

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

Gyunam Park^a, Julian Kofferath^b, Minsu Cho^{c,*}

^a*Fraunhofer Institute for Applied Information Technology FIT, Sankt Augustin, 53757, Germany*

^b*RWTH Aachen University, Aachen, 52074, Germany*

^c*School of Information Convergence, Kwangwoon University, Seoul, 01897, South Korea*

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, which largely depend on conventional Natural Language Processing (NLP) methods, face challenges such as limited generalization, difficulty handling noisy inputs, and constraints to a limited set of control-flow constructs. In response, we propose a novel approach that leverages Large Language Models (LLMs) to translate natural language text into declarative process models, specifically using the *Declare* modeling language. Our method enhances generalization, improves resilience to noisy input, and supports a broader spectrum of control-flow constructs. We fine-tune LLMs on a curated dataset, achieving substantial improvements in accuracy compared to existing approaches. Moreover, we contribute a new dataset of textual descriptions paired with corresponding *Declare* constraints to support ongoing research in this domain.

Keywords: Process Modeling, Automated Extraction, Large Language Models

*Corresponding author

Email addresses: gyunam.park@fit.fraunhofer.de (Gyunam Park), julian.kofferath@rwth-aachen.de (Julian Kofferath), mcho@kw.ac.kr (Minsu Cho)

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 methodology offers a promising approach for converting 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.

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].

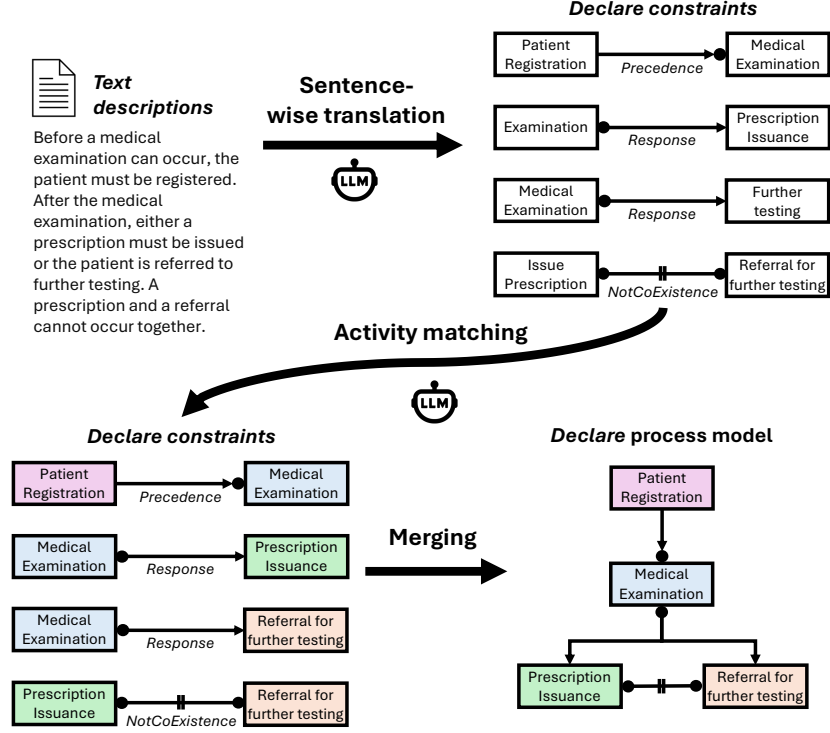


Figure 1: An illustration of the proposed method and a possible use case.

Our approach contributes to the automated process extraction in the following manners:

1. By utilizing LLMs, our approach significantly improves generalization capabilities and accuracy. Unlike rule-based methods, which are limited by strict rules, LLMs adapt better to diverse sentence structures and linguistic variations, effectively addressing the generalization challenges seen in existing approaches.
2. Unlike existing methods constrained by a limited set of control-flow constructs, our proposed approach facilitates the inclusion of various control-flow constructs based on various Declare templates.
3. To support systematic evaluation and further research, we provide a benchmark-quality dataset of textual descriptions paired with ground-truth *Declare* constraints. This collection comprises 969 sentences and covers 11 distinct *Declare*

43 templates.

44 The remainder of this paper is organized as follows: In Section 2, we introduce re-
45 lated work concerning the synthesis of process models from natural language. Section 3
46 introduces Declare and decoder-only LLMs, providing a comprehensive overview. Sec-
47 tion 4 and Section 5 describe the extraction pipeline and employed techniques. Follow-
48 ing this, Section 6 evaluates the performance of our approach and compares it with ex-
49 isting methods. In Section 7, we present a detailed discussion of our findings, including
50 limitations and opportunities for future research. Section 8, we conclude this paper.

51 2. Related Work

52 Recently, translating natural language texts to process models has gained atten-
53 tion. Bellan et al. [11] compare ten state-of-the-art approaches to extracting process
54 models from text. The study includes both qualitative and quantitative analyses,
55 examining characteristics, extracted elements, and performance evaluations. The
56 quantitative analysis identifies three performance groups, with no tool excelling in
57 all metrics. The study highlights the need for a common evaluation framework and
58 benchmarks, addressing challenges such as data variability, advanced NLP techniques,
59 and the complexity of end-to-end extraction pipelines.

60 Several traditional NLP-based approaches were proposed to discover procedural
61 process models from text. de AR Gonçalves et al. [5] introduced storytelling min-
62 ing, which generates BPMN representations from user-composed process instance
63 stories using text mining, information extraction, and NLP techniques. Friedrich
64 et al. [6] proposed extracting BPMN models from textual descriptions through NLP,
65 sentence-level analysis, text cleaning, and semantic parsing. Text Process Miner [7]
66 focuses on mining process models from archaeological reports by processing multiple
67 descriptions of process executions using NLP and rule-based techniques. Honkisz
68 et al. [8] presented a two-step method for automating BPMN model discovery from
69 text, involving syntactic and semantic analysis to extract and filter Subject-Verb-
70 Object constructs, with identified actors and gateways stored in an intermediate
71 representation for BPMN model generation. Sholiq et al. [9] proposed converting

72 textual requirements into BPMN diagrams using NLP, involving two main stages:
73 analyzing textual requirements to extract fact types and mapping these onto BPMN
74 elements to generate the diagrams.

75 Some papers have focused on extracting individual process elements, such as
76 activities, actors, or gateways, from textual sources [14, 15, 16, 17]. Additionally,
77 several approaches outside of *process discovery* have been proposed to compare or
78 align textual information with existing process models [18, 19, 20].

79 Only a few known approaches focus on the extraction of declarative process mod-
80 els. López et al. [4] present a framework for discovering business process models from
81 text using declarative models represented as Dynamic Condition Response (DCR)
82 graphs. Their method combines machine learning and expert systems, utilizing two
83 BERT models for sentence classification and Named Entity Recognition (NER) to
84 extract roles, events, and relationships.

85 van der Aa et al. [2] introduced a technique to extract declarative constraints
86 from sentences and represent them in the Declare language using an NLP pipeline.
87 The approach supports five Declare templates: *Init*, *End*, *Precedence*, *Response* and
88 *Succession*. This method overcomes challenges in identifying process model elements,
89 such as distinguishing noun-based activities and interpreting modal verbs. The
90 process involves three stages: linguistic processing to identify key semantic elements,
91 activity extraction to recognize verb-based and noun-based activities, and constraint
92 generation to create Declare constraints.

93 Grohs et al. [21] recreated the experimental setup of [2] for prompting GPT-4
94 to discover Declare constraints. This method outperformed the benchmark in most
95 templates, particularly in precision. However, some templates had lower recall
96 due to misclassifications. Overall, their study discusses the potential of LLMs,
97 particularly GPT-4, in Business Process Management (BPM) tasks, focusing on
98 mining imperative and declarative process models, and assessing the suitability of
99 process tasks for Robotic Process Automation (RPA).

100 To summarize, the existing work faces three limitations. First, rule-based methods
101 commonly struggle with generalizing across diverse text inputs due to their reliance
102 on predefined rules, resulting in poor adaptability to varying linguistic structures.

Second, most approaches support only a limited subset of control-flow constructs, reducing their applicability in complex and flexible process scenarios. Third, the field lacks a standardized evaluation framework, with no widely accepted datasets or benchmarks for consistent assessment and comparison. In this work, we address these limitations by harnessing the capabilities of Large Language Models (LLMs) to enhance generalization and accuracy, facilitating robust handling of varied sentence structures. Additionally, we introduce support for arbitrary *Declare* templates, extending the range of expressible process constraints. To further advance the field, we contribute a comprehensive dataset of textual descriptions paired with validated *Declare* constraints, establishing a benchmark resource for systematic evaluation and facilitating future research on process model extraction.

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). It provides a diverse array of templates tailored for expressing constraints associated with single or multiple activities. The foundational set of *Declare* templates originates from an extensive catalog of temporal logic patterns commonly utilized in model checking [22]. These patterns, categorized into occurrence (such as Absence, Existence, BoundedExistence, etc.) and order (including Precedence, Response, etc.), align with numerous process specification descriptions [23].

Given its focus on modeling business processes, *Declare* interprets LTL specifications over finite-length process executions. Each *Declare* template has its semantics derived from LTL, and a unique graphical representation. Table 1 provides an overview of common *Declare* templates and how those can be concisely expressed

Table 1: A selected set of common Declare templates with their textual description and graphical representation.

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	

131 using natural language, together with the graphical notation. Using such templates, a
 132 Declare constraint expresses a constraint associated with single or multiple activities.

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

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

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

139 Below is an exemplary collection of Declare constraints:

- 140 1. User login must be the first action in the system initialization process, denoted
 141 by *Init(LogIn)*.
- 142 2. After the user login, the main menu should appear, denoted by *Response(LogIn,*
 143 *EnterMenu)*.

- 144 3. Only in the main menu uploading a file is possible, denoted by *Prece-*
 145 *dence(EnterMenu, UploadFile)*
 146 4. The logout action marks the end of the user session, denoted by *End(LogOut)*.
 147 5. Logging out is only possible after logging in, denoted by *Precedence(LogIn,*
 148 *LogOut)*
 149 6. Only in the main menu, resetting the password is allowed, denoted by *Prece-*
 150 *dence(EnterMenu, ResetPassword)*
 151 7. The user may reset their password at most once, denoted by *AtMostOne(ResetPassword)*.
 152 8. Before a user can delete a file, they must upload a file, denoted by *Prece-*
 153 *dence(UploadFile, DeleteFile)*.
 154 9. If a file is uploaded, it must be processed, denoted by *Response(UploadFile,*
 155 *ProcessFile)*.
 156 10. Upon processing the file, the changes should be applied automatically without
 157 delay, denoted by *ChainResponse(ProcessFile, ApplyChanges)*.

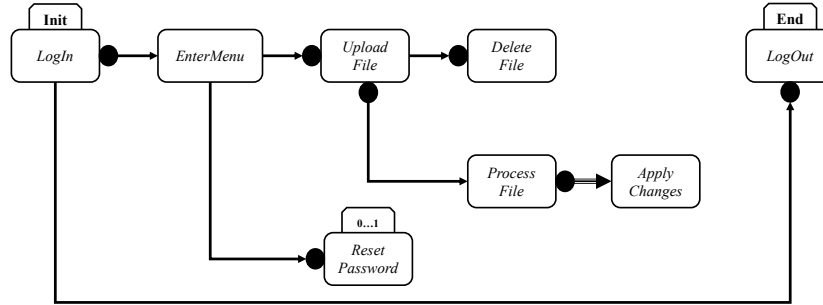


Figure 2: A graphical Declare model of the natural language specifications.

158 A Declare process model is a collection of Declare constraints. Each constraint in
 159 a Declare process model is defined over consistent activity names. Figure 2 depicts
 160 the process model that comprises the 13 constraints introduced above.

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

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

3.2. Decoder-only LLMs

In this subsection, we describe the architecture and functionality of decoder-only LLMs based on the Transformer model introduced in [24]. The main goal of decoder-only LLMs is to generate coherent and contextually relevant sequences of tokens based on an input sequence or prompt. The high-level architecture of such a model is depicted in Figure 3(a).

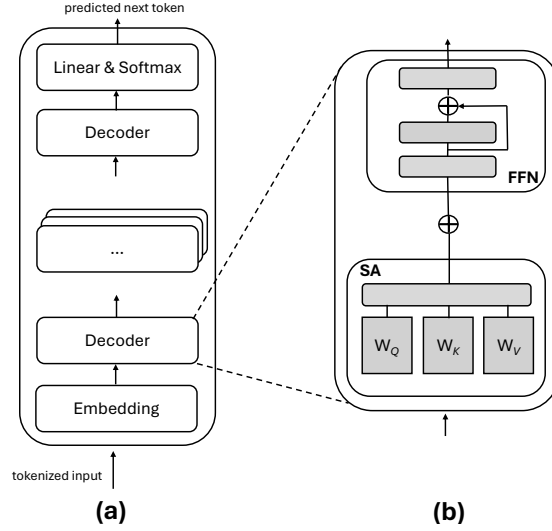


Figure 3: An overview of the LLM architecture. (a) shows the overall decoder-only model structure. (b) zooms into one decoder block, consisting of self-attention (SA), and feed-forward network (FFN) layers.

3.2.1. Input Tokenization and Embedding

The process starts with tokenizing the input text. For example, the input "What is the Declare constraint described in the following sentence? When a claim is created, it should be approved." is tokenized into individual tokens such as ["What", "is", "the", "Declare", "constraint", "described", "in", "the", "following", "sentence", "?", "When", "a", "claim", "is", "created", ",", "it", "should", "be", "approved", "."].

These tokens are then converted into a high-dimensional vector known as an embedding. This is achieved using an embedding layer that maps tokens to vectors

180 based on their vocabulary index. To incorporate positional information, positional
181 embeddings are added to each token embedding. This allows the model to understand
182 the order of tokens within the sequence.

183 3.2.2. Decoder Blocks

184 The model comprises several identical decoder blocks, stacked on top of the
185 embedding layer, each containing two fundamental layers:

186 *Masked Multi-Headed Self-Attention (SA)*. This mechanism computes attention
187 scores for each token but restricts attention to tokens that precede it in the sequence
188 (*masked* attention). This ensures the model processes tokens in a causal manner that
189 is suitable for tasks like language generation. The self-attention mechanism works
190 as follows: Each token in the input sequence undergoes linear projection into three
191 distinct vectors: key (K), query (Q), and value (V). This process involves three
192 separate weight matrices W_K , W_Q , and W_V . For each token, an attention score is
193 computed with every other token by calculating the dot product of their query and
194 key vectors. These raw attention scores are then normalized using a softmax function
195 to produce weights. The output for each token is generated as a weighted sum of
196 the value vectors, where the weights are determined by these attention scores. This
197 mechanism allows the model to prioritize relevant tokens for generating subsequent
198 tokens in the sequence.

199 For example, consider the input: "What is the Declare constraint described
200 in the following sentence: When a claim is created, it may be approved."
201 To calculate the attention scores for the token "constraint" using a single-head
202 attention mechanism:

203 1. **Linear Projections:** Assume the token embeddings for "constraint" (at
204 position t) and the preceding tokens have been projected to query (Q), key (K), and
205 value (V) vectors using the weight matrices W_Q , W_K , and W_V .

206 2. **Attention Scores Calculation:** For token t "constraint", we compute
207 the dot product of its query vector Q_t with the key vectors K_i of all preceding tokens
208 (including itself).

$$\text{score}_{t,i} = Q_t \cdot K_i \quad \text{for } i \leq t$$

209 Suppose the dot products yield the following raw scores: $[0.2, 0.1, 0.3, 0.4, 0.2]$ for
210 the tokens up to "constraint".

211 **3. Softmax Normalization:** Apply the softmax function to obtain normalized
212 attention weights:

$$\alpha_{t,i} = \frac{\exp(\text{score}_{t,i})}{\sum_{j=1}^t \exp(\text{score}_{t,j})}$$

213 For example,

$$\text{Softmax}([0.2, 0.1, 0.3, 0.4, 0.2]) \approx [0.191, 0.173, 0.211, 0.234, 0.191]$$

214 These values represent the attention weights $\alpha_{t,i}$.

215 **4. Weighted Sum of Values:** Compute the weighted sum of the value vectors
216 V_i using the attention weights:

$$\text{output}_t = \sum_{i=1}^t \alpha_{t,i} V_i$$

217 This yields the new representation for the token "constraint".

218 The self-attention mechanism operates across multiple heads in parallel, with each
219 head focusing on different segments of the sequence. To achieve this, separate key,
220 query, and value projections are performed for each head, and their dimensionality
221 is reduced to $\frac{d}{H}$ (where H is the number of attention heads). The outputs from all
222 heads are concatenated and then linearly projected back to the original dimension d .

223 *Feed-Forward Neural Network (FFN).* Finally, each token representation passes
224 through a feed-forward neural network composed of two linear layers with a nonlinear
225 activation in between. This transformation further refines the token representations
226 before passing them to the next decoder block or output layer.

227 To stabilize training and improve gradient flow, each sub-layer within the decoder
228 block is augmented with residual connections and layer normalization. This helps
229 in mitigating issues like vanishing gradients and enhances training efficiency.

230 3.2.3. Output Layer

231 After passing through multiple decoder blocks, the token representations are
232 processed by a final linear layer followed by a softmax activation. This layer computes
233 probabilities over the model’s output vocabulary, determining the likelihood of each
234 token appearing next in the sequence. The token with the highest probability is
235 selected as the model’s prediction.

236 To generate subsequent tokens, the model appends each predicted token to the
237 input sequence. This updated sequence is then fed back into the decoder blocks,
238 where it undergoes iterative processing to predict the next token in the sequence.
239 This process repeats until a predefined maximum sequence length is reached or an
240 end-of-sequence token is generated. The final output is a sequence of tokens that
241 represents the model’s prediction for the entire input sequence, capturing contextual
242 dependencies and generating coherent sequences. This way, we can obtain the fol-
243 lowing output sequence: "What is the Declare constraint described in the
244 following sentence? When a claim is created, it should be approved.
245 Response(CreateClaim, ApproveClaim)"

246 4. Fine-Tuning Large Language Models

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

251 4.1. Downstream Task

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

255 The fine-tuning process employs *causal language modeling*, where the model is
256 trained to predict the next token in a sequence given the previous tokens. This method
257 is particularly suitable for generation tasks, e.g., translating sentences into constraints,

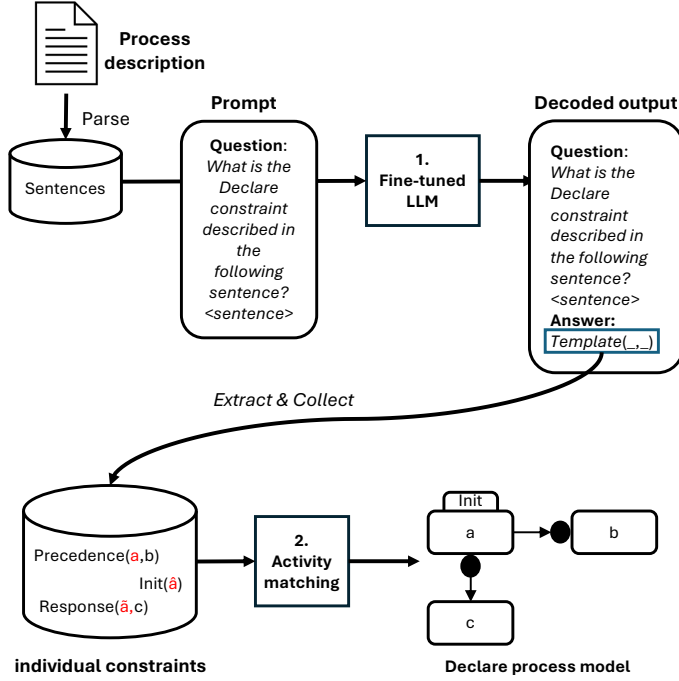


Figure 4: The overall pipeline required to transform a process description into a Declare model.

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, prepend-

ing 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) [25]. 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.

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 $h_{\text{out}} = W_0 h_{\text{in}}$. LoRA modifies this output by introducing an incremental update ΔW that encapsulates task-specific knowledge:

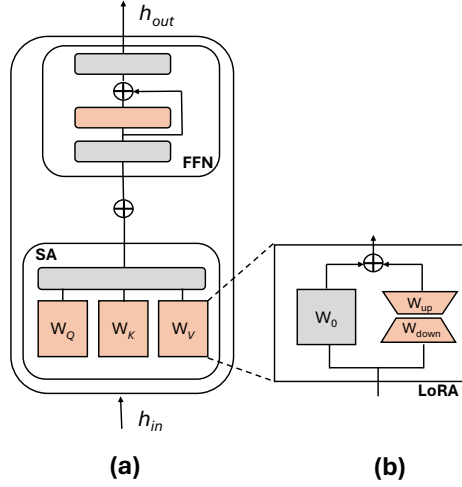


Figure 5: (a) zooms into one decoder block of the LLM, consisting of self-attention (SA), and feed-forward network (FFN) layers. (b) shows the detailed structure of the LoRA adaptation. The components shown in grey are kept frozen during fine-tuning, whereas the orange components contain updated weights.

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

where α is a scaling factor [26].

Figure 3 illustrates the architecture of the fine-tuned LLM with LoRA integration, highlighting the components that are adapted and those that remain frozen during fine-tuning.

4.2.1. Training Configuration

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 0.0002 to ensure a balanced update of the model weights, promoting convergence without overshooting the optimal parameter values.

311 5. Activity Matching

312 This section describes the second step in extracting a coherent Declare process
 313 model. Figure 4 shows how this step fits into the overall pipeline. In the following,
 314 we first introduce the problem that this step aims to tackle and then present several
 315 solutions to the problem.

316 5.1. Problem Definition

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

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

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

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

328 where $\text{UpdateMapping}(\kappa, Act)$ is a function that updates the mapping κ such
 329 that activities are drawn from the set of activity equivalence classes without changing
 330 the semantics of any of the activities $\kappa(x_i) = a_i$ for all i in the range $(1, \dots, m)$.

331 5.2. Solutions

332 We provide two solutions to the presented problem.

333 5.2.1. Zero-Shot Prompting

334 First, a zero-shot prompting approach prompts LLMs by providing only instruc-
 335 tions for the desired output without any examples of how the task has been performed
 336 correctly. The following is an example of zero-shot prompting: **"Take this list of**

337 Declare constraints containing various activities. Init(LogIn), AtMo-
 338 stOne(ResetPassword),AtLeastOne(CreateProfile), End(LogOut), Response(
 339 UserLogin, EnterMenu),Response(FileDeletion, ProcessFile), Precedence(
 340 UploadFile, DeleteFile), ChainResponse(ProcessingFile, ApplyChanges).
 341 Match semantically equivalent activities and provide an updated list of
 342 the constraints after agreeing on one unique label per activity equiv-
 343 alence class.". Note that variations exist in how activities are expressed, ranging
 344 from verb-based (e.g., *DeleteFile*) to noun-based (e.g., *FileDeletion*) and gerund-based
 345 (e.g., *ProcessingFile*) formulations.

346 5.2.2. Few-Shot Prompting

347 Another approach is to provide examples of input with correct output. Such a
 348 prompt, where the input includes a few examples to guide the model in performing
 349 a specific task, is called a *few-shot* prompt. Consider the excerpt from an exemplary
 350 few-shot prompt below: "Take this list of Declare constraints containing
 351 various activities. [...] Match semantically equivalent activities and
 352 provide an updated list of the constraints after agreeing on one unique
 353 label per activity equivalence class. Take the following examples for
 354 which the task has been solved correctly: Input: [Response(CreateClaim,
 355 ApproveClaim), Response(ClaimCreation, SendInvoice), Init(CreatingClaim)],
 356 Output: [Response(CreateClaim, ApproveClaim), Response(CreateClaim, Send-
 357 Invoice), Init(CreateClaim)] [...]."

358 6. Evaluation

359 This section aims to validate the efficiency of our proposed method both inter-
 360 nally and externally. The experimental results reported in the evaluation are fully
 361 reproducible at <https://anonymous.4open.science/r/text2declare/>.

362 6.1. Benchmark Dataset

363 To support the development and evaluation of our approach, we prepared a
 364 benchmark dataset comprising 969 text descriptions paired with corresponding

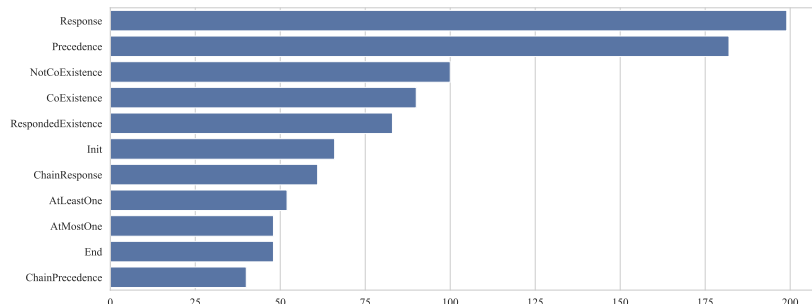


Figure 6: Distribution of Declare constraints in the benchmark dataset

365 gold-standard Declare constraints across a variety of domains, including healthcare,
 366 corporate compliance, education, and finance¹. Each constraint is carefully anno-
 367 tated by domain experts, providing a reliable reference for the best-fitting Declare
 368 constraint for each text input.

369 These gold-standard labels capture the most suitable constraint for each de-
 370 scription. However, given potential ambiguities in natural language, certain text
 371 inputs may correspond to multiple valid Declare templates. For instance, consider
 372 the sentence, "If a file is uploaded, it must be processed." Here, two in-
 373 terpretations are possible: *RespondedExistence(UploadFile, ProcessFile)* and *Re-*
 374 *sponse(UploadFile, ProcessFile)*. Although the context implies that uploading pre-
 375 cedes processing, this sequence is not explicitly stated, allowing for interpretation
 376 as a *RespondedExistence* relation.

377 This dataset spans 11 different Declare templates, including *Init*, *End*, *AtMostOne*,
 378 *AtLeastOne*, *Response*, *Precedence*, *ChainResponse*, *ChainPrecedence*, *RespondedEx-*
 379 *istence*, *CoExistence*, and *NotCoExistence*. Figure 6 illustrates the distribution of
 380 these constraints, with *Response* and *Precedence* being the most prevalent due to
 381 their central role in capturing process dependencies and order relations.

¹The dataset is publicly available at <https://anonymous.4open.science/r/text2declare/data/>

382 6.2. Internal Validation

383 In this section, we internally validate our proposed method by performing a cross-
384 validation using the benchmark dataset. To that end, we compare five different open-
385 source, decoder-only LLMs with similar sizes, namely seven to eight billion parameters.

386 6.2.1. Evaluation Metrics

387 In assessing the accuracy of the constraints generated by the fine-tuned LLM,
388 it is essential to identify a suitable metric. For each sentence in the respective test
389 dataset, we have both the gold standard constraint and the discovered constraint.

390 During our experiments, we encountered three types of errors:

- 391 • template error: the discovered template is incorrect
- 392 • activity error: at least one of the extracted activities is incorrect
- 393 • direction error: the extracted activities are correct, but the order is incorrect

394 The existing work [2] evaluates the approach by framing the problem as a slot-
395 filling problem. They assume that each constraint can be represented as a set of slots.
396 For example, a unary constraint like $Init(A)$ has two slots: one for the template
397 and one for the activity. Similarly, a binary constraint has three slots: one for the
398 template and one for each activity involved.

399 If we denote the set of slots found by the model as A and the set of slots com-
400 prising the gold standard constraint as G , we can define the common metrics recall,
401 precision, and F1-score as follows:

- 402 • $precision = \frac{|A \cap G|}{|A|}$
- 403 • $recall = \frac{|A \cap G|}{|G|}$
- 404 • $F1\ score = \frac{2 \times precision \times recall}{precision + recall}$

405 However, this implementation fails to penalize direction errors adequately. Con-
406 sider the scenario where the gold standard constraint is $Precedence(X, Y)$ while the
407 discovered constraint is $Precedence(Y, X)$. Then, we obtain:

$$\begin{aligned}
408 \quad \text{precision} &= \frac{|\{Precedence, X, Y\} \cap \{Precedence, Y, X\}|}{|\{Precedence, X, Y\}|} = 1 \text{ and} \\
409 \quad \text{recall} &= \frac{|\{Precedence, X, Y\} \cap \{Precedence, Y, X\}|}{|\{Precedence, Y, X\}|} = 1.
\end{aligned}$$

410 In this case, both precision and recall are calculated as 1, indicating perfect
411 performance despite the discovered constraint being semantically different from the
412 gold standard.

413 Furthermore, we contend that in our method, template errors should be penalized
414 more heavily than activity errors. This is primarily because activity extraction is a
415 task that the LLM executes seamlessly and without major difficulties. Typically, the
416 identified activities align closely with those in the gold standard constraint, albeit
417 with potential slight variations in wording. In the approach outlined by van der Aa
418 et al. [2], activity extraction was a distinct step in the NLP pipeline, considered a
419 notable challenge in itself. Conversely, template errors fundamentally change the
420 semantic meaning of the constraint.

421 To comprehensively assess the performance of our fine-tuned LLM, we propose
422 *precision*, *recall*, and *F1 score*. Algorithm 1 shows an algorithm to calculate the
423 metrics given a ground truth (*gs*), prediction (*pred*), and α that determines how
424 much the template slot should be weighted. In our evaluation, we use the weighting
425 factor $\alpha = 2$. This way, we ensure that template errors are penalized more heavily. In
426 addition, our implementation also penalizes direction errors. To compute the metrics,
427 we consider *true positives (TP)* as the correctly predicted instances of this specific
428 constraint (template and activities) compared to the gold standard instances of this
429 constraint. *False positives (FP)* are the incorrectly predicted instances of this specific
430 constraint, while *false negatives (FN)* refer to the gold standard instances of that
431 constraint that were not correctly predicted.

432 6.2.2. Experimental Setup

433 A 10-fold cross-validation was utilized to evaluate our approach. The benchmark
434 dataset was randomly partitioned into ten equally sized, non-overlapping subsets.
435 Each subset was used once as a test set, while the LLM was fine-tuned using the other
436 nine subsets as training data. We compare five different open-source, decoder-only
437 LLMs with similar sizes, namely seven to eight billion parameters. For each model,

Algorithm 1 Precision, Recall, and F1 Score

```
1: function EVALUATE( $gs, pred, \alpha$ ) ▷ gs: gold standard
2:   Given:  $gs = k(x_1, \dots, x_m)$  and  $pred = \tilde{k}(\tilde{x}_1, \dots, \tilde{x}_n)$ 
3:    $TP \leftarrow 0, FP \leftarrow 0, FN \leftarrow 0$ 
4:   if  $k = \tilde{k}$  then ▷ correct template
5:      $TP \leftarrow TP + \alpha * 1$ 
6:     for  $i \leftarrow 1, m$  do
7:       if  $x_i = \tilde{x}_i$  then  $TP \leftarrow TP + 1$ 
8:       else  $FP \leftarrow FP + 1, FN \leftarrow FN + 1$ 
9:   else ▷ incorrect template
10:     $FP \leftarrow FP + \alpha * 1, FN \leftarrow FN + \alpha * 1$ 
11:    for  $i \leftarrow 1, m$  do
12:      if  $\tilde{x}_i$  exists then
13:        if  $x_i = \tilde{x}_i$  then  $TP \leftarrow TP + 1$ 
14:        else  $FN \leftarrow FN + 1$ 
15:      else  $FN \leftarrow FN + 1$ 
16:    for  $i \leftarrow 1, n$  do
17:      if  $x_i$  exists then
18:        if  $x_i = \tilde{x}_i$  then  $TP \leftarrow TP + 1$ 
19:        else  $FP \leftarrow FP + 1$ 
20:      else  $FP \leftarrow FP + 1$ 
21:     $precision \leftarrow \frac{TP}{TP + FP}, recall \leftarrow \frac{TP}{TP + FN}$ 
22:     $F1 \leftarrow \frac{2 \times precision \times recall}{precision + recall}$ 
23:    return  $precision, recall, F1$ 
```

438 the training parameters were identical.

439 6.2.3. Experimental Results

440 Table 2 shows the experimental results. The template accuracy measures the
441 accuracy of translating templates. In other words, given a gold standard constraint
442 $k(x_1, \dots, x_m)$ and a found constraint $\tilde{k}(\tilde{x}_1, \dots, \tilde{x}_n)$, we obtain a template accuracy of

one if $k = \tilde{k}$, and zero otherwise.

Table 2: Performance Comparison of Different Models

Model	F1 ($\alpha = 2$)	F1 ($\alpha = 4$)	Template accuracy
Gemma 7B	0.884	0.918	0.980
Mistral 7B	0.839	0.868	0.978
Falcon 7B	0.853	0.894	0.971
Llama-2 7B	0.805	0.862	0.971
Llama-3 8B	0.849	0.891	0.970

The comparison of various similar LLMs in terms of structure and size reveals only minor differences in performance indicators. Specifically, template accuracy ranges from 0.97 to 0.98 in cross-validation averages. This indicates that all the considered LLMs perform similarly when distinguishing the semantics of Declare templates. Significant differences are noted only in the F1 scores, which account for activity errors as well. Overall, Google’s Gemma 7b achieved the highest scores for both F1 scores and template accuracy.

6.3. External Validation - Part A

Next, we compare our proposed approach with the state-of-the-art method by van der Aa [2].

6.3.1. Baseline

We evaluate our proposed method against the state-of-the-art approach proposed by [2]. For a given input sentence, this approach returns the template of the found Declare constraint together with the involved activities. The implementation is based on a tailored NLP pipeline, as discussed earlier.

6.3.2. Experimental Setup

Here, we limit ourselves to a subset of Declare templates - *Init*, *End*, *Precedence*, and *Response*. Apart from *Succession*, those are the only templates supported by the state-of-the-art approach. Since the *Succession* relation can be characterized

463 as a conjunction of a *Response* and *Precedence* relation between two activities, we
464 neglect this in our work.

465 6.3.3. Data Collection

466 For fine-tuning the Gemma-7b model, we employ a training set consisting of 850
467 sentences paired with the respective gold standard Declare constraint. In particular,
468 this dataset does not include any sentence from the data collection used in [2],
469 since part of their data is used for testing. To assess performance, we first use a
470 human-checked validation set (*V1*) consisting of 80 text inputs covering the four
471 mentioned templates, disjoint from the training data. In addition, we validate both
472 approaches on a subset of the validation set used by van der Aa et al., which we refer
473 to as *V2*. From the original data collection of 103 sentences, we excluded those that
474 either have *Succession* or a negative Declare template as the gold standard template,
475 as well as those that do not describe one of the introduced Declare templates but
476 rather a construct like an If-condition (e.g., *If an error is detected, another arbitrary*
477 *repair activity is executed; otherwise the repair is finished*).

478 6.3.4. Experimental Results

479 When comparing our approach to the state-of-the-art (Table 3) for selected
480 Declare templates across two distinct datasets, we observe a substantial improvement.
481 We evaluate the extracted constraints of our fine-tuned Gemma model and the
482 state-of-the-art approach on *V1* and *V2* and only compare their template accuracy.
483 Comparing overall performance (such as F1 score) is inappropriate, as it involves
484 considering activity errors. This is because the activities extracted by the state-of-
485 the-art approach often differ in wording. In our validation set, the gold standard
486 utilizes verb-based activities in Pascal style, consistent with the examples presented
487 in this paper and the labeled samples in our training set. For validation set *V1*, the
488 fine-tuned Gemma 7b model achieves nearly perfect template accuracy, compared
489 to just 53.8% accuracy by the state-of-the-art approach. This trend continues in the
490 validation set *V2*, albeit with less severity.

491 *6.4. External Validation - Part B*

492 Next, we compare our proposed approach with two prompting methods. Recent
 493 research [21] demonstrates that prompting methods with GPT-4 already surpass
 494 the benchmarks established by the state-of-the-art [2]. We aim to show that our
 495 approach based on fine-tuning is even more reliable than prompting. Specifically,
 496 prompting methods are not well-suited for covering arbitrary Declare templates, as
 497 the task becomes more complex and the prompt grows larger with each additional
 498 template that needs to be differentiated.

499 *6.4.1. Baselines*

500 *Few-Shot Prompting.* The first baseline uses few-shot prompting. For each template,
 501 we provide one example (shot), each one reflecting the textual description of the
 502 corresponding template as introduced in Table 1. Appendix A shows the sketch
 503 of the prompt.

Table 3: Comparison with the state-of-the-art [2]. Entries in the two rightmost columns specify the template accuracy.

		State-of-the-art		Our approach
		Cases	[2]	(Gemma 7B)
V1	Init	11	0.273	0.910
	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	0.963
	Overall	45	0.822	0.956

504 *Few-Shot Chain-of-Thought (CoT) Prompting.* The second baseline combines few-
 505 shot prompting with chain-of-thought (CoT) reasoning. In addition to the abstract
 506 template definitions at the beginning of the prompt, we provide an example transla-
 507 tion together with a chain-of-thought for each Declare template. Appendix B shows
 508 an excerpt of the prompt.

509 6.4.2. Experimental Setup

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

515 6.4.3. Experimental Results

Table 4: Comparing prompt engineering techniques to our fine-tuning approach.

Technique	F1 ($\alpha = 2$)	F1 ($\alpha = 4$)	Template accuracy
Few-shot prompt	0.268	0.334	0.459
Few-shot CoT prompt	0.598	0.588	0.565
Our approach	0.911	0.940	0.994

516 Comparisons with both prompting baselines Table 4 show that fine-tuning sig-
 517 nificantly enhances template accuracy, achieving 0.994. Furthermore, incorporating
 518 chain-of-thoughts into the prompt improves accuracy (0.565) compared to merely
 519 providing example translations (0.459). For both the few-shot prompt and the
 520 fine-tuned LLM, template accuracy surpasses F1 scores. Only for the few-shot-CoT
 521 prompt does template accuracy fall below the F1 scores. This suggests that this
 522 technique performs better at extracting activities and wording those exactly in the
 523 same style as in the provided shots rather than detecting the correct Declare template.

524 7. Discussion

525 7.1. Findings

526 When examining the cross-validation results of various LLMs, we observe that
527 template accuracy consistently surpasses F1 scores for $\alpha=2$ and $\alpha=4$. This suggests
528 that the LLMs are more proficient at distinguishing templates than at replicating the
529 exact wording of activities from the training dataset. As α increases, the F1-score
530 approaches template accuracy since activity errors are less penalized. Assuming that
531 matching equivalent but slightly differently worded activities is a rather straightfor-
532 ward post-processing step, template accuracy emerges as the most crucial indicator.

533 A direct comparison between the two Llama models reveals that having more
534 parameters does not necessarily translate to better performance, as evidenced by
535 their nearly identical template accuracies. However, the older Llama model has lower
536 F1 scores, indicating that the 8B version more effectively mimics the phrasing style
537 of the activities.

538 The few instances where the fine-tuned LLMs failed to translate sentences
539 correctly into the appropriate templates are primarily due to linguistic ambigu-
540 ities. For instance, in the sentence "Upon receiving a support ticket, the
541 troubleshooting process begins.", one can extract either an *Init* constraint
542 (*Init(ReceiveSupportTicket)*) or a *Response* constraint (*Response(ReceiveSupportTicket,*
543 *BeginTroubleshooting)*). The desired abstraction level of activities plays a crucial role
544 in determining the correct template here.

545 Comparing our approach with the state-of-the-art clearly demonstrates an im-
546 provement in accuracy for this translation task. Validation on *V1* shows that our
547 LLM-based classifier significantly outperforms the state-of-the-art in determining
548 the correct template. Notably, the largest discrepancies are seen in unary constraints
549 such as *Init* and *End*. The state-of-the-art method, as implemented by van der Aa,
550 relies on detecting meta-actions involving the term "process" and synonyms for
551 "start" and "end". Our LLM-based approach overcomes this with its robust nat-
552 ural language understanding capabilities. Also, for common templates like *Response*
553 and *Precedence*, our fine-tuned LLM shows a significant improvement, indicating

554 that it effectively learned to distinguish between these templates, which have subtle
555 but crucial semantic differences.

556 For validation set V2, a similar trend is observed for most templates, except
557 for *Precedence* constraints, where the state-of-the-art approach achieves a template
558 accuracy of 0.963. This value is not due to the sophisticated NLP capabilities of
559 their implementation but rather to the default interrelations between activities that
560 are generated in the absence of a matching pattern. These default interrelations are
561 later transformed into precedence constraints in their pipeline.

562 The comparison of our approach with two prompting techniques indicates that
563 fine-tuning is essential for reliable results in this translation task. Advanced prompt-
564 ing techniques, such as chain-of-thought prompting, have limitations because the
565 model weights are not adapted through prompting but rather help the model under-
566 stand the task better. Here, iterative refinements of the prompt to select appropriate
567 examples and detailed chain-of-thoughts without introducing biases were necessary
568 to enhance performance compared to the simple few-shot approach.

569 Extracting activities from textual descriptions is a vital first step in reliably
570 identifying constraints for declarative process models. This initial phase involves
571 recognizing and accurately parsing the core activities within a text, which serve
572 as the foundational elements for defining constraints. Although this is a complex
573 task in specialized NLP pipelines, LLMs can reliably perform this extraction and
574 subsequently use those activities to determine the correct template and, ultimately,
575 the whole constraint. This capability became particularly evident during the design
576 and testing of dedicated prompts, which later served as baselines.

577 7.2. Limitations

578 Our current approach has several limitations that need to be addressed. Firstly,
579 it operates under the assumption of a one-to-one correspondence between sentences
580 and constraints, which is not always realistic in practical scenarios. In real-world
581 process descriptions, one sentence can also contain multiple or no constraints at
582 all. Consider, for example, this sentence: "A claim must be created, before it
583 can be approved or rejected." This sentence contains two constraints: *Prece-*

584 *dence(CreateClaim, ApproveClaim)* and *Precedence(CreateClaim, RejectClaim)*. Cur-
 585 rently, our model translates this description into the following constraint: *Prece-*
 586 *dence(CreateClaim, ApproveOrRejectClaim)*. For such cases, a post-processing
 587 step is required to detect both described constraints. Furthermore, it is possi-
 588 ble that one constraint is described over multiple sentences. Therefore, consider
 589 the following specification: "Optionally, a claim is created. In this case,
 590 it must be approved afterwards." From this description, we can extract *Re-*
 591 *sponse(CreateClaim, ApproveClaim)*. Although for this specific example, our model
 592 is able to detect the constraint, it is questionable whether it is generally capable of
 593 doing that. Additionally, our fine-tuned model does not support advanced Declare
 594 constructs such as branched responses or time conditions, limiting its applicability
 595 in more complex scenarios. For instance, a branched response between activities a,
 596 b, and c enforces that every occurrence of a should eventually followed by b or c.
 597 The inherent ambiguity of natural language further complicates the task, leading
 598 to occasional misinterpretations and inaccuracies. The most significant challenge is
 599 the absence of comprehensive training data to further fine-tune LLMs, also for the
 600 above-mentioned scenarios.

601 7.3. Future Work

602 To address these limitations, future work will focus on several key areas. One
 603 priority is to train the LLMs for more advanced Declare constructs, such as branched
 604 responses. This will require generating adequate training data that accurately rep-
 605 resents these complex constructs. Additionally, developing tool support for the
 606 entire pipeline, including activity matching, will enhance the practical utility of our
 607 approach. Another important extension is to support sentences that map to multiple
 608 constraints or by managing multiple input sentences with potential inter-sentence
 609 dependencies that map to one or several output constraints. To accomplish this,
 610 post-processing steps can be helpful, e.g., splitting up activities, especially if they
 611 contain conjunctions. A more promising approach is to explicitly train the model on
 612 such scenarios. For instance, one can explore advanced techniques such as *curriculum*
 613 *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

The proposed approach leverages fine-tuned LLMs to translate natural language descriptions into Declare constraints, significantly advancing the field of declarative process model extraction. By demonstrating improved accuracy and generalization over traditional NLP methods, our approach highlights the potential of LLMs to handle diverse textual inputs. This method not only surpasses existing state-of-the-art techniques but also expands the scope of process model extraction to accommodate a wider range of Declare templates. Despite these advancements, several challenges remain. In particular, the dependency on high-quality, annotated training data poses a barrier to scalability. Future work will focus on addressing current limitations, such as supporting more complex constructs and enhancing tool support, to further refine and extend the applicability of this approach.

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 2019*, 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 2021*, Springer, 2021, pp. 109–117.
- [5] J. C. de AR Gonçalves, F. M. Santoro, F. A. Baiao, A case study on designing business processes based on collaborative and mining approaches, in: *14th International Conference on Computer Supported Cooperative Work in Design*, IEEE, 2010, pp. 611–616.
- [6] F. Friedrich, J. Mendling, F. Puhlmann, Process model generation from natural language text, in: *CAiSE 2011*, 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: *2015 IEEE 9th international conference on research challenges in information science (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 2018, Proceedings, Part I 11*, Springer, 2018, pp. 91–103.
- [9] S. Sholiq, R. Sarno, E. S. Astuti, Generating bpmn diagram from textual requirements, *Journal of King Saud University-Computer and Information Sciences* 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](https://arxiv.org/abs/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, *Enterprise Information Systems* 0 (2024) 2417404.
- [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 2021*, Springer, 2021, pp. 75–90.
- [15] L. Quishpi, J. Carmona, L. Padró, Extracting annotations from textual descriptions of processes, in: *Business Process Management: 18th International Conference, BPM 2020, Proceedings 18*, 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: *International Conference on Enterprise Information Systems*, volume 2, 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 2020*, 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, *Information Systems* 64 (2017) 447–460.
- [20] H. Van der Aa, H. Leopold, H. A. Reijers, Checking process compliance against natural language specifications using behavioral spaces, *Information Systems* 78 (2018) 83–95.
- [21] M. Grohs, L. Abb, N. Elsayed, J.-R. Rehse, Large language models can accomplish business process management tasks, in: *International Conference on Business Process Management*, Springer, 2023, pp. 453–465.
- [22] C. Di Ciccio, M. Montali, Declarative process specifications: reasoning, discovery, monitoring, in: *Process mining handbook*, Springer International Publishing Cham, 2022, pp. 108–152.
- [23] M. B. Dwyer, G. S. Avrunin, J. C. Corbett, Patterns in property specifications for finite-state verification, in: *Proceedings of the 21st international conference on Software engineering*, 1999, pp. 411–420.
- [24] 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.), *Annual Conference on Neural Information Processing Systems*, 2017, pp. 5998–6008.
- [25] 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: *The Tenth International Conference on Learning Representations, ICLR 2022*, 2022.
- [26] 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](https://arxiv.org/abs/2403.14608).

Appendix A. Baseline 1: Few Shot Prompting

Below is the prompt for the few-shot-prompting-based baseline used in the evaluation (cf. Subsubsection 6.4.1:

721

722 I am a highly intelligent AI capable of understanding and translating natural language descriptions

723 into Declare constraints. Here are some example translations of descriptions into Declare constraints:

724 1. Description: Breakfast must be the first to occur.

725 Task(s): Breakfast

726 Declare Constraint: Init(Breakfast)

727 more shots*

728 11. Description: Cooking and Biking never occur together.

729 Task(s): Cooking, Biking

730 Declare Constraint: NotCoExistence(Cooking, Biking)

731 Given the following description, we should first detect tasks and then translate the description into

732 a Declare constraint. The output format should exactly follow this pattern as in the example descrip-

733 tions. The answer should solely contain the constraint and nothing else, i.e., no additional descrip-

734 tion. Now, let's translate the following description into a Declare constraint:

735 12. Description: *description*

736

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

738 Below is the excerpt of the prompt for the few-shot-CoT-prompting-based base-

739 line used in the evaluation (cf. Subsubsection 6.4.1. Note that we employ a specific

740 syntax for this prompt. The instructional version of Google's Gemma 7b-it model

741 we chose for inference utilizes control tokens to enhance the clarity and structure

742 of conversations. This syntax includes tokens for indicating the user's turn (`user`),

743 the model's turn (`model`), the start of a dialogue turn (`<start_of_turn>`), and the

744 end of a dialogue turn (`<end_of_turn>`). This clear differentiation of roles and turns

745 improves the model's performance by providing explicit context and structure.

746

747 `<start_of_turn>user`

748 You are a highly intelligent AI capable of understanding and translating natural language descrip-

749 tions into Declare constraints. Here are definitions of some common Declare templates:

750 1. Definition: "Task A must be the first to occur."

751 Declare: Init(A)

752 more definitions*

753 11. Definition: "Task A and Task B never occur together."

754 Declare: NotCoExistence(A, B)

755 Think and write your step-by-step reasoning before responding.

756 Now, translate the following description into a Declare constraint: "They have to create the claim,

757 before they can approve it."

758 <end_of_turn>

759 <start_of_turn>model

760 Let's think step by step. In this example, we find two tasks, namely, creating a claim and approv-

761 ing it. We, therefore, define CreateClaim and ApproveClaim as tasks. We define Task A = CreateClaim

762 and Task B = ApproveClaim because of the temporal sequence of these tasks. According to the descrip-

763 tion, CreateClaim (Task A) is a condition for ApproveClaim (Task B). This matches the definition of

764 a Precedence constraint (If Task B occurs, then Task A must occur before).

765 Thus, the resulting constraint is: Precedence(CreateClaim, ApproveClaim)

766 <end_of_turn>

767 more shots with chain-of-thoughts*

768 <start_of_turn>user

769 Now, translate the following description into a Declare constraint: "Booking the venue should only

770 happen once."

771 <end_of_turn>

772 <start_of_turn>model

773 Let's think step by step. In this example, we find one task, namely booking the venue. We, there-

774 fore, define BookVenue as the task. We define Task A = BookVenue. According to the description, BookV-

775 enue (Task A) should only happen once. This matches the definition of an AtMostOne constraint (Task

776 A occurs at most once).

777 Thus, the resulting constraint is: AtMostOne(BookVenue)

778 <end_of_turn>

779 <start_of_turn>user

780 Now, let's translate the following description into a Declare constraint: description.

781 <end_of_turn>
782 <start_of_turn>model
783 Let's think step by step.