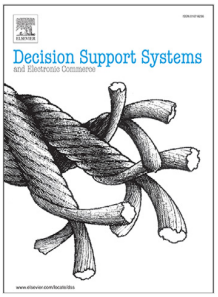
Journal Pre-proof



Extracting declarative constraints for process modeling from natural language descriptions with large language models

Gyunam Park, Julian Kofferath, Minsu Cho

PII: S0167-9236(26)00016-3
DOI: https://doi.org/10.1016/j.dss.2026.114627
Reference: DECSUP 114627

To appear in: *Decision Support Systems*

Extracting Declarative Constraints for Process Modeling From Natural Language Descriptions with Large Language Models

anonymous authors

Abstract

With the growing availability of unstructured text data in organizations, automating the extraction of process models from natural language descriptions has become increasingly crucial. Traditional rule-based techniques face challenges such as limited generalization and constraints to narrow sets of control-flow constructs. We present a systematic empirical study of Large Language Models (LLMs) for translating natural language sentences into declarative process constraints using the *Declare* modeling language. Through comprehensive evaluation across seven LLM architectures, we demonstrate that fine-tuned models achieve 97.8% template accuracy compared to 53.8% for existing rule-based approaches. Our fine-tuning approach substantially outperforms prompting techniques (99.4% vs. 56.5% template accuracy), establishing clear guidance for practical deployment. We contribute a benchmark dataset of 969 sentence-constraint pairs across 11 *Declare* templates, direction-sensitive evaluation metrics, and complete reproducibility materials. The extracted constraints enable practical decision support applications including compliance monitoring, conformance checking, and automated governance dashboards.

Keywords: Process Modeling, Automated Extraction, Large Language Models

1. Introduction

Process models are foundational to effective process management, serving as critical tools for understanding, analyzing, and optimizing business operations [1]. These models provide a structured representation of business activities, facilitating more precise communication and better compliance with standards. Accurately mapping out processes can significantly enhance operational efficiency and agility.

However, process actors and domain experts lack the expertise required to construct process models. Moreover, the process of eliciting these models can be exceedingly time-consuming [2]. Additionally, process-related information is often documented in more accessible formats, such as text documents [2, 3].

Recognizing this, researchers have turned to Natural Language Processing (NLP) approaches to extract process knowledge, especially process models, from textual sources [2, 3, 4, 5, 6, 7, 8, 9, 10]. This approach offers a promising way to convert unstructured textual data into formal process representations.

However, as described in [11, 12], the research field is fragmented and lacks systematic efforts. First, existing rule-based methods often lack flexibility, leading to poor generalization across varied text inputs. Second, existing methods support only a narrow range of control-flow constructs, limiting their applicability in complex process scenarios. Third, the field lacks a unified framework with robust datasets and evaluation metrics, which hinders consistent assessment and comparison of methods.

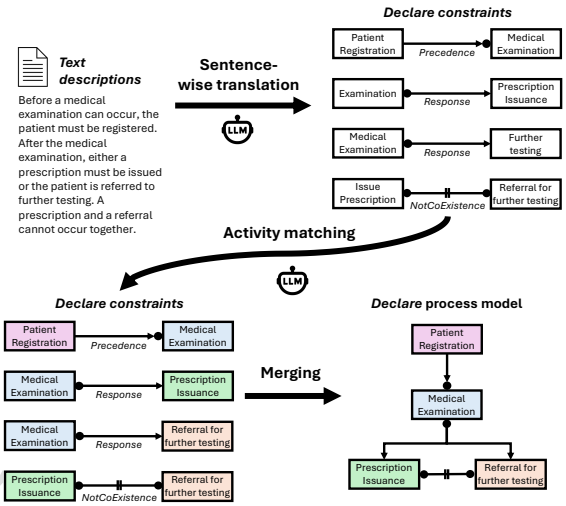


Figure 1: An illustration of the proposed method and a possible use case.

In this paper, we present a method to discover Declare process models from natural language descriptions using LLMs. First, as shown in Figure 1, a fine-tuned

LLM translates each sentence in the natural language description into a Declare constraint such as *Precedence(Patient Registration, Medical Examination)* and *Response(Examination, Prescription Issuance)*. Next, to build a cohesive Declare process model, we match activity names, e.g., *Examination* to *Medical Examination*. Finally, we merge Declare constraints to construct a Declare process model. The resulting process model can be used in various ways; it can serve as a benchmark for conducting declarative conformance checking against a de facto process model derived from event data [13].

Our approach contributes to the automated process extraction as follows:

1. *Reliable component for template translation:* By utilizing LLMs, our approach significantly improves generalization capabilities and accuracy over rule-based methods for sentence-to-constraint translation. The component adapts to diverse sentence structures and linguistic variations and supports 11 *Declare* templates.
2. *Benchmark dataset with provenance and policy:* We provide a benchmark-quality dataset of 969 textual descriptions paired with ground-truth *Declare* constraints across 11 templates, together with detailed provenance, annotation guidelines (including ambiguity policy), and intended use for benchmarking.
3. *Evaluation design and baselines:* We evaluate with separate metrics for template classification and activity extraction (direction-sensitive for ordered templates), and compare against both prompting and a rule-based state of the art, demonstrating higher reliability of the fine-tuned component.
4. *Reproducibility materials:* We release scripts and configuration to reproduce training, evaluation, and inference, along with hardware setup and runtime characteristics, enabling transparent replication.

This work focuses on a core component within a larger pipeline that transforms natural language documents into *Declare* constraints and models. The component translates a single sentence into a single constraint and currently assumes a one-sentence-to-one-constraint mapping. We discuss implications of this assumption and outline integration with upstream and downstream steps, such as sentence segmentation, multi-constraint detection and splitting (e.g., for conjunctions), and

activity matching, to form coherent models.

The remainder of this paper is organized as follows: In Section 2, we introduce related work concerning the synthesis of process models from natural language. Section 3 introduces Declare and decoder-only LLMs. Section 4 and Section 5 describe the extraction pipeline and employed techniques. Next, Section 6 evaluates the performance of our approach and compares it with existing methods. In Section 7, we present a detailed discussion of our findings, including limitations and opportunities for future research. Finally, Section 8 concludes this paper.

2. Related Work

Recently, translating natural language texts to process models has gained attention. Bellan et al. [11] compare ten state-of-the-art approaches to extracting process models from text, highlighting the need for common benchmarks and robust evaluation.

Classical NLP-based approaches translate text to imperative models (e.g., BPMN): De A. R. Gonçalves et al. [5] (storytelling mining), Friedrich et al. [6] (semantic parsing), Text Process Miner [7], Honkisz et al. [8], and Sholiq et al. [9]. Beyond full model discovery, works extract specific elements (activities, actors, gateways) [14, 15, 16, 17] and align text with existing models [18, 19, 20].

For declarative models, López et al. [4] use DCR graphs via BERT-based classification and NER, while Van der Aa et al. [2] propose an NLP pipeline to extract *Declare* constraints (five templates). Prompting LLMs has also been explored: Grohs et al. [21] show GPT-4 can outperform the pre-LLM baseline on several templates.

More recently, LLM-based methods have been applied to constraint extraction from documents and semantics-aware process mining. Barrientos et al. [22] and Mustroph et al. [23] translate compliance requirements from text to check them over event logs. Rebmann et al. [24] assess LLM capabilities for semantics-aware tasks in process mining, while Busch et al. [25] introduce explainable semantic anomaly detection using sequence-to-sequence models.

In parallel, conversational and imperative modeling with LLMs is emerging: Köpke and Safan [26] present efficient conversational process modeling; Klievtsova

et al. [27] survey conversational process modeling applications and implications.

Connections between imperative and declarative representations are relevant to our setting. Weidlich and Van der Werf [28] study relational semantics for Petri nets; Barbaro et al. [29] demonstrate deriving LTLf declarative specifications from sound workflow nets. Such mappings suggest that text-to-imperative datasets could be repurposed to derive text-to-declarative pairs in future work.

Finally, foundational LLM reports (e.g., Gemma, LLaMA, Mistral) and instruction tuning/PEFT literature underpin our design choices for fine-tuning and inference [30].

Our work focuses on a reusable component for sentence-to-*Declare* constraint translation across 11 templates, intended to plug into full NL-to-constraints pipelines. It complements (i) document-level constraint extraction and semantics-aware mining (broader granularity), (ii) conversational and imperative modeling (different artifacts), and (iii) imperative-to-declarative mappings (useful to enrich datasets). In this sense, our contribution lies at the intersection of fine-grained sentence translation and broader process modeling efforts.

Despite progress, prior work exhibits recurring limitations: (1) rule-based methods struggle to generalize across diverse linguistic structures; (2) many approaches support only a narrow subset of control-flow constructs; and (3) the field lacks standardized evaluation frameworks and benchmark datasets for consistent comparison. Our study addresses these gaps by (a) fine-tuning LLMs to improve generalization and direction-sensitive template recognition, (b) supporting a broad set of 11 *Declare* templates, and (c) contributing a benchmark dataset with clear annotation policy and an evaluation protocol that separates template and activity metrics.

3. Preliminaries

In this work, we design an approach to the automated extraction of *Declare* process models using *decoder-only LLMs*. In this section, we first introduce the process modeling language *Declare* and then explain the architecture and functionality of decoder-only LLMs.

3.1. Declare

Declare is a process modeling language built upon Linear Temporal Logic (LTL) that provides templates for expressing constraints on single or multiple activities. The templates originate from common temporal logic patterns used in model checking [31] and are categorized into occurrence patterns (e.g., Absence, Existence) and order patterns (e.g., Precedence, Response) [32].

Table 1: A selected set of common Declare templates.

Constraint	Textual	Graphical
Init(A)	A is the first to occur	
End(A)	A is the last to occur	
AtLeastOne(A)	A occurs at least once	
AtMostOne(A)	A occurs at most once	
Response(A,B)	If A occurs, then B occurs after A	
Precedence(A,B)	If B occurs, then A occurs before B	
RespondedExistence(A,B)	If A occurs, then B occurs before or after A	
ChainResponse(A,B)	If A occurs, then B occurs immediately afterwards	
ChainPrecedence(A,B)	If B occurs, then A occurs immediately beforehand	
CoExistence(A,B)	If A occurs, then B occurs before or after A, and vice versa	
Succession(A,B)	A occurs if and only if it is followed by B	
NotCoExistence(A,B)	A and B never occur together	
NotSuccession(A,B)	B cannot occur after A	

Given its focus on business processes, Declare interprets LTL specifications over finite-length process executions. Table 1 shows common Declare templates with their natural language descriptions and graphical notation.

Definition 1 (Declare Constraint). *Let $\mathcal{A}$ be the universe of activity names. A Declare constraint is characterized by the pair $(k(x_1,...,x_m),\kappa)$, where:*

- *$k(x_1,...,x_m)$ is a Declare template;*
- *κ is a mapping that associates each variable x_i (for $i \in \{1,...,m\}$) with an activity $\kappa(x_i)=a_i \in \mathcal{A}$.*

Such a Declare constraint can be represented as $k(a_1,...,a_m)$ for brevity.

A Declare process model is a collection of Declare constraints defined over consistent activity names. Figure 2 depicts an example of a declare model.

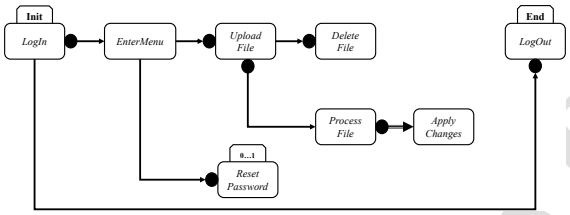


Figure 2: A graphical Declare model.

Definition 2 (Declare Process Model). *A Declare process model is a tuple* $DS = (Act, K)$*, comprising:*

- $Act \subseteq \mathcal{A}$ *is a set of activities;*
- K *is a set of Declare constraints defined over* Act*;*

3.2. Decoder-only LLMs

We adopt decoder-only Transformer architectures [33] trained with a causal language modeling objective: given a textual prompt, the model autoregressively predicts the next token under a causal mask. A high-level view is shown in Figure 3. For prompting baselines, we leverage instruction-following techniques (e.g., Chain-of-Thought and zero-shot reasoning) [34, 35], and note that self-consistency can further stabilize reasoning [36].

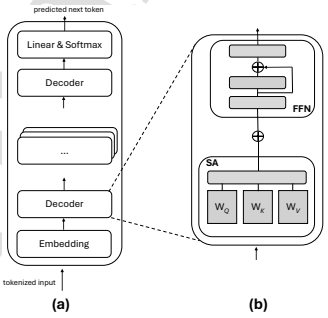


Figure 3: An overview of the LLM architecture. **(a)** shows the overall decoder-only model structure. **(b)** zooms into one decoder block, consisting of self-attention (SA), and feed-forward network (FFN) layers.

4. Fine-Tuning Large Language Models

This section outlines fine-tuning LLMs to translate text inputs to Declare constraints. Figure 4 demonstrates how the fine-tuned LLM is incorporated into the overall extraction pipeline. In the following, we first describe the downstream task for fine-tuning LLMs and then explain the fine-tuning process based on the downstream task.

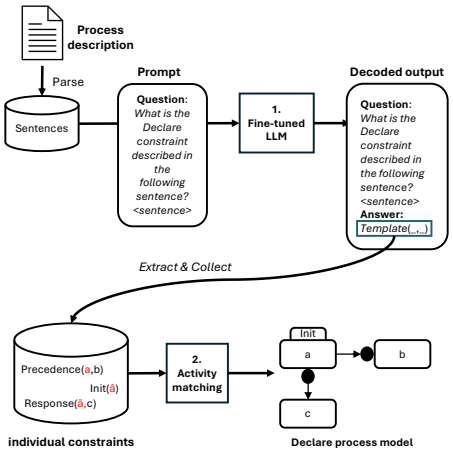


Figure 4: The overall pipeline required to transform a process description into a Declare model.

4.1. Downstream Task

The first step in the extraction pipeline involves passing each sentence from the input process description separately to the fine-tuned LLM to obtain the formal Declare constraint described by this sentence.

The fine-tuning process employs *causal language modeling*, where the model is trained to predict the next token in a sequence given the previous tokens. This method is particularly suitable for generation tasks, e.g., translating sentences into constraints, as it allows the model to learn the sequential nature of the output. The model generates constraints token-by-token, using the previously generated tokens and the input sentence context. During inference, the number of generated tokens can be bounded.

Additionally, a custom formatting function preprocesses the training data for the model. This function creates input sequences where the sentence (textual description) and its expected constraint are concatenated, ensuring that the model receives well-prepared inputs aligned with the task requirements. This leads to improved performance and generalization capabilities. The format is as follows:

`Question: What is the Declare constraint described in the following sentence? <description>`

`Answer: <gold standard>`

During inference, we prompt the fine-tuned model using the same format, prepending the question before the current sentence. For each inference iteration, i.e., for each input sentence, we obtain a response from the model that adheres to the above format.

To extract formal Declare constraints from the LLM's response at inference time, we use a regular expression pattern to search within the decoded output. This pattern is designed to match all considered types of Declare constraints (templates), such as *Precedence*, *Init*, *AtLeastOne*, etc., followed by activities enclosed in parentheses. By leveraging the fixed text-based format of Declare constraints, we can accurately identify and extract these constraints from the model's responses. Additionally, by defining the output format using the formatting function and limiting the number of generated tokens, we ensure that there is, at most, one match for the pattern in the decoded output.

4.2. Fine-Tuning

To adopt a decoder-only large language model (LLM) like Google's Gemma-7b for the task of translating input sentences into formal Declare constraints, we employ Low-Rank Adaptation (LoRA) [37]. LoRA is a parameter-efficient fine-tuning technique that modifies the LLM architecture by introducing low-rank adaptation matrices into the model's transformer layers, allowing efficient task-specific fine-tuning.

With LoRA, the fine-tuning process involves adding small, trainable low-rank matrices to the self-attention and feed-forward components of the model. These matrices adjust the query, key, and value projections in the self-attention mechanism, as well as the intermediate layers in the feed-forward network. Specifically, instead

of directly updating the large weight matrices, LoRA decomposes these updates into products of smaller matrices, significantly reducing the number of parameters that need to be fine-tuned. Figure 5 illustrates the affected components.

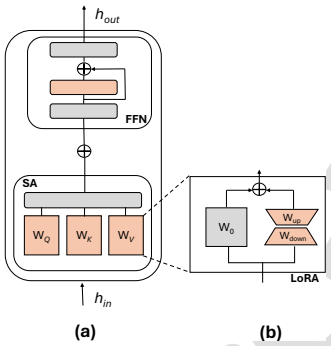


Figure 5: (a) zooms into one decoder block of the LLM, consisting of self-attention (SA), and feed-forward network (FFN) layers. **(b)** shows the detailed structure of the LoRA adaptation. The components shown in grey are kept frozen during fine-tuning, whereas the orange components contain updated weights.

For a given pre-trained weight matrix $W_0 \in \mathbb{R}^{d\times k}$, LoRA introduces two trainable weight matrices, $W_{\text{up}} \in \mathbb{R}^{d\times r}$ and $W_{\text{down}} \in \mathbb{R}^{r\times k}$, where the rank r is much smaller than the minimum of d and k. Normally, given input h_{in}, the output is computed as $h_{\text{out}} = W_0 \cdot h_{\text{in}}$. LoRA modifies this output by introducing an incremental update ΔW that encapsulates task-specific knowledge:

$$h_{\text{out}} = W_0 \cdot h_{\text{in}} + \alpha \cdot \Delta W \cdot h_{\text{in}} = W_0 \cdot h_{\text{in}} + \alpha \cdot W_{\text{up}} \cdot W_{\text{down}} \cdot h_{\text{in}},$$

where α is a scaling factor [30].

Training Configuration and Implementation Details. To fine-tune the large language model, we carefully select hyperparameters to optimize performance and accuracy. The training is conducted with a batch size of 2 per device and gradient accumulation steps set to 4, ensuring sufficient exposure to data and effective gradient updates. We include a warmup phase of 2 steps to stabilize the initial training and limit the

total training steps to 200 to prevent overfitting. The learning rate is set at 2×10^{-4} to ensure a balanced update of the model weights, promoting convergence without overshooting the optimal parameter values.

For reproducibility, we provide complete implementation details:

- *Models and tokenizers:* We consider `google/gemma-7b` [38], `mistralai/Mistral-7B-v0.3` [39], `tiiuae/falcon-7b` [40], `meta-llama/Llama-2-7b-hf` [41], `meta-llama/Llama-3-8b` [42], and small efficient baselines `meta-llama/Llama-3.2-1B-Instruct` and `meta-llama/Llama-3.2-3B-Instruct` [42]. Unless stated otherwise, we use base (non-instruct) variants and the corresponding official tokenizers (snapshots from October 2024 for 7B/8B models; August 2025 for 1B/3B).
- *Fine-tuning setup:* Parameter-efficient fine-tuning via PEFT/LoRA [30, 37] (`peft==0.8.2`) with rank $r = 8$ and scaling $\alpha = 8$, dropout at default; target modules include `q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj` for attention and MLP projections. Optimizer: AdamW [43] (Transformers default betas/eps; weight decay default); learning rate 2×10^{-4} with linear decay and warmup $=2$ steps; per-device batch size 2 with gradient accumulation 4 (effective 8); total training steps 200 (single epoch given dataset size).
- *Inference and parsing:* Greedy decoding (temperature 0.0, `max_new_tokens=15`). Outputs are post-processed with a regex that matches one *Declare* template name followed by activities in parentheses; only the first valid match is retained.
- *Hardware:* NVIDIA H100 96GB VRAM (single GPU) on a compute cluster; CPU/RAM standard server-class (not critical for reproduction).
- *Artifacts:* Code and dataset: `https://anonymous.4open.science/r/dss_rev/`.

Hardware and runtime. All fine-tuning experiments were executed on a single NVIDIA H100 (96 GB VRAM). With the LoRA configuration above, per-fold fine-tuning of a 7B model for 200 steps required approximately 10–20 minutes; complete 10-fold cross-validation finished within 2–4 hours per model. Typical inference

throughput was ∼2–5 ms per sentence on GPU (batch size 1) and ∼10–30 ms per sentence on a modern CPU, excluding I/O.

5. Activity Matching

This section describes the second step in extracting a coherent Declare process model (cf. Figure 4). We first introduce the problem that this step aims to tackle and then present several solutions to the problem.

5.1. Problem Definition

So far, in the process of extracting a process model from text, we have isolated Declare constraints, each extracted separately from a single sentence. To create a coherent process model, it is crucial that activities with the same meaning are labeled consistently across all constraints. In other words, we need to ensure that semantically equivalent activities share identical labels across all constraints.

To formalize the requirement of ensuring consistent labeling, we describe a dedicated function. Such a function takes as input a set of Declare constraints K and outputs Declare process model $DS=(Act,K')$ with a modified set of constraints K' where semantically equivalent activities have identical labels, and Act is the set of activity equivalence classes in K.

Formally, K' is obtained by modifying K such that:

$$K'=\{(k(\vec{x}),\kappa')|(k(\vec{x}),\kappa)\in K,\kappa'=\text{UpdateMapping}(\kappa,Act)\}$$

The function UpdateMapping(κ,Act) performs activity matching by replacing each activity label in constraint mapping κ with its canonical representative from the corresponding equivalence class in Act. Given a constraint with activities mapped by κ, the function identifies which equivalence class each activity belongs to and substitutes it with the chosen canonical label for that class, ensuring consistent labeling across all constraints while preserving the original constraint semantics.

Consider the following set of constraints with slightly different surface forms for the same activities:

$$K=\{Response(UserLogin,\ EnterMenu), Precedence(UploadFile,\ DeleteFile),\\ Response(FileDel,\ ProcessFile), ChainResponse(ProcessingFile,\ ApplyChange)\}$$

The activity matching process identifies semantically equivalent activities and groups them into equivalence classes:

$$Act=\{[UserLogin], [EnterMenu], [UploadFile],\\ [DeleteFile \equiv FileDel], [ProcessFile \equiv ProcessingFile], [ApplyChange]\}$$

where each class contains semantically equivalent activity labels, with the canonical representative listed first. The UpdateMapping function replaces each activity label with its canonical representative, yielding:

$$K'=\{Response(UserLogin,\ EnterMenu), Precedence(UploadFile,\ DeleteFile),\\ Response(DeleteFile,\ ProcessFile), ChainResponse(ProcessFile,\ ApplyChange)\}$$

5.2. Solutions

We provide two solutions to the presented problem.

Zero-Shot Prompting. First, a zero-shot prompting approach prompts LLMs by providing only instructions for the desired output without any examples of how the task has been performed correctly [35]. Full prompt templates are provided in Appendix A. Note that variations exist in how activities are expressed, ranging from verb-based (e.g., *DeleteFile*) to noun-based (e.g., *FileDel*) and gerund-based (e.g., *ProcessingFile*) formulations.

Few-Shot Prompting. Another approach is to provide examples of input with correct output. Such a prompt, where the input includes a few examples to guide the model in performing a specific task, is called a *few-shot* prompt [35]. Full few-shot prompt is in Appendix B.

We include the full list of constraints in the prompt because matching decisions often rely on cross-constraint evidence. Providing complete constraint context reduces ambiguity and promotes consistent canonical labels compared to processing constraints in isolation. Activities alone lose critical relational and directional signals needed for correct canonicalization. For example, from activities-only *{SubmitForm, ReviewRequest, ApproveRequest, ArchiveCase}*, a matcher may erroneously merge *ReviewRequest* and *ApproveRequest* into a single *Request* label. However, the full constraints *{Response(SubmitForm, ReviewRequest), Precedence(ApproveRequest, ArchiveCase)}* reveal distinct roles: *ReviewRequest* follows submission while *ApproveRequest* must precede archiving, indicating they should remain separate. Similarly, constraint context enables proper cross-constraint consolidation, identifying that *FileDel* and *DeleteFile* represent the same concept, as do *ProcessingFile* and *ProcessFile*, allowing unification to canonical forms.

6. Evaluation

This section aims to validate the efficiency of our proposed method both internally and externally. The experimental results reported in the evaluation are fully reproducible at `https://anonymous.4open.science/r/dss_rev/`.

6.1. Benchmark Dataset

To support the development and evaluation of our approach, we prepared a benchmark dataset comprising 969 text descriptions paired with corresponding gold-standard Declare constraints across a variety of domains, including healthcare, corporate compliance, education, and finance[1]. The corpus mixes curated domain sentences and synthetic paraphrases to ensure lexical and structural diversity. Sampling targeted balanced coverage over the 11 templates, with additional paraphrase variants for common templates (e.g., *Response*, *Precedence*). Figure 6 illustrates the distribution of 11 Declare tempaltes, with *Response* and *Precedence* being the most prevalent due to their central role in capturing process dependencies and order relations.

[1]The dataset is publicly available at `https://anonymous.4open.science/r/dss_rev/data`

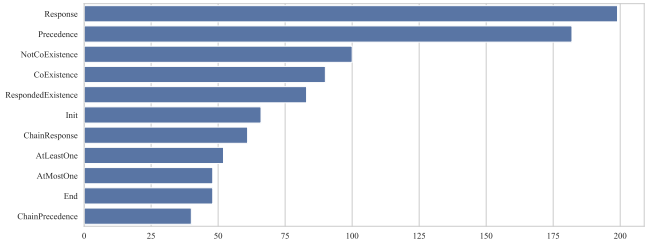


Figure 6: Distribution of Declare constraints in the benchmark dataset

Annotation guidelines and decision rules. Annotators were instructed to (i) extract verb- or noun-based activity mentions as concise Pascal-style labels (e.g., *UploadFile*, *FileDel*); (ii) map each text to the best-fitting *Declare* template considering order constraints (e.g., if B must follow A, prefer *Response(A,B)* over *RespondedExistence(A,B)* when ordering is explicit); and (iii) avoid conflating disjunctions or conjunctions into single activities unless explicitly intended. Canonicalization rules (e.g., verb-first preference) were applied consistently to reduce label drift.

Ambiguity and multiple-valid-templates policy. For under-specified sentences that admit multiple valid templates, we adopted a balanced annotation strategy. When a sentence could reasonably be interpreted with multiple valid templates, annotators were instructed to select the least represented template in the dataset to promote class balance. For example, `"If a file is uploaded, it must be processed."` could permit both *RespondedExistence(UploadFile, ProcessFile)* and *Response(UploadFile, ProcessFile)* depending on whether explicit ordering is assumed. In such cases, the less frequent template was chosen to mitigate dataset imbalance.

Adjudication and agreement. Each instance was labeled by one annotator and screened by a second.

6.2. Evaluation Metrics

In assessing the accuracy of the constraints generated by the fine-tuned LLM, it is essential to identify a suitable metric. For each sentence in the respective test

dataset, we have both the gold standard constraint and the discovered constraint.

There are three types of errors:

- *Template error*: the discovered template is incorrect
- *Activity error*: at least one of the extracted activities is incorrect
- *Direction error*: the extracted activities are correct, but the order is incorrect

The existing work evaluates the approach by framing the problem as a slot-filling problem [2]. They assume that each constraint can be represented as a set of slots. For example, a binary constraint like *Precedence(X, Y)* has three slots: one for the template and one for each activity involved. Although this method provides a straightforward way to compute precision and recall, it conflates different types of errors.

When the gold standard constraint is *Precedence(X, Y)* and the predicted constraint is *Precedence(Y,X)*, the slot-filling approach assigns partial credit based on overlapping slots. However, this error is not a simple activity mismatch; thus, it is a direction error that fundamentally reverses the meaning of the constraint. Similarly, when comparing *Precedence(X, Y)* with *Response(X, Y)*, the activities are identical but the templates differ, yet the slot-filling evaluation still grants a relatively high score, failing to capture the semantic gap between the two constraints. Therefore, it lacks the ability to separately capture template accuracy, activity correctness, and direction sensitivity.

To address these limitations and provide more precise evaluation, we introduce direction-sensitive metrics that properly distinguish template and activity accuracy.

Let the gold constraint be $g = k(a_1,...,a_m)$ and the prediction be $p = \tilde{k}(b_1,...,b_n)$. We evaluate template, activities, and direction with simple, direction-sensitive scores without weights:

$$\text{TemplateAcc}_i = \mathbb{1}[\tilde{k} = k],$$

$$\text{ActAcc}_i = \begin{cases} \frac{1}{m}\sum_{j=1}^{m} \mathbb{1}[a_j = b_j], & \text{if } \tilde{k} = k \text{ and } n = m, \\ 0, & \text{otherwise,} \end{cases}$$

$$\text{ConstraintScore}_i = \text{TemplateAcc}_i \cdot \text{ActAcc}_i.$$

Thus, a wrong template yields zero score; a correct template yields partial credit for correctly placed activities (direction-sensitive by construction). For symmetric

templates (e.g., *CoExistence*, *NotCoExistence*), we evaluate ActAcc_i as the maximum over argument permutations that preserve the template's semantics.

Over a dataset of N instances, we report macro-averaged metrics. Template accuracy is the fraction of instances with correct template prediction. Activity accuracy is computed only for instances with correct templates, measuring the fraction of correctly positioned activities. Constraint score is the overall fraction of instances with both correct template and correct activities.

6.3. Internal Validation

We internally validate which LLM architecture performs best for sentence-to-constraint translation through systematic comparison across seven open-source, decoder-only LLMs. We run 10-fold cross-validation on our benchmark dataset to identify the optimal model for external validation and analyze performance patterns across different model families and sizes.

6.3.1. Experimental Setup

A 10-fold cross-validation was utilized to evaluate our approach. The benchmark dataset was randomly partitioned into ten equally sized, non-overlapping subsets. Each subset was used once as a test set, while the LLM was fine-tuned using the other nine subsets as training data. For each model, the training parameters were identical, as described in Subsection 4.2, ensuring fair comparison across architectures.

6.3.2. Experimental Results

Table 2 reports macro-averaged TemplateAcc, conditional ActAcc, and the overall ConstraintScore as defined in Subsection 6.2.

Gemma 7B delivers the strongest performance across all three metrics, achieving 97.8% template accuracy, 78.2% conditional activity accuracy, and 76.4% overall constraint score. This establishes it as our primary model for subsequent external validations, with a substantial 9.5 percentage point advantage in template accuracy over the next-best performing Llama-3 8B. Clear generational improvements emerge within model families, most prominently between Llama variants. Llama-3 8B significantly outperforms Llama-2 7B (88.3% vs. 66.8% template accuracy).

Table 2: Cross-validation performance of different fine-tuned decoder-only LLMs.

Model	TemplateAcc	ActAcc (cond.)	ConstraintScore
Gemma 7B (ft)	**0.978**	**0.782**	**0.764**
Llama-3 8B (ft)	0.883	0.752	0.664
Falcon 7B (ft)	0.880	0.755	0.664
Mistral 7B (ft)	0.740	0.780	0.577
Llama-2 7B (ft)	0.668	0.733	0.491
Llama-3.2 3B-Instruct (ft)	0.920	0.701	0.644
Llama-3.2 1B-Instruct (ft)	0.885	0.664	0.588

An interesting pattern emerges when comparing template classification versus activity extraction performance across all models. Template accuracy consistently falls below conditional activity accuracy, indicating that LLMs are generally proficient at identifying relevant activities within text but struggle more with distinguishing between semantically similar constraint types, such as Response versus RespondedExistence. Mistral 7B exhibits a particularly unusual pattern with relatively low template accuracy (74.0%) but high conditional activity accuracy (78.0%).

The instruction-tuned lightweight models demonstrate competitive performance, particularly in template classification tasks. Llama-3.2 1B-Instruct achieves 88.5% template accuracy, which exceeds several 7B-8B base models. The 3B variant performs even better with 92.0% template accuracy, nearly matching the performance of much larger models while requiring significantly fewer computational resources.

6.4. External Validation - Part A

In this section, we conduct external validation by comparing our approach to the state-of-the-art method by Van der Aa et al. [2]. We use the best-performing model from the internal validation (Gemma 7B fine-tuned with LoRA as specified in Subsection 4.2). The comparison focuses on the four templates supported by both approaches: *Init*, *End*, *Precedence*, and *Response*. We evaluate performance on two distinct validation sets to assess generalization and reliability improvements over a traditional rule-based NLP pipeline.

6.4.1. Experimental Setup

We limit the evaluation to *Init*, *End*, *Precedence*, and *Response*, which are the templates supported by the baseline. We exclude *Succession*, as it is a conjunction of *Response* and *Precedence*.

For fine-tuning the best-performing model (Gemma 7B), we constructed a training set of 850 sentence–constraint pairs by starting from our benchmark dataset and removing all sentences that occur in the external validation data of Van der Aa et al. [2] (including minor paraphrases), thereby avoiding any overlap or leakage. No sentence from $V1$ or $V2$ is included in training.

We report only template accuracy (TemplateAcc) in this comparison. Activity-level metrics are not directly comparable due to differing activity vocabularies and canonicalization policies between approaches.

To assess performance, we use a human-checked validation set ($V1$) consisting of 80 inputs covering the four templates, disjoint from the training data. In addition, we validate both approaches on a subset of the validation set used by Van der Aa et al. [2], denoted $V2$. From the original 103 sentences, we exclude those with *Succession* or negative templates as gold standards, and those that do not describe one of the four templates (e.g., pure conditional constructs).

6.4.2. Experimental Results

As shown in Table 3, the best-performing fine-tuned model (Gemma 7B) substantially improves TemplateAcc on both $V1$ and $V2$. We report TemplateAcc only by design, since activity strings and canonicalization differ across systems and would confound activity-level comparisons. In our validation sets, gold labels use concise, verb-based Pascal-case activities; under this metric choice, the fine-tuned model attains nearly perfect template accuracy on $V1$ (versus 53.8% for the baseline) and improves over the baseline on $V2$ as well.

6.5. External Validation - Part B

Next, we compare our approach with prompting baselines. Prior work [21] shows that strong prompting can outperform traditional rule-based systems [2]. Here we

evaluate whether a small, fine-tuned model (Gemma 7B with LoRA) provides more reliable performance across our 11 templates.

6.5.1. Baselines

Few-shot prompting. One-shot per template (11 shots in total), each mirroring the abstract template description in Table 1. The prompt asks to output the constraint only (no prose). A sketch of the prompt is given in Appendix C.

Few-shot Chain-of-Thought (CoT) prompting. One-shot per template with short rationales preceding the answer, following Wei et al. [34], Kojima et al. [35]. We use the instruction-tuned Gemma 7B-it for CoT and standard Gemma 7B for non-CoT. An excerpt is shown in Appendix D.

Prompt formatting, token budgeting, and decoding.

- *Formatting.* Prompts are rendered in monospace blocks; CoT uses conversation control tokens required by Gemma 7B-it. When using other models, adapt or remove control tokens to match the target chat template. We instruct the model to output only a single constraint.

Table 3: Comparison with the state-of-the-art Van der Aa et al. [2]. Entries in the two rightmost columns specify TemplateAcc.

		Cases	State-of-the-art [2]	Our approach (Gemma 7B)
V1	Init	11	0.273	0.917
	End	4	0.0	1.0
	Response	38	0.684	1.0
	Precedence	27	0.519	1.0
	Overall	**80**	**0.538**	**0.988**
V2	Init	4	0.75	1.0
	End	1	0.0	1.0
	Response	14	0.643	0.929
	Precedence	27	0.963	0.962
	Overall	**45**	**0.822**	**0.956**

- *Token budgeting.* To stay within context limits we (i) keep shots concise, (ii) prefer schema-like definitions over verbose prose, (iii) reuse short, canonical activity names, and (iv) trim boilerplate/whitespace. If close to the window, reduce the number of shots first, then shorten rationales. We verify that system/instruction + shots + input fit within the model's maximum tokens.

- *Decoding parameters.* Decoding is greedy (temperature 0.0; no sampling; top-p=1.0; top-k disabled) with `max_new_tokens=15`. For CoT, we add stop sequences for end-of-turn tokens. Outputs are regex-validated to extract exactly one *Declare* constraint (template and activities) in canonical form.

- *Common failure cases.* Over-length prompts (truncated rationales), malformed outputs (multiple constraints/extra prose), and ambiguity-induced activity order swaps. Mitigations include stricter formatting, shorter shots, explicit stop sequences, and regex validation.

- *Common failure cases.* Over-length prompts (truncated rationales), malformed outputs (multiple constraints/extra prose), and ambiguity-induced activity order swaps. Mitigations include stricter formatting, shorter shots, explicit stop sequences, and regex validation.

6.5.2. Data Collection

We utilize the same dataset as for the internal validation. From this dataset, we allocate 80% as training data, while the remaining 20% serves as a test set for both the prompting approaches and the fine-tuned LLM.

6.5.3. Experimental Setup

We use Gemma-7b for both fine-tuning and the *few-shot* prompt. For the *few-shot CoT* approach, we utilize the instruct version of Gemma-7b. The prompts are designed to consider the same set of Declare templates on which we trained the LLM: *Init*, *End*, *AtMostOne*, *AtLeastOne*, *Response*, *Precedence*, *ChainResponse*, *ChainPrecedence*, *RespondedExistence*, *CoExistence*, and *NotCoExistence*.

6.5.4. Experimental Results

As shown in Table 4, fine-tuning yields the highest TemplateAcc (0.994) and the best overall *ConstraintScore* (0.809). CoT improves TemplateAcc over plain few-shot (0.565 vs. 0.459) and markedly boosts conditional *ActAcc* (0.734), indicating better activity placement when the template is correct. However, the fine-tuned model balances both template recognition and activity placement, leading to the strongest end-to-end constraint quality.

7. Discussion

7.1. Findings

Model Architecture and Performance Patterns. Our internal validation reveals clear performance hierarchies among LLM architectures for constraint extraction. Gemma 7B consistently delivers superior performance across all metrics (97.8% template accuracy, 78.2% conditional activity accuracy, 76.4% constraint score), establishing a 9.5 percentage point advantage over the next-best Llama-3 8B. Surprisingly, model size does not universally correlate with performance. The compact Llama-3.2 3B-Instruct (92.0% template accuracy) significantly outperforms the larger Llama-2 7B, highlighting the effectiveness of instruction tuning and knowledge distillation. The 1B-Instruct variant achieves 88.5% template accuracy, exceeding several 7B-8B base models and suggesting that instruction tuning provides substantial benefits for structured classification tasks.

Template vs. Activity Extraction Performance. An interesting pattern emerges across all models: template accuracy consistently exceeds conditional activity accuracy,

Table 4: Prompting baselines versus fine-tuned model using the direction-sensitive metrics.

Technique	TemplateAcc	ActAcc (cond.)	ConstraintScore
Few-shot prompt	0.459	0.051	0.024
Few-shot CoT prompt	0.565	0.734	0.415
Our approach	**0.994**	**0.814**	**0.809**

indicating that LLMs excel at distinguishing constraint types but face greater challenges in precise activity extraction and positioning. This suggests that template classification and activity extraction present distinct computational challenges, with template recognition being the more tractable task.

The few instances where fine-tuned LLMs fail to identify correct templates stem primarily from linguistic ambiguities inherent in natural language. For example, `"Upon receiving a support ticket, the troubleshooting process begins"` permits multiple valid interpretations: either *Init(ReceiveSupportTicket)* focusing on the initiation aspect, or *Response(ReceiveSupportTicket, BeginTroubleshooting)* emphasizing the causal relationship. The desired activity abstraction level significantly influences template selection in such cases.

Comparison with State-of-the-Art. External validation demonstrates substantial improvements over existing rule-based approaches. On validation set $V1$, our Gemma 7B model achieves 98.8% template accuracy compared to 53.8% for Van der Aa et al. [2]. The largest performance gaps occur in unary constraints (*Init*, *End*), where rule-based methods struggle due to their reliance on detecting specific meta-actions and keywords like `"process"`, `"start"`, and `"end"`. Our LLM-based approach overcomes these limitations through robust natural language understanding.

For common binary templates (*Response*, *Precedence*), the fine-tuned model demonstrates significant improvements, effectively learning to distinguish between semantically similar but structurally different constraint types. On validation set $V2$, our approach maintains competitive performance across most templates, with the exception of *Precedence* constraints where the baseline achieves artificially high accuracy (96.3%) due to default constraint generation rather than sophisticated NLP processing.

Fine-tuning vs. Prompting Strategies. Comparison with prompting baselines confirms that parameter adaptation through fine-tuning is essential for reliable constraint extraction. While chain-of-thought prompting improves over basic few-shot approaches (56.5% vs. 45.9% template accuracy), both fall substantially short of fine-tuned performance (99.4% template accuracy). This indicates that prompting techniques,

despite their sophistication, cannot substitute for task-specific weight adaptation when precise structural understanding is required.

Advanced prompting provides interpretable reasoning steps and better activity placement when templates are correctly identified (73.4% conditional activity accuracy for CoT), but struggles with consistent template classification. The fine-tuned model achieves superior balance across both template recognition and activity extraction, leading to the highest end-to-end constraint quality.

7.2. Practical Implications for Decision Support Systems

Discovered *Declare* constraints provide machine-checkable policy abstractions that can be operationalized in decision support:

- *Compliance and governance:* Constraints act as executable rules derived from textual policies and can be monitored continuously on event data to detect violations and support governance dashboards.
- *Policy automation:* Violations or risk conditions can trigger proactive alerts and workflow actions, supporting policy enforcement and case-handling automation.
- *Audit readiness and explainability:* Findings can be linked to concrete trace evidence (case IDs, timestamps, activity pairs), yielding auditable, explainable outcomes grounded in the original text-derived rules.

7.3. Academic Implications

Our contributions also have implications for research methodology and future studies in IS, BPM, and NLP:

- *Benchmark and comparability:* The curated benchmark (969 sentence–constraint pairs across 11 templates) and simplified, direction-sensitive metrics (TemplateAcc, conditional ActAcc, ConstraintScore) enable transparent, apples-to-apples comparisons and reduce evaluation ambiguity.
- *Reproducibility and extensibility:* Released scripts, prompts, and configurations, together with hardware/runtime notes, facilitate faithful replication and lower the barrier for follow-up work, including re-running with new base models or decoding settings.

- *Cross-paradigm bridges:* The dataset and protocol can support studies mapping between imperative and declarative representations, or repurposing text-to-imperative resources to synthesize text-to-declarative pairs.
- *Model-size insights:* Including 1B/3B instruction-tuned baselines highlights viable low-resource settings and opens avenues on efficiency–accuracy trade-offs, PEFT choices, and deployment constraints.

7.4. Limitations and Threats to Validity

Scope and expressiveness. Our component assumes a one-sentence-to-one-constraint mapping; multi-constraint sentences and cross-sentence constraints require additional logic (e.g., splitting/aggregation). We currently do not cover advanced *Declare* constructs such as branched responses or time conditions, which limits expressiveness in complex scenarios. Generalization to highly domain-specific phrasing remains challenging where data are scarce.

LLM-related threats. As highlighted by recent work on contamination and evaluation malpractices [44], LLM-based studies risk inadvertent data leakage (train/test overlap) and protocol biases. We mitigate this by (i) careful dataset curation and cross-validation splits, (ii) disjoint external validations, (iii) transparent reporting of models, prompts, and configs, and (iv) releasing code and data for reproducibility. Prompt overfitting is reduced by fine-tuning and by keeping decoding simple (greedy) with regex post-processing.

Model-size scope. Our primary comparisons focus on 7B–8B models; we additionally include small, efficient instruction-tuned baselines (1B/3B) where feasible. Comprehensive coverage of all size/variant combinations is beyond scope; we acknowledge this limitation and report model details and hardware requirements to inform practical deployments.

7.5. Future Work

To address these limitations, future work will focus on several key areas. One priority is to train the LLMs for more advanced Declare constructs, such as branched

responses. This will require generating adequate training data that accurately represents these complex constructs. Additionally, developing tool support for the entire pipeline, including activity matching, will enhance the practical utility of our approach. Another important extension is to support sentences that map to multiple constraints or by managing multiple input sentences with potential inter-sentence dependencies that map to one or several output constraints. To accomplish this, post-processing steps can be helpful, e.g., splitting up activities, especially if they contain conjunctions. A more promising approach is to explicitly train the model on such scenarios. For instance, one can explore advanced techniques such as *curriculum learning* where the idea is to start training the model on single sentences, continue with pairs of sentences, and finally learn this task for whole paragraphs. Another idea is multi-task learning (MTL), which addresses related tasks like activity extraction, constraint extraction, sentence classification, and activity matching simultaneously. By leveraging shared representations learned from these interconnected tasks, MTL can improve model generalization and accuracy. Additionally, integrating a resource perspective can provide further insights. For instance, another task might involve mapping each activity to the actor performing it. Finally, providing target event labels as context can further improve the model's performance by offering additional information that helps identify the desired abstraction level of activities. These enhancements will make our approach more robust, versatile, and applicable to a wider range of scenarios. In addition, testing the approach on real-world datasets from various industries can provide a more comprehensive evaluation of its effectiveness. To address the limitation of training data, implementing automated data augmentation techniques can generate a more extensive and diverse dataset. This helps improve the model's robustness and generalization capabilities.

8. Conclusion

This work provides a systematic empirical study of LLM-based sentence-to-Declare constraint extraction, establishing foundational infrastructure for automated constraint extraction from natural language. Through comprehensive evaluation

across multiple model architectures and training approaches, we demonstrate that fine-tuned LLMs significantly advance the state-of-the-art, achieving 97.8% template accuracy compared to 53.8% for existing rule-based approaches. Our primary contributions include a benchmark dataset of 969 sentence-constraint pairs across 11 Declare templates, direction-sensitive evaluation metrics that properly assess ordered constraint types, and complete reproducibility materials. The substantial performance advantages of fine-tuning over prompting approaches (99.4% vs. 56.5% template accuracy) provide clear guidance for practitioners, while competitive performance of lightweight instruction-tuned models demonstrates viable deployment across different computational constraints. Beyond technical advances, this work demonstrates clear pathways to practical decision support applications, where extracted constraints serve as machine-checkable policy abstractions enabling continuous compliance monitoring, automated conformance checking, and audit-ready governance dashboards.

References

[1] M. Camargo, M. Dumas, O. González, Automated discovery of business process simulation models from event logs, Decis. Support Syst. 134 (2020) 113284.

[2] H. van der Aa, C. Di Ciccio, H. Leopold, H. A. Reijers, Extracting declarative process models from natural language, in: CAiSE, Springer, 2019, pp. 365–382.

[3] O. Ettrich, S. Stahlmann, H. Leopold, C. Barrot, Automatically identifying customer needs in user-generated content using token classification, Decis. Support Syst. 178 (2024) 114107.

[4] H. A. López, R. Strømsted, J.-M. Niyodusenga, M. Marquard, Declarative process discovery: Linking process and textual views, in: CAiSE, Springer, 2021, pp. 109–117.

[5] J. C. de A. R. Gonçalves, F. M. Santoro, F. A. Baião, A case study on designing business processes based on collaborative and mining approaches, in: W. Shen, N. Gu, T. Lu, J. A. Barthès, J. Luo (Eds.), CSCWD, IEEE, 2010, pp. 611–616.

[6] F. Friedrich, J. Mendling, F. Puhlmann, Process model generation from natural language text, in: CAiSE, Springer, 2011, pp. 482–496.

[7] E. V. Epure, P. Martín-Rodilla, C. Hug, R. Deneckère, C. Salinesi, Automatic process model discovery from textual methodologies, in: RCIS, IEEE, 2015, pp. 19–30.

[8] K. Honkisz, K. Kluza, P. Wiśniewski, A concept for generating business process models from natural language description, in: KSEM, Springer, 2018, pp. 91–103.

[9] S. Sholiq, R. Sarno, E. S. Astuti, Generating BPMN diagram from textual requirements, J. King Saud Univ. Comput. Inf. Sci. 34 (2022) 10079–10093.

[10] S. Levich, B. Lutz, D. Neumann, Utilizing the omnipresent: Incorporating digital documents into predictive process monitoring using deep neural networks, Decis. Support Syst. 175 (2023) 114043.

[11] P. Bellan, M. Dragoni, C. Ghidini, H. van der Aa, S. P. Ponzetto, Process extraction from text: Benchmarking the state of the art and paving the way for future challenges, 2023. arXiv:2110.03754.

[12] W. V. Woensel, S. Motie, NLP4PBM: a systematic review on process extraction using natural language processing with rule-based, machine and deep learning methods, Enterp. Inf. Syst. 18 (2024).

[13] H. van der Aa, H. Leopold, M. Weidlich, Partial order resolution of event logs for process conformance checking, Decis. Support Syst. 136 (2020) 113347.

[14] L. Ackermann, J. Neuberger, S. Jablonski, Data-driven annotation of textual process descriptions based on formal meaning representations, in: CAiSE, Springer, 2021, pp. 75–90.

[15] L. Quishpi, J. Carmona, L. Padró, Extracting annotations from textual descriptions of processes, in: BPM, Springer, 2020, pp. 184–201.

[16] R. C. B. Ferreira, L. H. Thom, M. Fantinato, A semi-automatic approach to identify business process elements in natural language texts, in: S. Hammoudi, M. Smialek, O. Camp, J. Filipe (Eds.), ICEIS, SciTePress, 2017, pp. 250–261.

[17] C. Qian, L. Wen, A. Kumar, L. Lin, L. Lin, Z. Zong, S. Li, J. Wang, An approach for process model extraction by multi-grained text classification, in: CAiSE, Springer, 2020, pp. 268–282.

[18] J. Sànchez-Ferreres, H. van der Aa, J. Carmona, L. Padró, Aligning textual and model-based process descriptions, Data & Knowledge Engineering 118 (2018) 25–40.

[19] H. van der Aa, H. Leopold, H. A. Reijers, Comparing textual descriptions to process models - the automatic detection of inconsistencies, Inf. Syst. 64 (2017) 447–460.

[20] H. Van der Aa, H. Leopold, H. A. Reijers, Checking process compliance against natural language specifications using behavioral spaces, Inf. Syst. 78 (2018) 83–95.

[21] M. Grohs, L. Abb, N. Elsayed, J.-R. Rehse, Large language models can accomplish business process management tasks, in: BPM, Springer, 2023, pp. 453–465.

[22] M. Barrientos, K. Winter, J. Mangler, S. Rinderle-Ma, Verification of quantitative temporal compliance requirements in process descriptions over event logs, in: M. Indulska, I. Reinhartz-Berger, C. Cetina, O. Pastor (Eds.), CAiSE, volume 13901, Springer, 2023, pp. 417–433.

[23] H. Mustroph, M. Barrientos, K. Winter, S. Rinderle-Ma, Verifying resource compliance requirements from natural language text over event logs, in: C. D. Francescomarino, A. Burattin, C. Janiesch, S. Sadiq (Eds.), BPM, volume 14159, Springer, 2023, pp. 249–265.

[24] A. Rebmann, F. D. Schmidt, G. Glavas, H. van der Aa, Evaluating the ability of llms to solve semantics-aware process mining tasks, in: ICPM, IEEE, 2024, pp. 9–16.

[25] K. Busch, T. Kampik, H. Leopold, xsemad: Explainable semantic anomaly detection in event logs using sequence-to-sequence models, in: A. Marrella, M. Resinas, M. Jans, M. Rosemann (Eds.), BPM, volume 14940, Springer, 2024, pp. 309–327.

[26] J. Köpke, A. Safan, Efficient llm-based conversational process modeling, in: K. Gdowska, M. T. Gómez-López, J. Rehse (Eds.), BPM, volume 534, Springer, 2024, pp. 259–270.

[27] N. Klievtsova, J. Benzin, T. Kampik, J. Mangler, S. Rinderle-Ma, Conversational process modelling: State of the art, applications, and implications in practice, in: C. D. Francescomarino, A. Burattin, C. Janiesch, S. Sadiq (Eds.), BPM Forum, volume 490, Springer, 2023, pp. 319–336.

[28] M. Weidlich, J. M. E. M. van der Werf, On profiles and footprints - relational semantics for petri nets, in: S. Haddad, L. Pomello (Eds.), PETRI NETS, volume 7347, Springer, 2012, pp. 148–167.

[29] L. Barbaro, G. Varricchione, M. Montali, C. D. Ciccio, From sound workflow nets to ltlf declarative specifications by casting three spells, CoRR abs/2504.05114 (2025). arXiv:2504.05114.

[30] Z. Han, C. Gao, J. Liu, J. Zhang, S. Q. Zhang, Parameter-efficient fine-tuning for large models: A comprehensive survey, CoRR abs/2403.14608 (2024). arXiv:2403.14608.

[31] C. Di Ciccio, M. Montali, Declarative process specifications: reasoning, discovery, monitoring, in: Process mining handbook, Springer International Publishing Cham, 2022, pp. 108–152.

[32] M. B. Dwyer, G. S. Avrunin, J. C. Corbett, Patterns in property specifications for finite-state verification, in: B. W. Boehm, D. Garlan, J. Kramer (Eds.), ICSE, ACM, 1999, pp. 411–420.

[33] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: I. Guyon, et al. (Eds.), NeurIPS, 2017, pp. 5998–6008.

[34] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, D. Zhou, Chain-of-thought prompting elicits reasoning in large language models, in: S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, A. Oh (Eds.), NeurIPS, 2022.

[35] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, Y. Iwasawa, Large language models are zero-shot reasoners, in: S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, A. Oh (Eds.), NeurIPS, 2022.

[36] X. Wang, J. Wei, D. Schuurmans, Q. V. Le, E. H. Chi, S. Narang, A. Chowdhery, D. Zhou, Self-consistency improves chain of thought reasoning in language models, in: ICLR, OpenReview.net, 2023.

[37] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, Lora: Low-rank adaptation of large language models, in: ICLR, 2022.

[38] G. Team, T. Mesnard, C. Hardin, R. Dadashi, S. Bhupatiraju, S. Pathak, L. Sifre, M. Rivière, M. S. Kale, J. Love, et al., Gemma: Open models based on gemini research and technology, arXiv preprint arXiv:2403.08295 (2024).

[39] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, W. E. Sayed, Mistral 7b, 2023. arXiv:2310.06825.

[40] E. Almazrouei, H. Alobeidli, A. Alshamsi, A. Cappelli, R. Cojocaru, M. Debbah, Étienne Goffinet, D. Hesslow, J. Launay, Q. Malartic, D. Mazzotta, B. Noune, B. Pannier, G. Penedo, The falcon series of open language models, 2023. arXiv:2311.16867.

[41] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al., Llama 2: Open foundation and fine-tuned chat models, arXiv preprint arXiv:2307.09288 (2023).

[42] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, et al., The llama 3 herd of models, arXiv e-prints (2024) arXiv–2407.

[43] I. Loshchilov, F. Hutter, Decoupled weight decay regularization, in: ICLR, OpenReview.net, 2019.

[44] S. Balloccu, P. Schmidtová, M. Lango, O. Dusek, Leak, cheat, repeat: Data contamination and evaluation malpractices in closed-source llms, in: Y. Graham, M. Purver (Eds.), EACL, Association for Computational Linguistics, 2024, pp. 67–93.

Appendix A. Activity Matching Zero-Shot Prompting

> Take this list of Declare constraints containing various activities. Init(LogIn), AtMostOne(ResetPassword), AtLeastOne(CreateProfile), End(LogOut), Response(UserLogin, EnterMenu), Response(FileDel, ProcessFile), Precedence(UploadFile, DeleteFile), ChainResponse(ProcessingFile, ApplyChange). Match semantically equivalent activities and provide an updated list of the constraints after agreeing on one unique label per activity equivalence class.

Appendix B. Activity Matching Few-Shot Prompting (excerpt)

> Take this list of Declare constraints containing various activities. [...] Match semantically equivalent activities and provide an updated list of the constraints after agreeing on one unique label per activity equivalence class. Take the following examples for which the task has been solved correctly: Input: [Response(CreateClaim, ApproveClaim), Response(ClaimCreation, SendInvoice), Init(CreatingClaim)], Output: [Response(CreateClaim, ApproveClaim), Response(CreateClaim, SendInvoice), Init(CreateClaim)] [...]

Appendix C. Baseline 1: Few Shot Prompting

Below is the prompt for the few-shot-prompting-based baseline used in the evaluation (cf. Subsubsection 6.5.1:

> I am a highly intelligent AI capable of understanding and translating natural language descriptions into Declare constraints. Here are some example translations of descriptions into Declare constraints:
> 1. Description: Breakfast must be the first to occur.
> Task(s): Breakfast

Declare Constraint: Init(Breakfast)
more shots*
11. Description: Cooking and Biking never occur together.
Task(s): Cooking, Biking
Declare Constraint: NotCoExistence(Cooking, Biking)
Given the following description, we should first detect tasks and then translate the description into a Declare constraint. The output format should exactly follow this pattern as in the example descriptions. The answer should solely contain the constraint and nothing else, i.e., no additional description. Now, let's translate the following description into a Declare constraint:
12. Description: *description*

Appendix D. Baseline 2: Few Shot Chain-of-Thought (CoT) Prompting

Below is the excerpt of the prompt for the few-shot-CoT-prompting-based baseline used in the evaluation (cf. Subsubsection 6.5.1. Note that we employ a specific syntax for this prompt. The instructional version of Google's Gemma 7b-it model we chose for inference utilizes control tokens to enhance the clarity and structure of conversations. This syntax includes tokens for indicating the user's turn (`user`), the model's turn (`model`), the start of a dialogue turn (`<start_of_turn>`), and the end of a dialogue turn (`<end_of_turn>`). This clear differentiation of roles and turns improves the model's performance by providing explicit context and structure.

<start_of_turn>user
You are a highly intelligent AI capable of understanding and translating natural language descriptions into Declare constraints. Here are definitions of some common Declare templates:
1. Definition: "Task A must be the first to occur."
Declare: Init(A)
more definitions*
11. Definition: "Task A and Task B never occur together."
Declare: NotCoExistence(A, B)
Think and write your step-by-step reasoning before responding.
Now, translate the following description into a Declare constraint: "They have to create the claim, before they can approve it."
<end_of_turn>
<start_of_turn>model
Let's think step by step. In this example, we find two tasks, namely, creating a claim and approving it. We, therefore, define CreateClaim and ApproveClaim as tasks. We define Task A = CreateClaim and Task B = ApproveClaim because of the temporal sequence of these tasks. According to the description, CreateClaim (Task A) is a condition for

ApproveClaim (Task B). This matches the definition of a Precedence constraint (If Task B occurs, then Task A must occur before).
Thus, the resulting constraint is: Precedence(CreateClaim, ApproveClaim)
<end_of_turn>
more shots with chain-of-thoughts*
<start_of_turn>user
Now, translate the following description into a Declare constraint: "Booking the venue should only happen once."
<end_of_turn>
<start_of_turn>model
Let's think step by step. In this example, we find one task, namely booking the venue. We, therefore, define BookVenue as the task. We define Task A = BookVenue. According to the description, BookVenue (Task A) should only happen once. This matches the definition of an AtMostOne constraint (Task A occurs at most once).
Thus, the resulting constraint is: AtMostOne(BookVenue)
<end_of_turn>
<start_of_turn>user
Now, let's translate the following description into a Declare constraint: description.
<end_of_turn>
<start_of_turn>model
Let's think step by step.

Extracting Declarative Constraints for Process Modeling From Natural Language Descriptions with Large Language Models

Gyunam Park[a,b], Julian Kofferath[c], Minsu Cho[d,*]

[a]*Eindhoven University of Technology, Eindhoven, 5612 AZ, the Netherlands*
[b]*Fraunhofer Institute for Applied Information Technology FIT, Sankt Augustin, 53757, Germany*
[c]*RWTH Aachen University, Aachen, 52074, Germany*
[d]*School of Information Convergence, Kwangwoon University, Seoul, 01897, South Korea*

Acknowledgements.
This work was supported by the Ministry of Education of the Republic of Korea and the National Research Foundation of Korea (NRF-2024S1A5A8024081) and the Research Grant of Kwangwoon University in 2024.

[*]Corresponding author
Email addresses: g.park@tue.nl (Gyunam Park), julian.kofferath@rwth-aachen.de (Julian Kofferath), mcho@kw.ac.kr (Minsu Cho)

Highlights

1. LLM-based approach for automated extraction of declarative process models.
2. Novel dataset with 969 labeled text inputs covering 11 Declare templates.
3. Enhanced generalization and accuracy over existing rule-based NLP techniques.

Biographical Note

Gyunam Park is an Assistant Professor in computer science at Eindhoven University of Technology. Prior to this, he stayed at Fraunhofer FIT as a research group leader. He completed his Ph.D. in Computer Science at RWTH Aachen University. His research interests include process mining, machine learning, and the application of large language models. Dr. Park has published extensively in high-impact journals, including *Decision Support Systems*, *Expert Systems with Applications (ESWA)*, *Computers in Industry*, and *Engineering Applications of Artificial Intelligence*. His work contributes to advancing the understanding of data-driven decision support and intelligent process automation in various industrial contexts.

Julian Kofferath received his B.Sc. degree in Computer Science from RWTH Aachen University, where he is currently pursuing his M.Sc. in Computer Science. His research interests encompass data science, process mining, and the application of large language models. Julian has been involved in a research project funded by the Federal Ministry of Education and Research in Germany.

Minsu Cho received his Ph.D. degree in management engineering at UNIST (Ulsan National Institute of Science and Technology) in 2018. He is now an associate professor with the School of Information Convergence at Kwangwoon University. His research interests include process mining, natural language processing, business analytics, and industrial artificial intelligence. He has published more than 30 scientific papers in several top-level venues such as Decision Support Systems, Expert Systems with Applications, Journal of Biomedical Informatics, International Journal of Medical Informatics, etc.

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐ The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: