# Are Large Language Models Fluent in Declarative Process Mining?

**Valeria Fionda**[1] , **Antonio Ielo**[1] , **Francesco Ricca**[1]
[1]Department of Mathematics and Computer Science, University of Calabria, Italy
valeria.fionda@unical.it , antonio.ielo@unical.it , francesco.ricca@unical.it

## Abstract

Recent advancements in AI have made LLMs valuable tools for automating the interpretation of textual descriptions of business processes and for converting formal process specifications into natural language. However, there are no practical methodologies or systematic assessments to ensure these automatic translations are faithful. This paper proposes a novel approach, based on an auxiliary *bidirectional translation* task, to assess LLMs performance quantitatively; also, it empirically evaluates the performance of state-of-the-art LLMs for bidirectional translations between natural language and declarative formal process specifications. The results reveal substantial variability in performance among the LLMs, highlighting the importance of LLM selection and confirming the need for a robust method for assessing LLMs' outputs.

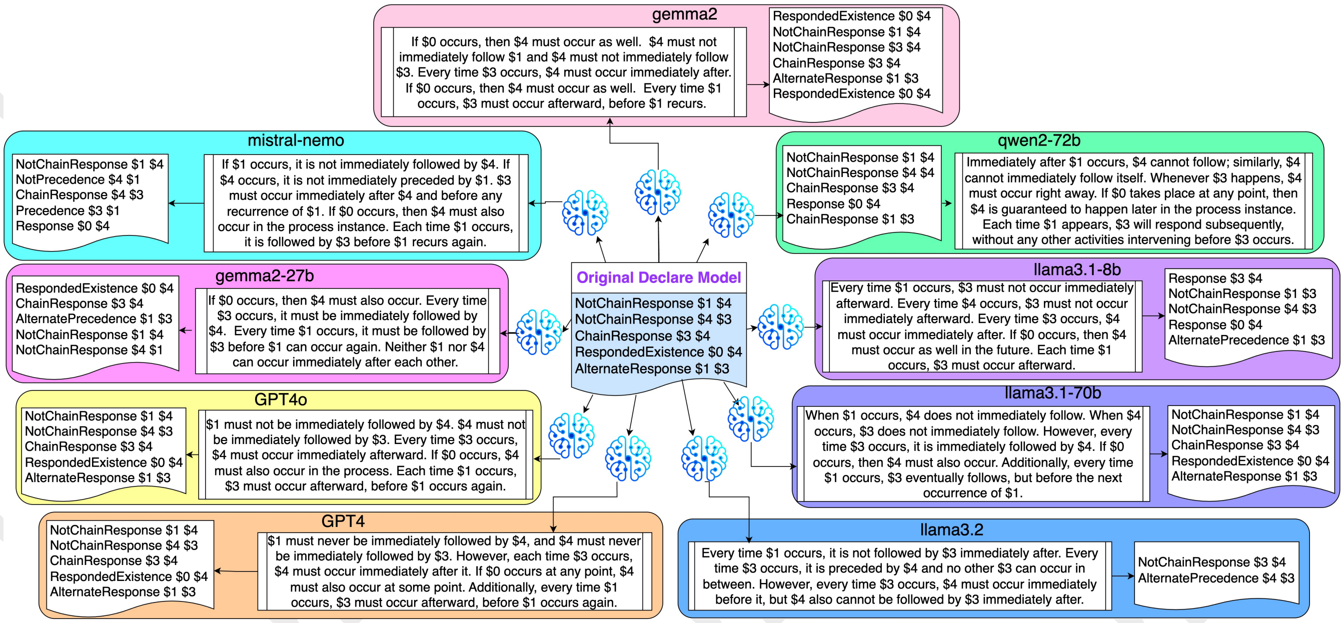
## 1 Introduction

Ensuring the reliability and correctness of business processes is critical, especially in domains like compliance management, healthcare, and financial services, where compliance to rules and regulations is essential [van der Aalst, 2022]. In Declarative Process Mining [Di Ciccio and Montali, 2022], formal specifications play a crucial role in verifying whether process executions conform to certain behavioral constraints. One widely used formalism for declarative process modeling is the Declare specification language [van der Aalst *et al.*, 2009], whose semantics is based on Linear Temporal Logic on Finite Traces (LTL$_f$) [De Giacomo and Vardi, 2013]. Declare allows the definition of flexible, constraint-based rules that describe the behavior of processes, such as *response(a,b)* (i.e., *b* must eventually follow *a*) or *not-coexistence(a,b)* (i.e., *a* and *b* must never occur concurrently). Despite the expressive power of Declare writing specifications manually is still a challenging, error-prone, and time-consuming task that requires significant expertise in both process modeling and formal logic. In fact, translating natural language (NL) descriptions of processes into formal Declare specifications is a challenge in process mining. For instance, consider a process requirement stated in natural language: *"After an order is received, it must be processed immediately"*. While this is easily understood in natural language, converting it into a formal Declare specification such as *chain-response(OrderReceived, OrderProcessed)* requires a deep understanding of Declare's syntax and semantics. This manual translation process can be time-intensive and prone to errors.

The reverse process – translating a formal Declare specification into natural language – is equally important for validating and communicating process rules. For example, a process modeler may generate a Declare rule such as *chain-precedence(Approval, Payment)* to express that "Payment must be immediately preceded by Approval". However, stakeholders or clients unfamiliar with the Declare formal language might struggle to understand this specification. Translating the formal rule back into natural language is essential for ensuring that non-expert stakeholders can validate whether the specification accurately reflects the intended process behavior. In fact, while formal specifications like Declare are mathematically precise, they can be difficult to understand for non-experts and thus, converting Declare constraints into natural language bridges the communication gap between verification engineers and non-expert stakeholders.

Recent advancements in Large Language Models (LLMs) offer an opportunity to automate this translation tasks. LLMs, trained on vast amounts of text, can assist in the translation between natural language and Declare specifications. Recent advancements in process mining have leveraged LLMs with the aim of enhancing various aspects of business process management [Berti, 2024; Berti *et al.*, 2024; Berti *et al.*, 2023; Busch *et al.*, 2023; Grohs *et al.*, 2023; Jessen *et al.*, 2023; Kourani *et al.*, 2024b]. These studies collectively showcase the increasing interest toward the use of LLMs as a tool for improving the accuracy, efficiency, and accessibility of process mining and modeling. However, ensuring that the translations obtained via LLMs are both accurate and consistent with the original input remains a significant challenge.

Figure 1 illustrates the performance differences among various LLMs in translating between Declare models and natural language. Starting from a Declare model (center of the figure), colored boxes represent the result of a two-step translation process (from Declare to NL and back to Declare) performed by different LLMs using identical prompts. The figure reveals high variability in how LLMs manage this translations. High-performing models, such as GPT-4,



Original Declare Model
NotChainResponse $1 $4
NotChainResponse $4 $3
ChainResponse $3 $4
RespondedExistence $0 $4
AlternateResponse $1 $3

gemma2

If $0 occurs, then $4 must occur as well. $4 must not immediately follow $1 and $4 must not immediately follow $3. Every time $3 occurs, $4 must occur immediately after. If $0 occurs, then $4 must occur as well. Every time $1 occurs, $3 must occur afterward, before $1 recurs.

RespondedExistence $0 $4
NotChainResponse $1 $4
NotChainResponse $3 $4
ChainResponse $3 $4
AlternateResponse $1 $3
RespondedExistence $0 $4

mistral-nemo

If $1 occurs, it is not immediately followed by $4. If $4 occurs, it is not immediately preceded by $1. $3 must occur immediately after $4 and before any recurrence of $1. If $0 occurs, then $4 must also occur in the process instance. Each time $1 occurs, it is followed by $3 before $1 recurs again.

NotChainResponse $1 $4
NotPrecedence $4 $1
ChainResponse $4 $3
Precedence $3 $1
Response $0 $4

qwen2-72b

NotChainResponse $1 $4
NotChainResponse $4 $4
ChainResponse $3 $4
Response $0 $4
ChainResponse $1 $3

Immediately after $1 occurs, $4 cannot follow; similarly, $4 cannot immediately follow itself. Whenever $3 happens, $4 must occur right away. If $0 takes place at any point, then $4 is guaranteed to happen later in the process instance. Each time $1 appears, $3 will respond subsequently, without any other activities intervening before $3 occurs.

gemma2-27b

If $0 occurs, then $4 must also occur. Every time $3 occurs, it must be immediately followed by $4. Every time $1 occurs, it must be followed by $3 before $1 can occur again. Neither $1 nor $4 can occur immediately after each other.

RespondedExistence $0 $4
ChainResponse $3 $4
AlternatePrecedence $1 $3
NotChainResponse $1 $4
NotChainResponse $4 $1

llama3.1-8b

Every time $1 occurs, $3 must not occur immediately afterward. Every time $4 occurs, $3 must not occur immediately afterward. Every time $3 occurs, $4 must occur immediately after. If $0 occurs, then $4 must occur as well in the future. Each time $1 occurs, $3 must occur afterward.

Response $3 $4
NotChainResponse $1 $3
NotChainResponse $4 $3
Response $0 $4
AlternatePrecedence $1 $3

GPT4o

$1 must not be immediately followed by $4. $4 must not be immediately followed by $3. Every time $3 occurs, $4 must occur immediately afterward. If $0 occurs, $4 must also occur in the process. Each time $1 occurs, $3 must occur afterward, before $1 occurs again.

NotChainResponse $1 $4
NotChainResponse $4 $3
ChainResponse $3 $4
RespondedExistence $0 $4
AlternateResponse $1 $3

llama3.1-70b

When $1 occurs, $4 does not immediately follow. When $4 occurs, $3 does not immediately follow. However, every time $3 occurs, it is immediately followed by $4. If $0 occurs, then $4 must also occur. Additionally, every time $1 occurs, $3 eventually follows, but before the next occurrence of $1.

NotChainResponse $1 $4
NotChainResponse $4 $3
ChainResponse $3 $4
RespondedExistence $0 $4
AlternateResponse $1 $3

GPT4

$1 must never be immediately followed by $4, and $4 must never be immediately followed by $3. However, each time $3 occurs, $4 must occur immediately after it. If $0 occurs at any point, $4 must also occur at some point. Additionally, every time $1 occurs, $3 must occur afterward, before $1 occurs again.

NotChainResponse $1 $4
NotChainResponse $4 $3
ChainResponse $3 $4
RespondedExistence $0 $4
AlternateResponse $1 $3

llama3.2

Every time $1 occurs, it is not followed by $3 immediately after. Every time $3 occurs, it is preceded by $4 and no other $3 can occur in between. However, every time $3 occurs, $4 must occur immediately before it, but $4 also cannot be followed by $3 immediately after.

NotChainResponse $3 $4
AlternatePrecedence $4 $3

Figure 1: An example of how different LLMs perform in the bidirectional translation between Declare models and natural language.

achieve precise and reliable translations in both directions. In contrast, models like Gemma2 often prioritize simplicity, occasionally at the expense of accuracy, while others, such as Mistral-Nemo, introduce redundancies and inconsistencies in their outputs.

**Contributions.** In this paper, we provide a framework for the quantitative assessment of the ability of Large Language Models to translate between natural language descriptions and formal Declare specifications. We also evaluate the accuracy and reliability of LLM-generated specifications using purpose-built metrics that address both syntactic and semantic correctness. The goal is to address the following research questions: *(i)* How accurately can LLMs translate between formal Declare specifications and natural language descriptions? *(ii)* How do different LLMs compare in terms of performance? *(iii)* What are the key challenges and limitations?

**Organization.** The rest of the paper is organized as follows. Section 2 reviews related literature. Section 3 provides some background. Section 4 introduces the framework, and Section 5 reports the results of the experimental evaluation. Finally, Section 6 concludes the paper.

## 2 Related Work

The integration of Large Language Models (LLMs) in process mining has gained attention, focusing on capabilities like process discovery and conformance checking. Berti et al. [Berti *et al.*, 2023] highlight the need for benchmarks to evaluate LLMs in process mining tasks. Other works, such as [Berti *et al.*, 2024], propose key LLM capabilities like text-to-SQL conversion and evaluation strategies for ensuring output accuracy. Tools like PM4Py.LLM [Berti, 2024] and ProMoAI [Kourani *et al.*, 2024a; Kourani *et al.*, 2024b] explore practical applications, leveraging LLMs to generate and refine process models. Recent studies [Busch *et al.*, 2023; Jessen *et al.*, 2023] focus on optimizing prompt engineering for better LLM performance in process management. Fahland et al. [Fahland *et al.*, 2024] explored LLMs for explainable AI in business processes through the SAX4BPM framework, integrating process discovery, causal modeling, and XAI. Bernardi et al. [Bernardi *et al.*, 2024] introduced an LLM framework using Retrieval-Augmented Generation (RAG) to enhance process knowledge in public administration. LLMs have also been applied to process discovery, for translating textual descriptions into imperative and declarative models [Fill *et al.*, 2023; Fuggitti and Chakraborti, 2023; Grohs *et al.*, 2023], and for learning temporal formulas that separate positive and negative examples [Duranti *et al.*, 2024] (also known as *passive learning* [Bordais *et al.*, 2025], where symbolic techniques are more typical [Camacho and McIlraith, 2019; Raha *et al.*, 2022; Ielo *et al.*, 2023]). Conversational modeling has similarly used LLMs to iteratively refine process models based on feedback [Klievtsova *et al.*, 2023].

LLMs have also been used to generate Linear Temporal Logic (LTL) formulas from natural language descriptions. Tools like nl2spec [Cosler *et al.*, 2023] and SYNTHTL [Hahn and Trippel, 2024] use LLMs to convert natural language into LTL, while benchmarks such as LTLBench [Tang and Belle, 2024] and TIMEBENCH [Chu *et al.*, 2024] provide standardized tests for evaluating LLMs' performance in handling complex temporal logic tasks.

## 3 Preliminaries

### 3.1 The Declare Language

Process Mining [van der Aalst, 2022] lies at the intersection of Process Science and Data Science, focusing on analyzing and improving processes, which are sequences of activities

| Class | Template name | $\mathrm{LTL_p}$ formula |
|---|---|---|
| Existence | $\mathit{existence}(x)$ | $\mathsf{F}\,(x)$; same as $\mathit{existence}(1, x)$ |
| | $\mathit{existence}(n, x)$ | $\mathsf{F}\,(x \wedge \mathsf{X}\,(\mathit{existence}(n\text{-}1, x)))$ |
| | $\mathit{absence}(x)$ | $\neg \mathit{existence}(x)$; same as $\mathit{absence}(1, x)$ |
| | $\mathit{absence}(n, x)$ | $\neg \mathit{existence}(n, x)$ |
| | $\mathit{exactly}(x)$ | $\mathit{existence}(x) \wedge \mathit{absence}(2, x)$ |
| | $\mathit{exactly}(n, x)$ | $\mathit{existence}(n, x) \wedge \mathit{absence}(n\text{+}1, x)$ |
| Choice | $\mathit{choice}(x, y)$ | $\mathsf{F}\,(x) \vee \mathsf{F}\,(y)$ |
| | $\mathit{exclusive\text{-}choice}(x, y)$ | $\mathit{choice}(x, y) \wedge \neg(\mathsf{F}\,(x) \wedge \mathsf{F}\,(y))$ |
| Relation | $\mathit{responded\text{-}existence}(x, y)$ | $\mathsf{F}\,(x) \rightarrow \mathsf{F}\,(y)$ |
| | $\mathit{coexistence}(x, y)$ | $\mathit{responded\text{-}existence}(x, y) \wedge \mathit{responded\text{-}existence}(y, x)$ |
| | $\mathit{response}(x, y)$ | $\mathsf{G}\,(x \rightarrow \mathsf{F}\,(y))$ |
| | $\mathit{precedence}(x, y)$ | $(\neg y \;\mathsf{U}\; x) \vee \mathsf{G}\,(\neg y)$ |
| | $\mathit{succession}(x, y)$ | $\mathit{response}(x, y) \wedge \mathit{precedence}(x, y)$ |
| | $\mathit{alternate\text{-}response}(x, y)$ | $\mathsf{G}\,(x \rightarrow \mathsf{X}\,(\neg x \;\mathsf{U}\; y))$ |
| | $\mathit{alternate\text{-}precedence}(x, y)$ | $\mathit{precedence}(x, y) \wedge \mathsf{G}\,(y \rightarrow \mathsf{X_w}(\mathit{precedence}(x, y)))$ |
| | $\mathit{alternate\text{-}succession}(x, y)$ | $\mathit{alternate\text{-}response}(x, y) \wedge \mathit{alternate\text{-}precedence}(x, y)$ |
| | $\mathit{chain\text{-}response}(x, y)$ | $\mathsf{G}\,(x \rightarrow \mathsf{X}\,(y))$ |
| | $\mathit{chain\text{-}precedence}(x, y)$ | $\mathsf{G}\,(\mathsf{X}\,(y) \rightarrow x) \wedge \neg y$ |
| | $\mathit{chain\text{-}succession}(x, y)$ | $\mathit{chain\text{-}response}(x, y) \wedge \mathit{chain\text{-}precedence}(x, y)$ |
| Negation | $\mathit{neg\text{-}response}(x, y)$ | $\mathsf{G}\,(x \rightarrow \neg\mathsf{X}\,\mathsf{F}\,(y))$ |
| | $\mathit{neg\text{-}chain\text{-}response}(x, y)$ | $\mathsf{G}\,(x \rightarrow \neg\mathsf{X}\,(y))$ |

Table 1: Declare templates organized in classes. $x$ and $y$ are placeholders for activities.

aimed at achieving specific goals. Process models can be categorized into two types: imperative models, which explicitly define all valid process executions, and declarative models, which specify desired properties through constraints that valid process executions must satisfy, rather than prescribing a step-by-step flow [Di Ciccio and Montali, 2022]. Declarative modeling is flexible, as it allows any behavior that does not violate the given set of constraints. The most widely used declarative modeling language is Declare [van der Aalst *et al.*, 2009], a constraint-based language, whose constraints are formalized as formulas using Linear Temporal Logic on Finite Traces ($\mathrm{LTL_f}$) [De Giacomo and Vardi, 2013], and actually, one of its variants called Linear Temporal Logic on Process Traces ($\mathrm{LTL_p}$) [Fionda and Greco, 2018], providing a robust framework for reasoning about temporal properties of processes.

$\mathrm{LTL_p}$ is a modal logic used to describe temporal relationships between events over a finite, linearly ordered timeline. Temporal operators of $\mathrm{LTL_p}$ relate events that occur at different points in time. The logic is built on a set $\mathcal{A}$ of activities, which are treated as propositional variables, and uses standard Boolean connectives and some temporal operators. Formally, $\mathrm{LTL_p}$ formulas $\varphi$ are constructed according to the following grammar:

$$\varphi ::= \top \mid a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{X}\,\varphi \mid \varphi\,\mathsf{U}\,\varphi,$$

where $a$ is any activity from the set $\mathcal{A}$, and $\varphi$, $\varphi_1$, and $\varphi_2$ are $\mathrm{LTL_p}$ formulas. Common propositional logic operators (e.g., disjunction $\vee$, implication $\rightarrow$, and co-implication $\leftrightarrow$) are assumed, along with several shorthand notations for temporal operators: *Eventually* $\mathsf{F}\,\varphi \equiv \top\,\mathsf{U}\,\varphi$, *Always* $\mathsf{G}\,\varphi \equiv \neg\mathsf{F}\,\neg\phi$, and *Weak Next* $\mathsf{X_w}\varphi \equiv \neg\mathsf{X}\,\neg\varphi \equiv \mathsf{X}\,\varphi\ \vee \neg\mathsf{X}\,\top$.

A *finite trace* is a sequence $\pi = \pi_0 \cdots \pi_{n-1}$ of $n$ *state*, where each state $\pi_i \subseteq \mathcal{A}$ and $|\pi_i| = 1$ for $i = 0, \ldots, n-1$, and $n \in \mathbb{N}$. The length of the trace, denoted $|\pi|$, is the number of states in the sequence. Let $\varphi$ be an $\mathrm{LTL_p}$ formula, $\pi$ a finite trace, and $0 \leq i < |\pi|$ an index. The *satisfaction* relation, denoted by $\pi, i \models \varphi$, indicates that the formula $\varphi$ is satisfied at position $i$ of the trace $\pi$ and is defined recursively as follows:

$$\begin{array}{lll}
\pi, i \models \top & \text{iff} & i < |\pi| \\
\pi, i \models a & \text{iff} & a \in \pi_i; \\
\pi, i \models \neg\varphi' & \text{iff} & \pi, i \models \varphi' \text{ does not hold;} \\
\pi, i \models (\varphi_1 \wedge \varphi_2) & \text{iff} & \pi, i \models \varphi_1 \text{ and } \pi, i \models \varphi_2; \\
\pi, i \models \mathsf{X}\,(\varphi') & \text{iff} & i < |\pi| \text{ and } \pi, i+1 \models \varphi'; \\
\pi, i \models (\varphi_1 \;\mathsf{U}\; \varphi_2) & \text{iff} & \exists j \text{ with } i \leq j < |\pi| \text{ such} \\
 & & \text{that } \pi, j \models \varphi_2 \text{ and } \forall k \text{ with} \\
 & & i {\leq} k {<} j \text{ it holds that } \pi, k \models \varphi_1.
\end{array}$$

Whenever $\pi, 0 \models\varphi$ holds, we say that $\pi$ satisfies $\varphi$ (also indicated by $\pi{\models}\varphi$).

The language underlying Declare provides a wide range of commonly used template, as shown in Table 1. These templates are categorized into four main classes [Maggi *et al.*, 2011]: $\mathit{existence}$ template that specify whether a given target activity must or must not be executed, potentially a certain number of times; $\mathit{choice}$ template, that model decisions about the execution of activities, indicating that one or more activities must be chosen; $\mathit{relation}$ template, that indicate that if a source activity is executed, a corresponding target activity must also be executed, possibly with additional conditions; $\mathit{negation}$ template, which ensure that if a source activity is executed, the target activity must not be executed, potentially with further restrictions. A Declare *formula* (over $\mathcal{A}$) is a conjunction of Declare constraints, that are Declare template instantiated with some activities in $\mathcal{A}$.

### 3.2 Large Language Models

Large Language Models (LLMs) represent a significant advancement in the field of artificial intelligence, particularly in natural language processing. These models are pre-trained on vast amounts of text data and fine-tuned to perform various tasks, such as text generation, summarization, translation, and question-answering. LLMs operate based on deep learning architectures, typically employing transformer models, such as BERT [Devlin *et al.*, 2019], GPT [Radford, 2018], and their numerous variations. The transformer architecture [Vaswani *et al.*, 2017] has become the foundation for modern LLMs due to its ability to handle long-range dependencies. The central concept behind LLMs is their capacity to learn language patterns and semantic meanings from

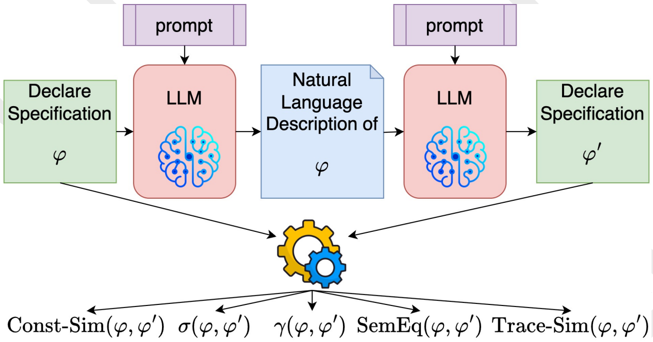

prompt
prompt
Declare Specification
$\varphi$
LLM
Natural Language Description of
$\varphi$
LLM
Declare Specification
$\varphi'$
Const-Sim$(\varphi, \varphi')$
$\sigma(\varphi, \varphi')$
$\gamma(\varphi, \varphi')$
SemEq$(\varphi, \varphi')$
Trace-Sim$(\varphi, \varphi')$

Figure 2: Framework for evaluating LLM capabilities over Declare specifications.

extensive datasets. This allows them to generalize well to various NLP tasks without the need for task-specific training from scratch. LLMs are trained using self-supervised learning, where the model predicts missing words or sequences in a sentence. This pre-training process, followed by fine-tuning on specific tasks, equips the model to handle a wide range of linguistic challenges, including ambiguity, context-switching, and even creativity in language generation.

## 4 Framework

The framework we propose is summarized in Figure 2. It consists in a bidirectional translation between formal Declare specifications and their corresponding natural language descriptions using Large Language Models (LLMs). The procedure begins with an initial formal specification in Declare, denoted as $\varphi$. First the LLM is asked to convert the formal Declare specification $\varphi$ into a natural language description. A pre-defined prompt including some information regarding Declare syntax and constraints semantics is used to guide the LLM in translating the set of constraints into an understandable natural language representation. This natural language description serves as a human-readable explanation of the constraints encoded in $\varphi$. In the next step, the natural language description produced by the LLM is used to ask to the same LLM to translate it back into a Declare specification (the result of this second translation is denoted as $\varphi'$). Using another tailored prompt, the LLM attempts to regenerate a formal specification that matches the original Declare specification $\varphi$. The final step of the framework involves evaluating the accuracy of the bidirectional translation. The two Declare specifications, $\varphi$ (the original specification) and $\varphi'$ (the regenerated specification), are compared using the metrics described below.

**Constraint-based Similarity.** This metric compares the set of constraints of the original and generated specifications. It allows to determine how closely the structure and syntax of the generated specification matches the original one. More formally, it is defined as the Jaccard similarity between the set of constraints in $\varphi$ and the set of constraints in $\varphi'$:

$$\text{Const-Sim}(\varphi, \varphi') = \frac{|C_\varphi \cap C_{\varphi'}|}{|C_\varphi \cup C_{\varphi'}|}$$

where $C_\varphi$ represents the set of constraints in the original specification, $C_{\varphi'}$ represents the set of constraints in the generated specification, $|\cdot|$ denotes the cardinality (i.e., the number of elements in the set), $C_\varphi \cap C_{\varphi'}$ represents the intersection of the two sets (common constraints), $C_\varphi \cup C_{\varphi'}$ represents the union of the two sets (all distinct constraints from both specifications). This measure provides a value between $0$ and $1$, where $1$ indicates perfect similarity (i.e., all constraints are identical), and $0$ indicates no similarity (i.e., no common constraints).

**Soundness, Completeness and Semantic Equivalence.** To verify that two Declare models, $\varphi$ and $\varphi'$, describe the same process behavior despite potential syntactic differences, we assess their *soundness* ($\sigma$), *completeness* ($\gamma$), and *semantic equivalence* (SemEq). This is achieved by translating both $\varphi$ and $\varphi'$ into equivalent $\text{LTL}_\text{p}$ formulas and evaluating whether they impose the same constraints on process executions, that is if $\varphi \rightarrow \varphi'$ and $\varphi' \rightarrow \varphi$. Formally, these measures are defined as follows:

$$\sigma(\varphi, \varphi') = \begin{cases} 1 & \forall \pi \in \mathcal{A}^*, \pi \models \text{LTL}_\text{p}(\varphi) \Leftarrow \pi \models \text{LTL}_\text{p}(\varphi') \\ 0 & \text{otherwise} \end{cases}$$

$$\gamma(\varphi, \varphi') = \begin{cases} 1 & \forall \pi \in \mathcal{A}^*, \pi \models \text{LTL}_\text{p}(\varphi) \Rightarrow \pi \models \text{LTL}_\text{p}(\varphi') \\ 0 & \text{otherwise} \end{cases}$$

$$\text{SemEq}(\varphi, \varphi') = \min(\sigma(\varphi, \varphi'), \gamma(\varphi, \varphi'))$$

Where: $\mathcal{A}^*$ is the set of all possible finite traces over the alphabet $\mathcal{A}$ of activities of the process under consideration, $\pi$ represents a specific finite trace (process execution), $\text{LTL}_\text{p}(\varphi)$ and $\text{LTL}_\text{p}(\varphi')$ correspond to the $\text{LTL}_\text{p}$ representations of the original $\varphi$ and generated $\varphi'$ Declare specifications, $\pi \models \phi$ denotes whether the formula $\phi$ is satisfied by the trace $\pi$ under $\text{LTL}_\text{p}$ semantics. The measures check that for every possible trace $\pi$, the relationships between $\varphi$ and $\varphi'$ accurately reflect their intended equivalence under $\text{LTL}_\text{p}$ semantics: soundness ensures that all behaviors allowed by $\varphi'$ are also allowed by $\varphi$, completeness verifies that all behaviors allowed by $\varphi$ are also allowed by $\varphi'$, and semantic equivalence ensures that $\varphi$ and $\varphi'$ yield the same truth value (both satisfying or both not satisfying) across all possible traces. When soundness and completeness both hold, the specifications are deemed *semantically equivalent*, indicating that they impose identical constraints on process executions and fully capture the same process behavior.

**Trace-based Similarity.** This metric compares how $\varphi$ and $\varphi'$ behave across a set of generated process traces. By examining their responses to execution traces up to a given length $k$, we quantify the behavioral similarity between the two specifications. More formally it is defined as:

$$\text{Trace-Sim}(\varphi, \varphi', k) = \\ 1 - \frac{1}{|\mathcal{T}_k|} \sum_{\pi \in \mathcal{T}_k} \mathbb{I}\left(\pi \models \text{LTL}_\text{p}(\varphi) \not\Leftrightarrow \pi \models \text{LTL}_\text{p}(\varphi')\right)$$

where $\mathcal{T}_k$ is the set of traces of length up to $k$, $\mathbb{I}(\cdot)$ is the indicator function, which returns $1$ if the condition inside is true (i.e., the $\mathrm{LTL_p}$ representations of $\varphi$ and $\varphi'$ do not have the same truth value on trace $\pi$), and 0 otherwise, $|\mathcal{T}_k|$ is the number of traces in the set $\mathcal{T}_k$. This metric provides a normalized score between 0 and 1, where 1 means $\varphi$ and $\varphi'$ behave identically on all traces and 0 means $\varphi$ and $\varphi'$ behave differently on all traces up to length $k$.

The proposed framework is agnostic to machine learning models and allows for a comprehensive assessment of the LLMs' performance in translating between formal Declare specifications and natural language. By evaluating both the syntactic and semantic properties of the generated specifications, we can determine the effectiveness and reliability of the LLMs in preserving the meaning of formal process models.

# 5 Experimental Evaluation

## 5.1 Experimental Setting

**Dataset.** The experimental evaluation is conducted on a synthetic benchmark dataset. The dataset aims to test the translation capabilities of LLMs under controlled complexity. The dataset consists of Declare specifications generated by varying two parameters: *(i)* the number $n$ of Declare constraints, and *(ii)* the number $m$ of distinct activities. For each pair of $(n, m)$, we generated 15 satisfiable models (i.e., models admitting at least one satisfying execution trace) by randomly picking $n$ Declare constraints and instantiating them with activities selected uniformly at random from the $m$ available ones. We considered the following combinations of $n$ and $m$: (5,3), (5,5), (5,8), (10,3), (10,5), (10,8), (15,5), (15,8), (15,10), (20,5), (20,8), (20,10).

**Prompts.** To inform the LLM regarding the syntax and semantics of Declare, we provide a fixed prompt to the LLM with a detailed list of Declare constraints, their associated semantics, and examples illustrating their application. This prompt establishes the context for the LLM, defining its task as that of translating between the formal Declare modelling language and natural language. For instance, the translation from Declare to natural language emphasizes precise, unambiguous English descriptions of constraints, such as "Response $0 $1" translating into "Every time $0 occurs, $1 must occur as well in the future." Similarly, translating from natural language to Declare involves the LLM interpreting structured sentences and outputting the corresponding formal constraints.

**LLMs.** We evaluate the performance of several state-of-the-art LLMs for translating between natural language and Declare specifications: Gemma2, Gemma2-27B, LLaMA3.1-8B, LLaMA3.1-70B, LLaMA3.2, LLaMA3.3, Mistral-Nemo, Qwen2-72B, GPT4-Turbo and GPT4o. The tested LLMs span a range of capabilities and architectures, offering different strengths and trade-offs. Lightweight models like Gemma2 prioritize speed and efficiency, making them suitable for simpler tasks, though they may struggle with complex logical reasoning. Mid-tier models such as LLaMA3.1-8B and Mistral-Nemo balance efficiency with enhanced contextual fluency, excelling in readability and interpretability, though they may sacrifice precision in intricate tasks. On the higher end, models like LLaMA3.3 and GPT4 have improved logical reasoning capabilities and leverage large parameter counts that allow to handle nuanced and complex inputs, showcasing higher robustness. Each model has been evaluated on the translation of Declare specifications into natural language descriptions and the reverse translation of natural language descriptions into Declare specifications.

**Implementation details.** The developed framework handles the translation of Declare specifications into natural language (and vice versa) using OpenAI APIs to interact with the various LLMs[1]. For the computation of semantic metrics, it leverages a symbolic approach based on Answer Set Programming (ASP [Brewka *et al.*, 2011; Gelfond and Lifschitz, 1991]) and $\mathrm{LTL_f}$ satisfiability solvers. Let $\varphi$ be the original Declare specification and $\varphi'$ the Declare specification resulting from the bidirectional translation. The Soundness and Completeness metrics are computed using the $\mathrm{LTL_f}$ solver `aaltaf` [Li *et al.*, 2020], which checks the validity of the $\mathrm{LTL_p}$ formulas $\varphi \rightarrow \varphi'$ and $\varphi' \rightarrow \varphi$. The results of these checks are then combined to derive the Semantic Equivalence metric. Trace-based Similarity is calculated using a system for *answer set model counting* [Eiter *et al.*, 2024], applied to an ASP encoding for bounded satisfiability [Fionda *et al.*, 2024] of $\mathrm{LTL_f}$ specifications. Informally, this approach *counts* the number of counterexamples (of bounded length) to the $\mathrm{LTL_p}$ formula $\varphi \leftrightarrow \varphi'$. Trace-based Similarity was computed using counterexamples of length up to 10. All data, prompts and code to reproduce the experiment is available in supplementary material.

## 5.2 Results

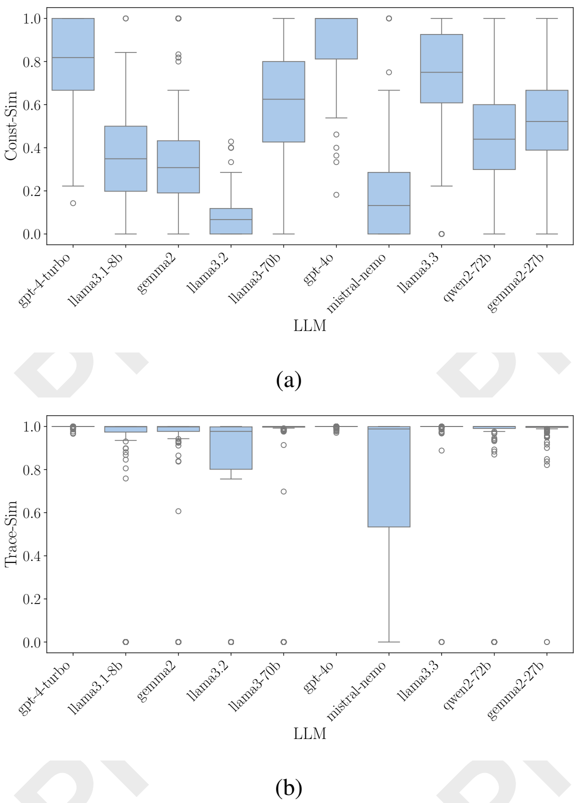
In this section, we analyze the translation capabilities of the considered LLMs via the specific metrics discussed in Section 4 (i.e., Constraint-based Similarity, Soundness, Completeness, Semantic Equivalence, and Trace-based Similarity) on the synthetic dataset discussed in the previous section.

### Performance Overview

Figure 3 provides a comparison of LLM performance using box plots for Constraint-based Similarity and Trace-based Similarity. The median values and interquartile ranges indicate significant differences across models, especially for Constraint-based Similarity. GPT4o shows the highest median performance across all metrics, with narrow interquartile ranges, indicating consistent accuracy. Among the top-performing LLMs, LLaMA3.3 and GPT4-Turbo show higher variability, suggesting its performance is less reliable for complex configurations.

The bar chart reported in Figure 4 illustrates the proportion of formulas for which the binary metrics –Semantic Equivalence, Completeness, and Soundness– hold across the various large language models (LLMs) on all input specifications. The LLMs are reported on the x-axis, while the y-axis shows the proportion of formulas (ranging from 0 to 1) where the corresponding semantic metric evaluates to 1. The chart reveals that models such as GPT-4-Turbo, GPT-4o and LLaMA3.3 demonstrate consistently high proportions across

[1] All LLMs, except for the GPT-family, were run locally using the `ollama` project.



(a)

(b)

Figure 3: Performance of all LLMs across the two metrics: (a) Constraint-based Similarity and (b) Trace-based Similarity.

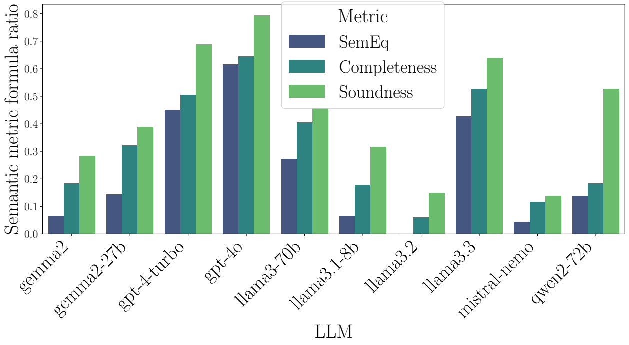


Figure 4: Semantic metrics evaluation across different LLMs.

all metrics, indicating robust performance in maintaining semantic properties. In contrast, variability is observed in other models; for instance, Qwen2-72B shows lower proportions for Completeness but high proportion for Soundness. Overall, LLMs appear to be more adept at generating Soundness translations than Complete ones, suggesting that the resulting output specifications are frequently overconstrained.

Figure 5 shows the performance of the various LLMs across different parameter configurations. The shaded regions around the lines represent confidence intervals, reflecting the variability among Declare models for the respective parameter settings. As it can be noted, as the number of activities increases, there is a consistent downward trend in both Constraint-based Similarity and Trace-based Similarity. In particular, some models, such as Gemma2 and LLaMA3.2, show higher variability as activities increase, indicating higher sensitivity to larger alphabets. Conversely, while an increase in constraints generally results in a decreasing performance for Constraint-based Similarity, Trace-based Similarity displays mixed trends, suggesting this metric is more robust wrt the complexity of logical constraints. Models like GPT4-Turbo, GPT4o and LLaMA3.3 shoe stronger performance in Constraint-based Similarity, while GPT4-Turbo, GPT4o and Gemma2-27B excel in Trace-based Similarity. Since larger shaded regions around the lines indicate inconsistent performance, we can notice that models such as Mistral-Nemo and LLaMA3.2 show a more accentuate variable behavior. In constrast, high-performing models, such as GPT4-Turbo and GPT4o, exhibit consistent and robust performance, as reflected by narrower confidence intervals across all plots.

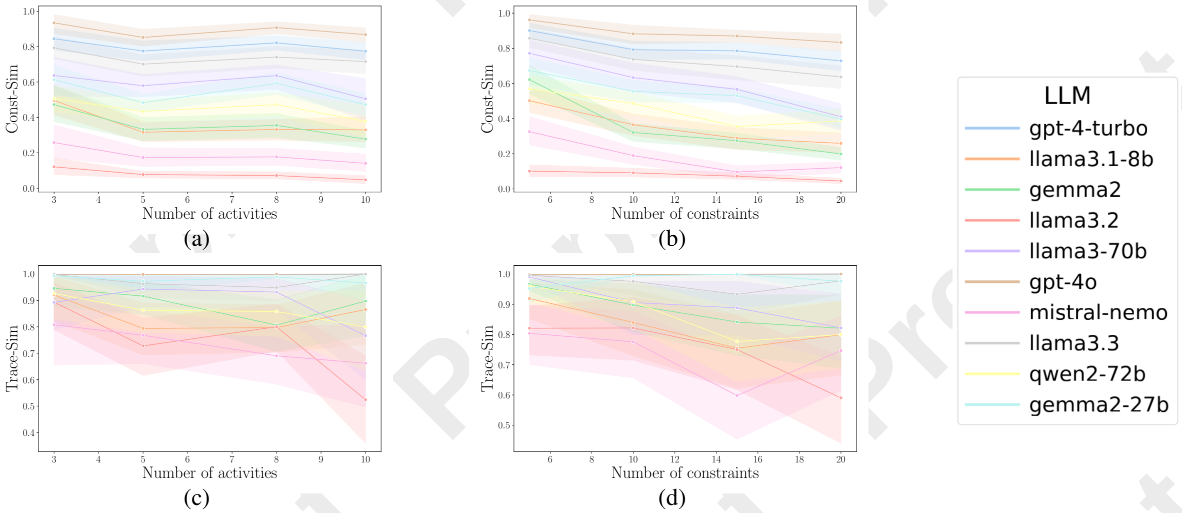
### Model-Specific Strengths

Radar charts reported in Figure 6 compare the performance of individual LLMs across all metrics. Each chart represents the performance of an individual LLM, highlighting its strengths and weaknesses. GPT4o shows strong and balanced performance across most metrics, with near-complete coverage in the radar chart, indicating robust capabilities according to both semantic and syntactic metrics. In contrast, models like LLAMA3.3 and GPT4-Turbo show moderate performance, achieving reasonable scores on Constraint-based Similarity and Trace-based Similarity while being weaker on Semantic Equivalence, Completeness, and Soundness. Mistral-Nemo and LLAMA3.2 exhibit weaker performance, with smaller coverage in the radar chart. Models such as Gemma2-27b and Qwen2-72b show mixed strengths, excelling in specific metrics like Trace-based Similarity while underperforming in other measures like Semantic Equivalence.

Many models struggle with Completeness and Soundness, which assess the logical implications between the original and generated formulas. These metrics are critical for ensuring semantic correctness, and the smaller coverage on these axes suggests that further improvement in logical reasoning capabilities is necessary for most models. However, most models perform well on Trace-based Similarity, suggesting that while semantic issues persist, the majority of process behaviors are accurately captured.

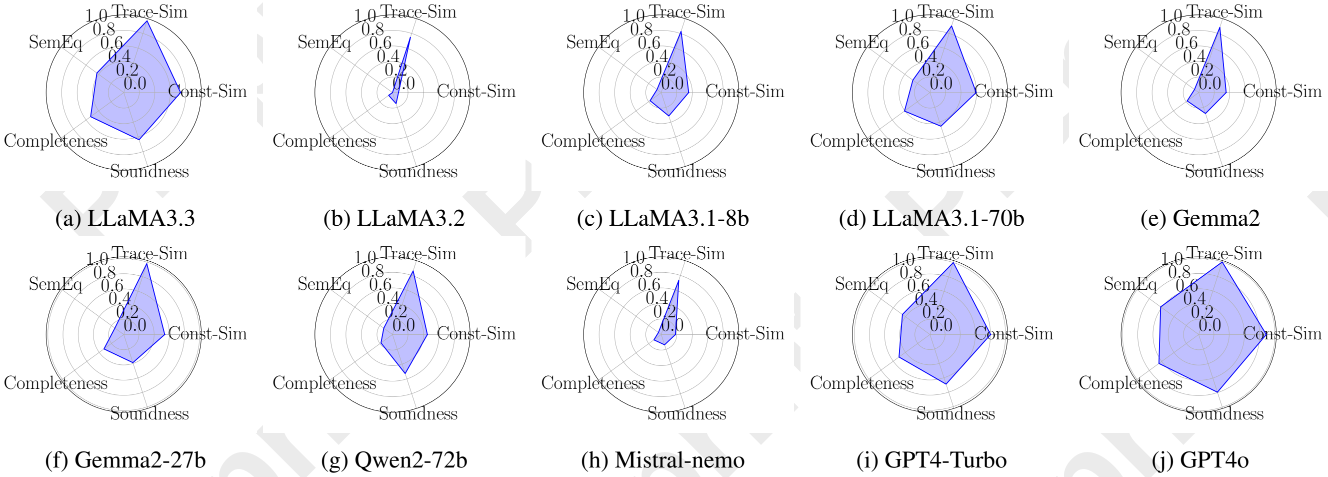
## 5.3 Discussion

Models such as GPT4-Turbo, GPT4o, LLaMA3.3 and LLaMA3.1-70B consistently show high performance, even when faced with high-complexity scenarios characterized by a large number of constraints and a smaller alphabet size. This indicates their strong ability to handle intricate logical relationships and maintain accurate translations under challenging conditions. In contrast, lightweight models like LLaMA3.2 and Mistral-Nemo, while offering efficiency in terms of computational resources, exhibit notable trade-offs in terms of accuracy and reliability.



(a) (b) (c) (d)

Figure 5: Line charts for the various metrics across all LLMs.



(a) LLaMA3.3 (b) LLaMA3.2 (c) LLaMA3.1-8b (d) LLaMA3.1-70b (e) Gemma2

(f) Gemma2-27b (g) Qwen2-72b (h) Mistral-nemo (i) GPT4-Turbo (j) GPT4o

Figure 6: Radar charts for the various LLM performance across all similarity measures.

A closer examination of GPT4o reveals its exceptional consistency and robustness. The narrow interquartile range observed in its performance metrics underscores its reliability, as it provides consistent results across diverse configurations. This reliability, combined with high accuracy, makes GPT4o a standout choice for both simple and complex tasks.

## 6 Concluding remarks

This paper presented an in-depth evaluation of Large Language Models (LLMs) for bidirectional translations between natural language descriptions and formal Declare specifications. Our results demonstrate that while some models, such as GPT4-Turbo and GPT4o, exhibit strong performance, other models like Mistral-Nemo and LLaMA3.2 show limitations, particularly in maintaining semantic equivalence and logical correctness. The findings emphasize the importance of model selection based on the task requirements. High-performing models provide a reliable foundation for applications requiring accurate and consistent translations, while lightweight models may be more suitable for simple process models or limited computational resources. Nonetheless, challenges persist when considering the soundness and completeness of LLM's translations, where even top-performing models struggle to maintain logical consistency between the original and generated specifications. Future works could focus on developing tailored fine-tuning strategies to address these limitations. Additionally, incorporating user feedback could further improve the applicability and reliability of LLMs in declarative process mining.

## Acknowledgments

This work was partially supported by the Italian Ministries MIMIT, under project EI-TWIN n. F/310168/05/X56 CUP B29J24000680005, project ASVIN n. F/360050/01-02/X75 CUP B29J2400020000; and MUR, under projects: PNRR FAIR - Spoke 9 - WP 9.1 and WP 9.2 CUP

H23C22000860006, Tech4You CUP H23C22000370006, and PRIN PINPOINT CUP H23C22000280006; and European Union – Next Generation EU through the MUR Project HypeKG within PRIN 2022 Program (CUP: H53D23003710006), and the MUR PRIN 2022-PNRR project DISTORT (CUP: H53D23008170001) under the Italian PNRR Mission 4 Component 1.

## References

[Bernardi *et al.*, 2024] Mario Luca Bernardi, Angelo Casciani, Marta Cimitile, and Andrea Marrella. A preliminary study on business process-aware large language models. In *Ital-IA*, 2024.

[Berti *et al.*, 2023] Alessandro Berti, Daniel Schuster, and Wil M. P. van der Aalst. Abstractions, scenarios, and prompt definitions for process mining with llms: A case study. In *BPM Workshops*, volume 492, pages 427–439. Springer, 2023.

[Berti *et al.*, 2024] Alessandro Berti, Humam Kourani, Hannes Hafke, Chiao-Yun Li, and Daniel Schuster. Evaluating large language models in process mining: Capabilities, benchmarks, evaluation strategies, and future challenges. *CoRR*, abs/2403.06749, 2024.

[Berti, 2024] Alessandro Berti. Pm4py.llm: a comprehensive module for implementing PM on llms. *CoRR*, abs/2404.06035, 2024.

[Bordais *et al.*, 2025] Benjamin Bordais, Daniel Neider, and Rajarshi Roy. The complexity of learning ltl, CTL and ATL formulas. In Olaf Beyersdorff, Michal Pilipczuk, Elaine Pimentel, and Kim Thang Nguyen, editors, *42nd International Symposium on Theoretical Aspects of Computer Science, STACS 2025, March 4-7, 2025, Jena, Germany*, volume 327 of *LIPIcs*, pages 19:1–19:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

[Brewka *et al.*, 2011] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.

[Busch *et al.*, 2023] Kiran Busch, Alexander Rochlitzer, Diana Sola, and Henrik Leopold. Just tell me: Prompt engineering in business process management. In *BPMDS/EMMSAD@CAiSE*, volume 479, pages 3–11. Springer, 2023.

[Camacho and McIlraith, 2019] Alberto Camacho and Sheila A. McIlraith. Learning interpretable models expressed in linear temporal logic. In J. Benton, Nir Lipovetzky, Eva Onaindia, David E. Smith, and Siddharth Srivastava, editors, *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2019, Berkeley, CA, USA, July 11-15, 2019*, pages 621–630. AAAI Press, 2019.

[Chu *et al.*, 2024] Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Haotian Wang, Ming Liu, and Bing Qin. Timebench: A comprehensive evaluation of temporal reasoning abilities in large language models. In *ACL (1)*, pages 1204–1228, 2024.

[Cosler *et al.*, 2023] Matthias Cosler, Christopher Hahn, Daniel Mendoza, Frederik Schmitt, and Caroline Trippel. nl2spec: Interactively translating unstructured natural language to temporal logics with large language models. In *CAV (2)*, volume 13965, pages 383–396. Springer, 2023.

[De Giacomo and Vardi, 2013] Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, pages 854–860, 2013.

[Devlin *et al.*, 2019] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*, pages 4171–4186, 2019.

[Di Ciccio and Montali, 2022] Claudio Di Ciccio and Marco Montali. Declarative process specifications: Reasoning, discovery, monitoring. In *PM Handbook*, volume 448, pages 108–152. Springer, 2022.

[Duranti *et al.*, 2024] Damiano Duranti, Paolo Giorgini, Andrea Mazzullo, Marco Robol, and Marco Roveri. Llm-driven knowledge extraction in temporal and description logics. In Mehwish Alam, Marco Rospocher, Marieke van Erp, Laura Hollink, and Genet Asefa Gesese, editors, *Knowledge Engineering and Knowledge Management - 24th International Conference, EKAW 2024, Amsterdam, The Netherlands, November 26-28, 2024, Proceedings*, volume 15370 of *Lecture Notes in Computer Science*, pages 190–208. Springer, 2024.

[Eiter *et al.*, 2024] Thomas Eiter, Markus Hecher, and Rafael Kiesel. aspmc: New frontiers of algebraic answer set counting. *Artif. Intell.*, 330:104109, 2024.

[Fahland *et al.*, 2024] Dirk Fahland, Fabiana Fournier, Lior Limonad, Inna Skarbovsky, and Ava J. E. Swevels. How well can large language models explain business processes? *CoRR*, abs/2401.12846, 2024.

[Fill *et al.*, 2023] Hans-Georg Fill, Peter Fettke, and Julius Köpke. Conceptual modeling and large language models: Impressions from first experiments with chatgpt. *Enterp. Model. Inf. Syst. Archit. Int. J. Concept. Model.*, 18:3, 2023.

[Fionda and Greco, 2018] Valeria Fionda and Gianluigi Greco. LTL on finite and process traces: Complexity results and a practical reasoner. *J. Artif. Intell. Res.*, 63:557–623, 2018.

[Fionda *et al.*, 2024] Valeria Fionda, Antonio Ielo, and Francesco Ricca. Ltlf2asp: Ltlf bounded satisfiability in ASP. In *LPNMR*, volume 15245 of *Lecture Notes in Computer Science*, pages 373–386. Springer, 2024.

[Fuggitti and Chakraborti, 2023] Francesco Fuggitti and Tathagata Chakraborti. NL2LTL – a python package for converting natural language (NL) instructions to linear temporal logic (LTL) formulas. In *ICAPS*, 2023.

[Gelfond and Lifschitz, 1991] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Gener. Comput.*, 9(3/4):365–386, 1991.

[Grohs *et al.*, 2023] Michael Grohs, Luka Abb, Nourhan Elsayed, and Jana-Rebecca Rehse. Large language models can accomplish business process management tasks. In *BPM Workshops*, volume 492, pages 453–465. Springer, 2023.

[Hahn and Trippel, 2024] Christopher Hahn and Caroline Trippel. Translating natural language to temporal logics with large language models and model checkers. In *FMCAD*, page 119, 2024.

[Ielo *et al.*, 2023] Antonio Ielo, Mark Law, Valeria Fionda, Francesco Ricca, Giuseppe De Giacomo, and Alessandra Russo. Towards ilp-based LTL f passive learning. In Elena Bellodi, Francesca Alessandra Lisi, and Riccardo Zese, editors, *Inductive Logic Programming - 32nd International Conference, ILP 2023, Bari, Italy, November 13-15, 2023, Proceedings*, volume 14363 of *Lecture Notes in Computer Science*, pages 30–45. Springer, 2023.

[Jessen *et al.*, 2023] Urszula Jessen, Michal Sroka, and Dirk Fahland. Chit-chat or deep talk: Prompt engineering for process mining. *CoRR*, abs/2307.09909, 2023.

[Klievtsova *et al.*, 2023] Nataliia Klievtsova, Janik-Vasily Benzin, Timotheus Kampik, Juergen Mangler, and Stefanie Rinderle-Ma. Conversational process modelling: State of the art, applications, and implications in practice. *CoRR*, abs/2304.11065, 2023.

[Kourani *et al.*, 2024a] Humam Kourani, Alessandro Berti, Daniel Schuster, and Wil M. P. van der Aalst. Process modeling with large language models. *CoRR*, abs/2403.07541, 2024.

[Kourani *et al.*, 2024b] Humam Kourani, Alessandro Berti, Daniel Schuster, and Wil M. P. van der Aalst. Promoai: Process modeling with generative AI. *CoRR*, abs/2403.04327, 2024.

[Li *et al.*, 2020] Jianwen Li, Geguang Pu, Yueling Zhang, Moshe Y. Vardi, and Kristin Y. Rozier. Sat-based explicit ltlf satisfiability checking. *Artif. Intell.*, 289:103369, 2020.

[Maggi *et al.*, 2011] Fabrizio Maria Maggi, Arjan J. Mooij, and Wil M. P. van der Aalst. User-guided discovery of declarative process models. In *CIDM*, pages 192–199, 2011.

[Radford, 2018] Alec Radford. Improving language understanding by generative pre-training. 2018.

[Raha *et al.*, 2022] Ritam Raha, Rajarshi Roy, Nathanaël Fijalkow, and Daniel Neider. Scalable anytime algorithms for learning fragments of linear temporal logic. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 263–280. Springer, 2022.

[Tang and Belle, 2024] Weizhi Tang and Vaishak Belle. Ltlbench: Towards benchmarks for evaluating temporal logic reasoning in large language models. *CoRR*, abs/2407.05434, 2024.

[van der Aalst *et al.*, 2009] Wil M. P. van der Aalst, Maja Pesic, and Helen Schonenberg. Declarative workflows: Balancing between flexibility and support. *Comput. Sci. Res. Dev.*, 23(2):99–113, 2009.

[van der Aalst, 2022] Wil M. P. van der Aalst. Process mining: A 360 degree overview. In *PM Handbook*, volume 448, pages 3–34. Springer, 2022.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30:5998–6008, 2017.