2025 5th International Conference on Neural Networks, Information and Communication Engineering (NNICE) | 979-8-3315-0796-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/NNICE64954.2025.11064395

Complex Flowchart Text Generation Based on Large Language Models

Yuchen Fang
Artificial Intelligence Institute of China Electronics Technology Group Corporation
Beijing, China
e-mail: bob1301@163.com

Zhiqiang Fan*
Artificial Intelligence Institute of China Electronics Technology Group Corporation
Beijing, China
*Corresponding author: zhiqiang.fan@buaa.edu.cn

***Abstract*—As a common business process modeling tool, flowcharts are widely used in software engineering, business process management, architecture framework modeling, and other fields. However, with the increase of business complexity, the scale and complexity of flowcharts also increase, which brings challenges to the understanding and text description of flowcharts. To solve this problem, this paper proposes a text generation method for complex process diagrams based on large language model (LLM), which realizes high-quality text description of complex process diagrams through RPST (Refined Process Structure Tree) decomposition and Least-to-Most prompting strategy.**

Keywords-Graph decomposition, Machine Learning, Text Generation

I. INTRODUCTION

In the development process and application process of architecture tools, there exists a large amount of text processing work, such as transforming the design scheme into architecture model and deriving the design report from the architecture model. Due to its large workload, strong specialization and high writing threshold, there is a large amount of human resources wasted in the process of document processing.

The architecture model itself belongs to the category of structured data, and its content model can be simply summarized into several categories such as tree diagrams, architecture diagrams, flowcharts, and forms. The structure of tree diagrams and architecture diagrams is relatively simple, and LLMs can easily convert them into textual information, supplemented by reasonable prompts without errors or illusions. However, the structure of flowcharts is relatively complex, and it is easy to lose information or make mistakes in relationship matching by directly utilizing LLMs to generate corresponding text descriptions.

Currently, investigations on flowcharts focus on the following aspects: recognition of handwritten flowcharts[1]; generation of corresponding text descriptions from flowchart images using LLMs, such as TextFlow proposed by Ye et al.[2]; use of generative AI to improve the parsing and understanding of flowcharts, such as GenFlowChart proposed by Arbaz et al.[3]; or opposite, generate from natural language descriptions to flowcharts[4], [5]. Research related to the direct generation of text from flowchart model data is still in the pre-LLM era, e.g., MeZiane proposes a method for generating natural language specification descriptions from UML diagrams[6], and Dalianis proposed a method to generate natural language descriptions from conceptual models[7], etc. All of these methods rely on transformed templates, and the generated text is more rigid and not as flexible as the text generated by LLMs.

Text generation for flowchart descriptions based on LLMs generally uses Zero-Shot and One-Shot prompting methods.

Zero-Shot[8] involves providing no samples at all, only natural language instructions describing the task, guiding LLMs to complete the specified task. Due to the inherent limitations of natural language, it is difficult to fully describe all task requirements in a brief prompt. Figure 1 shows the lacked information when using Zero-Shot Prompt. As a result, the LLMs may struggle to understand and generate the desired result directly.

Zero-Shot Prompt:

Describe this flowchart into a paragraph:

graph TD
A[Start Job Evaluation Process] --> B{Is Job Analysis Complete?}
B -- Yes --> C[Prepare Job Descriptions]
B -- No --> D[Conduct Job Analysis]
D --> C
C --> E{Evaluate Job Worth}
E -- Using Evaluation Method --> F[Rank Jobs by Value]
E -- If Method Requires --> G[Assign Points for Job Factors]
F --> H[Create Job Hierarchy]
G --> H
H --> I[Review and Validate Results]
I --> J{Are Results Acceptable?}
J -- Yes --> K[Implement Job Hierarchy]
J -- No --> L[Adjust Evaluation Process]
L --> E
K --> M[End Job Evaluation Process]

Zero-Shot Answer:

This flowchart outlines the job evaluation process, starting with the initiation of the evaluation. If the job analysis is complete, job descriptions are prepared, and the job's worth is evaluated using appropriate methods. Depending on the method, jobs are either ranked by value or assigned points for specific job factors. A job hierarchy is then created based on the evaluation results, which are reviewed and validated. If the results are acceptable, the job hierarchy is implemented, and the process ends. If not, readjust the evaluation process.

Figure 1. An Example about Zero-Shot prompt. Information lacked in the answer has been marked as italic

One-Shot[9] only allows one sample to be provided in addition to the natural language description of the task. The expectation is that LLMs can fully define the task based on this

single sample. One-Shot reduces reliance on a large amount of labeled data, making the single sample instance provided to LLMs particularly important, as it directly influences whether LLMs can extract useful information from the instance and apply it to the specified task. Figure 2 is an example of illusions and errors when using One-Shot Prompt.

One-Shot Prompt:

Describe this mermaid flowchart into a paragraph:

graph TD
A[Start Job Evaluation Process] --> B{Is Job Analysis Complete?}
B -- Yes --> C[Prepare Job Descriptions]
B -- No --> D[Conduct Job Analysis]
D --> C
C --> E{Evaluate Job Worth}
E -- Using Evaluation Method --> F[Rank Jobs by Value]
E -- If Method Requires --> G[Assign Points for Job Factors]
F --> H[Create Job Hierarchy]
G --> H
H --> I[Review and Validate Results]
I --> J{Are Results Acceptable?}
J -- Yes --> K[Implement Job Hierarchy]
J -- No --> L[Adjust Evaluation Process]
L --> E
K --> M[End Job Evaluation Process]

For example, input is like:

flowchart TD
A[Start] --> B{Is the item in stock?}
B -- Yes --> C[Add item to cart]
C --> D[Proceed to checkout]
B -- No --> E[Notify customer]
E --> F[End]
D --> F

And the output is like: This Mermaid flowchart illustrates a decision-making process for an online shopping system. It begins with 'Start', followed by a decision point 'Is the item in stock?'. If yes, the flow continues to 'Add item to cart', then to 'Proceed to checkout'. If no, it moves to 'Notify customer' and ends. The process provides a simple and logical flow for handling stock availability in an e-commerce platform.

One-Shot Answer:

This mermaid flowchart illustrates a decision-making process for an *online job evaluation system*. It begins with a starting point labeled "Start Job Evaluation Process", followed by a decision point "Is Job Analysis Complete?" which is followed by two possible actions: "Prepare Job Descriptions" and "Conduct Job Analysis". *If the job analysis is complete, the flow continues to "Prepare Job Descriptions", then to "Conduct Job Analysis". If the job analysis is not complete, the flow moves to "Conduct Job Analysis".* After the job descriptions are prepared, the next step is to "Evaluate Job Worth", which can be done either by ranking jobs by value or, if the evaluation method requires, by assigning points for job factors. Both paths converge at "Create Job Hierarchy". Following this, the results are reviewed and validated. If the results are acceptable, the job hierarchy is implemented, and the process ends. If the results are not acceptable, the evaluation process is adjusted, and the evaluation step is revisited.

Figure 2. An Example about One-Shot prompt. Illusions and errors in the answer has been marked as italic

Few-Shot is also a popular prompting method, needs at least 10-100 examples[9] to generate a better result. For users, obtaining these data is a task with a large workload, making it difficult to complete in a short amount of time. Additionally, the accuracy of the provided samples cannot be guaranteed. Although the output results are better compared to Zero-Shot and One-Shot, it leads to additional waste of human and time resources, making it ultimately not cost-effective.

Therefore, this paper proposes to start from the diagram properties of the flowchart itself, first decompose the relatively complex flowchart into small segments that LLMs can better understand, supplemented by prompting engineering, to help LLMs better understand and analyze the flowchart, and finally obtain a more accurate description of the flowchart.

In section 2, we introduce two main algorithms used in this method: RPST decomposition and prompt generating by Least-to-Most or its concepts. In section 3, we test different methods on several LLMs. And the conclusion is given in section 4.

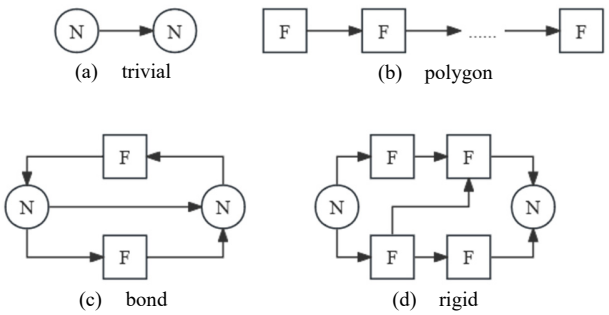
II. Materials And Methods

A. RPST Decomposition

In a flowchart, the structure of the graph reflects the execution path of the process. RPST[10], [11] is a graph decomposition algorithm that decomposes a graph into different types of indivisible process fragments, storing the fragments and their interrelations using a process structure tree. At the same time, the analysis results of the flowchart using the RPST algorithm have the characteristics of uniqueness and modularity, which facilitates further analysis and processing of the flowchart.

RPST includes the following four types of process fragments[12]:

- **Trivial**: A fragment containing only a single edge, representing the basic unit of the flowchart, as Figure 3(a).
- **Polygon**: The largest sequence of fragments with connected endpoints, used to represent a series of activities or operations executed sequentially, as Figure 3(b).
- **Bond**: The largest set of fragments with the same pair of boundary nodes, used to represent activities executed in parallel or activities that are cyclic in nature, as Figure 3(c).
- **Rigid**: Process fragments other than trivial, polygon, and bond types, representing more complex structural parts, as Figure 3(d).



(a) trivial (b) polygon (c) bond (d) rigid

Figure 3. Four types of process fragments. N means a node, and F means a fragment

To generate prompts for complex flowcharts, we do not need to determine the specific type of every process fragment and construct the RPST. Instead, we only need to process the more complex parts of the flowchart. Based on the definitions of the process fragments mentioned above, trivial and polygon are simple fragments that can be described in a streaming manner, while bond and rigid are complex fragments that cannot be described in such a manner. Therefore, we only need to identify all the bond and rigid fragments in the flowchart and apply specific methods to describe them, which will allow us to generate the corresponding prompts. Hereafter, bond and rigid fragments are collectively referred to as complex fragments.

A method for constructing the RPST by building a dominance tree and a post-dominance tree is proposed. This method can identify all complex fragments in order of size, from small to large, and follows an upward direction in the process structure tree. Compared to the RPST defined by triconnected components, this approach is more intuitive and easier to implement.

In a flowchart, a node d dominates node n if and only if every path from the start node to node n passes through node d, denoted as $d \in Dom(n)$. Conversely, a node p post-dominates node n if and only if every path from node n to the end node passes through p, denoted as $p \in Pdom(n)$. Each node dominates and post-dominates itself. Based on these rules, we can construct trees rooted at the start node, or the end node, referred to as the dominance tree and the post-dominance tree, respectively. From this, we can define inverse dominance and inverse post-dominance, where $Dom^{-1}(n)$ is the set of all nodes dominated by node n, and $Pdom^{-1}(n)$ is the set of all nodes post-dominated by node n.

A complex fragment Rg(s, t) starting at node s and ending at node t can be obtained under the following conditions:

1. $n \in Dom^{-1}(s) \cap Pdom^{-1}(t)$
2. n connects only to any other nodes in Rg(s, t)
3. $| Adj(s) \cap Rg(s, t) | \geq 2$ and $| Adj(t) \cap Rg(s, t) | \geqslant 2$
4. $| Adj(s) \cap Rg(s, t) | > | Adj(s) \cap Rg(s, w) |$ or $| Adj(s) \cap Rg(s, t) | > | Adj(s) \cap Rg(s, w) |$ for any Rg(s, w) and Rg(u, t) nested in Rg(s, t)

And the specific algorithm flow is as follows[13]:

ALGORITHM I. RPST DECOMPOSITION

Input: Graph G

Output: All complex fragments Rg

```
Compute dominance tree DT and post-dominance tree PDT
cndEntry = bottom-up ordered DT
cndExit = bottom-up ordered PDT
Rg = {}
for s in cndEntry do
    for t in cndExit do
        IN = Dom^-1(s) ∩ Pdom^-1(t) - {s, t}
        if n ∈ IN doesn't connect only to other nodes in IN ∪ {s, t},
    remove n from IN, repeat until all nodes in IN connect only to nodes
    in IN ∪ {s, t}
        if IN follows condition 3 and condition 4 then
            Rg[(s, t)] = IN ∪ {s, t}
        end if
    end for
end for
return Rg
```

An example of flowgraph and its RPST decomposition is in Figure 4.

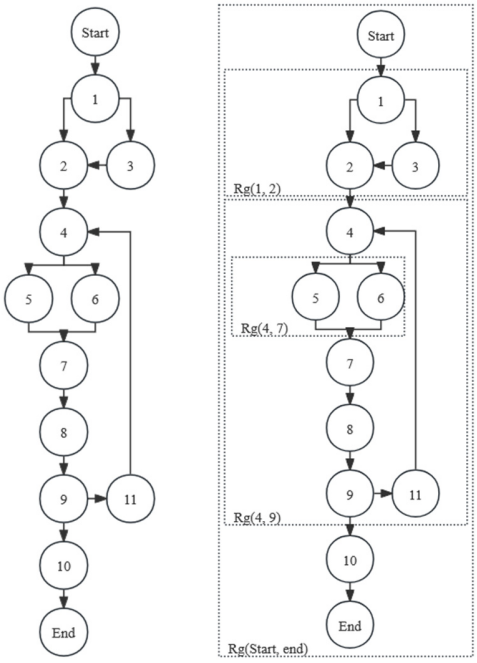


Figure 4. An abstract flowchart and all the complex fragments found in it

B. Generating Prompts for LLM

After identifying all complex fragments, we need to find a method to describe these fragments and generate prompts for LLMs. Since complex fragments may contain loops, traditional prompt generation methods based on depth-first or breadth-first search can easily mislead LLMs, causing it to lose structural or node relationship information. This would undermine the goal of providing an accurate description of the flowchart and reducing the workload.

The description approach we adopt is as follows: Since all complex fragments contain at least one path from the start node to the end node, we can find the longest executable path within the fragment to serve as the main path. Then, the remaining paths are sorted by length and treated as complementary paths for the complex fragment.

The main path begins at the start node and is found by performing a depth-first search (DFS) to reach the end node. The longest path discovered during this search becomes the main path. For the complementary paths, we sort the remaining nodes in the complex fragment by their in-degree and start searching

from nodes with an in-degree of 0. We then search for the farthest node they can reach, which defines one complementary path. This process is repeated until all edges in the complex fragment are assigned to one and only one path.

ALGORITHM II. PATHS OF COMPLEX FRAGMENT

Input: Complex fragment Rg(s, t)
Output: Main path main_path, Complementary paths sub_paths

```
Using DFS to explore all paths from s to t, select the longest on as main_path
sub_graph is a subgraph of Rg(s, t) removing edges in main_path
sub_paths = []
while sub_graph is not empty:
    Select a node n in sub_graph with in-degree 0
    Using DFS to explore the longest path p started at node n
    sub_paths.append(p)
    Remove all the edges of p from sub_graph
end while
return main_path, sub_paths
```

After obtaining all the paths corresponding to the complex fragments, we can construct a prompt for each path using a simple template. For a path (s, t, description, condition), if s is a regular node, the prompt can be directly described as: "s is connected to t by description." If s is a decision branch node, the prompt can be described as: "If s is condition, it is connected to t by description." Once we have the prompts for all the paths, we can concatenate them to form the complete prompt for the entire complex fragment.

While it is possible to directly connect all the remaining edges in the flowchart with the complex fragments and generate the entire flowchart's prompt, we can also apply certain prompt engineering techniques to enhance the LLM's understanding of the flowchart and complex fragments.

The Least-to-Most (LtM)[14] prompting method teaches LLMs how to break down a complex problem into a series of simpler subproblems. It consists of two consecutive processes: task decomposition and subtask solving. In the task decomposition phase, prompts include demonstrations of how tasks are decomposed and the specific problems to be decomposed. In the subtask solving phase, examples are provided to show how each subproblem is solved, including the list of previous subproblems and their corresponding solutions, along with the next questions to be answered.

Unlike LtM, in our case, the task decomposition phase has already been completed using the RPST algorithm, which decomposes the flowchart into a sequence of process fragments. We have obtained the prompt for each fragment, so we can directly proceed to the subtask solving phase. Notice that complex fragments have a natural order from smaller to larger and from partial to whole. Therefore, we directly provide the prompts for the complex fragments to LLMs and let them generate a textual description of each individual complex fragment. Then, we embed the generated descriptions back into the higher-level prompts, repeating this process to generate a complete textual description of the entire flowchart.

III. RESULT AND DISCUSSION

Mermaid flowcharts are one of the general formats for representing flowcharts. They describe all the edges in the flowchart, with each edge specifying the names of the nodes on either side and an annotation for the edge. We selected 112 Mermaid flowcharts from open-source datasets[15] and manually wrote a description for each flowchart to participate in a comparative analysis of the results.

A. Metrics

BERTScore[16] is one of the most commonly used model-based evaluation metrics. It uses contextual token similarity to measure the overlap between the generated text and reference text, applying cosine similarity to match the word vectors of the generated text and reference text. BERTScore considers **Recall** (R), **Precision** (P), and **F1-Score**, making it applicable to a variety of text generation tasks, with a strong correlation to human judgment. BERTScore overcomes the limitations of metrics like BLEU and ROUGE, which focus solely on lexical matching, and it can distinguish the meaning of the same word in different contexts through contextual embedding models, making it suitable for more complex text generation tasks.

Readability is generally assessed through metrics such as total sentence count, total word count, and total syllable count. Different evaluation methods assign varying weights to the ratios between these factors in order to assess readability from different angles. **Flesch Reading Ease (FRE)**[17] aims to measure the difficulty of understanding an English paragraph, indicating whether the material is easy to read. On the other hand, **Flesch-Kincaid Grade Level (FKGL)**[17] maps readability to the grade level of American students, indicating the years of education required to comprehend the text, making it easier to intuitively judge the readability level of a given text.

B. Result

Table I shows the experimental results on five LLMs. In the row "Method", **D** indicates that LLMs are invoked directly from the mermaid data to generate the description content; **S** indicates that without graph decomposition and path excavation, the template is sequentially applied to the mermaid data to generate the prompt content, and then the LLMs are invoked to generate the description content; **RPST** indicates that the **RPST** decomposition is used alone, and the prompts of each complex segment are generated in the order of paths, and then collocated by the LLMs to generate description content; **R+LtM** means applying all algorithms mentioned in section 2.

TABLE I. RESULTS

LLM	Method	BERTScore			Reading Ease	
		R	***P***	***F1***	***FRE***	***FKGL***
Qwen2.5-72B-Instruct	D	79.93	84.20	82.00	29.39	14.73
	S	84.91	87.61	86.23	40.26	11.92
	RPST	85.74	88.50	87.10	42.15	11.88
	R+LtM	**87.10**	**90.58**	**88.80**	**43.92**	**11.44**
Qwen2.5-32B-Instruct	D	79.95	83.74	81.79	17.52	16.94
	S	82.51	84.89	83.67	22.49	15.11

	RPST	**82.65**	85.54	84.04	25.45	14.67
	R+LtM	82.18	**86.29**	**84.16**	**27.50**	**14.40**
Qwen2.5-7B-Instruct	D	78.49	83.18	80.76	31.34	13.90
	S	84.18	87.75	85.90	40.19	11.91
	RPST	84.08	87.80	85.88	**40.78**	**11.79**
	R+LtM	**84.46**	**88.43**	**86.38**	37.93	12.39
GLM-4-9B-Chat	D	78.63	83.47	80.97	23.70	15.70
	S	82.68	85.87	84.24	26.78	14.43
	RPST	**83.06**	86.72	84.84	**32.28**	**13.45**
	R+LtM	82.83	**87.06**	**84.89**	25.30	14.49
TeleChat2-115B	D	79.18	83.99	81.51	23.26	15.27
	S	87.06	90.06	88.53	44.08	11.20
	RPST	87.42	90.66	89.01	**44.90**	**11.11**
	R+LtM	**87.46**	**91.03**	**89.21**	43.18	11.37

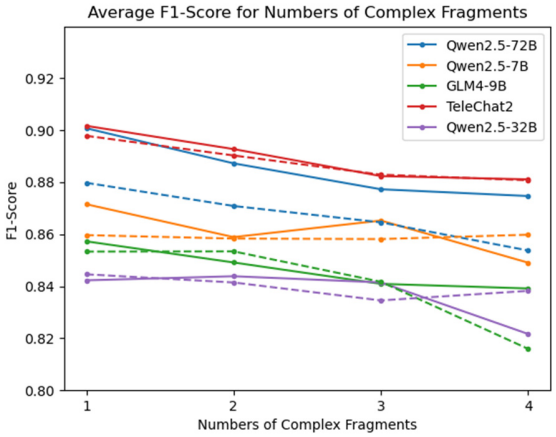


Figure 5. Average F1-Score for Numbers of Complex Fragments in different methods. The solid line represents the RPST+LtM method, while the dashed line represents the RPST method.

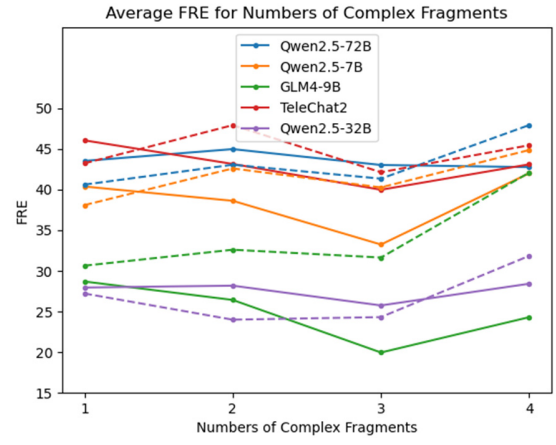


Figure 6. Average FRE for Numbers of Complex Fragments in different methods. The solid line represents the RPST+LtM method, while the dashed line represents the RPST method.

Figure 5 and Figure 6 show F1-Score and FRE change by the numbers of complex fragments in flowcharts in different methods. As we can see, for the F1-Score, RPST+LtM method in larger LLMs is always better than RPST. However, for F1-Score under other LLMs, or FRE across all LLMs, the RPST+LtM method performs worse than the RPST method when the number of complex segments is bigger than 3.

Therefore, in the practical use of the algorithm, the number of complex fragments involved simultaneously should be controlled to be less than or equal to 3. When the complex fragments are too large or numerous, smaller complex fragments should be prioritized and implemented using the RPST method and prompt, while controlling the number of LtM prompt, in order to ensure the accuracy and readability of the generated results.

IV. Conclutions

The complex flowchart text generation method based on RPST decomposition and Least-to-Most prompt strategy proposed in this paper has achieved significant results. The experimental results show that the RPST+LtM prompt strategy has a significant improvement in BERTScore scores compared to the direct generation method in all LLMs tested, with the Qwen2.5-72B-Instruct model achieving the optimal result (F1-score of 88.80) when using this strategy. The method also performed well on readability metrics, with higher Flesch Reading Ease (FRE) scores for the generated text, indicating that the generated descriptions were easier to understand.

On smaller-sized LLMs, the RPST+LtM prompt method instead performs less well than the RPST decomposition alone, which we hypothesize is mainly due to the fact that multiple iterative invocations of the smaller model can lead to information distortion and cumulative errors in the processing of complex fragments. On the other hand, on LLMs with a larger number of parameters, although the method proposed in this paper still delivers a performance improvement, the magnitude of the improvement is relatively small, which may be due to the fact that large-scale language models inherently have stronger information extraction and comprehension capabilities, which makes the marginal benefits brought by the graph decomposition and prompt engineering strategies relatively lower. At the same time, in practical applications, the excessive use of the RPST+LtM prompt method should be avoided. Smaller complex fragments should be handled using the RPST prompt method to ensure that the generated results are both readable and effective.

References

[1] B. Schäfer, M. Keuper, and H. Stuckenschmidt, "Arrow R-CNN for handwritten diagram recognition," *Int. J. Doc. Anal. Recognit. IJDAR*, vol. 24, no. 1–2, pp. 3–17, Jun. 2021, doi: 10.1007/s10032-020-00361-1.

[2] J. Ye, A. Dash, W. Yin, and G. Wang, "Beyond End-to-End VLMs: Leveraging Intermediate Text Representations for Superior Flowchart Understanding," Dec. 21, 2024, *arXiv*: arXiv:2412.16420. doi: 10.48550/arXiv.2412.16420.

[3] A. Arbaz, H. Fan, J. Ding, M. Qiu, and Y. Feng, "GenFlowchart: Parsing and Understanding Flowchart Using Generative AI," in *Knowledge Science, Engineering and Management*, vol. 14884, C. Cao, H. Chen, L. Zhao, J. Arshad, T. Asyhari, and Y. Wang, Eds., in Lecture Notes in Computer Science, vol. 14884. , Singapore: Springer Nature Singapore, 2024, pp. 99–111. doi: 10.1007/978-981-97-5492-2_8.

[4] F. Friedrich, J. Mendling, and F. Puhlmann, "Process Model Generation from Natural Language Text," in *Active Flow and Combustion Control 2018*, vol. 141, R. King, Ed., in Notes on Numerical Fluid Mechanics and Multidisciplinary Design, vol. 141. , Cham: Springer International Publishing, 2011, pp. 482–496. doi: 10.1007/978-3-642-21640-4_36.

[5] H. Kourani, A. Berti, D. Schuster, and W. M. P. van der Aalst, "Process Modeling With Large Language Models," vol. 511, 2024, pp. 229–244. doi: 10.1007/978-3-031-61007-3_18.

[6] F. Meziane, N. Athanasakis, and S. Ananiadou, "Generating Natural Language specifications from UML class diagrams," *Requir. Eng.*, vol. 13, no. 1, pp. 1–18, Jan. 2008, doi: 10.1007/s00766-007-0054-0.

[7] H. Dalianis, "A method for validating a conceptual model by natural language discourse generation," in *Active Flow and Combustion Control 2018*, vol. 141, R. King, Ed., in Notes on Numerical Fluid Mechanics and Multidisciplinary Design, vol. 141. , Cham: Springer International Publishing, 1992, pp. 425–444. doi: 10.1007/BFb0035146.

[8] J. Wei *et al.*, "Finetuned Language Models Are Zero-Shot Learners," Feb. 08, 2022, *arXiv*: arXiv:2109.01652. doi: 10.48550/arXiv.2109.01652.

[9] T. B. Brown *et al.*, "Language Models are Few-Shot Learners," Jul. 22, 2020, *arXiv*: arXiv:2005.14165. doi: 10.48550/arXiv.2005.14165.

[10] C. Gutwenger and P. Mutzel, "A Linear Time Implementation of SPQR-Trees," in *Graph Drawing*, vol. 1984, J. Marks, Ed., in Lecture Notes in Computer Science, vol. 1984. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 77–90. doi: 10.1007/3-540-44541-2_8.

[11] J. Vanhatalo, H. Völzer, and J. Koehler, "The refined process structure tree," *Data Knowl. Eng.*, vol. 68, no. 9, pp. 793–818, Sep. 2009, doi: 10.1016/j.datak.2009.02.015.

[12] Y. Choi, N. L. Ha, P. Kongsuwan, and K. H. Han, "An alternative method for refined process structure trees (RPST)," *Bus. Process Manag. J.*, vol. 26, no. 2, pp. 613–629, Oct. 2019, doi: 10.1108/BPMJ-11-2018-0319.

[13] C. Tang, "Automatically Block-structuring of BPMN Models," Eindhoven University of Technology, 2022.

[14] D. Zhou *et al.*, "Least-to-Most Prompting Enables Complex Reasoning in Large Language Models," Apr. 16, 2023, *arXiv*: arXiv:2205.10625. doi: 10.48550/arXiv.2205.10625.

[15] rakitha, "mermaid-flowchart-transformer." Accessed: Jan. 01, 2025. [Online]. Available: https://huggingface.co/datasets/rakitha/mermaid-flowchart-transformer

[16] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "BERTScore: Evaluating Text Generation with BERT," Feb. 24, 2020, *arXiv*: arXiv:1904.09675. doi: 10.48550/arXiv.1904.09675.

[17] J. P. Kincaid, Jr. Fishburne, R. Robert P., C. Richard L., and Brad S., "Derivation of New Readability Formulas (Automated Readability Index, Fog Count and Flesch Reading Ease Formula) for Navy Enlisted Personnel:," Defense Technical Information Center, Fort Belvoir, VA, Feb. 1975. doi: 10.21236/ADA006655.