LLM-based Business Process Models Generation from Textual Descriptions

Xiaoxuan Li[1,2], **Lin Ni**[1], **Xin Wang**[2], **Yitong Tang**[3], **Ruoxuan Li**[3], **Jiamou Liu**[1], **Zhongsheng Wang**[1]

[1]The University of Auckland, New Zealand
[2]Uxtrata Limited, New Zealand
[3]Southwest University, China

xli443@aucklanduni.ac.nz, lni600@aucklanduni.ac.nz,
info@uxtrata.com, jiamou.liu@auckland.ac.nz, zwan516@aucklanduni.ac.nz

Abstract

Business process modeling has traditionally depended on manual efforts or rigid rule-based techniques, limiting scalability and flexibility. Recent progress in Large Language Models (LLMs) enables automatic generation of process models from text, yet a systematic evaluation remains lacking. This paper explores the ability of LLMs to produce structurally and semantically valid business process workflows using five approaches: zero-shot, zero-shot CoT, few-shot, few-shot CoT, and fine-tuning. We assess performance under increasing control-flow complexity (e.g., nested gateways, parallel branches) using the publicly available large-scale MaD dataset, and introduce a masked-input setting to test semantic robustness. Results show that while fine-tuning achieves the best accuracy, few-shot CoT excels in handling complex logic and incomplete inputs. These findings reveal the strengths and limits of LLMs in process modeling and offer practical guidance for enterprise Business Process Management (BPM) automation.

1 Introduction

Business Process Management (BPM) plays a critical role in discovering, designing, analyzing, and optimizing business processes across industries (Jeston, 2014). A key step in the BPM lifecycle is the generation of Business Process Model and Notation (BPMN) diagrams, which offer a standardized visual framework for business processes, enhancing operational efficiency and decision-making support (von Rosing et al., 2015). Figure 1 shows the possible atomic components in BPMN and a simple example of BPMN diagram. Generating accurate BPMN diagrams remains a complex and labor-intensive task, demanding significant domain expertise and access to structured data(Friedrich et al., 2011; van der Aa et al., 2019; Reijers et al., 2003). This dependence on expert knowledge and structured information often serves as a critical barrier to the efficiency of process improvement efforts.

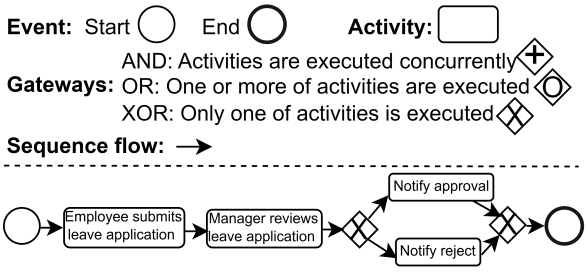

Figure 1: Basic components of a BPMN Diagram and a simple BPMN diagram for "the approval process of an employee applying for leave."

Although NLP has been applied to BPM, challenges such as natural language inherent ambiguity and lack of structured input hinder accurate text-to-BPMN conversion (Van der Aa et al., 2018). Rule-based and NLP-hybrid methods typically rely on syntactic patterns and handcrafted templates (Friedrich et al., 2011; Ferreira et al., 2017), offering interpretability but suffering from rigidity and low scalability. Neural models like BiLSTM-CRF or ON-LSTM require large annotated datasets and also struggle with complex flows (Suárez-Paniagua et al., 2019; Qian et al., 2020; Han et al., 2020). Tools like NLP4BPM and BPMN Sketch Miner (Delicado Alcántara et al., 2017; Ivanchikj et al., 2020) support representation, but not full diagram generation directly. Recent advances in LLMs (e.g., GPT) offer strong capabilities in understanding and generating structured outputs (Ye et al., 2023), and some research has demonstrated the feasibility of automating BPMN generation from text using LLMs (Grohs et al., 2023; Kourani et al., 2024). However, many critical questions remain unanswered: How structurally accurate and semantically meaningful are the generated business process models? How do LLMs handle increasing process complexity? And how robust are these

Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, pages 523–533
December 20-24, 2025 ©2025 Association for Computational Linguistics

models when the input descriptions are incomplete or under-specified?

In this paper, we investigate how LLMs generate business process models from text, evaluating five strategies: zero-shot prompting, zero-shot Chain-of-Thought (CoT), few-shot prompting, few-shot CoT, and fine-tuning. These methods are assessed on their ability to produce syntactically valid, structurally accurate, and semantically coherent DOT representations of BPMN. After that, we categorize evaluation instances according to structural complexity such as nested gateways and multiple parallel branches, and assess how increasing control-flow complexity influences model performance across these methods. For consistent evaluation, experiments are conducted on the MaD dataset (Li et al., 2023), a large-scale benchmark containing text–BPMN pairs across diverse business domains. Finally, we introduce a masked-input setting to test the models' capacity of infering missing activities and reconstructing plausible control-flow logic, thereby testing their reasoning and generalization abilities. We utilize Graphviz to render model-generated DOT code into BPMN diagrams and evaluate outputs using syntactic validity checks, structural similarity (AGSS), and semantic similarity (ANSS), as long as human evaluation by domain experts. Figure 2 illustrates the overall pipeline from natural language descriptions to DOT representations and finally to BPMN outputs.

2 Related Work

Prompt Engineering for Large Language Models: Recent advances in large language models (LLMs) have significantly improved performance across a range of NLP tasks. Prompt engineering, as a technique to elicit desirable behaviors from LLMs, has gained prominence in various domains. Instruction tuning (Wei et al., 2021) enhances zero-shot capabilities by aligning model behavior with human intent. Few-shot prompting (Logan IV et al., 2021) further improves performance by providing contextual examples. For complex reasoning tasks, Chain-of-Thought (CoT) prompting (Wei et al., 2022) and its variants such as Zero-Shot-CoT (Kojima et al., 2022) and Auto-CoT (Zhang et al., 2022) offer a structured approach to reasoning by decomposing problems into intermediate steps. While these methods have proven effective in general text-to-code tasks (Chen et al., 2021), their applicability to highly structured and domain-specific outputs, such as BPMN representations in DOT language, remains underexplored.

Traditional Methods for BPMN Generation: BPMN is a standardized diagramming notation that provides a graphical representation of business processes. It is widely used in BPM to visually illustrate and communicate processes in a clear way (von Rosing et al., 2015). Early efforts in automating BPMN generation primarily relied on rule-based systems and constrained natural language input. Rule-based methods, as discussed by (Friedrich et al., 2011) and (Schumacher and Minor, 2014), often struggle with the complexities of dynamic business processes due to their dependence on pre-set rules. (Ivanchikj et al., 2020) explored using a constrained natural language for BPMN Sketch Miner, aiming for a balance between expressiveness and usability, though this method faces limitations in flexibility. (Sonbol et al., 2023) incorporated machine translation to maintain semantic integrity during BPMN generation, achieving an 81% similarity to manual models. (Sholiq et al., 2022) introduced a two-stage method that first analyzes textual requirements with NLP and then generates BPMN diagrams from these analyses. Despite their contributions, these methods either impose rigid input constraints or fail to handle complex constructs like nested gateways and ambiguous semantics.

Large Language Models for Business Process Modeling: Recent studies have begun to explore the use of large language models (LLMs) for generating business process models from text. Some approaches extract structured information such as activities, actors, and control-flow dependencies using zero-shot or few-shot in-context learning, returning outputs in JSON or tuple format (Bellan et al., 2022; Neuberger et al., 2024). These methods adopt fixed output schemas to ensure consistency and generalization across different process descriptions and LLM backends. To move beyond information extraction, several studies employ prompt-based methods to generate formal or semi-formal process representations. One approach designs structured prompts that instruct GPT-4 to produce outputs in a pre-specified intermediary notation, where tasks are written in plain text and control-flow logic is encoded using symbols such as arrows ($\rightarrow$), XOR, and AND. This lightweight representation facilitates downstream parsing into BPMN-XML (Grohs et al., 2023). Another method first prompts the LLM to extract fact

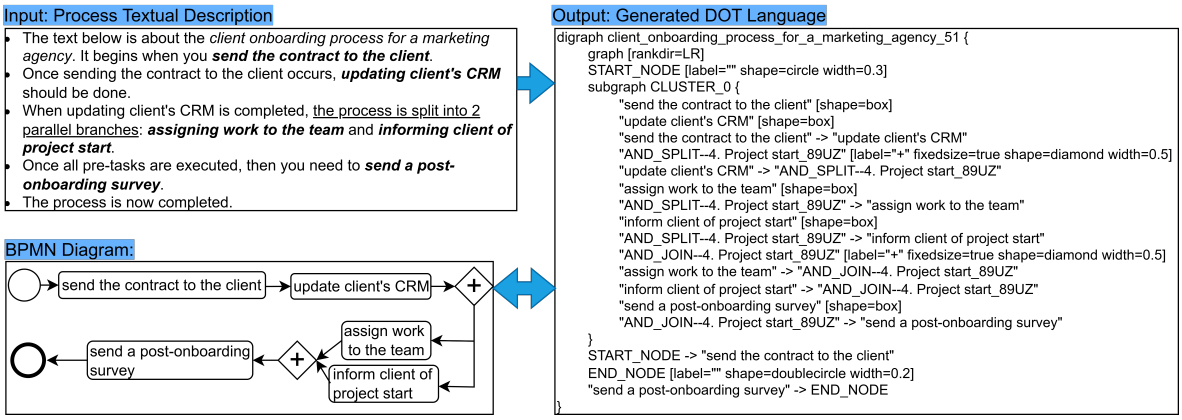

Figure 2: A BPMN Diagram (in DOT Language) generated from textual description using LLMs. In the upper left corner is a text description of the process. LLMs are used to generate the DOT language on the right as the intermediate code of the diagram description language. Then, external tools are used to translate the code into the corresponding BPMN diagram (lower right corner).

triples from text and then maps them into the Partially Ordered Workflow Language (POWL), which is subsequently compiled into executable BPMN or Petri net models (Kourani et al., 2024). Other approaches employ fine-tuned models trained on paired text-structure examples. For instance, one method uses a GPT-based model to output activities and their pertinent information from business process descriptions, leveraging training supervision to enforce schema adherence (Ajmal et al., 2024). Another method extracts regular-expression-like structures from input text, constructs abstract syntax trees (ASTs), and transforms them into BPMN using rule-based mappings (Nivon and Salaün, 2024).

Overall, prior work demonstrates the feasibility of using LLMs for BPMN generation, but existing methods often rely on intermediate layers, constrained templates, or require extensive prompt tuning. In contrast, our approach systematically compares five prompting and fine-tuning strategies for generating DOT-based BPMN representations and investigates how structural complexity affects model performance.

3 Task Definition

The core of this task can be described as follows: for a given natural language business process description sequence $T = \{t_1, t_2, \ldots, t_n\}$, we make full use of the large language model $\mathcal{L}$ to accurately generate the corresponding DOT language description $D = \{N, E\}$, which $N = \{n_1, n_2, \ldots, n_k\}$ is the node-set, representing BPMN elements such as activities, events, and gateways, and $E = \{e_{i,j} \mid n_i, n_j \in N\}$ is the collection of edges representing sequence flows between activities or events. It can be formalized as:

$$\mathcal{L}(T; P) = D$$

$P = \{P_{Zero-Shot}, P_{Zero-Shot-COT}, P_{Few-Shot}, P_{Few-Shot-COT}, P_{fine-tune}\}$ represents five different prompt learning approaches to comprehensively evaluate LLMs' ability to generate BPMN diagrams. The generated intermediate language D will be combined with external tools such as the local executor E to generate the corresponding BPMN diagram G. In this way, we have completed the automation process using LLMs.

$$G = LocalExcution(D, E)$$

LocalExecution represents a conceptual process involving the parsing of D, the mapping of nodes and edges to BPMN elements, and the rendering of the final BPMN diagram. By integrating $\mathcal{L}$ with E, we complete the automation pipeline from natural language descriptions to fully realized BPMN diagrams, thereby addressing the challenge of translating unstructured textual inputs into formal process models.

4 Methods

To enable BPM automation, we position LLMs as the core reasoning engine for transforming textual descriptions into process models. We introduce DOT language as an intermediate textual representation, enabling precise graph-based output that external tools (e.g., Graphviz) can convert into BPMN diagrams. DOT is a plain-text graph description

language widely used with Graphviz[1]. It allows straightforward encoding of process logic through nodes and edges, making it suitable for generating BPMN structures from text.

We evaluate five LLM-based strategies for generating BPMN models from text (Figure 3): Zero-Shot, Zero-Shot Chain-of-Thought (CoT), Few-Shot, Few-Shot CoT, and Fine-Tuning. Prompt-based methods guide LLMs using task-specific instructions, optionally including in-context examples or reasoning steps. Zero-Shot and Few-Shot methods simulate real-world scenarios by testing LLMs with minimal input data. CoT approaches focus on assessing the logical coherence of the generated outputs, while fine-tuning trains models on domain-specific pairs to improve structure accuracy and reduce prompt dependency.

Zero-Shot Method: Provides the model with a plain task instruction without examples. It tests the model's ability to generalize based on pre-trained knowledge.

Zero-Shot Prompt

Input: Generate the DOT language to present the Business Process Model and Notation (BPMN) based on the given process textual description: ⟨*Process_Textual_Description*⟩
Output:

Few-Shot Method: Extends Zero-Shot with a few examples (each consist of a process description and its corresponding DOT representation) to help the model learn task-specific patterns.

Few-Shot Prompt

Input: Generate the DOT language to present the Business Process Model and Notation (BPMN) based on the given process textual description: ⟨*Process_Textual_Description_Example_01*⟩
Output: ⟨*Generated_DOT_Language_Example_01*⟩
Input: Generate the DOT language to present the Business Process Model and Notation (BPMN) based on the given process textual description: ⟨*Process_Textual_Description_Example_02*⟩
Output: ⟨*Generated_DOT_Language_Example_02*⟩
Input: Generate the DOT language to present the Business Process Model and Notation (BPMN) based on the given process textual description: ⟨*Process_Textual_Description*⟩
Output:

Zero-Shot CoT Method: Zero-Shot Chain-of-Thought (CoT) prompting is a simple yet effective technique where the prompt is augmented with a phrase like "Let's think step by step." to encourage large language models to generate intermediate reasoning steps before arriving at the final answer.

Zero-Shot CoT Prompt

Input: Generate the DOT language to present the Business Process Model and Notation (BPMN) based on the given process textual description: ⟨*Process_Textual_Description*⟩. Please think about it step by step.
Output: OK! Let's think step by step ...

[1] https://graphviz.org/doc/info/lang.html

Few-Shot CoT Method: Drawing inspiration from the Chain-of-Thought approach (Wei et al., 2022), we construct coherent reasoning traces (illustrated in Appendix A.1). Combines in-context examples with CoT traces to guide stepwise reasoning and enhance accuracy in complex process modeling.

Few-Shot CoT Prompt

Input: Generate the DOT language to present the Business Process Model and Notation (BPMN) based on the given process textual description: ⟨*Process_Textual_Description_Example_01*⟩
Output:
⟨*Steps_of_CoT*⟩
⟨*Generated_DOT_Language_Example_01*⟩
Input: Generate the DOT language to present the Business Process Model and Notation (BPMN) based on the given process textual description: ⟨*Process_Textual_Description_Example_02*⟩
Output:
⟨*Steps_of_CoT*⟩
⟨*Generated_DOT_Language_Example_02*⟩
Input: Generate the DOT language to present the Business Process Model and Notation (BPMN) based on the given process textual description: ⟨*Process_Textual_Description*⟩.
Output:

Fine-Tuned LLMs: Fine-tuning enables large language models (LLMs) to more effectively perform domain-specific tasks, such as generating BPMN representations in DOT Language. We utilize GPT-3.5-turbo-0613[2] as the basic model. The fine-tuning process involves the following steps: (1) **Dataset Preparation**: Collect diverse textual descriptions of business processes and their corresponding BPMN diagrams in DOT Language. (2) **Training Example Creation**: Format the data (shown in the textbox) as training examples according to OpenAI's fine-tuning schema. (3) **Model Training**: Fine-tune the base model using the uploaded dataset. (4) **Progress Monitoring**: Evaluate training performance and adjust the data or hyperparameters as needed to improve learning. (5) **Model Evaluation**: Assess the model's generalization using a held-out validation set. (6) **Model Deployment**: Integrate the fine-tuned model into the target application or system.

{"**messages**":[{"**role**": "system", "**content**": "Your position in a company is to visualize a business process using a graphical representation of a given piece of text in BPMN. Your response should use DOT language to do the graphical representation."}, {"**role**": "user", "**content**": "⟨*Process_Textual_Description*⟩"}, {"**role**": "assistant", "**content**": "⟨*Generated_DOT_Language*⟩"}]}

Together, the five methods offer complementary strategies for BPMN generation. Prompt-based approaches simulate task behavior with or without examples, while fine-tuning embeds domain knowledge to improve consistency. This reduces reliance on expert-crafted rules and lowers the barrier for

[2] https://platform.openai.com/docs/models/continuous-model-upgrades

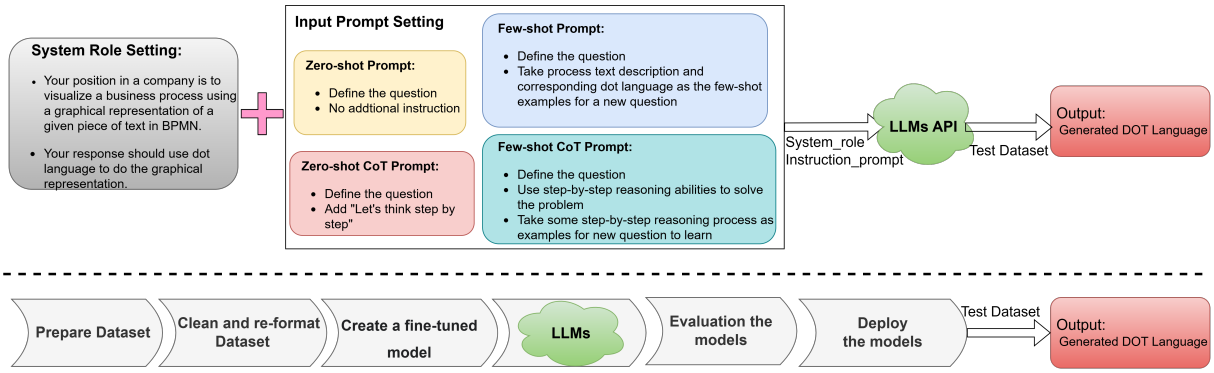

Figure 3: The method framework for generating BPMN diagrams in DOT language from process text descriptions. The first half shows the use of four different prompt engineering methods to guide LLMs to generate a specified DOT description language as an intermediate component for generating BPMN diagrams. The second half is a detailed workflow description of fine-tuning LLMs for this specific generation task.

business process automation.

5 Experiments

This section outlines the experimental procedure for applying the five proposed LLM-based generation methods. Each method is used to convert natural language descriptions of business processes into structured DOT representations, using the MaD dataset (Li et al., 2023) as the evaluation benchmark. The resulting outputs form the basis for subsequent analysis.

Dataset: The MaD dataset includes 30,000 text–BPMN pairs across 15 business categories (Table 1). Each category has 1,600 training, 200 validation, and 200 test samples.

Abbreviation	Full Name
APP	Accounts Payable Process
ARP	Accounts Receivable Process
BPP	Budget Preparation Process
CRPP	Churn Rate Prevention Process
COPMA	Client Onboarding Process for Marketing Agency
CPP	Content Promotion Process
CSPTM	Customer Support Process for the Ticket Management
EOP	Employee Onboarding Process
FGSP	Final Grades Submission Process
LAP	Loan Application Process
OFP	Order Fulfillment Process
POP	Process for Optimizing a Process
PMP	Project Management Process
POW	Purchase Order Workflow
SDDVC	Startup Due Diligence for Venture Capitalist

Table 1: The business process categories in MaD dataset

LLM Utilization and Experiment Setting: We use OpenAI's GPT-3.5-turbo-0613[3] via API to implement all five strategies. Prompt-based methods use the Chat Completion API[4], with a consistent system role guiding DOT generation from text. The system role is defined as follows: "*Your position in a company is to visualize a business process using a graphical representation of a given piece of text in BPMN. Your response should use DOT language to create the graphical representation.*". Fine-tuning is performed on MaD dataset with $n_{epochs} = 3$, $learning\_rate = 0.001$, and $batch\_size = 20$, followed by testing on a 200-sample set per category.

6 Evaluation

We evaluate the DOT-based BPMN diagrams generated by each LLM method using both automatic and human-involved metrics, focusing on syntactic validity, structural fidelity, and semantic alignment.

6.1 Syntax Error Check

To evaluate LLMs' ability to generate DOT language representations, it is essential to examine their syntactic accuracy, which may result in compilation failures or invalid diagram rendering. Tools like Graphviz Online can help identify syntax errors and validate the correctness of the generated DOT language[5]. We analyze 3,000 BPMN diagrams generated by each method to assess syntax validity, and the DOT language compilation error rates are shown in Figure 2. Both Zero-Shot and Few-Shot methods exhibited modest syntax error rates, with Few-Shot prompting significantly reducing errors compared to Zero-Shot. Providing in-context examples improved output reliability, particularly when combined with Chain-of-Thought prompting. The fine-tuned model achieved perfect syntactic validity, generating DOT representations

[3] https://platform.openai.com/docs/models/continuous-model-upgrades

[4] https://platform.openai.com/docs/guides/text-generation/chat-completions-api

[5] https://dreampuf.github.io/GraphvizOnline/

without any compilation errors. One of the common compilation errors is missing a semicolon after defining an edge or node, such as in '$A \rightarrow B$" instead of $A \rightarrow B$;".

Methods	Zero_shot	Zero_shot CoT	Few_shot	Few_shot CoT	Fine_tuned GPT-3.5 Turbo
Syntax Error Rate	6.1%	3.7%	1.7%	0.43%	0

Table 2: Syntax error rate of different methods

6.2 Generated DOT Language Evaluation

For each process textual description within the MaD test dataset, there are DOT string of standard BPMN models as well as five additional corresponding BPMN models represented in DOT langauge generated using proposed methods. To assess the effectiveness of these methods, we can evaluate the similarity between each generated BPMN diagram and its standard counterpart. The evaluation involves two main aspects: **Node Check**: identifying key BPMN elements (presented in Figure 1), such as events, activities, and gateways (AND, OR, XOR), which must be correctly labeled in the DOT representation to ensure clear communication and consistency; **Graph Check**: analyzing the sequence flow between BPMN elements to confirm it aligns with the intended sequence of operations, thus maintaining coherence of the business process. We parse the DOT string into a graph representation comprising nodes (BPMN elements) and edges (sequence flows), facilitating a detailed analysis. For example, as illustrated in Figure 2, the parsed DOT language appears as follows:

{**Node_list** = {start, send the contract to the client, update client's CRM, and_split-1, assign work to the team, inform client of project start, and_join-1, send a post-onboarding survey, end}; **Edge_list** = {(start, send the contract to the client), (send the contract to the client, update client's CRM), (update client's CRM, and_split-1), (and_split-1, assign work to the team), (and_split-1, inform client of project start), (assign work to the team, and_join-1), (inform client of project start, and_join-1),(and_join-1, send a post-onboarding survey), (send a post-onboarding survey, end)}}

Node Similarity Check: To compute the similarity between two sets of labeled nodes, we use a method based on semantic analysis. We begin by encoding the textual labels of each node using the *sentence-transformers/paraphrase-MiniLM-L6-v2* model from HuggingFace[6], which generates embeddings capturing their semantic representations. Next, we compute the cosine similarity between the embedding of corresponding labels to quantify their semantic resemblance. We also establish a similarity matrix to systematically compare each node from one graph to every node in the other. This matrix encapsulates the similarity scores computed for each pair of nodes. By evaluating both rows and columns, we identify the maximum similarity scores for each node in both graphs. Aggregating these maximum similarities allows us to compute the average similarity between the two sets of nodes. This holistic approach enables a comprehensive assessment of the semantic correspondence between the labeled nodes.

Graph Similarity Check: To assess the structural similarity between graphs, the node2vec (Grover and Leskovec, 2016) algorithm is employed. Node2vec is an algorithm for generating graph embedding that map nodes in a graph to low-dimensional vector spaces while preserving structural information between nodes. By performing random walks on the graph and employing a Word2Vec-like approach to learn vector representations of nodes, node2vec can capture structural information between nodes. When comparing the structural similarity of two graphs, we can use node2vec to generate graph-level embedding for each graph and then use cosine similarity or other similarity measures to compare these embeddings. This method helps us determine whether the structures of two graphs are similar, beyond just the quantity or connectivity of their nodes and edges.

Results Analysis: Table 3 highlights the performance of various methods using ANSS and AGSS. Fine-tuned GPT-3.5 Turbo excels, achieving a perfect ANSS score of 1.000 and high AGSS scores, with only the CRPP category scoring 0.898 in AGSS due to its complexity. CoT reasoning significantly boosts performance, especially in the CRPP category, where Zero-Shot CoT improved ANSS by 2.3% and AGSS by 1.1%. In the APP category, Few-Shot CoT increased ANSS by 3.8% and AGSS by 1.7%. Few-Shot methods generally outperform Zero-Shot methods, with notable improvements seen in categories like LAP. These results emphasize the effectiveness of fine-tuning and CoT reasoning in enhancing model performance.

While Table 4 assesses the performance of various methods in generating BPMNs from textual descriptions, particularly examining the effects of gateways and nested gateways using ANSS and AGSS metrics. ANSS consistently shows high node similarity across scenarios, while AGSS exhibits notable variations. Generally, the presence of gateways tends to reduce AGSS, with instances

[6]https://huggingface.co/sentence-transformers/paraphrase-MiniLM-L6-v2

Categories	Zero-shot		Zero-shot CoT		Few-shot		Few-shot CoT		Fine-tuned GPT-3.5 Turbo	
	ANSS	AGSS	ANSS	AGSS	ANSS	AGSS	ANSS	AGSS	ANSS	AGSS
APP	0.764	0.760	0.777	0.808	0.928	0.925	0.966	0.941	**1.000**	**0.977**
ARP	0.725	0.838	0.739	0.830	0.927	0.924	0.972	0.934	**1.000**	**0.961**
BPP	0.719	0.828	0.708	0.834	0.906	0.872	0.975	0.892	**1.000**	**0.916**
CRPP	0.707	0.822	0.723	0.831	0.907	0.863	0.971	0.882	**1.000**	**0.898**
COPMA	0.692	0.780	0.696	0.783	0.931	0.932	0.970	0.931	**1.000**	**0.970**
CPP	0.686	0.700	0.779	0.725	0.972	0.833	0.953	0.846	**1.000**	**0.896**
CSPTM	0.366	0.782	0.744	0.749	0.926	0.905	0.939	0.915	**0.999**	**0.957**
EOP	0.723	0.808	0.715	0.825	0.912	0.901	0.973	0.921	**1.000**	**0.951**
FGSP	0.732	0.806	0.724	0.817	0.908	0.917	0.959	0.932	**0.996**	**0.961**
LAP	0.731	0.832	0.733	0.848	0.918	0.920	0.945	0.932	**1.000**	**0.959**
OFP	0.730	0.836	0.733	0.834	0.928	0.932	0.957	0.931	**1.000**	**0.966**
POP	0.698	0.862	0.694	0.870	0.916	0.901	0.972	0.919	**1.000**	**0.937**
PMP	0.723	0.826	0.708	0.851	0.911	0.915	0.958	0.934	**1.000**	**0.957**
POW	0.754	0.829	0.754	0.834	0.946	0.938	0.976	0.943	**1.000**	**0.930**
SDDVC	0.753	0.788	0.740	0.809	0.905	0.877	0.974	0.901	**1.000**	**0.927**
Average	**0.700**	**0.806**	**0.731**	**0.816**	**0.923**	**0.904**	**0.964**	**0.917**	1.000	0.944

Table 3: Performance Comparison of Various Methods on Different Categories Using Average Node Similarity Score (ANSS) and Average Graph Similarity Score (AGSS)

Types	Num	Zero-Shot		Zero-Shot CoT		Few-Shot		Few-Shot CoT		Fine-tuned GPT-3.5 Turbo	
		ANSS	AGSS	ANSS	AGSS	ANSS	AGSS	ANSS	AGSS	ANSS	AGSS
nogateway	387	0.692	**0.841**	0.824	**0.848**	0.976	**0.935**	0.950	**0.939**	1.000	**0.965**
gateway	2613	0.701	0.801	0.717	0.812	0.915	0.899	0.966	0.914	1.000	0.941
gateway=1	805	0.699	**0.816**	0.739	**0.823**	0.932	**0.921**	0.960	**0.930**	1.000	**0.956**
gateway=2	748	0.699	0.810	0.714	0.820	0.916	0.920	0.965	0.930	1.000	0.950
gateway=3	432	0.702	0.799	0.701	0.805	0.904	0.901	0.970	0.922	1.000	0.953
gateway=4	152	0.701	0.782	0.689	0.800	0.901	0.848	0.975	0.866	1.000	0.893
gateway=5	36	0.685	0.746	0.670	0.767	0.909	0.802	0.973	0.812	1.000	0.826
gateway=6	2	0.630	0.634	0.602	0.650	0.938	0.742	0.983	0.778	1.000	0.781
nest=0	2175	0.700	**0.807**	0.718	**0.816**	0.918	**0.910**	0.965	**0.922**	1.000	**0.947**
nest=1	414	0.708	0.775	0.715	0.792	0.898	0.849	0.972	0.877	1.000	0.918
nest=2	24	0.724	0.743	0.701	0.761	0.898	0.789	0.975	0.808	1.000	0.841

Table 4: Impact of gateways and nested gateways on Average Node Similarity Score (ANSS) and Average Graph Similarity Score (AGSS)

without gateways achieving higher scores. For graphs with fewer gateways, AGSS remains relatively stable, but it decreases significantly as the number of gateways increases. Similarly, increased nesting levels also lead to a decline in AGSS. These findings indicate that complex graph structures, such as higher gateway counts and nested gateways, pose challenges in maintaining graph similarity when generating BPMNs from textual descriptions.

6.3 Human-involved Evaluation

We also conducted a human evaluation using a web-based survey on Qualtrics[7]. Participants with demonstrated proficiency in both business process management and English were asked to evaluate BPMN diagrams generated by five methods against 15 randomly selected business process descriptions. Criteria included evaluating activity representation and relationships between activities using a scale from 1 to 5 (1: Very poor; 2: Poor; 3: Neutral; 4: Good; 5: Very good). See questionnaire details in Appendix A.2. In total, 32 participants submitted valid responses, covering 197 diagram & description pairs. As shown in Table 5, the results indicate clear differences across methods. The Zero-Shot method received the lowest average score (1.89), while the fine-tuned GPT-3.5 Turbo achieved the highest (4.85). Notably, Few-Shot CoT showed significant improvement over Few-Shot, indicating the positive impact of CoT method on the quality of BPMN generation.

[7] https://auckland.au1.qualtrics.com/jfe/form/SV_broysxDbppmJPo0

Methods	Zero_shot	Zero_shot CoT	Few_shot	Few_shot CoT	Fine_tuned GPT-3.5 Turbo
Average Score	1.89	2.82	3.63	4.11	4.85

Table 5: Average human evaluation score

6.4 Effectiveness Evaluation under Masked Information

While previous evaluations assume access to complete and well-structured process descriptions, real-world scenarios often involve incomplete or under-specified inputs. To further assess the robustness and inferential capabilities of LLM-based methods, we introduce a masked-input setting in which key information is intentionally removed from the input. This setting allows us to evaluate how well each method performs when required to generate BPMN models from partial descriptions.

We simulate information incompleteness by applying a masking strategy to the MaD test dataset. First, we identify all activity phrases within a process description. Then, 20% of these activities are randomly selected and replaced with the placeholder token "******" (See Appendix A.3 for an example). If the selected percentage does not yield an integer, it is rounded up to ensure that at least one activity is masked. The masking process is applied uniformly across all test samples.

Once the masked descriptions are prepared, we apply each of the five generation methods—Zero-Shot, Zero-Shot CoT, Few-Shot, Few-Shot CoT, and Fine-Tuned GPT-3.5 Turbo—without any additional instructions or adaptations. The generated DOT representations are then evaluated against their complete-reference counterparts using the same graph-based metrics introduced earlier: Average Node Similarity Score (ANSS) and Average Graph Similarity Score (AGSS). Table 6 presents the comparison between non-masked and masked scenarios.

Masked-input scenarios reveal substantial differences in model robustness. Zero-Shot and Zero-Shot CoT methods exhibit the largest performance drops, with AGSS declining by 9.6% and 7.1%, respectively. This suggests that these approaches rely heavily on the presence of complete contextual cues. In contrast, Few-Shot CoT demonstrates strong resilience to missing information, with AGSS dropping by only 2.8%. The fine-tuned model also maintains relatively high performance, though it exhibits a moderate decline in both node and graph similarity scores. Overall, Few-Shot CoT consistently achieves the best balance between performance and robustness, demonstrating its suitability for real-world applications involving incomplete or under-specified business process descriptions.

7 Conclusion

This paper presented a systematic investigation into the capabilities of large language models (LLMs) for generating business process models from natural language descriptions. We framed the task as structured text-to-graph generation using DOT language as an intermediate representation and evaluated five methods—Zero-Shot, Zero-Shot CoT, Few-Shot, Few-Shot CoT, and Fine-Tuning—on the MaD dataset. Fine-tuned models achieved the best overall performance, but Few-Shot CoT emerged as a robust alternative, especially under complex control-flow and incomplete input conditions. Human assessments further confirmed the benefits of CoT-based prompting in preserving logical consistency. Our findings demonstrate the practical value of LLMs in BPM automation and offer guidance for deploying prompt-based or fine-tuned strategies in enterprise settings. Future work may explore hybrid symbolic-LLM frameworks, real-time modeling interfaces, or extensions for multilingual and multimodal inputs.

Limitations

While our study provides a comprehensive evaluation of LLM-based methods for process model generation, several limitations remain. First, the use of DOT language as an intermediate format may not capture all BPMN-specific semantics, such as execution semantics or exception handling. Second, although we evaluated robustness using masked inputs, real-world ambiguity and user intent variability were only partially simulated. Third, fine-tuning was conducted on a single model (GPT-3.5-turbo), and generalization across other architectures was not explored. Finally, our evaluation focused on textual inputs in English; future work should assess multilingual capabilities.

Acknowledgments

This work was supported by Uxtrata Limited and the Callaghan Innovation R&D Fellowship Grants, New Zealand.

Scenarios	Zero-Shot		Zero-Shot CoT		Few-Shot		Few-Shot CoT		Fine-tuned GPT-3.5 Turbo	
	ANSS	AGSS	ANSS	AGSS	ANSS	AGSS	ANSS	AGSS	ANSS	AGSS
Masked Average	0.621	0.710	0.651	0.745	0.857	0.874	0.935	0.889	0.932	0.911
Non-masked Average	**0.700**	**0.806**	**0.731**	**0.816**	**0.923**	**0.904**	**0.964**	**0.917**	**1.000**	**0.944**
Difference	0.079	0.096	0.080	0.071	0.066	0.030	0.029	0.028	0.068	0.033

Table 6: Performance comparison of non-masked and masked process textual descriptions Using Average Node Similarity Score (ANSS) and Average Graph Similarity Score (AGSS)

References

Farhath Ajmal, Poorna Wijekoon, Haritha Dhanamina, Yasiru Ravishan, Dasuni Nawinna, and Buddhima Attanayaka. 2024. Automated bpmn diagram generation. In *2024 6th International Conference on Advancements in Computing (ICAC)*, pages 7–12. IEEE.

Patrizio Bellan, Mauro Dragoni, and Chiara Ghidini. 2022. Extracting business process entities and relations from text using pre-trained language models and in-context learning. In *International Conference on Enterprise Design, Operations, and Computing*. Springer.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Luis Delicado Alcántara, Josep Sánchez Ferreres, Josep Carmona Vargas, and Lluís Padró. 2017. Nlp4bpm: Natural language processing tools for business process management. In *BPM Demo and Industrial Track*.

Renato César Borges Ferreira, Lucinéia Heloisa Thom, and Marcelo Fantinato. 2017. A semi-automatic approach to identify business process elements in natural language texts. In *ICEIS (3)*, pages 250–261.

Fabian Friedrich, Jan Mendling, and Frank Puhlmann. 2011. Process model generation from natural language text. In *CAiSE*. Springer.

Michael Grohs, Luka Abb, Nourhan Elsayed, and Jana-Rebecca Rehse. 2023. Large language models can accomplish business process management tasks. In *International Conference on Business Process Management*, pages 453–465. Springer.

Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *ACM SIGKDD*.

Xue Han, Lianxue Hu, Lijun Mei, Yabin Dang, Shivali Agarwal, Xin Zhou, and Pengwei Hu. 2020. A-bps: Automatic business process discovery service using ordered neurons lstm. In *ICWS*.

Ana Ivanchikj, Souhaila Serbout, and Cesare Pautasso. 2020. From text to visual bpmn process models: Design and evaluation. In *Proceedings of MODELS*, pages 229–239.

John Jeston. 2014. *Business process management: practical guidelines to successful implementations*. Routledge.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *NeurIPS*.

Humam Kourani, Alessandro Berti, Daniel Schuster, and Wil MP van der Aalst. 2024. Process modeling with large language models. In *International Conference on Business Process Modeling, Development and Support*, pages 229–244. Springer.

Xiaoxuan Li, Lin Ni, Renee Li, Jiamou Liu, and Mengxiao Zhang. 2023. Mad: A dataset for interview-based bpm in business process management. In *IJCNN*. IEEE.

Robert L Logan IV, Ivana Balažević, Eric Wallace, Fabio Petroni, Sameer Singh, and Sebastian Riedel. 2021. Cutting down on prompts and parameters: Simple few-shot learning with language models. *arXiv preprint arXiv:2106.13353*.

Julian Neuberger, Lars Ackermann, Han van der Aa, and Stefan Jablonski. 2024. A universal prompting strategy for extracting process model information from natural language text using large language models. In *International Conference on Conceptual Modeling*, pages 38–55. Springer.

Quentin Nivon and Gwen Salaün. 2024. Automated generation of bpmn processes from textual requirements. In *International Conference on Service-Oriented Computing*, pages 185–201. Springer.

Chen Qian, Lijie Wen, Akhil Kumar, Leilei Lin, Li Lin, Zan Zong, Shu'ang Li, and Jianmin Wang. 2020. An approach for process model extraction by multi-grained text classification. In *International Conference on Advanced Information Systems Engineering*, pages 268–282. Springer.

Hajo A Reijers, Selma Limam, and Wil MP Van Der Aalst. 2003. Product-based workflow design. *Journal of management information systems*.

Pol Schumacher and Mirjam Minor. 2014. Extracting control-flow from text. In *Proceedings of the IEEE IRI*.

Sholiq Sholiq, Riyanarto Sarno, and Endang Siti Astuti. 2022. Generating bpmn diagram from textual requirements. *Journal of King Saud University-Computer and Information Sciences*.

Riad Sonbol, Ghaida Rebdawi, and Nada Ghneim. 2023. A machine translation like approach to generate business process model from textual description. *SN Computer Science*.

Víctor Suárez-Paniagua, Renzo M Rivera Zavala, Isabel Segura-Bedmar, and Paloma Martínez. 2019. A two-stage deep learning approach for extracting entities and relationships from medical texts. *Journal of biomedical informatics*, 99:103285.

Han Van der Aa, Josep Carmona Vargas, Henrik Leopold, Jan Mendling, and Lluís Padró. 2018. Challenges and opportunities of applying natural language processing in business process management. In *COLING 2018: The 27th International Conference on Computational Linguistics: Proceedings of the Conference: August 20-26, 2018 Santa Fe, New Mexico, USA*, pages 2791–2801. Association for Computational Linguistics.

Han van der Aa, Claudio Di Ciccio, Henrik Leopold, and Hajo A Reijers. 2019. Extracting declarative process models from natural language. In *Advanced Information Systems Engineering: 31st International Conference, CAiSE 2019, Rome, Italy, June 3–7, 2019, Proceedings 31*, pages 365–382. Springer.

Mark von Rosing, Stephen White, Fred Cummins, and Henk de Man. 2015. Business process model and notation-bpmn.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *NeuralIPS*.

Junjie Ye, Xuanting Chen, Nuo Xu, Can Zu, Zekai Shao, Shichun Liu, Yuhan Cui, Zeyang Zhou, Chao Gong, Yang Shen, and 1 others. 2023. A comprehensive capability analysis of gpt-3 and gpt-3.5 series models. *arXiv preprint arXiv:2303.10420*.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2022. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493*.

A Appendix

A.1 Steps of Chain_of_thought (CoT)

To improve the reasoning capability of LLMs, we employ Chain-of-Thought (CoT) prompting to guide the model in step-by-step generation. The following textbox outlines the structured steps used in our CoT strategy to transform business process descriptions into DOT language representations.

Steps of Chain_of_Thought (CoT)

Following the steps to answer the question:
STEP1: Claim the task: In order to generate the dot language for the given text, we should generate the subgraph for each sentence.
STEP2: Before start, identify the condition type in each sentence. There are three types of conditions in the subprocess:
a. If the sentence contains the phrases like "the process is split into" and "should be done in parallel", it is the **AND** condition. For this condition, there are two special nodes in the subgraph: "AND_SPLIT" and "AND_JOIN".
b. If the sentence contains a phrase like "one or more of the following paths", it is **OR** condition. For this condition, there are two special nodes in the subgraph: "OR_SPLIT" and "OR_JOIN".
c. If the sentence contains phrases like "should be considered" and "should be taken into account", it is **XOR** condition. For this condition, there are two special nodes in the graph: "XOR_SPLIT" and "XOR_JOIN".
STEP3: Generate the subgraph for each sentence. Based on the given text, there are $\langle i \rangle$ sentences.
Sentence_1: The process starts with the start node, and then the process goes to $\langle activity \rangle$. So the subgraph is:
subgraph sentence_1 $\{start \rightarrow \langle activity \rangle;\}$
.....
Sentence_i: The process is completed. The previous node is the last node in sentence_i-1. So the subgraph is:
subgraph sentene_i $\{\langle activity \rangle \rightarrow end;\}$
STEP4: Join all the subgraphs together. The generated dot language is:
$\langle Generated\_DOT\_Language \rangle$

A.2 Questionnaire

The following textbox outlines the purpose and scope of the user study conducted to evaluate the quality of BPMN diagrams generated by different LLM-based methods, as presented to the participants.

The purpose of our survey is to investigate the ability of Large Language Models (LLMs) to generate Business Process Model and Notation (BPMN) diagrams from business process textual descriptions. You will be presented with 15 business process textual descriptions. For each, it consist of five generated BPMN diagrams based on different methods, and you should rate these five generated BPMNs.

The second textbox outlines the evaluation criteria for the BPMN diagrams.

Please provide a holistic evaluation of the generated BPMN diagrams. You may base your evaluation on the following criteria:
a. Activity Representation: Assess whether the activities in the generated BPMN diagrams perfectly capture those in business process textual descriptions. Determine if each activity is accurately represented and described in the generated BPMN.
b. Relationship Between Activities: Evaluate whether the relationships between various activities are correctly depicted in the generated BPMN diagrams. Verify if the sequence flows and connections between activities accurately reflect the flow and dependencies described in business process textual descriptions.

The following textbox explains the 1-to-5 rating scale used to assess the quality of the BPMN diagrams.

For scoring, you can use a scale from 1 to 5:
1: Very poor - The generated BPMN diagrams poorly represent the activities and relationships in the business process textual descriptions.
2: Poor - The representation of activities and relationships in the generated BPMN diagrams is inadequate in the business process textual descriptions.
3: Neutral - The generated BPMN diagrams partially capture the activities and relationships described in the business process textual descriptions.
4: Good - The generated BPMN diagrams adequately represent the activities and relationships, but some improvements could be made for better alignment with the business process textual descriptions.
5: Very good - The generated BPMN diagrams perfectly capture the activities and relationships described in the business process textual descriptions, and they can effectively replace the corresponding business process textual descriptions.

A.3 An example of masked business process text description

The following description is about the account payable process. It starts with ******. After ******, creating a receiving report should be done.
When creating a receiving report is completed, one or more of the following paths should be executed: ******, reporting errors to relevant team member.
After reporting errors to relevant team member, double-checking a three-way match needs to be done.
After that, you need to enter the invoice into the accounts payable account.
The process is now completed.