From Rule-Based NLP to Prompt-Guided Multi-LLM Pipelines for Text-to-UML and Code Generation

Zakaria Babaalla
GLISI Team, Faculty of Sciences and Techniques of Errachidia, Moulay Ismail University, Morocco
za.babaalla@edu.umi.ac.ma (corresponding author)

Hamza Abdelmalek
GLISI Team, Faculty of Sciences and Techniques of Errachidia, Moulay Ismail University, Morocco
h.abdelmalek@edu.umi.ac.ma

Abdeslam Jakimi
GLISI Team, Faculty of Sciences and Techniques of Errachidia, Moulay Ismail University, Morocco
ajakimi@yahoo.fr

Rachid Saadane
Electrical Engineering Department, Hassania School of Public Works, Casablanca, Morocco
saadane@ehtp.ac.ma

Received: 8 February 2026 | Revised: 18 March 2026 and 27 March 2026 | Accepted: 6 April 2026

 DOI: https://doi.org/10.48084/etasr.18049

ABSTRACT

Automatic transformation of textual specifications into formal software models is a key challenge in model-driven engineering. Despite progress, existing approaches remain fragmented and rarely integrate into a coherent methodological framework. This article offers a progressive and analytical review of UML and text-based code generation methods, structured around three successive contributions reflecting the natural evolution of the field: a first approach based on linguistic analysis and rules, a second based on language models trained on an annotated corpus, and a third relying on prompt-guided LLMs within a multi-model MDA pipeline stabilized by an intermediate Domain-Specific Language (DSL). A comparative study underscores the respective advantages and shortcomings of the considered paradigms and demonstrates that DSL-guided multi-LLM orchestration markedly enhances structural consistency, multilingual robustness, and the quality of generated code.

Keywords-MDA; text-to-UML; LLM; DSL; prompt engineering; code generation; software modeling

I. INTRODUCTION

Transforming textual specifications into formal software models (and ultimately into executable code) remains a major challenge in software engineering [1]. In most development processes, system requirements are expressed in natural language, often in heterogeneous and ambiguous forms that complicate their direct use during the design and implementation phases [2]. Therefore, analysts and developers must manually interpret these descriptions to construct structured models, a process that is both time-consuming and prone to inconsistencies. Automating the transformation of textual requirements into Unified Modeling Language (UML) diagrams and subsequently into executable code can significantly improve the traceability between requirements and implementation, reduce interpretation errors, and enhance development efficiency [3].

A promising framework for such automation was provided by the Model-Driven Architecture (MDA) paradigm [4], which structures software development across multiple abstraction levels, including the Computation Independent Model (CIM), Platform Independent Model (PIM), and Platform Specific Model (PSM). In this perspective, textual requirements can be considered CIM-level descriptions, whereas UML models correspond to PIM-level representations formalizing system structure and behavior. Consequently, transforming textual specifications into UML diagrams represents an automated CIM-to-PIM transition that can support further refinements and code generation. MDA-based approaches have proven to be effective in domains such as software modernization and

reverse engineering, where model transformations help manage system complexity and maintain traceability across development stages [5, 6].

Among UML representations, class diagrams play a central role in model-driven development because they describe the static structure of systems using classes, attributes, operations, and relationships [7]. When sufficiently precise, these diagrams can be translated into programming constructs using transformation rules that map UML elements to programming language structures [8]. Other UML diagrams, such as use case, sequence, and activity diagrams, can also contribute to code generation by describing the behavioral aspects of the system [7]. However, inaccuracies in extracting UML elements from textual specifications may propagate through the development pipeline and affect the generated code, highlighting the need for reliable transformation mechanisms.

Numerous studies have attempted to automate the generation of UML diagrams from textual requirements using Natural Language Processing (NLP) techniques. In [9], NLP and heuristic rules were combined to automatically extract classes, attributes, and relationships from textual specifications to generate UML class diagrams. In [10], NLP techniques were applied to support object-oriented design from requirement documents, while in [11], a similar framework was presented for identifying classes, attributes, and relationships. In [12], NLP analysis and classification rules were combined to transform textual requirements into UML class diagrams, whereas in [13], an intelligent framework integrated NLP and the K-Nearest Neighbors (KNN) algorithm to generate both use case and class diagrams by identifying actors, entities, and relationships.

Beyond rule-based NLP approaches, several studies have explored machine learning and deep learning techniques to improve the automation of UML diagram generation. In [14], a machine learning-based method was proposed to generate use case diagrams by identifying actors and use cases in textual descriptions. In [15], machine learning was applied to class diagram generation by extracting entities, attributes, and relationships with improved accuracy compared to rule-based methods. In [16], deep learning techniques were used to automatically identify structural elements in textual specifications, reducing reliance on manually crafted linguistic rules. In [17], an interactive approach combined artificial intelligence and machine learning to generate a traceable UML model, allowing user intervention to improve model quality.

Recently, Large Language Models (LLMs) have attracted attention for their ability to interpret complex natural language descriptions and generate structured outputs, such as UML models or source code. In [18], it was shown that few-shot prompt learning can guide software model completion tasks. In [19], it was shown that LLMs generate methods to complete UML class diagrams, although human supervision is still required to correct certain inconsistencies. In [20], the use of GPT-4 [21] was investigated to reverse engineer class diagrams from source code, successfully extracting attributes, operations, and types while highlighting the limitations of higher-level abstractions. In [22], parameter-efficient fine-tuning techniques were explored to adapt LLMs for code generation tasks.

Despite recent advances, challenges remain regarding the structural reliability and formal consistency of automatically generated models [23]. Generative approaches may produce outputs that violate modeling constraints or introduce structural inconsistencies. To mitigate these issues, recent research has explored hybrid solutions that combine LLMs with structured intermediate representations. Domain-Specific Languages (DSLs) serve as an intermediate layer between textual specifications and UML models, helping to normalize extracted elements, reduce ambiguity, and support validation before model or code generation [24, 25].

In this context, this work presents a progressive study of text-to-UML-to-code transformation approaches. Three complementary methodologies are analyzed: a rule-based linguistic approach, a supervised LLM-based approach trained on annotated corpora, and a third approach—introduced in this paper—based on a prompt-guided multi-LLM pipeline integrated into an MDA workflow and stabilized through an intermediate DSL. By comparing these approaches within a unified evaluation framework, this study highlights their strengths, limitations, and contributions to building reliable automated pipelines for transforming textual specifications into UML models and executable code.

II. METHODOLOGY AND PROGRESSIVE CONTRIBUTIONS

This section traces the progressive evolution of the automatic transformation of specification texts into UML models and then into code. Three contributions are presented: a rule-based linguistic approach, an approach based on an LLM trained on annotated corpora, and a multi-LLM MDA pipeline guided by prompts, each aiming to overcome the limitations of the previous approach.

A. UML Extraction Using a Rule-Based Linguistic Approach

In the first contribution, the study in [26] focused on automating the extraction of UML models from natural language specifications to reduce manual effort during the early analysis and design phases. Unlike approaches that require controlled or reformulated inputs, the proposed method accepts relatively free text descriptions and applies a progressive linguistic analysis to identify structured modeling elements.

1) General Principle and Architecture of the Approach

This approach relies on a two-module architecture (Figure 1) that combines an NLP module and a rule-based knowledge extraction module. The input text was first processed through standard NLP steps, including sentence segmentation, tokenization, part-of-speech tagging, and lemmatization to yield a normalized linguistic representation. Based on this analysis, heuristic rules derived from common syntactic patterns and object-oriented modeling principles were applied to detect UML elements, including classes, attributes, methods, relationships, actors, and use cases.

2) Generated Outputs

The system automatically produces UML class and use case diagrams. Classes are mainly inferred from candidate nouns, attributes from nominal structures or possession patterns, and

methods from action verbs associated with the classes. Relationships are estimated using simple syntactic patterns, and normalization rules are applied to reduce redundancy and ensure naming consistency. The resulting models can be visualized graphically or exported in a structured format to facilitate toolchain integration.

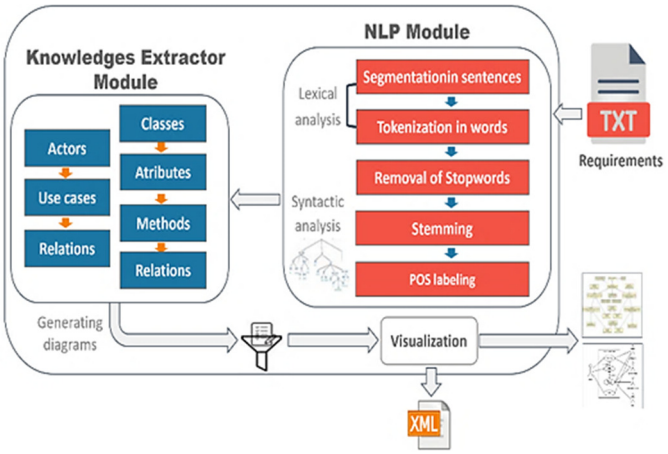

Fig. 1. Rule-based NLP pipeline for UML extraction.

3) *Experimental Results and Limitations*

Experiments conducted on a set of textual specifications show high extraction performance for classes, attributes, and methods, with F1-scores generally ranging between 80% to 100%. The extraction of relationships and use case links remains challenging because of their implicit and contextual expressions in natural language. Moreover, this approach is sensitive to linguistic variability and complex sentence structures, and its rule-based nature limits generalization across languages and domains.

These limitations motivate the transition toward learning-based solutions, which are addressed in the next section through supervised LLMs trained on annotated corpora.

B. *Supervised LLM Trained on Annotated Corpus*

The second contribution [25] addressed the limitations of purely heuristic approaches by framing the text-to-UML transformation as a supervised learning problem. More precisely, UML extraction is modeled as a Named Entity Recognition (NER) [27] task, where each token is assigned a label corresponding to a UML entity (class, attribute, or method) or a relationship type (association, composition, aggregation, inheritance). This formulation enables the use of transformer-based models [27] to capture contextual and semantic information, even when UML concepts are implicitly expressed.

1) *Principle and Supervised Learning*

The proposed architecture follows a three-stage pipeline (Figure 2). First, several pretrained language models (BERT, RoBERTa, SpanBERT, XLNet, MiniLM, and Electra) were fine-tuned on an annotated corpus to learn UML concept recognition [28-33]. Second, an extraction module applies the trained model to new textual requirements to identify the UML elements. Finally, the extracted entities are structured into an intermediate DSL through a visualization interface, allowing validation and correction before generating the final UML class diagram. This diagram can then be automatically transformed into Python code, ensuring continuity between the modeling and implementation within the MDA workflow. To support supervised learning, a dedicated corpus of 132 specifications from heterogeneous sources was manually annotated using an enhanced IOB schema [34]. The dataset comprised 915 sentences and 11,460 annotated tokens covering diverse domains, writing styles, and syntactic structures. All models were trained and evaluated under the same conditions.

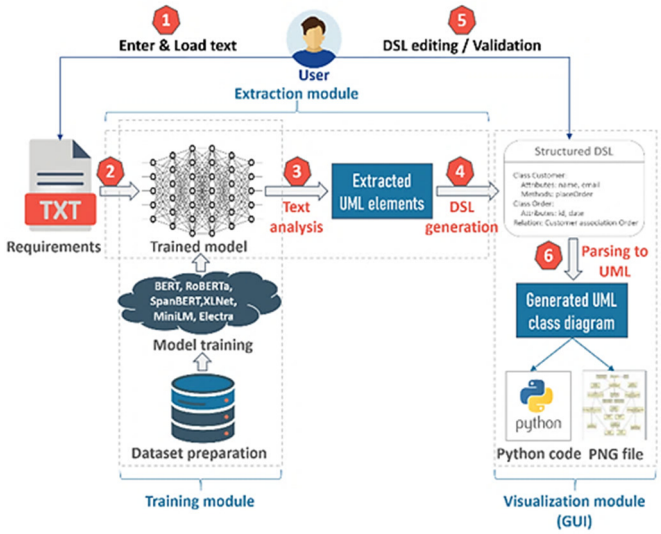

Fig. 2. Supervised UML extraction and UML/Code generation via intermediate DSL.

2) *Results and Gains Over Rule-Based Methods*

The experimental results show a significant improvement in robustness compared to the rule-based approach. Class extraction achieved very high performance (F1 ≈ 0.98 for BERT, RoBERTa, and XLNet), whereas method and attribute detection also benefited from contextual learning, despite remaining more challenging. UML relationship extraction, the most complex task, was notably improved with XLNet, which better captured long-range dependencies (F1 score ≈ 0.92). In addition to these extraction gains, this study introduced automatic UML-to-Python code generation, extending the scope beyond modeling and reinforcing MDA integration.

3) *Limitations*

The main limitation of this approach is its dependence on high-quality annotated data, which are costly to produce and domain-dependent. Moreover, although an intermediate DSL was already used for structuring and validation, additional refinement was required to ensure completeness, consistency, and scalability, particularly for long, ambiguous, and multilingual specifications. These limitations motivate the next contribution, which explores a prompt-guided multi-LLM pipeline with enhanced DSL-based controls.

C. Prompt-Driven Multi-LLM MDA Pipeline

This third contribution introduces a methodological shift by proposing a fully controlled MDA pipeline driven by prompt engineering [35] and multi-LLM orchestration. Unlike previous approaches—rule-based extraction [26] and supervised learning on annotated corpora [25]—this strategy avoids handcrafted rules and costly annotation efforts while ensuring a structured and industrializable transformation chain from text to software artifacts.

1) MDA-Oriented Architecture

The proposed pipeline follows a progressive transformation chain aligned with the MDA principles (Figure 3). Starting from a textual specification at the CIM level, an LLM is used to extract the conceptual elements. Instead of directly generating UML diagrams, the system first produces a textual UML DSL that acts as a pivot representation at the PIM level. This intermediate DSL is then validated through syntactic and structural checks to ensure model consistency before being transformed into a UML class diagram and subsequently into source code (PSM-to-Code).

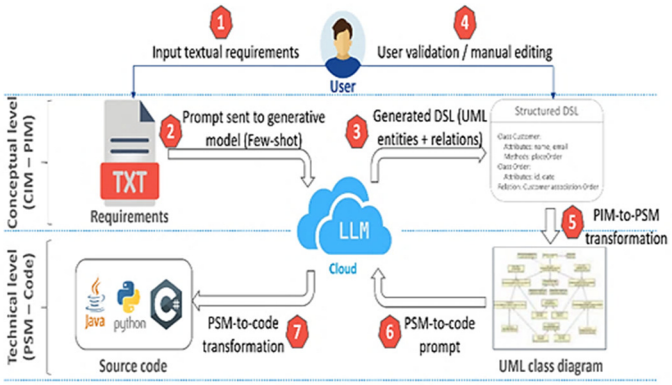

Fig. 3. Prompt-guided multi-LLM MDA pipeline for UML and code generation

2) Prompt Engineering and Generation Control

A key contribution is the use of structured and constrained prompts to limit the variability and hallucinations typically associated with generative models. The extraction prompt (Figure 4) enforces a strict output format, standardized identifiers, and explicit constraints to prevent the introduction of elements not grounded in the input text. A separate prompt (Figure 5) is dedicated to code generation, focusing on producing consistent and compilable code aligned with the validated UML structure.

3) Multi-LLM Orchestration

Rather than relying on a single model, the pipeline adopts a multi-LLM strategy and selects models according to task requirements. Models optimized for global comprehension (e.g., LLaMA-3-8B, Mistral-7B) are preferred for complex or ambiguous specifications, whereas reasoning-oriented and code-specialized models (e.g., Deepseek-Coder, CodeLLaMA) are used for consistency-critical phases and final code generation. This adaptive orchestration improves robustness while maintaining architectural flexibility [36-39].

4) Contributions and Limitations

This contribution presents a comprehensive text-to-UML-to-code pipeline that embeds LLMs within a formally controlled MDA framework. The combination of DSL-based validation and prompt-guided generation significantly improves the structural consistency and reduced output variability. Additionally, the pipeline supports multi-language code generation (e.g., Python, Java, and C#), thereby enhancing its applicability in heterogeneous industrial contexts.

However, some limitations persist, particularly regarding business-level ambiguities and prompt sensitivity, which may still affect the modeling decisions. These aspects motivate the comparative evaluation presented in the next section.

```
prompt = f"""
You are an expert in UML modeling and Model-Driven Engineering (MDA).
Generate a structured textual UML DSL from the following description.

### Objective

Produce a complete and strictly formatted UML DSL including:
-All classes
- Attributes (names only, comma-separated)
- Methods (names only, no parameters)
- All UML relations

### Structure

Class ClassName:
    Attributes: attr1, attr2, …
    Methods: meth1(), meth2(), …
Relation: SourceClass relation_type TargetClass

Allowed relation types: association, aggregation, composition, dependency,
realization, implements, operation, use, inheritance, association_class, …

### Example

    Class Library:
        Attributes: libraryId
        Methods: getLibraries()
    Class Book:
        Attributes: title, author, publicationYear, ISBN
        Methods: getBookDetails()
    Class Reader:
        Attributes: name, address, cardNumber
        Methods: borrowBook()
    Relation: Library composition Book
    Relation: Reader association Book

### Rules

- 3 lines per class; keep empty lines if needed.
- One relation per line; reflexive allowed (Source = Target).
- Every class in a relation must be defined; create minimal ones.
- Class names: PascalCase; attributes/methods: camelCase.
- No comments, no cardinalities, no narrative text, no extra punctuation.

### Output

Only the final DSL.

### User specification:

{spec_text}
"""
```

Fig. 4. Prompt for extracting UML class diagram entities from textual specifications.

```
prompt = f"""
You are an expert in software engineering and object-oriented programming.
Your task is to generate the corresponding {language} code from the UML model
below.

### Objective

Produce **directly executable code** based strictly on the UML model:
- Classes
- Attributes
- Methods
- Relationships

### Mandatory Rules

- Output **only the code**, with no explanations or comments outside the code.
- Follow the syntax and OOP conventions of the chosen language: {language}.
- If a method appears in a class without an implementation in the UML mode
 --> You must implement it.

### Expected Output

Clean, valid, executable {language} code derived from the UML model.

### UML Model

{uml_description}
"""
```

Fig. 5. Prompt for generating source code from the UML class diagram.

III. EXPERIMENTAL EVALUATION AND COMPARATIVE ANALYSIS

This section presents a comparative evaluation of the three proposed approaches: (i) rule-based UML extraction, (ii) supervised LLMs trained on annotated corpora, and (iii) a prompt-driven multi-LLM MDA pipeline. Their trade-offs are analyzed in terms of precision, robustness, structural consistency, and code generation, while positioning them with respect to existing approaches. Experiments were conducted on a diverse set of textual specifications, including short and long structured and narrative descriptions across several domains (e.g., management systems, e-commerce, and libraries). Each specification was associated with a manually designed UML reference model that operated as the gold standard. All approaches were evaluated on the same dataset, focusing on two stages: UML generation (CIM/PIM to PSM) and code generation (PSM to Code).

A. Evaluation Metrics

1) UML Extraction Quality

UML extraction performance is evaluated based on classes, attributes, methods, and relationships (association, inheritance, composition, and aggregation). For each type of entity, Precision, Recall, and F1-score are calculated. This fine-grained evaluation was necessary because strong performance on classes alone does not guarantee a complete or exploitable UML model.

2) Code Generation Quality

The code generation is evaluated using three complementary indicators:

- Compilation / Execution: Code that can be compiled without major errors.
- UML-to-code conformity: structural correspondence (classes, attributes, method signatures, and translated relations).
- Logical consistency: respect for inter-class dependencies and consistency of types and calls.

This analysis goes beyond the mere presence of classes and aims to verify the pipeline's ability to produce a truly usable final artifact.

B. Quantitative Results

Table I summarizes the results, reporting the performance of the three approaches in terms of UML extraction accuracy and code-generation capabilities. Overall, the performance varied across the three paradigms, each offering specific advantages.

- The rule-based approach (Contribution 1) works well for short, structured texts, achieving good precision for classes and attributes but struggles with implicit information, complex relationships, and method extraction.
- The supervised LLM (Contribution 2) improves overall extraction, especially for relations and methods, benefiting from semantic pattern learning and the intermediate DSL for validation; however, it remains dependent on the domain of the annotated corpus, with errors reappearing in out-of-distribution texts.
- The prompt-driven multi-LLM pipeline (Contribution 3) achieves the highest structural consistency with refined DSL controls, constrained prompts, and multi-model orchestration, offering improved multilingual stability and fewer internal inconsistencies.

Regarding code generation (Contributions 2 and 3), the supervised LLM produces richer but occasionally unstable code, whereas the multi-LLM pipeline achieves the highest rate of compilable code and the best alignment with UML models, supporting Python, Java, and C#.

TABLE I. QUANTITATIVE COMPARISON OF THE THREE CONTRIBUTIONS

Criteria		**Measure**	**Approaches (Contributions)**		
			Cont1 [26]	**Cont2 [25]**	**Cont3 (Current)**
UML extraction	**Classes**	**Precision**	0.84	0.89	0.92
		Recall	0.71	0.83	0.88
		F1-score	0.77	0.86	0.90
	Attributes	**Precision**	0.79	0.86	0.90
		Recall	0.65	0.80	0.86
		F1-score	0.71	0.83	0.88
	Methods	**Precision**	0.62	0.81	0.86
		Recall	0.45	0.72	0.82
		F1-score	0.52	0.76	0.84
	Relations	**Precision**	0.68	0.83	0.88
		Recall	0.50	0.74	0.83
		F1-score	0.58	0.78	0.85
Code generation		**Generated source code**		Python	Python + Java + C#
		Compilability / Executability		0.78	0.88
		UML-to-code conformity		0.80	0.90
		Logical consistency		0.74	0.86

C. Comparative Discussion with Existing Approaches

To set the proposed approaches within the state of the art, they were compared with representative works on UML generation from natural language requirements. The approach proposed in [40] reports approximately 93% precision and 83% recall, whereas the study [41] reports 90% precision and 90% recall. Similarly, in [42], 92% precision and 100% recall were achieved, whereas machine-learning approaches such as [43] obtained 64% precision and 70% recall, and in [17], 94.33% precision and 84% recall were reported.

Despite these results (Figure 6), most existing approaches focus only on UML extraction, with limited support for automatic code generation. Only a few studies, such as [11, 44, 45], generate code (respectively in Java, Java, and C#), usually for a single language and without evaluating compilation success or UML-code consistency. In contrast, the approaches proposed in this study extend the process toward end-to-end model-driven development, including the generation of compilable source code. In particular, the prompt-driven multi-LLM pipeline (Contribution 3) provided the most consistent results in terms of UML structural accuracy and reliability of the generated implementations.

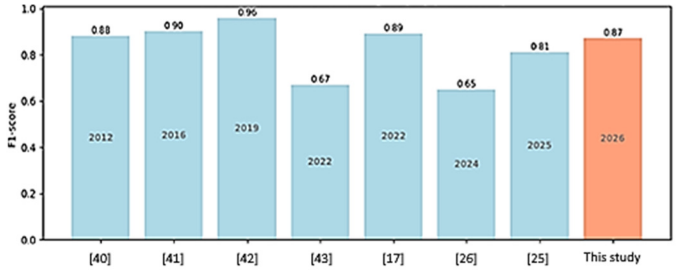

Fig. 6. Comparison of F1-scores for UML extraction across approaches and contributions

IV. DISCUSSION

The results show that transforming natural language requirements into UML models and executable code requires entity extraction, semantic interpretation, and structural consistency. These three approaches address these aspects differently. Rule-based methods remain effective for short and structured requirements but are less robust when dealing with linguistic variability and implicit information. Supervised LLMs improve the balance between precision and recall by learning semantic patterns, although their performance depends on the quality and coverage of the training corpus.

However, some limitations remain, particularly the ambiguity of natural language requirements and the computational cost associated with orchestrating multiple models. Nevertheless, by combining LLMs with structured prompts, DSL representations, and model-driven principles, it significantly improves the reliability of automated text-to-UML-to-code transformation pipelines.

V. CONCLUSION

This study presented a progressive review of automatic text-to-UML-to-code transformation, structured around three complementary contributions. The first contribution relies on UML extraction using linguistic rules, which performs well for simple texts but is limited in handling implicit meanings. The second contribution introduces a supervised LLM trained on annotated corpora, improving extraction robustness, and integrates an intermediate DSL to structure and validate the model. The third contribution is a multi-LLM MDA pipeline guided by prompts, where the DSL is refined to stabilize generation and improve UML-to-code consistency, particularly in multilingual contexts.

The main scientific contribution lies in demonstrating that a reliable chain depends not only on the model used but also on the combination of integration of LLMs, formal control via DSLs, and multi-model orchestration adapted to the complexity of the texts. Compared to existing studies, this approach addresses recurring limitations (rule fragility, dependence on annotated data, and instability of generative outputs) by proposing a more robust MDA-oriented integration.

In the short term, the prospects concern the extension of the pipeline to other UML diagrams (sequence, activity), the strengthening of semantic validation (business constraints, cardinalities, and inter-model consistency), as well as the implementation of standardized multilingual benchmarks allowing a more reproducible and comparable evaluation in real conditions.

DECLARATION OF COMPETING INTERESTS

The authors declare that they have no competing interests.

FUNDING STATEMENT

This research received no external funding.

DATA AVAILABILITY

The data and materials supporting the findings of this study are publicly accessible in [46].

REFERENCES

[1] O. S. Dawood and A. E. K. Sahraoui, "From Requirements Engineering to UML using Natural Language Processing – Survey Study," *European Journal of Engineering and Technology Research*, vol. 2, no. 1, pp. 44–50, Jan. 2017, https://doi.org/10.24018/ejeng.2017.2.1.236.

[2] L. Zhao *et al.*, "Natural Language Processing for Requirements Engineering: A Systematic Mapping Study," *ACM Computing Surveys*, vol. 54, no. 3, Dec. 2021, Art. no. 1-41, https://doi.org/10.1145/3444689.

[3] J. Shin and J. Nam, "A Survey of Automatic Code Generation from Natural Language," *Journal of Information Processing Systems*, vol. 54, no. 3, pp. 537-555, 2021.

[4] S. J. Mellor, MDA Distilled: Principles of Model-driven Architecture. Addison-Wesley Professional, 2004.

[5] H. Abdelmalek, Z. Babaalla, C. Ouaddi, L. Benaddi, and A. Jakimi, "Model-Driven Engineering and Machine Learning for Legacy System Modernization," *Engineering, Technology & Applied Science Research*, vol. 16, no. 1, pp. 32285–32291, Feb. 2026, https://doi.org/10.48084/etasr.16444.

[6] A. G. Kleppe, J. B. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture : Practice and Promise*. Addison-Wesley Professional, 2003

[7] P. Evitts, *A UML pattern language*. Macmillan Technical Publications, pp. 386-393, 2000.

[8] Y. Cheon, "LLMs as Code Generators for Model-Driven Development:," in *Proceedings of the 20th International Conference on Software Technologies*, 2025, pp. 386–393, https://doi.org/10.5220/0013580300003964.

[9] E. A. Abdelnabi, A. M. Maatuk, T. M. Abdelaziz, and S. M. Elakeili, "Generating UML Class Diagram using NLP Techniques and Heuristic Rules," in *20th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, Monastir, Tunisia, Sept. 2020, pp. 277–282, https://doi.org/10.1109/STA50679.2020.9329301.

[10] N. Bashir, M. Bilal, M. Liaqat, M. Marjani, N. Malik, and M. Ali, "Modeling Class Diagram using NLP in Object-Oriented Designing," in *2021 National Computing Colleges Conference (NCCC)*, Taif, Saudi Arabia, Mar. 2021, pp. 1–6, https://doi.org/10.1109/NCCC49330.2021.9428817.

[11] F. Alharbia, S. R. Masadeh, F. Alshrouf, "A Framework for the Generation of Class Diagram from Text Requirements using Natural language Processing," *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 10, no. 1, pp. 25–31, Feb. 2021, https://doi.org/10.30534/ijatcse/2021/041012021.

[12] Y. Meng and A. Ban, "Automated UML Class Diagram Generation from Textual Requirements Using NLP Techniques," *JOIV : International Journal on Informatics Visualization*, vol. 8, no. 3–2, pp. 1905–1915, Nov. 2024, https://doi.org/10.62527/joiv.8.3-2.3482.

[13] T. A. Alrawashdeh, A. A. Hnaif, M. Alrifaee, and M. S. Kamel, "An Intelligent Framework to Generate Use Case Diagrams and Class Diagrams from Requirements Documents." In Review, Aug. 14, 2024, https://doi.org/10.21203/rs.3.rs-4764870/v1.

[14] M. S. Osman, N. Z. Alabwaini, T. B. Jaber, and T. Alrawashdeh, "Generate use case from the requirements written in a natural language using machine learning," in *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, Apr. 2019, pp. 748–751, https://doi.org/10.1109/JEEIT.2019.8717428.

[15] M. S. Osman, N. Z. Alabwaini, T. B. Jaber, and T. Alrawashdeh, "Generate use case from the requirements written in a natural language using machine learning," in *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, Amman, Jordan, Apr. 2019, pp. 748–751, https://doi.org/10.1109/JEEIT.2019.8717428.

[16] Y. Rigou and I. Khriss, "A Deep Learning Approach to UML Class Diagrams Discovery from Textual Specifications of Software Systems," in *Intelligent Systems and Applications*, Amsterdam, Netherlands, 2023, pp. 706–725, https://doi.org/10.1007/978-3-031-16078-3_49.

[17] R. Saini, G. Mussbacher, J. L. C. Guo, and J. Kienzle, "Automated, interactive, and traceable domain modelling empowered by artificial intelligence," *Software and Systems Modeling*, vol. 21, no. 3, pp. 1015–1045, June 2022, https://doi.org/10.1007/s10270-021-00942-6.

[18] M. B. Chaaben, L. Burgueño, and H. Sahraoui, "Towards using Few-Shot Prompt Learning for Automating Model Completion," in *2023 IEEE/ACM 45th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, May 2023, pp. 7–12, https://doi.org/10.1109/ICSE-NIER58687.2023.00008.

[19] D. Rouabhia and I. Hadjadj, "Behavioral Augmentation of UML Class Diagrams: An Empirical Study of Large Language Models for Method Generation." arXiv, June 01, 2025, https://doi.org/10.48550/arXiv.2506.00788.

[20] V. Campanello, S. Shahbaz, V. Indykov, and D. Strüber, "On the Use of GPT-4 in the Reverse Engineering of Class Diagrams.," *The Journal of Object Technology*, vol. 24, no. 2, 2025, https://doi.org/10.5381/jot.2025.24.2.a14.

[21] J. A. Baktash and M. Dawodi, "Gpt-4: A Review on Advancements and Opportunities in Natural Language Processing." arXiv, May 04, 2023, https://doi.org/10.48550/arXiv.2305.03195.

[22] M. Weyssow, X. Zhou, K. Kim, D. Lo, and H. Sahraoui, "Exploring Parameter-Efficient Fine-Tuning Techniques for Code Generation with Large Language Models," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 7, May 2025, Art. no. 201, https://doi.org/10.1145/3714461.

[23] P. Giannouris and S. Ananiadou, "NOMAD: A Multi-Agent LLM System for UML Class Diagram Generation from Natural Language Requirements," arXiv, Nov. 27, 2025. https://arxiv.org/abs/2511.22409v1.

[24] K. Hölldobler, B. Rumpe, and I. Weisemöller, "Systematically deriving domain-specific transformation languages," in *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, Sept. 2015, pp. 136–145, https://doi.org/10.1109/MODELS.2015.7338244.

[25] Z. Babaalla, A. Jakimi, and M. Oualla, "LLM-Driven MDA Pipeline for Generating UML Class Diagrams and Code," *IEEE Access*, vol. 13, pp. 171266–171286, 2025, https://doi.org/10.1109/ACCESS.2025.3615828.

[26] Z. Babaalla, E. M. Bouziane, A. Jakimi, and M. Oualla, "From text-based system specifications to UML diagrams: A bridge between words and models," in *2024 International Conference on Circuit, Systems and Communication (ICCSC)*, Fes, Morocco, June 2024, pp. 1–6, https://doi.org/10.1109/ICCSC62074.2024.10616686.

[27] M. Hagiwara, Real-World Natural Language Processing: Practical applications with deep learning. Simon and Schuster, 2021.

[28] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Mar. 2019, pp. 4171–4186, https://doi.org/10.18653/v1/N19-1423.

[29] Y. Liu *et al.*, "RoBERTa: A Robustly Optimized BERT Pretraining Approach." arXiv, July 26, 2019, https://doi.org/10.48550/arXiv.1907.11692.

[30] M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy, "SpanBERT: Improving Pre-training by Representing and Predicting Spans," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 64–77, Jan. 2020, https://doi.org/10.1162/tacl_a_00300.

[31] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "XLNet: Generalized Autoregressive Pretraining for Language Understanding," in *Advances in Neural Information Processing Systems*, Vancouver, Canada, 2019, vol. 32.

[32] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, "MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers," in *Advances in Neural Information Processing Systems*, 2020, vol. 33, pp. 5776–5788.

[33] K. Clark, M. T. Luong, Q. V. Le, and C. D. Manning, "ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators." arXiv, Mar. 23, 2020, https://doi.org/10.48550/arXiv.2003.10555.

[34] Z. Babaalla, A. Jakimi, R. Saadane, and A. Chehri, "Towards automatic extraction of UML class diagrams: Creation of an annotated dataset for training deep models," *Procedia Computer Science*, vol. 270, pp. 3152–3161, Jan. 2025, https://doi.org/10.1016/j.procs.2025.09.440.

[35] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing," *ACM Computing Surveys*, vol. 55, no. 9, Jan. 2023, Art. no. 1-35, https://doi.org/10.1145/3560815.

[36] B. Liu, "Comparing sparse Llama 3 and Llama 2 models for on-device AI assistants." In Review, Sept. 09, 2024, https://doi.org/10.21203/rs.3.rs-4927672/v2.

[37] H. Thakkar and A. Manimaran, "Comprehensive Examination of Instruction-Based Language Models: A Comparative Analysis of Mistral-7B and Llama-2-7B," in *2023 International Conference on Emerging Research in Computational Science (ICERCS)*, Dec. 2023, pp. 1–6, https://doi.org/10.1109/ICERCS57948.2023.10434081.

[38] D. Guo *et al.*, "DeepSeek-Coder: When the Large Language Model Meets Programming -- The Rise of Code Intelligence." arXiv, Jan. 26, 2024, https://doi.org/10.48550/arXiv.2401.14196.

[39] H. Ashraf, S. M. Danish, S. Rahman, and Z. Sattar, "Toward Green Code: Prompting Small Language Models for Energy-Efficient Code Generation." arXiv, Oct. 07, 2025, https://doi.org/10.48550/arXiv.2509.09947.

[40] H. Herchi and W. B. Abdessalem, "From user requirements to UML class diagram." arXiv, Nov. 04, 2012, https://doi.org/10.48550/arXiv.1211.0713.

[41] R. Naik, "UDRA: Reflecting Natural Language Text in to UML Diagrams," *Spvryan's International Journal of Engineering Sciences & Technology*, vol. 2, no. 1, 2016.

[42] A. Z. Utama and D. S. Jang, "An Automatic Construction for Class Diagram from Problem Statement using Natural Language Processing," *Journal of Korea Multimedia Society*, vol. 22, no. 3, pp. 386–394, Mar. 2019.

[43] K. A. D. O. K. K. Arachchi, "AI Based UML Diagrams Generator," Ph.D. dissertation, University of Colombo, Sri Lanka.

[44] D. K. Deeptimahanti and R. Sanyal, "Semi-automatic generation of UML models from natural language requirements," in *Proceedings of the 4th India Software Engineering Conference*, Thiruvananthapuram, India, Oct. 2011, pp. 165–174, https://doi.org/10.1145/1953355.1953378.

[45] S. K.Shinde, V. Bhojane, and P. Mahajan, "NLP based Object Oriented Analysis and Design from Requirement Specification," *International Journal of Computer Applications*, vol. 47, no. 21, pp. 30–34, June 2012, https://doi.org/10.5120/7475-0574.

[46] Z. Babaalla, "zakariababaalla/Text-to-uml-to-code." Feb. 09, 2026, [Online]. Available: https://github.com/zakariababaalla/Text-to-uml-to-code.