual adjustments in three instances and failed to generate a valid model once after 15 iterations. Similarly, Codestral encountered two such instances of manual adjustments and one failure. The results for Codestral, which is optimized for code generation, were particularly surprising. Despite its specialization, this model performed worse than the other two Mistral models we considered. This suggests that task-optimized models face challenges on their primary task when applied to new domains.

### 6.3.2 Quality of the generated models

As described in Sect. 6.1.3, we assess the quality of the generated process models using the harmonic mean of fitness and precision scores obtained from conformance checking the models against the ground truth event logs. The average obtained quality scores for each LLM are reported in Table 3.

The results demonstrate significant variation in the quality of generated models across different LLMs. Claude-3.5-Sonnet leads the pack with an impressive average score of 0.93, closely approaching the average score of the ground truth models (0.98).[1] Following Claude-3.5-Sonnet, the O1-Preview and O1-Mini models also exhibit strong performance, achieving average scores of 0.92 and 0.91, respectively. These high scores underscore the effectiveness of the O1 series in generating accurate and reliable results, attributable to their advanced reasoning capabilities.

A closer examination of the results reveals the following observations and patterns:

**(1) Positive Correlation Between Iteration Performance and Quality:** There is a notable correlation between error-handling performance and model quality. Models like Claude-3.5-Sonnet, O1-Mini, O1-Preview, and Gemini-1.5-Pro-002, which produced less errors and required fewer iterations to generate valid process models, also achieved higher quality scores. Conversely, models that required more iterations or experienced failures, such as Mixtral-8x22B, Codestral, and WizardLM-2-8x22B, tended to have lower scores, reflecting potential compromises in model accuracy due to extended correction processes. This indicates that the ability to generate error-free outputs promptly contributes to the overall quality of the process models.

**(2) Quality Consistency on Average:** Despite the inherent non-determinism in LLM outputs, we observe consistent quality trends within similar groups of models. Specifically, the O1 models achieved average scores of 0.92 and 0.91, while the two instances of the Llama 3.1 family scored on average 0.86 and 0.83. The three models within the GPT-4 family recorded average scores ranging from 0.74 to 0.76. Additionally, across four independent runs of Gemini-1.5-Pro-002 (cf. Sect. 7.1), the model consistently exhibited similar performance with average scores ranging between 0.86 and 0.88. The observed patterns suggest that, although re-executing the same or similar LLMs on identical tasks can yield varying results, the average quality remains stable within each model group.

**(3) Impact of Speed Optimization:** Models optimized for speed exhibited varied behaviors in terms of quality scores. Notably, the OpenAI models O1-Mini and GPT-4o-

[1] A perfect score (1.0) is unattainable in many cases due to the presence of loops within the process models, which inherently allow for infinite behavior.

**Table 3** Average quality scores

| Model | Avg. Score |
|---|---|
| Ground Truth | 0.98 |
| Claude-3.5-Sonnet | 0.93 |
| O1-Preview | 0.92 |
| O1-Mini | 0.91 |
| Gemini-1.5-Pro-002 | 0.87 |
| Llama-3.1-405B-Instruct | 0.86 |
| Llama-3.1-Nemotron-70B-Instruct | 0.83 |
| Llama-3.2-90B-Vision-Instruct | 0.80 |
| Qwen2.5-72B-Instruct | 0.80 |
| Mistral-Large-2 | 0.78 |
| GPT-4 | 0.76 |
| GPT-4o | 0.76 |
| GPT-4o-Mini | 0.74 |
| WizardLM-2-8x22B | 0.73 |
| Codestral | 0.73 |
| Gemini-1.5-Flash-002 | 0.73 |
| Mixtral-8x22B | 0.72 |

Mini maintained quality scores comparable to their base counterparts, O1-Preview and GPT-4o, respectively. This consistency suggests that optimization for speed in these models does not significantly compromise the quality of their outputs. In contrast, Gemini-1.5-Flash-002 demonstrated lower quality scores compared to the Gemini-1.5-Pro-002 variant. However, Google promotes Gemini-1.5-Flash-002 as a small, lightweight model designed for tasks where speed and cost-effectiveness matter the most; therefore, it is unfair to compare its performance with the Pro model. These examples illustrate that speed optimization strategies can significantly differ based on the desired balance between speed enhancement and quality preservation, either aiming for moderate speed improvements with minimal quality loss or prioritizing speed as the primary optimization goal.

### 6.3.3 Time efficiency

Time efficiency is a critical factor in evaluating the practicality and scalability of different LLMs in our framework. We assessed the performance of each LLM by measuring both the average cumulative time taken across all iterations to generate the process model and the average time per a single iteration. Table 4 reports these metrics, ordered by the average total time in ascending order.

Gemini-1.5-Flash-002 and Codestral emerge as the fastest models in terms of per-iteration processing speed, with average times of 4.03 s and 7.98 s, respectively. However, their rapid response times come at the cost of requiring more iterations and a compromise in quality, diminishing their overall effectiveness. In contrast, Claude-3.5-Sonnet and Gemini-1.5-Pro-002 exemplify models that strike an excellent balance between speed and quality. Claude-3.5-Sonnet is particularly noteworthy, ranking second in total average time at 23.63 s while ranking first in average quality at a score of 0.93. Similarly, Gemini-1.5-Pro-002 demonstrate robust performance with average total times around 24.86 s and a strong average quality score of 0.87. These models achieve their efficiency not only through relatively fast per-iteration times but also by requiring less iterations and effectively handling errors.

The reasoning models of the O1 series (O1-Mini and O1-Preview) exhibit longer per-iteration processing times (39.16 s and 90.84 s, respectively). Despite their slower speeds per iteration, they maintain moderate total processing times due to their high efficiency in producing valid models quickly, minimizing the need for error handling iterations. O1-Mini, in particular, achieves a similar quality score to O1-Preview but at a substantially lower total time, making it a highly efficient option within the O1 series. This pattern extends to other OpenAI models, where GPT-4o-Mini outperforms GPT-4 and GPT-4o in speed while maintaining similar quality levels.

At the lower end of the efficiency spectrum, WizardLM-2-8x22B, Mistral-Large-2, and Llama-3.1-Nemotron-70B-Instruct stand out as the least time-efficient models, with average total times between 181.47 and 167.98 s.

### 6.3.4 Summary of results

In summary, our evaluation demonstrates that Claude-3.5-Sonnet stands out as the best-performing LLM in our framework, delivering the highest quality process models with minimal iterations and efficient error handling. Gemini-1.5-Pro-002 follows closely behind, with lower quality scores but also offering a good balance between quality and efficiency. The O1 series models (O1-Mini and O1-Preview) exhibit excellent performance but fall behind in the overall time performance. Speed-optimized models like Gemini-1.5-Flash-002 provide faster per-iteration times but often require more iterations and may compromise on quality.

These findings highlight the trade-offs between speed, iteration count, quality, and cost across different LLMs, emphasizing the importance of selecting the right LLM to achieve the best balance. Note that we did not include cost information in our evaluation due to the rapidly changing landscape of LLM pricing. However, when taking cost into account, Gemini-1.5-Pro-002 offered the best trade-off as it was freely available through APIs for limited, small-scale testing.

**Table 4** Time efficiency metrics

| Model | Avg. Total Time (s) | Avg. Time per Iteration (s) |
|---|---|---|
| Gemini-1.5-Flash-002 | 14.51 | 4.03 |
| Claude-3.5-Sonnet | 23.63 | 16.88 |
| Gemini-1.5-Pro-002 | 24.86 | 12.06 |
| Codestral | 38.27 | 7.98 |
| Mixtral-8x22B | 52.88 | 12.20 |
| O1-Mini | 55.29 | 39.16 |
| GPT-4o-Mini | 56.20 | 12.11 |
| Llama-3.1-405B-Instruct | 67.86 | 24.69 |
| Llama-3.2-90B-Vision-Instruct | 72.97 | 21.56 |
| GPT-4o | 78.98 | 33.72 |
| GPT-4 | 108.55 | 26.56 |
| Qwen2.5-72B-Instruct | 126.98 | 20.59 |
| O1-Preview | 145.48 | 90.84 |
| Llama-3.1-Nemotron-70B-Instruct | 167.98 | 38.20 |
| Mistral-Large-2 | 169.66 | 59.46 |
| WizardLM-2-8x22B | 181.47 | 29.57 |

## 6.4 Supplementary analyses

In addition to the main benchmarking results, we conducted supplementary analyses to explore specific aspects of LLM behavior within our framework, namely the effect of standardizing activity labels and the influence of background knowledge.

### 6.4.1 Effect of standardizing activity labels

To explore the potential impact of standardizing activity labels, we conducted a supplementary experiment focused on understanding how LLMs perform without predefined labels. We selected two processes from our benchmark: the R&D process (described in Listing 4, 13 activities in its ground truth model) and the university enrollment process (described in Listing 5, 26 activities in its ground truth model). We used two LLMs for this experiment: Gemini-1.5-Pro-002 and GPT-4o. We generated process models based only on the natural language descriptions, omitting the list of ground truth activity labels from the prompt. Subsequently, we manually analyzed the generated models, mapping the LLM-created activity labels back to the ground truth labels to identify similarities and differences. The mappings are shown in Tables 5 and 6.

For both processes, the LLMs generated activity labels that corresponded well with the ground truth activities, often involving only minor variations in phrasing (e.g., "Identify Idea" vs. "Identify idea for new product or improvement"). The core difference observed for the R&D process, however, was a tendency for both LLMs to aggregate concurrent activities into a single step. As detailed in Table 5, concurrent activities like "Conduct initial research" and "Conduct feasibility studies" were combined by the LLMs into composite activities such as "Research and Feasibility Study". Similarly, distinct testing activities ("Test functionality", "Test safety", "Test market potential") were merged into a single "Test Prototype" activity. This suggests that without explicit label guidance, LLMs may naturally generate models at a slightly higher level of abstraction, though the fundamental process steps were still captured.

For the more complex university enrollment process, 24 out of 26 of activities were correctly identified and labeled by both LLMs, with only minor phrasing differences, as shown in Table 6. However, a notable deviation concerned the final outcome specified in the description: "The process repeats each semester until the student graduates or withdraws". When operating without the predefined label list, both Gemini-1.5-Pro-002 and GPT-4o omitted explicit activities corresponding to "Graduate" and "Withdraw" (highlighted in Table 6).

This supplementary analysis confirms that providing activity labels influences the LLM output, primarily by ensuring granularity consistency and preventing the omission of details. Nonetheless, the core process structure and main activities were largely captured correctly in our tests even without this standardization. Therefore, while acknowledging its effect, we assess the standardization of activity labels as a suitable method for our main evaluation, serving its necessary purpose of enabling rigorous conformance checking for automated quality assessment.

**Table 5** Mapping activities between ground truth and LLM-generated models for the R&D process without predefined labels

| Ground Truth | Gemini-1.5-Pro-002 | GPT-4o |
|---|---|---|
| Identify idea for new product or improvement | Identify Idea | Identify idea for new product or improvement |
| **Conduct initial research** | | |
| **Conduct feasibility studies** | **Research and Feasibility Study** | **Conduct initial research and feasibility studies** |
| Draft design concepts | Draft Design Concepts | Draft design concepts |
| Select promising design | Select Design | Select promising design |
| Build prototype | Build Prototype | Build prototype using available materials and resources |
| **Test functionality** | | |
| **Test safety** | | |
| **Test market potential** | **Test Prototype** | **Test prototype functionality, safety, and market potential** |
| Collect feedback from testing phase | Collect Feedback | Collect feedback from testing phase |
| Refine prototype | Refine Prototype | Refine prototype |
| Approve prototype for further development | Approve Prototype | Approve prototype for further development |
| Discard prototype | Discard Prototype | Discard prototype |

Bold rows show where LLMs combined multiple ground truth activities

**Table 6** Mapping activities between ground truth and LLM-generated models for the university enrollment process without predefined labels

| Ground Truth | Gemini-1.5-Pro-002 | GPT-4o |
|---|---|---|
| Submit application online | Submit application online | Submit application |
| Review application and documents | Review application and documents | Review application and supporting documents |
| Notify applicant of missing documents | Notify about missing documents | Notify applicant to provide missing documents |
| Provide missing documents | Receive all documents | Provide missing documents |
| Evaluate application by admissions committee | Evaluate application | Evaluate application |
| Process fees or waivers | Process application fees/waivers | Process application fees or waivers |
| Send acceptance letter | Send acceptance letter | Send acceptance letter |
| Send rejection letter | Send rejection letter | Send rejection letter |
| Confirm enrollment | Confirm enrollment | Confirm enrollment by deadline |
| Cancel application | Cancel application | Application canceled |
| Send orientation materials | Send orientation materials | Send orientation materials |
| Set up IT accounts | Setup student accounts | Set up student accounts |
| Assist with visa processing | Visa processing | Assist with visa processing |
| Obtain student ID card | Obtain student ID card | Obtain student ID card |
| Meet with academic advisor | Meet with academic advisor | Meet with academic advisor |
| Select courses | Select courses | Select courses |
| Resolve schedule conflicts | Resolve schedule conflicts | Resolve schedule conflicts |
| Begin attending classes | Attend classes | Begin attending classes |
| Add/drop courses | Add/drop courses | Add/drop courses during add/drop period |
| Post grades | Post grades | Post grades |
| Review grades online | Review grades online | Review grades online |
| Submit appeal form | Submit appeal form | Submit appeal form |
| Meet with appeals committee | Meet with appeals committee | Meet with appeals committee |
| Await decision | Await decision | Await decision |
| **Withdraw** | | |
| **Graduate** | | |

Bold rows show activities omitted by the LLMs

### 6.4.2 Effect of background knowledge

Beyond the direct impact of predefined activity labels, an important consideration when using LLMs for process modeling is how they balance strict adherence to the input text with their built-in domain knowledge. LLMs are more than text-processing engines—they also carry extensive knowledge acquired during pre-training. This leads to a key question in process modeling: when presented with a textual description, does the LLM follow the exact sequence and structure? Or does it draw on its internal knowledge to adjust the process accordingly? Understanding how LLMs handle this trade-off is important, especially since real-world process descriptions often include ambiguities and inconsistencies.

To explore this, we conducted a focused experiment using a process description that intentionally included a counter-intuitive causal sequence in a typical order handling scenario. The process description is shown in Listing 6. This description clearly reverses the normal flow (e.g., delivery happens before login). We gave this description to two different LLMs, Gemini-1.5-Pro-002 and GPT-4o, using two types of prompts: (i) the standard prompt used in our main experiments, and (ii) a prompt that added the following instruction: “Strictly stick to the process description and do not use your domain knowledge”.

The results, shown in Fig. 5, revealed interesting differences between the models’ default behavior. With the standard prompt, Gemini-1.5-Pro-002 chose to ignore the illogical order in the text and instead generated a model following a standard process flow (login first, delivery last), as seen in Fig. 5b. In contrast, GPT-4o using the same standard prompt followed the unusual order given in the text (Fig. 5a). Notably, even with just these two LLMs, we observed distinct default approaches. This suggests that the way different LLMs balance textual instructions against their internal knowledge can vary considerably. However, the experiment also showed that this behavior can be controlled. When we used the prompt that specifically told Gemini-1.5-Pro-002 to stick strictly to the description, it successfully ignored its domain knowledge and created a model matching the illogical sequence from the text (Fig. 5c).

These findings show that while LLMs might indeed apply their background knowledge, potentially correcting perceived errors or ambiguities in the input, this tendency varies and can be directed using prompt engineering. In our process modeling framework, we do not explicitly prohibit the LLM from using domain knowledge by default. We believe that in many practical situations, users might provide descriptions that are slightly incomplete or ambiguous. In these cases, allowing the LLM to use its general process understanding could be beneficial, helping to fill gaps or clarify steps in a useful way.

## 7 Evaluating LLM self-improvement strategies

In this section, we investigate the ability of LLMs to improve the quality of their outputs through self-improvement strategies. We aim to examine how LLMs can autonomously evaluate, refine, and optimize their performance within our framework for process modeling. Specifically, we explore three techniques: *self-evaluation* (cf. Sect. 7.1), *self-optimization of input* (cf. Sect. 7.2), and *self-optimization of output* (cf. Sect. 7.3).

We primarily use Gemini-1.5-Pro-002 in this section due to its optimal trade-off between performance, time, and cost, as discussed in Sect. 6. We utilize the same set of 20 processes previously described in Sect. 6. All results are available at https://github.com/humam-kourani/EvaluatingLLMsProcessModeling.

### 7.1 LLM self-evaluation

In this section, we explore the potential of self-evaluation by LLMs to enhance the quality of their outputs within our process modeling framework.

Self-evaluation capitalizes on the reasoning capabilities of LLMs, enabling them to assess and potentially refine their own outputs. The aim is to reduce errors and hallucinations, thereby increasing the reliability and accuracy of LLM outputs [79]. Given the inherent non-determinism of LLM outputs, where responses can vary significantly across different sessions [80], self-evaluation might help in stabilizing outputs and mitigating variability.

Self-evaluation can be utilized in several ways: providing a quality score to the user to indicate the LLM’s confidence in its own answers, generating multiple candidate outputs and selecting the best one, or combining several responses into a single, more robust output. In our framework, we implement LLM self-evaluation by generating multiple candidate process models for each process description, and then we instruct the LLM to assess them based on predefined criteria and select the best model.

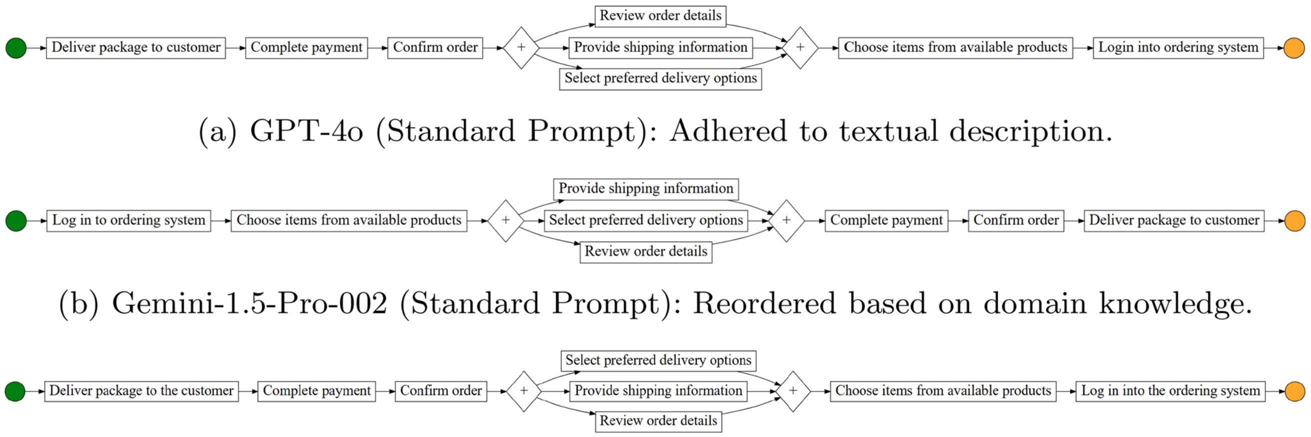
### 7.1.1 Implementation and experimental setup

For each process description, we conducted four independent runs, generating four candidate models (labeled `R1` to `R4`). After generating the four candidate models, we tasked the LLM with self-evaluating them and selecting the best one. In addition to Gemini-1.5-Pro-002, we replicated the experiment using Gemini-1.5-Flash-002 to assess whether the impact of self-evaluation differs between high-performing and lower-performing LLMs.

We crafted a comprehensive prompt that includes the initial task description, the specific process description, the four

**Listing 6** Textual description for the reversed order handling process.

```
The process starts by delivering the package to the customer. Then, the customer
completes the payment and confirms the order. After that, the customer carefully
reviews the order details, provides shipping information, and selects preferred
delivery options. Next, the customer chooses items from the available products.
Finally, the entire procedure concludes when the customer logs in into the ordering
 system.
```


Deliver package to customer
Complete payment
Confirm order
+
Review order details
Provide shipping information
Select preferred delivery options
+
Choose items from available products
Login into ordering system
Log in to ordering system
Choose items from available products
+
Provide shipping information
Select preferred delivery options
Review order details
+
Complete payment
Confirm order
Deliver package to customer
Deliver package to the customer
Complete payment
Confirm order
+
Select preferred delivery options
Provide shipping information
Review order details
+
Choose items from available products
Log in into the ordering system

(a) GPT-4o (Standard Prompt): Adhered to textual description.

(b) Gemini-1.5-Pro-002 (Standard Prompt): Reordered based on domain knowledge.

(c) Gemini-1.5-Pro-002 (Strict Prompt): Adhered to textual description.

**Fig. 5** LLM-generated models for the reversed order handling process description, illustrating the effect of domain knowledge and prompt instructions

generated candidate models, and a request for the LLM to self-evaluate these candidates and provide an overall quality score for each. Additionally, we computed the ground truth quality scores as described in Sect. 6.1.3 to validate whether the LLM's selections align with these scores, thereby verifying the effectiveness of its self-evaluation capabilities.

***Evaluation Criteria***

We instructed the LLM to evaluate the generated models based on sets of criteria to ensure a comprehensive assessment of each candidate model's quality according to different perspectives. We created two sets of evaluation criteria: a general set (cf. Listing 7) and a conformance-based set (cf. Listing 8). The first set of criteria focuses on broader aspects of model quality, while the second set aligns with the quality metrics used to compute the ground truth scores.

### 7.1.2 Results and discussion

The results of the LLM self-evaluation experiments are summarized in Table 7. It reports the average quality scores of the process models without self-evaluation, the number of instances where the LLM's selected process models are a subset of the best process models based on the quality assessment, the number of instances of exact matches, and the average quality scores of the process models post LLM self-evaluation and selection. To account for minor variations in quality scores, we applied a 0.02 buffer when determining the best process models in our match calculations.

It is important to note that the variations between the candidate runs (R1 to R4) were minimal in many cases, particularly for simpler processes or when the LLM consistently converged on a specific interpretation. When candidates are nearly identical in quality, the potential benefit of self-evaluation through selection is inherently constrained, and assessing the 'correctness' of the LLM's choice becomes less meaningful. Therefore, while we report the alignment between the LLM's selection and the highest-scoring models (subset matches and exact matches in Table 7), we emphasize the change in the average quality score of the selected models compared to the average scores for the single candidate runs. This provides a clearer perspective on the overall practical impact of this self-evaluation strategy on the final output quality, mitigating the issue of limited variability in some instances.

For Gemini-1.5-Pro-002, the initial average quality of the candidate process models ranged between 0.86 and 0.88. After applying self-evaluation and selection, the average quality increased to 0.91, indicating an overall improvement. However, these improvements were not consistent across all cases. In 5 out of 20 cases, the LLM's selected best models did not align with the selection based on the quality assessment. Despite these discrepancies, the increase in average

**Listing 7** Evaluation criteria in the prompt for general self-evaluation.

```
**Evaluation Criteria:**
- **Behavior Accuracy:** How accurately does the model capture the intended process
behavior?
- **Completeness:** Does the model include all necessary activities as described?
- **Correctness:** Are the control flows (e.g., partial orders, choices, loops)
 correctly implemented?
```

**Listing 8** Evaluation criteria in the prompt for conformance-based self-evaluation.

```
**Evaluation Criteria:**
- **Fitness:** Evaluate how well the process model can reproduce the behaviors of
the process according to the process description.
- **Precision:** Evaluate the extent to which the process model exclusively
represents behaviors that are allowed in the process according to the process
description.
```

**Table 7** Impact of LLM self-evaluation on process model quality

| LLM | Avg. Quality Without Self-Eval. (R1-R4) | Evaluation Criteria | Subset Match | Exact Match | Avg. Quality With Self-Eval. |
|---|---|---|---|---|---|
| Gemini-1.5-Pro-002 | 0.86–0.88 | General | 15/20 | 5/20 | 0.91 |
| | | Conformance | 15/20 | 7/20 | 0.91 |
| Gemini-1.5-Flash-002 | 0.73–0.75 | General | 4/20 | 0/20 | 0.72 |
| | | Conformance | 3/20 | 0/20 | 0.72 |

Under both sets of evaluation criteria (general and conformance-based), we report the average quality scores of the process models before self-evaluation and selection, the number of instances where the LLM's selected process models are a subset of the best process models based on the quality assessment, the number of instances of exact matches, and the average quality scores of the process models post LLM self-evaluation and selection

quality suggests that the self-evaluation strategy was generally effective for Gemini-1.5-Pro-002. The LLM's ability to select models that improved the overall average quality demonstrates its capacity to critically assess its outputs.

In contrast, Gemini-1.5-Flash-002 exhibited lower performance. The initial average quality of its candidate models was between 0.73 and 0.75. After self-evaluation, the average quality slightly decreased to 0.72. The LLM selected the wrong models in 16 or 17 out of 20 cases, depending on the evaluation criteria used. Although the final impact on average quality was not high, these results suggest that the self-evaluation strategy may be disadvantageous for Gemini-1.5-Flash-002.

Regarding the evaluation criteria, the results indicate that there was no significant difference between using the general evaluation criteria and the conformance-based criteria for both LLMs. The LLMs made similar assessments regardless of the prompted criteria, suggesting that the choice of evaluation criteria may not significantly influence the effectiveness of the self-evaluation.

Effective self-evaluation inherently relies on the LLM's instruction-following and reasoning capabilities. Our results align with this expectation: Gemini-1.5-Pro-002, the more capable model in our test pair, led to an overall improvement in average quality, while Gemini-1.5-Flash-002, the smaller and faster model, struggled significantly with the task. This suggests that the utility of this self-evaluation approach may be limited for models lacking a sufficient level of reasoning ability, potentially even proving detrimental. However, drawing broad conclusions is challenging, as this observation is based on comparing only two models within the Gemini family.

In summary, the effectiveness of employing LLM self-evaluation to select the best output among multiple candidates appears to be highly dependent on the selection of the LLM. Our results highlight that this strategy might be more beneficial for higher-performing models like Gemini-1.5-Pro-002; however, even for such models, the improvements are not consistent across all cases. Consequently, while LLM self-evaluation holds potential, the question of whether the potential improvements justify the additional time and costs remains unresolved, as the performance gains vary depending on the chosen LLM.

We acknowledge that these findings are limited as we only used two LLMs in our experiments. It is also important to note that our conclusions cannot be generalized to broader applications. Given that all candidate process models were generated by the same LLM, their quality levels are inherently close. We anticipate that LLM self-evaluation might yield better results when applied to outputs with larger disparities in quality.

## 7.2 LLM self-optimization of input

In this section, we investigate the potential of LLMs to enhance the quality of process models by self-optimizing the input process descriptions. Unlike traditional self-improvement methods, which aim at refining the output against a fixed input [11], we explore whether improving the clarity and detail of the input itself leads to a better output. The hypothesis is that by allowing LLMs to refine and enrich the initial process descriptions, they might produce higher-quality process models.

### 7.2.1 Implementation and experimental setup

For each process description, we created two additional versions: a summarized version that retains 50–80% of the original description's length (medium-length version), and a compact, very high-level version with 15–35% of the original length (short version). By crafting shorter versions of the process descriptions, we aimed to introduce varying levels of detail and specificity. The motivation behind this was to give the LLM more latitude to enhance and clarify the descriptions, potentially leading to improved process models upon self-optimization.

For each of the three versions (long, medium-length, and short), we instructed Gemini-1.5-Pro-002 to improve the description using the prompt illustrated in Listing 9. It is important to acknowledge a conceptual distinction between this input optimization experiment and self-improvement techniques focusing purely on output refinement. Modifying the input description, even with the goal of improvement, inherently risks altering the target specification against which the final model is generated. Recognizing this potential issue, the prompt (Listing 9) was deliberately designed with constraints: while encouraging the LLM to enrich the description by adding relevant details, clarifying ambiguities, and making process constructs explicit, it crucially instructed the LLM to ensure all additions were relevant, accurate, and directly related to the original process. This aimed to guide the LLM toward clarification within the bounds of the original process, rather than introducing unrelated information or fundamentally altering the described behavior.

### 7.2.2 Results and discussion

We evaluated the quality of the process models generated from both the original and the LLM-improved process descriptions. A summary of the results is presented in Table 8.

Contrary to our hypothesis, our investigation reveals that, for our specific application and framework, LLM self-optimization of input does not yield consistent benefits and may even be counterproductive. For the original long descriptions, the average quality score was 0.87, which decreased to 0.79 after LLM self-optimization, with only 6 out of 20 cases showing improvement. For the medium-length descriptions, the average score increased from 0.75 to 0.82 post-optimization, with 11 cases showing improvement and 9 not. For the short descriptions, the average score decreased from 0.78 to 0.72 after self-optimization, with improvements in only 8 cases.

These results suggest that the changes introduced by the LLM during the self-optimization process did not systematically contribute to better model quality. While LLMs can generate coherent text, they may lack the specific domain knowledge required to accurately enrich process descriptions in a way that leads to better process models.

Based on our observations across various processes, the prompt designed to discourage hallucinations (cf. Listing 9) appeared largely effective; quality degradation primarily stemmed from instances where the LLM introduced complicating details that it subsequently struggled to model accurately, rather than from generating entirely irrelevant information. To better understand how such quality degradation can occur through this self-optimization process, we examine a specific case where the negative impact was particularly pronounced: the inventory replenishment process ($p3$). For this process, self-optimization for the long description by Gemini-1.5-Pro-002 resulted in a drop in the quality score from 0.94 to 0.59. One notable modification involved the description of the reorder trigger:

- Original: “This process begins with monitoring inventory levels in a warehouse or store. When stock reaches a predefined threshold, an automated alert or manual check signals the need to reorder”.
- LLM-Optimized: “This process begins with continuous monitoring of inventory levels for each product in a warehouse or store. When the stock level of a product reaches a predefined minimum threshold, a system generates an automated alert. In parallel, a manual check of inventory levels is performed periodically. If the manual check reveals a product's stock level below the threshold, it also triggers a reorder signal. These two reorder triggers (automated alert or manual check) initiate the next step”.

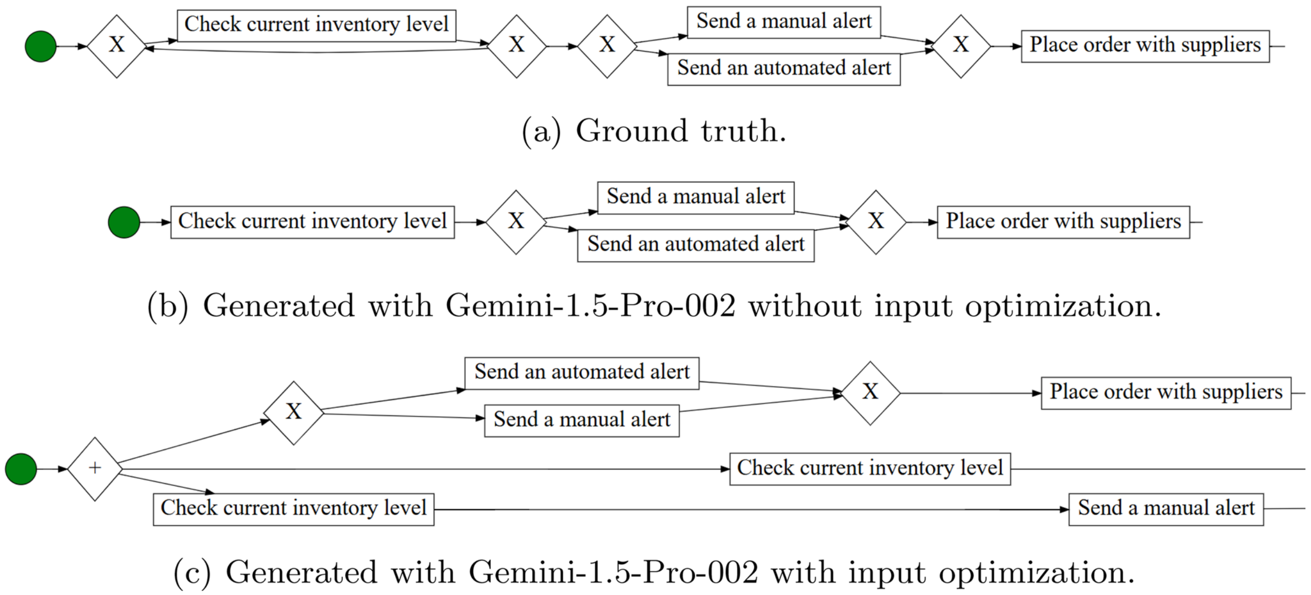
**Listing 9** Prompt for input optimization.

```
You are provided with a process description. Your task is to optimize this
description to make it richer and more detailed, while ensuring that all additions
are relevant, accurate, and directly related to the original process. The goal is
to make the description more comprehensive and suitable for process modeling
purposes.

Possible areas for enhancement include:
- **Detail Enhancement:** Add specific details that are missing but crucial for
understanding the process flow.
- **Clarity Improvement:** Clarify any ambiguous or vague statements to ensure that
the description is clear and understandable.
- **Explicit Process Constructs:** Rephrase parts of the description to explicitly
incorporate constructs. For example, change 'X happens in most cases' to 'there
is an exclusive choice between performing X or skipping it'.
```

**Table 8** Comparison of average quality scores before and after LLM self-improvement of process descriptions

| Description Length | Avg. Quality Before Self-Improvement | Avg. Quality After Self-Improvement | Cases With Increased Quality |
|---|---|---|---|
| Long (Original) | 0.87 | 0.79 | 6/20 |
| Medium-Length (50–80%) | 0.75 | 0.82 | 11/20 |
| Short (15–35%) | 0.78 | 0.72 | 8/20 |


X
Check current inventory level
X
X
Send a manual alert
Send an automated alert
X
Place order with suppliers
Check current inventory level
X
Send a manual alert
Send an automated alert
X
Place order with suppliers
+
X
Send an automated alert
Send a manual alert
X
Place order with suppliers
Check current inventory level
Check current inventory level
Send a manual alert

(a) Ground truth.

(b) Generated with Gemini-1.5-Pro-002 without input optimization.

(c) Generated with Gemini-1.5-Pro-002 with input optimization.

**Fig. 6** Process models fragments for the inventory replenishment process (p3)

While the original text simply implies an exclusive choice between an automated alert or a manual check, the LLM significantly elaborated, describing potentially parallel activities. This self-introduced complexity subsequently confused the LLM during the process model generation phase. Figure 6 shows fragments from the ground truth model for this process alongside the models generated by Gemini-1.5-Pro-002 with and without self-optimization. With the self-optimized input, the LLM generated a flawed model where the activities related to monitoring the inventory levels and sending the manual alert were duplicated and placed to run in parallel with the rest of the activities. This example highlights the risk of the input self-optimization strategy within our framework. Seemingly plausible enrichments introduced by the LLM can increase complexity to the point where the LLM itself fails to model the described process accurately, thereby harming overall model quality.

In conclusion, our findings suggest that relying on LLMs to autonomously enhance process descriptions without domain-specific guidance or constraints does not effectively improve the resultant process models.

## 7.3 LLM self-optimization of output

In this section, we examine the potential of LLMs to enhance the quality of their outputs by self-optimizing the generated process models. The underlying hypothesis is that by enabling LLMs to critically evaluate and refine their own outputs, they may identify and correct flaws, leading to higher-quality process models.

### 7.3.1 Implementation and experimental setup

To investigate this approach, we extended our previous experiments by instructing the LLMs to perform self-optimization on their initial outputs. Specifically, after generating the initial process model from the process description, we prompted the LLM to critically evaluate the model against the initial description and improve it accordingly. The prompt used is shown in Listing 10. To prevent unnecessary alterations that might degrade the model's quality, we crafted the prompt with intentional restrictiveness, emphasizing that the LLM should only make genuinely beneficial changes and encouraging it to retain the same model if no areas for improvement are identified. This approach addresses the tendency of LLMs to respond affirmatively to requests, even when they may lack the knowledge or capability to perform the task, potentially leading to unintended *hallucinations*.

We conducted experiments using three different LLMs: Gemini-1.5-Pro-002, Gemini-1.5-Flash-002, and GPT-4o. The selection of these models was motivated by the desire to assess whether self-optimization could yield more significant improvements in less-performing models (Gemini-1.5-Flash-002 and GPT-4o) compared to a high-performing model (Gemini-1.5-Pro-002).

For each process in our dataset, we generated the initial process model using each LLM and then applied the self-optimization prompt. We then evaluated the quality of the initial and improved models. For this experiment, we disabled the error refinement loop. Instead, in cases where errors occurred, we repeated the same self-optimization prompt multiple times until an error-free response was achieved. By bypassing error-handling loops, we aimed to avoid distracting the LLM with error correction, which might steer it away from the primary goal of enhancing the already successfully generated model.

### 7.3.2 Results and discussion

The results of the experiment are summarized in Table 9. This table includes, for each LLM, the average quality scores before and after LLM self-optimization of output, as well as the maximum improvement and maximum decline observed. Furthermore, the table reports the average number of attempts required for the LLM to produce an error-free response to the self-optimization prompt.

The results highlight that while the average improvements may appear modest, self-optimization of output can yield significant benefits in specific instances. GPT-4o showed the most substantial benefit, with the highest average quality gain (+0.05) and an instance of very large improvement (+0.84). Gemini-1.5-Flash-002 also saw significant enhancements, achieving gains up to +0.29. For Gemini-1.5-Pro-002, the impact of self-optimization was relatively low, with a slight average increase (+0.005), reflecting the high quality of its initial outputs. We also note that in some cases, self-optimization of output led to declines in quality. However, the maximum declines were relatively smaller compared to the maximum improvements, suggesting that while there is a risk of degradation, the potential for significant enhancement is greater.

The average number of attempts required, reported in Table 9, offers insight into the feasibility of this self-optimization step. All tested LLMs required relatively few attempts (between 1.1 and 1.4 on average) to produce an error-free response to the optimization prompt. These low numbers suggest that the process of modifying an initially valid output was generally efficient; the LLMs typically succeeded within one or two tries without frequently introducing new errors.

In conclusion, our analysis indicates that allowing LLMs to self-optimize their outputs can, in general, be beneficial within our framework, especially for models that initially produce lower-quality outputs. However, there is a risk of quality degradation and it is crucial to design the prompt carefully to discourage hallucinations.

# 8 Discussion and future work

In this section, we discuss the practical and research implications of our work, outline the current limitations and threats to validity, and suggest directions for future research.

## 8.1 Implications for practice and research

Our findings suggest several potential implications for both the practical application of BPM and future research directions. For practitioners, our framework points toward methods that could potentially reduce the time and effort involved in initial process modeling. By leveraging LLMs to generate draft process models from textual descriptions like procedural documents or interview notes, analysts might shift some focus from manual modeling toward validation, refinement, and analysis. This approach could also contribute to making process modeling more accessible; a natural language interface may lower the barrier for stakeholders less familiar with

**Listing 10** Prompt for output optimization.

```
Could you further improve the model? Please critically evaluate the process model
against the initial process description and improve it accordingly **only where
genuinely beneficial**. If you see no significant areas for enhancement, it is
perfectly acceptable to return the same model without any changes.
```

**Table 9** Impact of LLM self-optimization of output on model quality

| LLM | Avg. Num. Attempts | Avg. Quality Before Self-Improvement | Avg. Quality After Self-Improvement | Max. Improvement | Max. Decline |
|---|---|---|---|---|---|
| Gemini-1.5-Pro-002 | 1.2 | 0.87 | 0.87 | +0.14 | −0.04 |
| Gemini-1.5-Flash-002 | 1.4 | 0.73 | 0.76 | +0.29 | −0.08 |
| GPT-4o | 1.1 | 0.76 | 0.81 | +0.84 | −0.03 |

formal notations like BPMN, encouraging broader participation in process documentation efforts. The availability of our open-source tool, ProMoAI, provides a concrete implementation of these ideas, offering the community a platform to explore and build upon this LLM-driven approach. There is also potential for integrating such LLM functionalities into existing BPM platforms to assist users.

From a research standpoint, our work provides a specific context for examining how LLMs handle tasks requiring structured output generation, adherence to formal constraints, and logical reasoning from procedural text. Our study contributes to the broader challenge of automatically translating unstructured language into formal models, highlighting areas where current LLMs succeed and struggle. Our evaluation methodology, employing conformance checking, offers a reusable approach for benchmarking future LLMs on similar tasks. Finally, our initial exploration of self-improvement strategies suggests that this is a complex area requiring more research to understand the limits of LLM self-correction, develop effective prompting techniques, and compare autonomous refinement with human-guided or knowledge-augmented approaches.

## 8.2 Framework limitations and future directions

Our framework, while pioneering in leveraging LLMs for process modeling, has limitations. In this section, we outline areas for improvement and propose ideas for addressing them in future work.

*Expanding Process Perspectives:* Our framework addresses the control-flow perspective of process modeling, omitting the data, resource, and operational perspectives, which are crucial for a comprehensive understanding of business processes. The inherent flexibility and understanding capabilities of LLMs present a significant potential for extending our framework to incorporate additional process perspectives.

*Direct BPMN Generation:* The current implementation of our framework utilizes POWL for intermediate process representation. A possible direction for future research is the exploration of the direct generation of BPMN models without an intermediate process representation. This approach promises to offer greater flexibility in representing intricate process structures and dynamics and allows for the enrichment of process models with context-rich annotations. However, moving away from the structured guarantees provided by POWL necessitates the development of more advanced process model generation and validation techniques.

*Enhanced Interactivity:* We intend to enhance the model refinement loop to support more nuanced and interactive feedback mechanisms. For example, we aim to empower users to not only provide textual feedback on generated process models but also to manually edit the generated models.

## 8.3 Evaluation limitations and threats to validity

Our experimental evaluation is subject to several limitations and potential threats to validity that should be considered when interpreting the results.

*Manual Design of Processes:* Process descriptions and ground truth models were co-designed, constraining them to POWL-expressible structures. While this approach enables a realistic assessment of the framework's capabilities by avoiding elements it cannot capture, it may not reflect the full complexity or ambiguity encountered in unconstrained real-world processes. Future studies should consider incorporating real-world datasets to evaluate performance on more varied inputs.

*Conformance Checking Assessment:* Our evaluation relies primarily on conformance checking metrics and simulation to assess model quality. While this approach offers an objective measure of performance, it does not fully account for the practical usability or interpretability of the models from a human expert's perspective. This reliance may limit the generalizability of our findings to scenarios where qualitative human feedback is essential.

*Effect of LLM Design on Model Quality:* While our evaluation clearly shows differences in model quality across LLMs, pinpointing the exact causes is challenging as the LLMs are treated as black boxes. Although architectural variations (e.g., transformer-based vs. mixture-of-experts) and design goals (such as task-specific training) may contribute to the observed performance differences, our study does not directly isolate these factors. Instead, our focus is on providing a robust, empirical benchmark of overall performance within our framework. Further research could systematically investigate how underlying design choices impact quality in BPM tasks.

*Standardizing Activity Labels:* Another limitation is the use of predefined activity labels, provided to the LLM to enable automated conformance checking. This simplifies the task, as the LLM does not need to identify and consistently name activities from the raw text. Although our supplementary analysis in Sect. 6.4.1 suggests that the effect of standardizing activity labels is limited, it remains a factor that could affect the validity of our results, particularly concerning the activity identification sub-task.

*Generalizability of Findings on LLM Self-Improvement:* A key threat to the validity of our LLM self-improvement experiments (Sect. 7) arises because they were conducted using a small number of LLMs. While our results provide initial insights (e.g., suggesting that more capable models might benefit more from self-evaluation, or that output optimization holds potential), these findings cannot be reliably generalized to the broader LLM landscape. Validation across a more diverse set of LLMs, exhibiting varying sizes and architectures, is necessary. Future work should undertake a more extensive investigation involving additional state-of-the-art models to ascertain the broader applicability and robustness of these self-improvement techniques.

*Absence of Explicit Improvement Parameters for Output Optimization:* A limitation of our current output self-optimization experiment (Sect. 7.3) is the intentionally open-ended nature of the prompt, which instructs the LLM to improve the model without specifying explicit parameters, performance measures, or dimensions for enhancement. The rationale was to assess the LLM's ability to autonomously identify improvements based on the process description, mimicking user requests for general enhancement without pre-diagnosed flaws. While this approach tests a valuable capability, future research could explore the effectiveness of more structured self-optimization prompts that provide specific criteria. For example, these prompts could ask the LLM to verify adherence to certain modeling principles, check for specific patterns mentioned in the description, or even optimize toward metrics like simplicity or alignment with known best practices.

## 9 Conclusion

In this paper, we extended our LLM-driven process modeling framework by designing a reusable evaluation approach for benchmarking LLMs on process modeling and exploring LLM self-improvement strategies. Our assessment of 16 state-of-the-art LLMs revealed substantial performance variations, with Claude-3.5-Sonnet demonstrating exceptional capabilities in generating high-quality process models efficiently. We found a positive correlation between error-handling performance and the overall quality of the generated models. Additionally, our analysis indicated consistent quality trends within similar model families.

The investigation of LLM self-improvement strategies revealed that while self-evaluation depends heavily on the chosen LLM and input optimization shows limited reliability, output optimization demonstrates promising potential for enhancing quality. This underscores the possibility of leveraging LLMs to autonomously refine their outputs, potentially reducing the need for manual intervention. However, carefully crafted prompts are crucial to discourage hallucinations.

Our work contributes valuable insights into the application of LLMs for automated process modeling. The benchmark provides a foundation for comparing LLM performance in this domain, while the self-improvement analysis identifies promising avenues for further enhancing LLM-generated process models. Future research directions include incorporating additional process perspectives beyond control-flow, exploring direct BPMN generation without intermediate representations, investigating alternative prompting strategies, and exploring the integration of external knowledge sources to further enhance the accuracy and reliability of LLM-generated process models.

**Funding** Open Access funding enabled and organized by Projekt DEAL.

## References

1. Rosing, M., White, S., Cummins, F., Man, H.: Business process model and notation—BPMN. In: The Complete Business Process Handbook: Body of Knowledge from Process Modeling to BPM, Volume I, pp. 429–453 (2015). https://doi.org/10.1016/B978-0-12-799959-3.00021-5
2. Hee, K.M., Sidorova, N., Werf, J.M.E.M.: Business process modeling using petri nets. Trans. Petri Nets Other Model. Concurr. **7**, 116–161 (2013). https://doi.org/10.1007/978-3-642-38143-0_4
3. OpenAI: GPT-4 technical report (2023). CoRR arXiv:2303.08774. https://doi.org/10.48550/ARXIV.2303.08774
4. Anil, R., et al.: Gemini: a family of highly capable multimodal models (2023). CoRR arXiv:2312.11805. https://doi.org/10.48550/ARXIV.2312.11805
5. Li, J., Tang, T., Zhao, W.X., Wen, J.: Pretrained language model for text generation: a survey. In: Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19–27 August 2021, pp. 4492–4499 (2021). https://doi.org/10.24963/IJCAI.2021/612
6. Vidan, A., Fiedler, L.H.: A composable just-in-time programming framework with LLMs and FBP. In: IEEE High Performance Extreme Computing Conference, HPEC 2023, Boston, MA, USA, September 25–29, 2023, pp. 1–8 (2023). https://doi.org/10.1109/HPEC58863.2023.10363587
7. Zhou, Y., Muresanu, A.I., Han, Z., Paster, K., Pitis, S., Chan, H., Ba, J.: Large language models are human-level prompt engineers. In: The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1–5, 2023 (2023)
8. Kourani, H., Berti, A., Schuster, D., Aalst, W.M.P.: Process modeling with large language models. In: Enterprise, Business-Process and Information Systems Modeling—25th International Conference, BPMDS 2024, and 29th International Conference, EMMSAD 2024, Limassol, Cyprus, June 3-4, 2024, Proceedings, pp. 229–244 (2024). https://doi.org/10.1007/978-3-031-61007-3_18
9. Kourani, H., Zelst, S.J.: POWL: partially ordered workflow language. In: Business Process Management—21st International Conference, BPM 2023, Utrecht, The Netherlands, September 11–15, 2023, Proceedings, pp. 92–108 (2023). https://doi.org/10.1007/978-3-031-41620-0_6
10. Carmona, J., Dongen, B.F., Weidlich, M.: Conformance checking: foundations, milestones and challenges. In: Process Mining Handbook, pp. 155–190 (2022). https://doi.org/10.1007/978-3-031-08848-3_5
11. Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B.P., Hermann, K., Welleck, S., Yazdanbakhsh, A., Clark, P.: Self-refine: iterative refinement with self-feedback. In: Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10–16, 2023 (2023)
12. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, 2nd edn (2018). https://doi.org/10.1007/978-3-662-56509-4
13. Forster, S., Pinggera, J., Weber, B.: Toward an understanding of the collaborative process of process modeling. In: Proceedings of the CAiSE'13 Forum at the 25th International Conference on Advanced Information Systems Engineering (CAiSE), Valencia, Spain, June 20th, 2013, pp. 98–105 (2013)
14. Sintoris, K., Vergidis, K.: Extracting business process models using natural language processing (NLP) techniques. In: 19th IEEE Conference on Business Informatics, CBI 2017, Thessaloniki, Greece, July 24–27, 2017, Volume 1: Conference Papers, pp. 135–139 (2017). https://doi.org/10.1109/CBI.2017.41
15. Qian, C., Wen, L., Kumar, A., Lin, L., Lin, L., Zong, Z., Li, S., Wang, J.: An approach for process model extraction by multi-grained text classification. In: Advanced Information Systems Engineering—32nd International Conference, CAiSE 2020, Grenoble, France, June 8–12, 2020, Proceedings, pp. 268–282 (2020). https://doi.org/10.1007/978-3-030-49435-3_17
16. Aa, H., Carmona, J., Leopold, H., Mendling, J., Padró, L.: Challenges and opportunities of applying natural language processing in business process management. In: Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20–26, 2018, pp. 2791–2801 (2018)
17. Salimifard, K., Wright, M.: Petri net-based modelling of workflow systems: An overview. Eur. J. Oper. Res. **134**(3), 664–676 (2001). https://doi.org/10.1016/S0377-2217(00)00292-7
18. Unruh, E., Delfmann, P., Thimm, M.: Quantitative deadlock analysis in petri nets using inconsistency measures. In: 23rd IEEE Conference on Business Informatics, CBI 2021, Bolzano, Italy, September 1–3, 2021. Volume 1, pp. 42–51 (2021). https://doi.org/10.1109/CBI52690.2021.00015
19. Menezes, P.B., Costa, J.F.: Synchronization in petri nets. Fundam. Informaticae **26**(1), 11–22 (1996). https://doi.org/10.3233/FI-1996-2612
20. Datta, A.K., Ghosh, S.: High-level petri-net model for a resource-sharing problem. Inf. Sci. **51**(2), 213–220 (1990). https://doi.org/10.1016/0020-0255(90)90027-8
21. Leemans, S.J.J.: Robust Process Mining with Guarantees—Process Discovery, Conformance Checking and Enhancement. Lecture Notes in Business Information Processing, vol. 440 (2022). https://doi.org/10.1007/978-3-030-96655-3
22. Kourani, H., Schuster, D., Aalst, W.M.P.: Scalable discovery of partially ordered workflow models with formal guarantees. In: 5th International Conference on Process Mining, ICPM 2023, Rome, Italy, October 23–27, 2023, pp. 89–96 (2023). https://doi.org/10.1109/ICPM60904.2023.10271941
23. Kourani, H., Zelst, S.J., Schuster, D., Aalst, W.M.P.: Discovering partially ordered workflow models. Inf. Syst. **128**, 102493 (2025). https://doi.org/10.1016/J.IS.2024.102493
24. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4–9, 2017, Long Beach, CA, USA, pp. 5998–6008 (2017)
25. Brown, T.B., et al.: Language models are few-shot learners. In: Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6–12, 2020, Virtual (2020)
26. Wil M. P. van der Aalst et al.: Process mining manifesto. In: Business Process Management Workshops—BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part I, pp. 169–194 (2011). https://doi.org/10.1007/978-3-642-28108-2_19
27. Freitas, A.P., Pereira, J.L.M.: Process simulation support in BPM tools: the case of BPMN (2015)
28. Aalst, W.M.P.: Business process simulation revisited. In: Enterprise and Organizational Modeling and Simulation—6th International Workshop, EOMAS 2010, Held at CAiSE 2010, Hammamet, Tunisia, June 7–8, 2010. Selected Papers, pp. 1–14 (2010). https://doi.org/10.1007/978-3-642-15723-3_1
29. Dunzer, S., Stierle, M., Matzner, M., Baier, S.: Conformance checking: a state-of-the-art literature review. In: Proceedings of the 11th International Conference on Subject-Oriented Business Process Management, S-BPM ONE 2019, Seville, Spain, June 26-28, 2019, pp. 4–1410 (2019). https://doi.org/10.1145/3329007.3329014

30. Woensel, W.V., Motie, S.: NLP4PBM: a systematic review on process extraction using natural language processing with rule-based, machine and deep learning methods. Enterp. Inf. Syst. **18**(11) (2024). https://doi.org/10.1080/17517575.2024.2417404
31. Gonçalves, A.R., Santoro, J.C., Baião, F.M.F.A.: Let me tell you a story—on how to build process models. J. Univers. Comput. Sci. **17**(2), 276–295 (2011). https://doi.org/10.3217/JUCS-017-02-0276
32. Friedrich, F., Mendling, J., Puhlmann, F.: Process model generation from natural language text. In: Advanced Information Systems Engineering—23rd International Conference, CAiSE 2011, London, UK, June 20–24, 2011. Proceedings, pp. 482–496 (2011). https://doi.org/10.1007/978-3-642-21640-4_36
33. Sholiq, S., Sarno, R., Astuti, E.S.: Generating BPMN diagram from textual requirements. J. King Saud Univ. Comput. Inf. Sci. **34**(10 Part B), 10079–10093 (2022). https://doi.org/10.1016/J.JKSUCI.2022.10.007
34. Ivanchikj, A., Serbout, S., Pautasso, C.: From text to visual BPMN process models: design and evaluation. In: MoDELS '20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems, Virtual Event, Canada, 18-23 October, 2020, pp. 229–239 (2020). https://doi.org/10.1145/3365438.3410990
35. Aalst, W.M.P.: Process Mining—Discovery, Conformance and Enhancement of Business Processes (2011). https://doi.org/10.1007/978-3-642-19345-3
36. Busch, K., Rochlitzer, A., Sola, D., Leopold, H.: Just tell me: Prompt engineering in business process management. In: Enterprise, Business-Process and Information Systems Modeling - 24th International Conference, BPMDS 2023, and 28th International Conference, EMMSAD 2023, Zaragoza, Spain, June 12-13, 2023, Proceedings, pp. 3–11 (2023). https://doi.org/10.1007/978-3-031-34241-7_1
37. Vidgof, M., Bachhofner, S., Mendling, J.: Large language models for business process management: Opportunities and challenges. In: Business Process Management Forum - BPM 2023 Forum, Utrecht, The Netherlands, September 11–15, 2023, Proceedings, pp. 107–123 (2023). https://doi.org/10.1007/978-3-031-41623-1_7
38. Norouzifar, A., Kourani, H., Dees, M., Aalst, W.M.P.: Bridging domain knowledge and process discovery using large language models. In: Business Process Management Workshops—BPM 2024 International Workshops, Krakow, Poland, September 1–6, 2024, Revised Selected Papers, pp. 44–56 (2024). https://doi.org/10.1007/978-3-031-78666-2_4
39. Kourani, H., Berti, A., Henrich, J., Kratsch, W., Weidlich, R., Li, C., Arslan, A., Schuster, D., Aalst, W.M.P.: Leveraging large language models for enhanced process model comprehension (2024). CoRR arXiv:2408.08892. https://doi.org/10.48550/ARXIV.2408.08892
40. Nicola, A.D., Formica, A., Mele, I., Missikoff, M., Taglino, F.: A comparative study of LLMs and NLP approaches for supporting business process analysis. Enterp. Inf. Syst. **18**(10) (2024). https://doi.org/10.1080/17517575.2024.2415578
41. Bellan, P., Dragoni, M., Ghidini, C.: Process knowledge extraction and knowledge graph construction through prompting: A quantitative analysis. In: Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing, SAC 2024, Avila, Spain, April 8–12, 2024, pp. 1634–1641 (2024). https://doi.org/10.1145/3605098.3635957
42. Chen, S., Liao, H.: Bert-log: anomaly detection for system logs based on pre-trained language model. Appl. Artif. Intell. **36**(1) (2022). https://doi.org/10.1080/08839514.2022.2145642
43. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2–7, 2019, Volume 1 (Long and Short Papers), pp. 4171–4186 (2019). https://doi.org/10.18653/V1/N19-1423
44. Fill, H., Fettke, P., Köpke, J.: Conceptual modeling and large language models: Impressions from first experiments with ChatGPT. Enterp. Model. Inf. Syst. Archit. Int. J. Concept. Model. **18**, 3 (2023). https://doi.org/10.18417/EMISA.18.3
45. Muff, F., Fill, H.: Limitations of ChatGPT in conceptual modeling: Insights from experiments in metamodeling. In: Modellierung 2024 - Workshop Proceedings, Potsdam, Germany, March 12–15, 2024, p. 8 (2024). https://doi.org/10.18420/MODELLIERUNG2024-WS-008
46. Busch, K., Leopold, H.: Towards a benchmark for large language models for business process management tasks (2024). CoRR arXiv:2410.03255. https://doi.org/10.48550/ARXIV.2410.03255
47. Klievtsova, N., Benzin, J., Kampik, T., Mangler, J., Rinderle-Ma, S.: Conversational process modelling: State of the art, applications, and implications in practice. In: Business Process Management Forum—BPM 2023 Forum, Utrecht, The Netherlands, September 11–15, 2023, Proceedings, pp. 319–336 (2023). https://doi.org/10.1007/978-3-031-41623-1_19
48. Ziche, C., Apruzzese, G.: LLM4PM: A case study on using large language models for process modeling in enterprise organizations. In: Business Process Management: Blockchain, Robotic Process Automation, Central and Eastern European, Educators and Industry Forum—BPM 2024 Blockchain, RPA, CEE, Educators and Industry Forum, Krakow, Poland, September 1–6, 2024, Proceedings, pp. 472–483 (2024). https://doi.org/10.1007/978-3-031-70445-1_35
49. Fontenla-Seco, Y., Winkler, S., Gianola, A., Montali, M., Penín, M.L., Diz, A.J.B.: The droid you're looking for: C-4PM, a conversational agent for declarative process mining. In: Proceedings of the Best Dissertation Award, Doctoral Consortium, and Demonstration & Resources Forum at BPM 2023 Co-located with 21st International Conference on Business Process Management (BPM 2023), Utrecht, The Netherlands, September 11th to 15th, 2023, pp. 112–116 (2023)
50. Grohs, M., Abb, L., Elsayed, N., Rehse, J.: Large language models can accomplish business process management tasks. In: Business Process Management Workshops - BPM 2023 International Workshops, Utrecht, The Netherlands, September 11–15, 2023, Revised Selected Papers, pp. 453–465 (2023). https://doi.org/10.1007/978-3-031-50974-2_34
51. Wenger, S., Spahic-Bogdanovic, M., Martin, A.: Large language models for democratizing business process modeling: BPMN model generation and style guide adherence. In: Southern African Conference for Artificial Intelligence Research, pp. 372–389. Springer, Berlin (2024)
52. Eldin, A.N., Assy, N., Anesini, O., Dalmas, B., Gaaloul, W.: Nala2BPMN: Automating BPMN model generation with large language models. In: Cooperative Information Systems—30th International Conference, CoopIS 2024, Porto, Portugal, November 19–21, 2024, Proceedings, pp. 398–404 (2024). https://doi.org/10.1007/978-3-031-81375-7_27
53. Ayad, S., Alsayoud, F.: Prompt engineering techniques for semantic enhancement in business process models. Bus. Process. Manag. J. **30**(7), 2611–2641 (2024). https://doi.org/10.1108/BPMJ-02-2024-0108
54. Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y., Madotto, A., Fung, P.: Survey of hallucination in natural language generation. ACM Comput. Surv. **55**(12), 248–124838 (2023). https://doi.org/10.1145/3571730
55. Pan, L., Saxon, M., Xu, W., Nathani, D., Wang, X., Wang, W.Y.: Automatically correcting large language models: surveying the landscape of diverse automated correction strategies. Trans. Assoc. Comput. Linguist. **12**, 484–506 (2024). https://doi.org/10.1162/TACL_A_00660

56. Huang, J., Chen, X., Mishra, S., Zheng, H.S., Yu, A.W., Song, X., Zhou, D.: Large language models cannot self-correct reasoning yet. In: The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7–11, 2024 (2024)
57. Song, Y., Zhang, H., Eisenach, C., Kakade, S.M., Foster, D.P., Ghai, U.: Mind the gap: Examining the self-improvement capabilities of large language models (2024). CoRR arXiv:2412.02674. https://doi.org/10.48550/ARXIV.2412.02674
58. Yuntao Bai et al.: Constitutional AI: harmlessness from AI feedback (2022). CoRR arXiv:2212.08073. https://doi.org/10.48550/ARXIV.2212.08073
59. Lianmin Zheng et al.: Judging LLM-as-a-judge with MT-Bench and chatbot arena. In: Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10–16, 2023 (2023)
60. Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., Yao, S.: Reflexion: language agents with verbal reinforcement learning. In: Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10–16, 2023 (2023)
61. Huang, A., Block, A., Foster, D.J., Rohatgi, D., Zhang, C., Simchowitz, M., Ash, J.T., Krishnamurthy, A.: Self-improvement in language models: the sharpening mechanism (2024). CoRR arXiv:2412.01951. https://doi.org/10.48550/ARXIV.2412.01951
62. Kaplan, J., McCandlish, S., Henighan, T., Brown, T.B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., Amodei, D.: Scaling laws for neural language models (2020). CoRR arXiv:2001.08361
63. Hoffmann, J., et al.: Training compute-optimal large language models (2022). CoRR arXiv:2203.15556. https://doi.org/10.48550/ARXIV.2203.15556
64. Lopes, I.F., Ferreira, D.R.: A survey of process mining competitions: The BPI challenges 2011–2018. In: Business Process Management Workshops—BPM 2019 International Workshops, Vienna, Austria, September 1–6, 2019, Revised Selected Papers, pp. 263–274 (2019). https://doi.org/10.1007/978-3-030-37453-2_22
65. Berti, A., Kourani, H., Aalst, W.M.P.: PM-LLM-Benchmark: Evaluating large language models on process mining tasks (2024). CoRR arXiv:2407.13244. https://doi.org/10.48550/ARXIV.2407.13244
66. Fournier, F., Limonad, L., Skarbovsky, I.: Towards a benchmark for causal business process reasoning with LLMs. In: Business Process Management Workshops—BPM 2024 International Workshops, Krakow, Poland, September 1–6, 2024, Revised Selected Papers, pp. 233–246 (2024). https://doi.org/10.1007/978-3-031-78666-2_18
67. Fahland, D., Fournier, F., Limonad, L., Skarbovsky, I., Swevels, A.J.E.: How well can large language models explain business processes? (2024). CoRR arXiv:2401.12846. https://doi.org/10.48550/ARXIV.2401.12846
68. Rebmann, A., Schmidt, F.D., Glavas, G., Aa, H.: Evaluating the ability of LLMs to solve semantics-aware process mining tasks. In: 6th International Conference on Process Mining, ICPM 2024, Kgs. Lyngby, Denmark, October 14–18, 2024, pp. 9–16 (2024). https://doi.org/10.1109/ICPM63005.2024.10680677
69. Xu, B., Yang, A., Lin, J., Wang, Q., Zhou, C., Zhang, Y., Mao, Z.: Expertprompting: Instructing large language models to be distinguished experts (2023). CoRR arXiv:2305.14688. https://doi.org/10.48550/ARXIV.2305.14688
70. Martino, A., Iannelli, M., Truong, C.: Knowledge injection to counter large language model (LLM) hallucination. In: The Semantic Web: ESWC 2023 Satellite Events—Hersonissos, Crete, Greece, May 28–June 1, 2023, Proceedings, pp. 182–185 (2023). https://doi.org/10.1007/978-3-031-43458-7_34
71. Bellan, P., Aa, H., Dragoni, M., Ghidini, C., Ponzetto, S.P.: PET: an annotated dataset for process extraction from natural language text tasks. In: Business Process Management Workshops—BPM 2022 International Workshops, Münster, Germany, September 11–16, 2022, Revised Selected Papers, pp. 315–321 (2022). https://doi.org/10.1007/978-3-031-25383-6_23
72. Miyake, D., Iohara, A., Saito, Y., Tanaka, T.: Negative-prompt inversion: fast image inversion for editing with text-guided diffusion models (2023). CoRR arXiv:2305.16807. https://doi.org/10.48550/ARXIV.2305.16807
73. Kourani, H., Berti, A., Schuster, D., Aalst, W.M.P.: ProMoAI: Process modeling with generative AI. In: Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3–9, 2024, pp. 8708–8712 (2024)
74. Kourani, H., Zelst, S.J., Lehmann, B., Einsdorf, G., Helfrich, S., Liße, F.: PM4KNIME: process mining meets the KNIME analytics platform (extended abstract). In: Proceedings of the ICPM Doctoral Consortium and Demo Track 2022 Co-located with 4th International Conference on Process Mining (ICPM 2022), Bolzano, Italy, October, 2022, pp. 65–69 (2022)
75. Kourani, H., Park, G., Aalst, W.M.P.: Translating workflow nets into the partially ordered workflow language. arXiv preprint arXiv:2503.20363 (2025)
76. Berti, A., Zelst, S.J., Schuster, D.: PM4Py: A process mining library for python. Softw. Impacts **17**, 100556 (2023). https://doi.org/10.1016/J.SIMPA.2023.100556
77. Berti, A., Aalst, W.M.P.: Reviving token-based replay: Increasing speed while improving diagnostics. In: Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2019 Satellite Event of the Conferences: 40th International Conference on Application and Theory of Petri Nets and Concurrency Petri Nets 2019 and 19th International Conference on Application of Concurrency to System Design ACSD 2019, ATAED@Petri Nets/ACSD 2019, Aachen, Germany, June 25, 2019, pp. 87–103 (2019)
78. Munoz-Gama, J., Carmona, J.: A fresh look at precision in process conformance. In: Business Process Management—8th International Conference, BPM 2010, Hoboken, NJ, USA, September 13–16, 2010. Proceedings, pp. 211–226 (2010). https://doi.org/10.1007/978-3-642-15618-2_16
79. Zhang, X., Peng, B., Tian, Y., Zhou, J., Jin, L., Song, L., Mi, H., Meng, H.: Self-alignment for factuality: mitigating hallucinations in LLMs via self-evaluation. In: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11–16, 2024, pp. 1946–1965 (2024). https://doi.org/10.18653/V1/2024.ACL-LONG.107
80. Song, Y., Wang, G., Li, S., Lin, B.Y.: The good, the bad, and the greedy: evaluation of LLMs should not ignore non-determinism (2024). CoRR arXiv:2407.10457. https://doi.org/10.48550/ARXIV.2407.10457

**Humam Kourani** is a is a Research Associate at the Fraunhofer Institute for Applied Information Technology (FIT). He is a member of the Center for Process Intelligence and contributes to research and software development projects within the Data Science and Artificial Intelligence Department. In addition, Humam serves as a process mining examiner for the Fraunhofer Personnel Certification Authority. Humam is currently pursuing his Ph.D. at RWTH Aachen University. His research focuses on business process modeling, process discovery, and the integration of generative AI into BPM and process mining.

**Alessandro Berti** is a Software Engineer in the Process and Data Science (PADS) group at RWTH Aachen University. He is Doctor Designatus in Computer Science at RWTH, with a thesis on object-centric process mining. Alessandro co-founded the widely-used process mining library pm4py, which has achieved significant recognition within the academic and practitioner communities. His research interests include the development of innovative process mining algorithms and tools, GPU-accelerated computations for event log analysis, and integrating generative AI with process mining.

**Daniel Schuster** is a researcher specializing in process mining. He earned his Ph.D. from RWTH Aachen University on Incremental Process Discovery, which received the Best Ph.D. Dissertation Award from the IEEE Task Force on Process Mining. He previously led the Process Mining Research Group at the Fraunhofer Institute for Applied Information Technology and is now Managing Director of Process Intelligence Solutions, a Fraunhofer spin-off focused on process mining software solutions such as PM4Py. He has published extensively in international conferences and journals, with research focusing on interactive and incremental process discovery, partially ordered event data, and user-centric process mining.

**Wil M. P. van der Aalst** is a full professor at RWTH Aachen University, leading the Process and Data Science (PADS) group. He is also the Chief Scientist at Celonis, part-time affiliated with the Fraunhofer FIT. He also has an unpaid professorship position at Queensland University of Technology (since 2003). Currently, he is also deputy CEO of the Internet of Production (IoP) Cluster of Excellence, co-director of the RWTH Center for Artificial Intelligence, and speaker of the RWTH ICT Profile Area. His research interests include process mining, data science, Petri nets, process management, workflow management, process modeling, and process analysis. Wil M. P. van der Aalst has published over 300 journal papers, 35 books (as author or editor), 720 refereed conference/workshop publications, and 90 book chapters.