Context-Aware Graph Querying for LLM-Based Code Generation on Low-Code platforms

Arden Wołowiec[1], **Dawid Korzępa**[1], **Anna Śmigiel**[1],
Krzysztof Raczyński[1,2], **Patryk Żywica**[1*,[0000−0003−3542−8982]]

[1]*Adam Mickiewicz University, Poznań*
Faculty of Mathematics and Computer Science
Uniwersytetu Poznańskiego 4, 61-614 Poznań, Poland
[2]*DomData S.A.*
Aleje Solidarności 46, 61-696 Poznań, Poland
[*]*Corresponding author: patryk.zywica@amu.edu.pl*

Abstract. *This study examines the integration of a multi-agent LLM architecture into low-code systems. Key insights derived from a BPMN model are obtained using graph clustering and LLM-based querying. Subsequently, this information is used to construct an optimal prompt, which is then provided to a code generation agent. We evaluated the generated code using expert assessments and an external LLM. Both evaluations were conducted using six user prompts that reflect real-world use cases. The results indicate that our method dynamically generates context-aware code, thereby supporting low-code developers with limited programming expertise.*
Keywords: *LLM, Graph, Low-Code, No-Code, Code Generation, BPMN*

1. Introduction

In recent years, low-code and no-code systems have gained popularity due to high developer costs and the scarcity of qualified specialists. These platforms enable users with minimal programming experience to develop applications visually, focusing on business logic [1] rather than on the technicalities of infrastructure and backend management. Although low-code systems are challenging to define precisely, they typically incorporate components such as data model designers, GUI builders, external API integrations, and BPMN-based process flow editors

[2], which can reduce development time by a factor of 5 to 10 [3] and meet the demand for rapid, flexible solutions. Despite these advances, ongoing innovation – particularly in AI integration – is necessary. Incorporating advanced AI features can overcome challenges in customization, scalability, and complex logic implementation, ultimately fostering more intuitive and adaptable development environments essential for the next generation of user-centric, intelligent software tools.

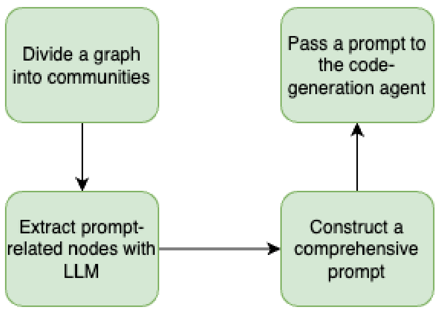
Despite the numerous advantages offered by low-code systems, their effective use still requires a foundational understanding of technical concepts and basic programming skills. The primary motivation for our research is to lower the entry barrier for individuals without programming experience while simultaneously improving the quality and efficiency of application development. In this context, integrating AI-powered assistants into low-code systems offers significant benefits, allowing users to design processes using natural language without the need for coding. Our research aims to accelerate the transformation of low-code systems into no-code solutions, which would greatly enhance their accessibility and utility across various industries.

2. Methods

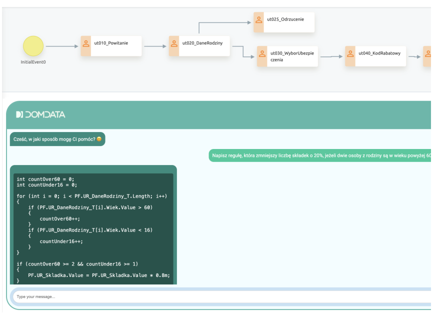
This research explores how using user prompts to extract contextual information from nodes in BPMN diagram graphs can automate code generation, transforming low-code platforms into no-code platforms (Figure 1b). Unlike static knowledge graph-based systems, our approach dynamically queries BPMN graphs in real-time with natural language prompts. The paper proposes the methods, algorithms, and implementation of such a solution.

In BPMN, diagrams are represented as graphs where nodes denote tasks, gateways, and events, while edges represent transitions. Besides defining process flows, BPMN also encapsulates predefined data models. For efficient processing, the graph is partitioned into community structures via Graph Spectral Clustering [4], which leverages eigenvalues of the similarity matrix. This flexible technique allows adjusting the number of clusters based on task requirements, with fewer clusters for simple processes and more for complex diagrams.

Upon receiving a user prompt, a Large Language Model (LLM) identifies the most relevant nodes within each cluster. The prompt includes context-aware instructions, such as: *You are a BPMN expert and assistant on a low-code platform. Select the graph nodes that are most relevant to the given user query or coding*


Divide a graph into communities
Extract prompt-related nodes with LLM
Construct a comprehensive prompt
Pass a prompt to the code-generation agent

(a)



Cześć, w jaki sposób mogę Ci pomóc? 😊

Napisz regułę, która zmniejszy liczbę składek o 20%, jeżeli dwie osoby z rodziny są w wieku powyżej 60

```
int countOver60 = 0;
int countUnder16 = 0;

for (int i = 0; i < PF.UR_DaneRodziny_T.Length; i++)
{
    if (PF.UR_DaneRodziny_T[i].Wiek.Value > 60)
    {
        countOver60++;
    }
    if (PF.UR_DaneRodziny_T[i].Wiek.Value < 16)
    {
        countUnder16++;
    }
}

if (countOver60 >= 2 && countUnder16 >= 1)
{
    PF.UR_Skladka.Value = PF.UR_Skladka.Value * 0.8m;
}
```

(b)

Figure 1: Prompt construction flow (a), and a screenshot from a low-code platform (b).

task. This clustering structure facilitates parallel processing, thereby improving performance and scalability. To manage parallelization, we adopt a Map-reduce-like framework [5]. Each cluster is independently processed to extract prompt-related nodes, then the results are aggregated and refined into a comprehensive prompt forwarded to the code generation agent (Figure 1a).

The contextual information extracted from the BPMN graph is used for automatic code generation. To achieve this, in our solution we employ several tools and models: *Bielik-11B-v2.2-Instruct-FP8* extracts relevant nodes from the context and corresponding data model variables; *Codestral-24.05* generates concise, contextually appropriate code snippets; *LangGraph 0.2.39* supports a multi-agent architecture for graph-based querying; *LangChain 0.3.0* enables seamless integration with LLMs; and *Ollama* enhances model interaction and management.

3. Results and Discussion

Evaluating code generated by large language models (LLMs) poses significant challenges, especially as our assistant produces concise snippets intended to integrate into an existing codebase – limiting the applicability of traditional unit or end-to-end testing. To address this, we devised an alternative evaluation strategy that combines expert knowledge with LLM evaluation. Our manual evaluation, based on six user prompts reflecting real-world use cases within the business process, divided the code generation flow into two parts: (1) selecting relevant data model variables from BPMN's context using graph clustering and LLM extraction, and (2) generating code that incorporates this contextual information, with evaluators comparing snippets from two distinct models. For code generation, we compared two models: Codestral-24.05 and GPT-4. Both models were queried with the same structured prompts to ensure consistency in all test cases.

A panel of five experts conducted the evaluation – two professional low-code developers, two educators specializing in low-code solutions, and one full-stack developer with 10 years of experience. They evaluated variable selection on a yes/no basis according to three criteria: the relevance of the selected variables to the prompt, the inclusion of all necessary variables, and the exclusion of redundant ones. Additionally, they rated each code snippet on a five-point scale based on the absence of syntax errors, proper use of variables, clarity and conciseness, and compliance with user requirements. The original evaluation sheet and prompts were constructed in Polish. An example testing scenario is presented below.

User prompt: If the discount code has the value `"UAM5"`, reduce the insurance premium by 5% and display the insurance premium field.

Variables selected via graph clustering & LLM: `PF.UR_KodZnizkowy: String`, `PF.UR_Skladka: Decimal`

Generated code:

```
if (PF.UR_KodZnizkowy.Value == "UAM5") {
    PF.UR_Skladka.Value = PF.UR_Skladka.Value * 0.95m;
    PF.UR_Skladka.SetVisible(true);
}
```

An LLM evaluation was performed using GPT-4. Analogously, to the manual assessment, we used the same six scenarios and the same previously defined criteria, yet this time the criteria were evaluated binary.

The results (Table 1) show that in terms of variable selection, Bielik demonstrated strong performance in prompt relevance (0.917 ± 0.167) and variable completeness (0.925 ± 0.158), indicating its general effectiveness in selecting relevant variables. However, its ability to eliminate redundant variables was comparatively weaker (0.500 ± 0.424). Although redundancy may impact efficiency, retaining additional variables is generally preferable to excluding essential ones, as omission can lead to incomplete or erroneous code.

The second part of Table 1 highlights the differences in code generation quality between GPT-4 and Codestral-24.05. Codestral consistently outperforms GPT-4 in syntax correctness, variable usage, clarity, and compliance with user requirements. GPT-4 struggles with syntax errors and effective variable usage, while Codestral

Table 1: Average results of quality comparison for variable selection and code generation based on expert and LLM evaluation.

Scope	Question	our model	
Variable selection	Relevance to Prompt	0.917 ± 0.167	
	All Variables Included	0.925 ± 0.158	
	No Redundant Variables	0.500 ± 0.424	
		GPT-4	our model
Code generation (expert)	No Syntax Errors	4.467 ± 0.729	**4.767** ± 0.522
	Variable Usage	4.567 ± 0.396	**4.733** ± 0.596
	Clarity & Conciseness	**4.733** ± 0.438	4.467 ± 0.620
	User Requirements	4.267 ± 0.800	**4.733** ± 0.515
Code generation (LLM)	No Syntax Errors	0.667	**1.000**
	Variable Usage	0.833	0.833
	Clarity & Conciseness	1.000	1.000
	User Requirements	0.667	**0.833**

excels in both areas. Although GPT-4 scores reasonably well for clarity, this often comes at the expense of oversimplification, whereas Codestral aligns more accurately with user specifications which results in higher complexity.

The evaluation carried out by human experts and GPT-4 itself indicates that GPT-4's performance is lower than Codestral's. Furthermore, the LLM-based evaluation framework revealed that the majority of errors made by both language models were related to null safety in C#. However, Codestral addressed these issues much more effectively than GPT-4. The implementation of the framework is available in project repository [1].

4. Conclusions

The proposed approach demonstrates superior accuracy, structure, and reliability, positioning it as a robust tool for automated coding. In particular, expert evaluations highlighted a significant advantage for our model in one of the more challenging tasks. Its performance was comparable to simpler scenarios, while the GPT-4 model received notably lower ratings. However, given the relatively small

[1]`https://github.com/korzepadawid/lowcode-ai/`

test sample in relation to the broad applicability of our solution, these results may be somewhat overoptimistic. Consequently, further testing on a larger sample is recommended.

Proposed approach is distinct from traditional methods like Retrieval-Augmented Generation (RAG) [6] and GraphRAG [7], which rely on predefined databases or knowledge graphs. By querying the BPMN diagram directly in real-time, we achieve greater flexibility and responsiveness to user prompts. Additionally, the use of Spectral Clustering for graph decomposition provides a more task-specific partitioning compared to other clustering algorithms.

References

[1] Sahay, A., Indamutsa, A., Di Ruscio, D., and Pierantonio, A. Supporting the understanding and comparison of low-code development platforms. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications*, pages 171–178. IEEE, 2020. doi:10.1109/SEAA51224.2020.00036.

[2] Bock, A. C. and Frank, U. Low-code platform. *Business & Information Systems Engineering*, 63:733–740, 2021. doi:10.1007/s12599-021-00726-8.

[3] Yan, Z. The impacts of low/no-code development on digital transformation and software development. *arXiv preprint 2112.14073*, 2021.

[4] Ng, A., Jordan, M., and Weiss, Y. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 14, 2001.

[5] Dean, J. and Ghemawat, S. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008. doi:10.1145/1327452.1327492.

[6] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[7] Edge, D., Trinh, H., Cheng, N., Bradley, J., Chao, A., Mody, A., Truitt, S., and Larson, J. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint 2404.16130*, 2024.