Systematic Review of Automatic UML Diagram Generation

Nimsha Fernando and Chaman Wijesiriwardana

Abstract Unified Modeling Language (UML) diagrams are widely used to support system understanding, design communication, and traceability in software engineering, yet their manual creation and maintenance have become increasingly difficult in fast-paced, requirements-driven development environments. This systematic literature review addresses the problem of how effectively UML diagram generation can be automated and which approaches and diagram types demonstrate the strongest empirical performance. Following the PRISMA framework, peer-reviewed studies published between 2015 and 2025 were systematically identified, screened, and analyzed. The reviewed methods span rule-based techniques, hybrid machine-learning pipelines, transformer-based models, large language models, and code-driven extraction approaches. The findings reveal a clear methodological shift from deterministic rule-oriented solutions toward data-driven and learning-based techniques that better accommodate linguistic variability and evolving requirements. Structural UML artefacts, particularly class diagrams, consistently exhibit the highest reported effectiveness, frequently achieving accuracy levels above 90%, while behavioral and interaction-oriented diagrams such as sequence and activity diagrams show lower and more variable performance. Despite this progress, persistent challenges remain, including sensitivity to ambiguous requirements, difficulties in relationship inference, and semantic inconsistencies in generative outputs, indicating that fully autonomous UML generation is not yet feasible. Overall, the results suggest that automated UML generation is most effective when applied as an assistive capability supported by validation mechanisms and human oversight. From a translational perspective, these insights support the development of more reliable modelling tools that can improve traceability, design consistency, and decision-making in real-world software engineering practice, particularly in large-scale and safety-critical systems.

***Index Terms*— Automated UML generation, Model-driven development, Requirements engineering, Unified Modeling Language**

I. Introduction

UNIFIED Modeling Language (UML) diagrams are commonly used in software engineering to describe complex systems in a clear and structured way. Koç et al. report that UML diagrams are widely used to support system analysis, design communication, and documentation across the software development lifecycle, making them one of the most enduring modelling formalisms in the field [1]. In a similar vein, Mornie et al. explain that UML diagrams play a key role in visualizing requirements artefacts such as user stories, thereby supporting shared understanding between technical and non-technical stakeholders [2]. From a conceptual standpoint, Valiente et al. demonstrate that ontology-based and model-driven approaches strengthen UML's importance by explicitly linking problem-domain concepts with technical design representations [3]. Bozyigit et al. further observe that UML models are frequently used as a bridge between software requirements and conceptual models, reinforcing their relevance in requirements-driven and model-driven development contexts [4]. Collectively, these studies establish UML as a central artefact for communication, traceability, and design reasoning in software engineering.

Although UML diagrams are still widely used, keeping them up to date has become increasingly difficult in modern development work. Stettina and Heijstek note that agile ways of working place emphasis on short iterations and frequent changes, which often means documentation is postponed or treated as a secondary task [5]. Umar et al. describe a similar situation, observing that ongoing requirement changes and fast-moving codebases make it challenging to keep UML models aligned with what is actually implemented in agile, model-driven projects [6].Torre et al. observe in their mapping study that this misalignment commonly leads to UML diagrams becoming outdated or only partially accurate, which in turn reduces their practical value [7]. In many projects, diagrams are therefore no longer treated as dependable design artefacts but are instead used informally or abandoned altogether. This situation weakens traceability across requirements, design, and implementation, limiting the usefulness of UML for long-term system understanding and maintenance.

In response to these issues, researchers have increasingly examined ways of generating UML diagrams automatically by making use of artefacts already produced during

Nimsha Fernando is with the APIIT School of Computing, Colombo, Sri Lanka. (Email: cb011990@students.apiit.lk)

Chaman Wijesiriwardana is a Senior Lecturer in the Department of Information Technology, University of Moratuwa, Sri Lanka. (Email: chaman@uom.lk)

ISSN: 2961-5992

software development. Ahmed et al. describe the conversion of natural language requirements into UML models as an important research direction, primarily motivated by the need to reduce manual modelling effort and maintain closer alignment between requirements and design representations [8]. Li et al. point out that machine learning has gradually found its way into requirements engineering, especially for practical tasks like pulling out relevant information and sorting it in ways that can later support UML modelling [9]. However, studies also make it clear that these techniques do not work equally well in all situations. Akay et al. report that machine learning techniques are able to pick out design-related information from requirements text, but they also note that results vary greatly depending on how features are defined and how outputs are checked [10]. Rahman et al. describe a comparable limitation, showing that deep learning can assist with identifying non-functional requirements only when the training data has been prepared carefully and labelled in a consistent manner [11].

More recent studies have begun to widen the scope of this work by looking beyond text alone. Meng et al. illustrate that class diagrams can be produced directly from written requirements through the use of natural language processing techniques [12]. By contrast, Conrardy and Cabot take a different approach, focusing on the recovery of UML models from sketches and existing diagram images using language-based methods [13]. Zhang et al. look at how ChatGPT can be used to assist with class diagram generation from natural language descriptions, positioning it as a supporting aid rather than a replacement for human judgement in modelling tasks [14]. Related studies by Bates et al. and Johnson and Martinez explore multimodal language models for generating code from UML artefacts and report that, although the results are encouraging, technical limitations remain [15], [16].At the same time, Pei et al. and De Bari point out that issues such as ambiguous language, reliance on unstated domain knowledge, and the ongoing need for human checking continue to limit how far UML generation can be automated in real settings [17], [18].

Although many automated approaches have been proposed, the literature remains fragmented across UML diagram types, input modalities, and technical strategies. Abdelnabi et al. observe that most studies concentrate on class-diagram generation, while behavioral and interaction-oriented diagrams receive limited attention [19]. Ahmad et al. and Rozo et al. report uneven empirical evidence across diagram types and note that differing evaluation methods make results hard to compare [20], [21]. Recent reviews by Zadenoori et al. and Ronanki et al. indicate that advances in large language models have expanded the solution space without yet providing consolidated, comparable evidence of effectiveness across UML artefacts [22], [23]. Taken together, prior work reveals a clear gap: despite many proposals, there is no single synthesis that compares diagram types, input sources, and evaluation practices from a consistently UML-centered perspective.

This review addresses that gap by examining how automated UML techniques published between 2015 and 2025 have been applied in practice, focusing on the modelling challenges they confront and the diagram types that perform most consistently. Its contribution is threefold: it consolidates fragmented evidence into a UML-centered synthesis, it provides a quantitative comparison of reported performance across diagram types and techniques, and it derives practical implications for tool builders and practitioners in industrial and safety-critical contexts. The following research questions (RQs) guide the discussion.

- *RQ1: To what extent have automated approaches addressed the challenges of UML diagram generation, and what limitations remain unresolved?*

- *RQ2: Which UML diagram types and supporting techniques demonstrate the strongest empirical effectiveness across different input artefacts and application contexts?*

The remainder of this paper is organized as follows. Section II reviews related work; Section III outlines the methodology; Section IV presents the results and discussion; and Section V concludes with future research directions and industrial applications.

II. Related Work

Research on automatic UML generation has developed across several adjacent fields, including requirements engineering, model-driven development, and natural language processing, rather than along a single unified line. To keep the discussion focused, prior work is grouped into three themes: earlier consolidating reviews and the coverage gaps they expose; requirements-driven generation through rule-based, transformation, and learning techniques; and the recent turn to large language models.

A. Prior Reviews and Coverage Gaps

Several mapping studies and surveys have attempted to consolidate the field. Torre et al. [7] classify synthesis approaches by input source, target diagram, and degree of automation, observing that class and sequence diagrams dominate the literature while state-based and activity diagrams are comparatively neglected, and that many techniques are validated only on small, controlled examples. Ahmed et al. [8] reach similar conclusions for approaches generating UML from natural-language requirements, repeatedly identifying ambiguous input and incomplete diagram coverage as limiting factors, while

ISSN: 2961-5992

Mornie et al. [2] report that techniques derived from user stories rarely progress beyond early design. In the testing domain, Ahmad et al. [20] find that model-based testing increasingly relies on automatically generated activity diagrams yet still depends on rule-based transformation when requirements are ambiguous, and Rozo et al. [21] confirm that class and sequence diagrams receive disproportionate attention relative to behavioral artefacts. Across these reviews the message is consistent: coverage across diagram types and development stages is uneven, and inconsistent evaluation makes cross-study comparison difficult.

B. Requirements-Driven Generation:Rules, Transformation, and Learning

A large body of work derives UML models directly from textual requirements. Early and still widely used approaches rely on linguistic rules and heuristics, treating nouns as candidate classes and verbs as relationships, or on domain ontologies [4], [24], [25]. Such methods are transparent and effective within narrow, well-structured domains but transfer poorly to new contexts without substantial rework, and their evaluation on small or bespoke benchmarks limits comparability [4], [26]. Natural language processing techniques such as part-of-speech tagging, dependency parsing, and named entity recognition underpin many of these pipelines, but Dalpiaz et al. [27] show that ambiguous, incomplete, and domain-specific requirements directly degrade the quality of the resulting artefacts.

To address this brittleness, the field has moved toward learning-based methods. Li et al. [9] document a broad shift toward data-driven techniques in requirements engineering, while Abdelnabi et al. [19] find that, although automation accelerates early class diagram modelling, manual refinement of classes, attributes, and relationships is still required. Model transformation research complements this line. Kahani et al. [28] show that declarative transformation languages are easier to maintain at low complexity but lose that advantage as transformations grow, and several studies report that combining declarative and imperative styles is more robust for complex or loosely defined requirements [29], [30]. Closely related is requirements traceability, which sustains the links between requirements, models, and code on which automated generation depends. Goulao et al. [31] and García García et al. [32] demonstrate that model driven traceability reduces the manual, error prone effort of maintaining these links in large, evolving systems.

C. Large Language Models and the Current Frontier

The most recent work applies large language models to UML tasks, typically in zero-shot or few-shot configurations [22], [33], [34]. Their appeal lies in strong generalization and the ability to produce diagrams without task-specific retraining, but empirical evaluations expose new failure modes. Ferrari et al. [33] document hallucinated entities, missing interactions, and incorrect relationships in generated sequence diagrams, and show that careful prompt design measurably reduces these errors. Bouali et al. [35] find that human review remains necessary to ensure correctness, and Rouabhia et al. [36] report persistent difficulty in maintaining semantic consistency during class diagram augmentation.

TABLE I
SUMMARY OF KEY LITERATURE ON AUTOMATIC UML DIAGRAM GENERATION

Study	Main Contribution	Key Limitation
Torre et al. (2018)	Systematic mapping of UML synthesis approaches	Limited validation on industrial datasets
Ahmad et al. (2019)	Use of activity diagrams for model-based testing	Requires clearly defined requirements
Bozyigit et al. (2021)	Linking requirements to UML models	Heuristic-based approach; inconsistent evaluation
Abdelnabi et al. (2021)	UML class diagram generation from requirements	Requires human refinement
Koç et al. (2021)	Analysis of UML usage across software engineering	Fragmented effectiveness evidence
Dalpiaz et al. (2018)	Application of NLP in requirements engineering	Ambiguity and incompleteness issues
Kahani et al. (2018)	Comparison of model transformation languages	Hybrid approaches require expertise
Zadenoori et al. (2025)	Use of LLMs in requirements engineering	Limited use of advanced prompting
Ronanki et al. (2024)	Generative AI for requirements processing	Hallucinations; need for domain-specific data

Beyond single prompt use, Ronanki et al. [37] propose NOMAD, a multi agent architecture for class diagram generation that improves accuracy over single agent baselines. Synthesizing the LLM literature, Zadenoori et al. [22] and Ronanki et al. [23] note that most studies still rely on simple prompting rather than retrieval augmented generation or fine tuning, and that grounding, domain

ISSN: 2961-5992

accuracy, and reliability remain unresolved, indicating that these models are not yet suitable for unsupervised use. Across all three themes the evidence is uneven, structural diagrams are better served than behavioral ones, and evaluation inconsistencies limit comparability, motivating the UML centered synthesis undertaken here [1], [7], [22].

III. METHODOLOGY

This systematic literature review was carried out with the aim of carefully examining existing research on the automatic generation of Unified Modeling Language (UML) diagrams. To guide the review process, the PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) framework was followed, as outlined by Page et al. [38]. This framework was chosen because it offers a practical way to organize the review process, from identifying relevant studies to deciding which papers should ultimately be included. Using this structure helped keep the selection process consistent and reduced the likelihood of overlooking relevant work, resulting in a more reliable overview of current research in automated UML diagram generation.

A. Keywords Utilized

A structured keyword strategy was adopted to ensure comprehensive retrieval of relevant literature while limiting false positives. Keywords were organized into three conceptual categories reflecting the interdisciplinary nature of UML automation research.

- UML terminology and included terms: *"UML", "Unified Modeling Language", "Class diagram", "Sequence diagram", "Activity diagram", "Use case diagram*").
- Automation and Transformation Concepts: *("Automatic generation", "Model transformation", "Reverse engineering", "Diagram synthesis", "Model-based testing"*).
- Requirements and Modelling Contexts: *("Natural Language Requirements", "User Stories","Requirements-To-Model"*).

These keyword groups were combined using Boolean operators (AND/OR) and iteratively refined during pilot search executions to improve recall and precision. This refinement process ensured that the final search strategy captured both established and emerging approaches to UML automation.

B. Databases and Frameworks

The search was carried out across several established digital libraries, including IEEE Xplore, the ACM Digital Library, SpringerLink, ScienceDirect, Scopus, and Web of Science, selected because they regularly publish peer reviewed journal and conference papers in UML, requirements engineering, and model driven development. Google Scholar was also consulted to identify relevant grey literature, including technical reports and preprints, where their methodological quality could be assessed. Searching across multiple sources inevitably produced duplicate records; rather than a drawback, this overlap improved coverage and reduced missing key studies.

C. PRISMA Workflow

An overview of the study selection process is presented in the PRISMA flow diagram shown in Figure 1.

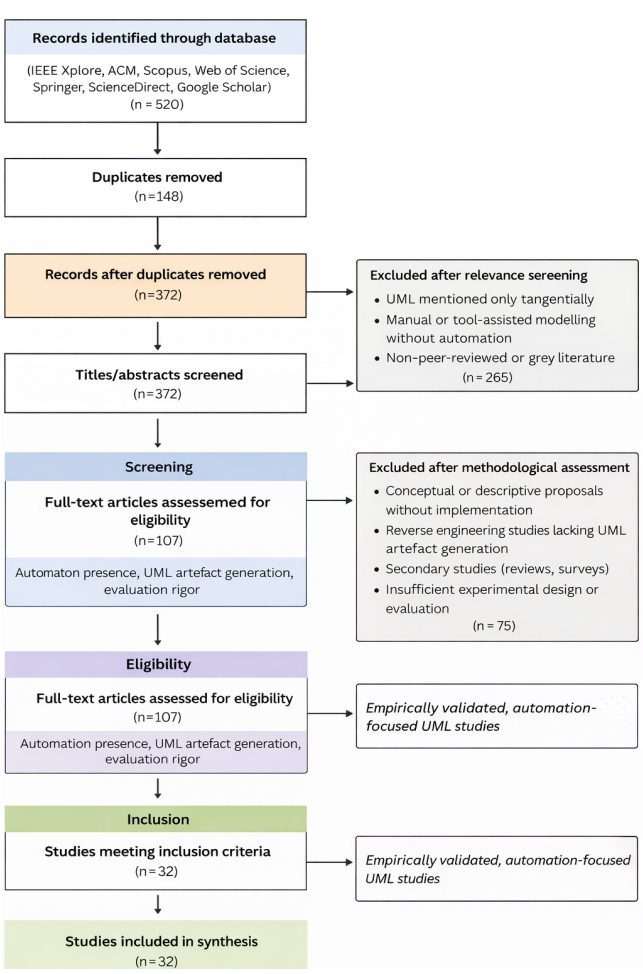

Fig.1. PRISMA 2020 flow diagram of the study-selection process, showing the identification, screening, and eligibility stages that narrowed 520 initial records to 32 primary studies included in the synthesis.

1) *Identification Phase:*

The review began by collecting material from a range of academic databases, including IEEE Xplore, ACM, SpringerLink, ScienceDirect, Scopus, and Web of Science, alongside Google Scholar, selected for their strong coverage of software engineering research, which initially produced 520 publications dated between 2015 and 2025.

ISSN: 2961-5992

At this stage, the aim was not to narrow the scope too quickly, but to ensure that relevant work on automated UML diagram generation was not missed due to overly restrictive search terms. After the initial set of records had been compiled, repeated entries were identified and removed, as overlapping coverage across databases meant duplication was expected. This was done first with the help of reference management software and then checked manually by comparing titles, author information, and DOI details. After deduplication, 372 studies were retained for review.

2) *Screening Phase:*

The next stage involved reviewing the titles and abstracts of these 372 studies. Each record was assessed against a set of relevance criteria defined in advance. Papers were excluded when they did not involve UML diagrams, focused only on manual modelling practices or educational contexts, addressed modelling approaches unrelated to UML, or were not peer-reviewed. Through this extensive filtering process, a large number of papers were found to be outside the scope of the review, resulting in the exclusion of 265 studies. In several cases, exclusions were based on a lack of technical detail or an absence of any clear automation component. When uncertainty arose, a conservative decision was taken to retain the study for further examination at the full-text stage. The remaining 107 papers were considered suitable for full-text review.

3) *Eligibility Phase:*

A more detailed assessment was then carried out by reading the full texts of these 107 studies. To remain within the scope of this review, eligible studies needed to either propose or evaluate techniques for automated UML diagram generation, treat UML diagrams as core or intermediate artefacts, and provide empirical evidence or clear methodological contributions. Only peer-reviewed publications published between 2015 and 2025 were considered. At this point, several studies were removed because they did not provide enough substance for detailed analysis. Some papers discussed UML generation only at a conceptual level, while others were secondary reviews that did not contribute to new findings. A number of studies focused entirely on reverse engineering from source code and did not consider requirements as an input, which placed them outside the scope of this review. In a few cases, the methodological details were too limited to allow proper evaluation. After applying these exclusions, 75 studies were not taken forward.

4) *Inclusion Phase:*

The final group consisted of 32 studies that were examined in more detail. This was not intended to represent the largest possible collection of papers, but rather a set that could be analyzed carefully without losing sight of the wider research landscape. Working with this group made it possible to see broader patterns in how automated UML generation has developed over time, including a gradual move away from strictly rule-based solutions toward more data-driven techniques such as machine learning, transformer models, and recent language-model-based approaches. Studies were favored when they explained their automation mechanisms in sufficient detail and provided experimental evidence to support their claims, as this made it easier to assess practical usefulness as well as remaining limitations. Looking across this set also helped reveal recurring strengths and weaknesses that appear across different lines of work, which in turn informed how the research questions of this review were addressed. This also helped highlight areas where further investigation is needed, particularly in terms of evaluation consistency and real-world applicability.

IV. Results and Discussion

A. Analytical Overview of the Selected Studies

The 32 primary studies can be analyzed along three dimensions: the methodological paradigm they adopt, the UML artefacts they target, and the maturity of the evidence they provide. Viewed this way, the corpus reveals an interpretable structure rather than a simple chronological progression. Methodologically, the studies cluster into four paradigms: rule based, hybrid combining machine learning with rule-based validation, transformer based, and large language model based, whose relative prominence shifts across the 2015 to 2025 window, as summarized in Fig. 2. The analytical significance of this distribution is not that newer techniques simply appear later, but that each paradigm enters the field to address a specific documented weakness of its predecessor. Hybrids emerge to reduce the brittleness of hand-crafted rules under linguistic variability [9], [39], transformers are adopted to capture contextual dependencies that local pattern methods miss [40], [41], and large language models are introduced to reduce the need for task specific retraining altogether [33], [34]. The trajectory therefore reflects a problem driven response to recurring limitations rather than the adoption of novelty for its own sake [8], [7].

A second observation concerns the mismatch between methodological effort and artefact coverage. Although the methods grow more sophisticated, research attention remains concentrated on structural artefacts, especially class diagrams, while behavioral and interaction-oriented diagrams remain comparatively under studied across all four paradigms [7], [21], [19]. The field is thus methodologically deep but representationally narrow, as advances in technique have not been matched by a broadening of modelling targets.

ISSN: 2961-5992

Finally, the strength of evidence varies systematically with paradigm. Rule based and hybrid studies tend to report results on small curated requirement sets, whereas transformer based and LLM based studies increasingly report quantitative accuracy figures yet often omit dataset descriptions and apply inconsistent evaluation criteria [1], [9], [42]. The apparent improvement in reported performance over time must therefore be interpreted cautiously, since it partly reflects changes in evaluation practices rather than true improvement under comparable conditions. These three observations, paradigm substitution, narrow artefact coverage, and uneven evidence quality, collectively frame the analysis presented for the two research questions below.

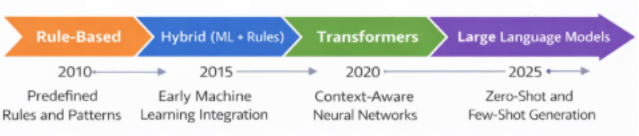

Fig. 2. Evolution of automated UML-generation approaches, 2010–2025, showing the progressive shift from rule-based methods to hybrid, transformer-based, and large-language-model techniques, each emerging to address limitations of the preceding paradigm.

B.RQ1: To what extent have automated approaches addressed the challenges of UML diagram generation, and what limitations remain unresolved?

Across the studies reviewed, several distinct ways of approaching UML automation can be observed, each reflecting different attempts to deal with problems that have existed in this area for some time [8], [9], [7]. Much of the earliest work relies on rule-based techniques [24], [25]. In these approaches, linguistic patterns are explicitly defined, such as treating nouns as candidate classes and verbs as indicators of relationships, as illustrated in the methods proposed by Bashir et al. [25] and Waragde and Sharma [24]. These systems are generally straightforward to implement and can work reasonably well when requirements are written in a structured manner and use consistent terminology [39]. An additional advantage often noted is the clear traceability they provide between the original text and the generated model, which is particularly valued in regulated or safety-critical contexts [7], [24].

At the same time, the literature consistently points out that such approaches struggle once they are applied beyond controlled settings. Ambiguous language, inconsistent naming, and informal phrasing all have a noticeable impact on performance [9], [27]. Since real requirements rarely follow strict syntactic rules, purely rule-based solutions tend to behave unpredictably when exposed to realistic inputs [39], [43]. As limitations in purely rule-based methods became clearer, some researchers began experimenting with combinations of learning-based techniques and explicit rules [43]–[45]. In these approaches, machine learning is typically used to handle variation in language, while rules are kept in place to represent domain knowledge and maintain semantic consistency [43], [46]. Reported results suggest that this balance can make systems more stable. For example, Nair and Thushara [45] note improvements of around 15–20% in actor identification when hybrid methods are used instead of a single technique, and Sanyal et al. [43] describe gains in consistency when learned outputs are checked using rule-based validation. Other alternatives have also been considered. Omer and Eltyeb [46], for instance, show that case-based reasoning can reach similar performance levels without relying on large training datasets. Even so, these studies make it clear that effective automation still depends heavily on how well domain knowledge is defined, rather than on learning mechanisms alone [45], [46].

Alongside these developments, attention has increasingly shifted toward transformer-based neural models [40], [41], [47]. Because these models use attention mechanisms to capture context across longer spans of text, they are better suited to the complexity typically found in real requirements descriptions than earlier methods focused on local patterns [40], [41]. Empirical findings from recent studies suggest that transformer-based models can be effective for certain sub-tasks in UML generation. For example, Shweta et al. report improvements when RoBERTa is fine-tuned on requirements datasets, while work by Babaalla et al. shows high accuracy for entity extraction using models from the BERT family [41], [47], [48]. These results indicate clear progress in identifying relevant elements from text. At the same time, several studies point out that success at the extraction level does not automatically extend to higher-level modelling tasks. In particular, inferring relationships and cardinalities remains difficult, limiting the extent to which UML generation can be fully automated [40], [47].

More recent research turns to large language models in zero-shot or few-shot configurations [33], [35], [34], attractive for their generalization and ability to generate diagram representations without retraining [33], [34]. Yet empirical evaluations reveal new challenges: Ferrari et al. document hallucinated entities, missing interactions, and incorrect relationships in generated sequence diagrams [33], and Bouali et al. show that human review remains necessary to ensure correctness [35]. Reviews by Zadenoori et al. and Ronanki et al. highlight unresolved problems of grounding, domain accuracy, and reliability, indicating these models are not yet suitable for unsupervised use [22], [23]. The persistence of these challenges across techniques is summarized in Table II, which shows that no single category of approach simultaneously resolves ambiguity, semantic correctness, and scalability.

An alternative research direction focuses on code-driven extraction, where UML diagrams are derived from source code or execution traces rather than requirements text [49], [7]. These approaches often achieve high precision when implementation artefacts are available, addressing the

ISSN: 2961-5992

common situation in which documentation lags behind code [49]. However, their applicability is limited to systems with accessible and stable implementations, making them unsuitable for early-stage design or greenfield projects [7], [24]. This confines their usefulness largely to maintenance and reverse-engineering scenarios rather than initial system modelling, and reduces their relevance in requirement-centric development contexts where system behavior is still being defined. Because it operates on a fundamentally different input modality, code-driven extraction sits outside the four-paradigm progression described earlier, representing a complementary rather than competing line of work, one that could potentially be paired with requirements-based methods to cross-validate generated models against implementation artefacts [22], [33], [49]. This comparative perspective is consolidated in Table II, which contrasts the challenges addressed and the remaining limitations of the main categories of UML automation approaches.

TABLE II
COMPARISON OF APPROACHES FOR AUTOMATED UML DIAGRAM GENERATION

Approach	Challenges Addressed	Remaining Limitations and Key Studies
Rule-Based	Transparent extraction from structured text	Fails on ambiguous or varied requirements; Key studies: [24], [39]
Hybrid (ML + Rules)	Flexibility with semantic validation	Requires domain knowledge engineering; Key studies: [44], [43], [46]
Transformers	Context-aware entity extraction (≈95% accuracy)	Relationship inference and cardinality challenging; Key studies: [40], [41], [47]
Large Language Models	Zero-shot generalization across diagram types	Hallucinations, omissions, semantic errors; Key studies: [33], [35], [34]
Code-Driven	High precision when source code available	Limited to reverse-engineering scenarios; Key studies: [49]

C.RQ2: Which UML diagram types and supporting techniques demonstrate the strongest empirical effectiveness across different input artefacts and application contexts?

Reported effectiveness differs markedly across UML diagram types, largely because of differences in structural complexity and semantic demands [1], [9], [7]. Class diagrams receive the most attention and consistently achieve the strongest results [8], [7]; their close correspondence with object-oriented constructs makes them relatively straightforward to derive from both requirements text and source code [49]. This correspondence builds on the noun-to-class heuristics underlying earlier rule-based work, which later learning-based methods have refined rather than replaced [24], [25]. Across studies, accuracy for class-related entity extraction is frequently above 90%, and in some deep-learning approaches exceeds 95% [47], [48]. This disparity across diagram types is illustrated in Fig. 3, which shows the consistently stronger effectiveness reported for class diagrams relative to behavioral and interaction-oriented artefacts. These findings suggest that class-diagram generation has reached a comparatively mature stage of automation in well-defined, structurally consistent contexts [8], [7], consistent with the artefact-coverage imbalance noted earlier [7], [21], [19]. The lower, more variable performance of behavioral diagrams reflects the greater difficulty of inferring temporal dynamics and interaction semantics from natural language alone [33], [39]; consequently, automation effectiveness is increasingly evaluated per diagram type [9], [7]. The remainder of this section examines that variation in detail.

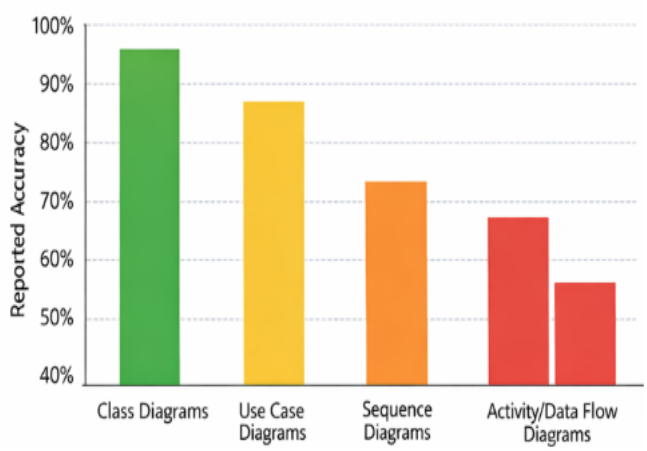

Fig. 3. Reported accuracy by UML diagram type, showing class diagrams (≈96%) outperforming use-case (≈87%) and sequence diagrams (≈73%), with activity and data-flow diagrams lowest (≈56–67%), reflecting the difficulty of automating behavioral semantics.

In contrast, use case and sequence diagrams show more moderate performance. These diagram types must represent temporal behavior and interactions, which are inherently more difficult to infer from textual descriptions alone [33], [39]. Reported correctness typically falls within the mid-70% to low-80% range, depending on the quality of the input requirements and the techniques applied [33], [44], [45]. Data flow and activity diagrams receive considerably

ISSN: 2961-5992

less attention and generally demonstrate lower empirical performance [34], [50]. The available studies report reduced correctness, reflecting ongoing difficulties in modelling control flow, concurrency, and dynamic system behavior [50]

Certain supporting techniques recur in studies reporting stronger outcomes. Careful preprocessing of requirements text, including normalization and sentence segmentation, is consistently associated with reduced downstream errors [39], [43]. Context-aware representations, particularly from transformer-based models, improve robustness by capturing relationships beyond local word patterns [40], [41]. Hybrid strategies that combine machine learning with rule-based validation further enhance reliability by enforcing domain constraints [43], [45], [46]. For large language models, prompt design is especially influential: Ferrari et al. [33] show that structured prompts—specifying output formats, providing in-context examples, or requesting step-by-step reasoning—substantially reduce the range of error types in generated sequence diagrams. A smaller body of work treats specifications and implementation artefacts as relational structures rather than linear text; Shivamurthy et al. [50] show that this perspective uncovers connections difficult to identify through sentence-level processing alone. These supporting techniques and their reported impact are summarized in Table III.

TABLE III
SUPPORTING TECHNIQUES WITH THE STRONGEST EMPIRICAL EFFECTIVENESS

Technique	Applicable Diagrams	Empirical Impact and Key Studies
Text Preprocessing	All Types	15–25% error reduction, key studies [17], [25], [44]
Context-Aware Transformers	Class, Use Case	15–20% accuracy gain with approximately 95% entity extraction, key studies [40], [26], [48]
Hybrid ML + Rules	Class, Use Case, Sequence	20–30% improvement over single methods, key studies [43], [45], [46]
Prompt Engineering	All Types (LLM-Based)	Score improvement from 3.2 to 4.1 out of 5, reduction of error types from 23 to 6–8, key studies [33], [35]
Graph-Based Analysis	Sequence, Data Flow	Superior relationship extraction, key studies [50]

Table IV consolidates these strands, with its recurring "NR" entries underscoring the benchmarking gap noted throughout this review.

TABLE IV
REPORTED QUANTITATIVE PERFORMANCE ACROSS REPRESENTATIVE PRIMARY STUDIES

Approach (refs)	TARGET DIAGRAM	Reported Quantitative Result	Evaluation Basis / Dataset
Rule-based [24], [25], [39]	Class, use case	Reliable only on structured text; degrades sharply on ambiguous input	Small curated requirement sets (size NR)
Hybrid ML + rules [43], [45], [46]	Class, use case, sequence	+15–20% actor identification [45]; 20–30% gain over single methods; CBR reaches comparable accuracy without large training data [46]	Project-specific requirement sets (size NR)
Transformer-based [40], [41], [47], [48]	Class	≈95% entity extraction; class accuracy >90%, exceeding 95% in some deep-learning models	BERT/RoBERTa fine-tuned on requirement corpora (size NR)
Large language models [33], [35]	Sequence, class	Prompt design raised quality score from 3.2 to 4.1 of 5 and reduced distinct error types from 23 to 6–8 [33]; human review still required [35]	Requirement-to-diagram prompts; expert / teaching-assistant scoring
Code-driven extraction [49]	Class, sequence	High precision when source code is available	Implementation artefacts (size NR)
Aggregate by diagram type (Fig. 3)	Class / use case / sequence / activity–DFD	≈96% / ≈87% / ≈73% / ≈56–67%	Aggregated across reviewed studies

D. Synthesis and Implications

Bringing the preceding analysis together, three implications emerge for both research and practice. First, progress in UML automation is best understood as cumulative refinement rather than paradigm replacement, as each generation of techniques addresses specific weaknesses of the previous one while inheriting unresolved problems such as relationship inference and semantic validation [40], [41], [47]. Second, the maturity of automation is artefact dependent, with class diagram generation reaching a level of reliability suitable for tool assisted use, whereas behavioral and architectural diagrams remain research challenges because the temporal and dynamic information they encode is typically under specified in natural language input [20], [21], [33]. Third, and most consequentially for adoption, even the strongest large language models continue to produce substantive errors, including hallucinated entities, omitted interactions, and inconsistent relationships, which necessitate human verification [22], [23], [33]. These implications converge on a single practical conclusion: automated UML generation is most dependable when organized as a layered, human in the loop pipeline. Fig. 4 illustrates such a configuration, in

ISSN: 2961-5992

which preprocessing, learning based extraction, and rule-based validation form successive stages whose output is reviewed and refined by a human modeler, positioning automation as an assistive rather than autonomous capability [1], [7].

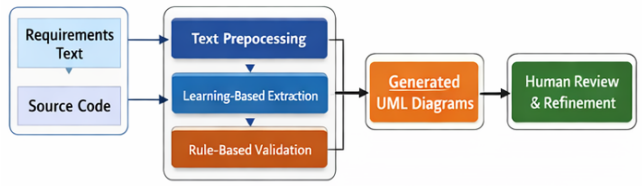

Fig.4. Conceptual human-in-the-loop pipeline for automated UML generation, combining preprocessing, learning-based extraction, and rule-based validation with human review, reflecting an assistive rather than autonomous role.

V. Conclusion

This systematic literature review examined research on the automatic generation of UML diagrams and synthesized evidence across rule-based, machine-learning, transformer-based, and large-language-model-driven approaches. The findings demonstrate a clear evolution from rigid rule-oriented pipelines toward hybrid and learning-based techniques that better accommodate linguistic variability and evolving requirements. Structural UML artefacts, particularly class diagrams, consistently exhibit the highest automation effectiveness owing to their strong correspondence with object-oriented abstractions and source-code structures. In contrast, behavioral and architectural diagrams remain harder to automate, reflecting challenges in modelling temporal interactions, concurrency, and implicit system behavior from textual artefacts.

Overall, the evidence indicates that automatic UML generation is most effective as an assistive capability that accelerates modelling and improves consistency while preserving human control over interpretation and validation. Persistent challenges, including requirement ambiguity, domain-specific semantics, and factual inconsistencies in generative outputs, limit the feasibility of fully autonomous modelling.

Building on these findings, several research directions warrant focused attention. First, the pronounced imbalance in artefact coverage calls for dedicated work on behavioral and interaction-oriented diagrams, namely sequence, activity, and state-machine generation, where temporal and concurrency semantics remain poorly handled [20], [21], [33]. Second, the field would benefit substantially from shared, openly available benchmark datasets and standardized evaluation protocols, without which the rising accuracy figures reported across studies cannot be compared on equal terms [1], [9], [42]. Third, hybrid pipelines that combine requirements-based generation with code-level extraction offer a promising route to cross-validate generated models against implementation artefacts, mitigating the hallucination and omission errors observed in purely generative approaches [22], [33], [49]. Fourth, for large language models specifically, research should move beyond zero- and few-shot prompting toward retrieval-augmented generation, fine-tuning on domain corpora, and multi-agent architectures, which early results suggest can improve grounding and accuracy [22], [37]. Finally, systematic study of prompt design and of automated consistency-checking mechanisms is needed to make generative outputs verifiable rather than merely plausible [33], [35].

From an industrial standpoint, the assistive model of UML automation has immediate practical relevance. Integrated into CASE and model-driven development tools, requirements-to-model generation can accelerate early design and reduce the effort of keeping diagrams synchronized with rapidly changing agile codebases [5], [6]. Automated traceability between requirements, models, and code can support impact analysis and change management in large, long-lived systems, and is especially valuable in regulated or safety-critical domains where demonstrable links between requirements and design are mandated [31], [32]. Code-driven extraction can assist the maintenance and documentation of legacy systems whose implementation has outpaced its design records [49], while model-based testing workflows can draw on generated behavioral models to derive and prioritise test cases [20], [21]. In each setting, the evidence reviewed here indicates that the greatest near-term value lies not in replacing the modeller but in augmenting expert judgement with fast, validated, and traceable automation. Achieving this requires standardized evaluation and hybrid validation.

References

[1] H. Koç, A. M. Erdoğan, Y. Barjakly, and S. Peker, "UML diagrams in software engineering research: A systematic literature review," Proceedings, vol. 74, no. 1, p. 13, 2021, doi: 10.3390/proceedings2021074013

[2] M. N. Mornie, N. Jali, S. N. Junaini, E. Mit, W. S. Cheah, and S. Saee, "Visualisation of user stories in uml models: A systematic literature review," Acta Informatica Pragensia, vol. 12, no. 2, pp. 419–438, 2023. [Online]. Available: https://doi.org/10.18267/j.aip.212

[3] M. C. Valiente, G. Escalona, and M. J. Escalona, "An ontology-based and model-driven approach for software requirements," Journal of Software and Systems Modeling, vol. 10, no. 2, pp. 213–232, 2011. [Online]. Available: http://www.cc.uah.es/drg/j/Valiente

[4] F. Bozyiğit, Ö. Aktaş, and D. Kılınç, "Linking software requirements and conceptual models: A systematic literature review," Eng. Sci. Technol., Int. J., vol. 24, no. 1, pp. 71–82, 2021, doi: 10.1016/j.jestch.2020.11.006

[5] C. J. Stettina and W. Heijstek, "Necessary and neglected? An empirical study of internal documentation in agile software development teams," in Proc. 29th ACM Int. Conf. Design of Communication (SIGDOC), Pisa, Italy, 2011, pp. 159–166, doi: 10.1145/2038476.2038509

ISSN: 2961-5992

[6] M. A. Umar, K. Lano, and A. K. Abubakar, "Automated requirements engineering framework for agile model-driven development," Frontiers in Computer Science, vol. 7, p. 1537100, 2025. [Online]. Available: https://www.frontiersin.org/journals/computer-science/articles/10.3389/fcomp.2025.1537100/pdf

[7] D. Torre et al., "Uml diagram synthesis techniques: A systematic mapping study," in Proc. ACM/IEEE Int. Workshop on Modelling in Software Engineering (MiSE), 2018, pp. 33–40. [Online]. Available: https://doi.org/10.1145/3193954.3193957

[8] S. Ahmed, A. Ahmed, and N. U. Eisty, "Automatic transformation of natural to unified modeling language: A systematic review," in Proc. IEEE/ACIS 20th Int. Conf. Software Engineering Research, Management and Applications (SERA), 2022, pp. 112–119. [Online]. Available: https://doi.org/10.1109/SERA54885.2022.9806783

[9] T. Li, N. Ghaff, K. Hassani, and A. Zarai, "Machine learning for requirements engineering: A systematic review," Journal of Systems and Software, vol. 198, p. 111601, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095058492400082X

[10] H. Akay, F. Morbitzer, and K. H. Law, "Automating design requirement extraction from text with machine learning," in ASME Int. Design Engineering Technical Conf., 2021, pp. 1–12. [Online]. Available: https://dspace.mit.edu/bitstream/handle/1721.1/154885/v03bt03a035-detc2021-66898.pdf

[11] A. Rahman, M. K. Hassan, and S. Parvez, "A deep learning framework for non-functional requirement classification," PeerJ Computer Science, vol. 10, p. e2229, 2024. [Online]. Available: https://peerj.com/articles/cs-2229

[12] Y. Meng, Y. Liu, and X. Zhang, "Automated uml class diagram generation from textual requirements using nlp techniques," Journal of Visualization and Interaction, vol. 8, no. 4, pp. 1–15, 2024. [Online]. Available: http://joiv.org/index.php/joiv/article/view/3482

[13] A. Conrardy and J. Cabot, "From image to uml: First results of image-based uml diagram generation using llms," in CEUR Workshop Proc., vol. 3727, 2024. [Online]. Available: https://ceur-ws.org/Vol-3727/paper5.pdf

[14] K. Zhang, L. Wang, and M. Chen, "Enhancing class diagram dynamics: A natural language approach with chatgpt," arXiv preprint, 2024. [Online]. Available: https://arxiv.org/abs/2406.11002

[15] A. Bates, R. Kumar, and S. Patel, "Unified modeling language code generation from diagram images using multimodal large language models," Software and Systems Modeling, 2025, in press. [Online]. Available: https://arxiv.org/abs/2503.12293

[16] K. Johnson and L. Martinez, "Toward a new era of rapid development: Assessing gpt-4-vision's capabilities in uml-based code generation," in Proc. ACM/IEEE Int. Conf. Software Engineering (ICSE), 2024, pp. 1–10. [Online]. Available: https://dl.acm.org/doi/10.1145/3643795.3648391

[17] Z. Pei, L. Liu, C. Wang, and J. Wang, "Requirements engineering for machine learning: A review and reflection," in Proc. AIRE Workshop, 2022. [Online]. Available: https://aire-ws.github.io/aire22/papers/AIRE 05.pdf

[18] D. De Bari, "Evaluating large language models in software design: A comparative analysis of uml class diagram generation," Master's thesis, Politecnico di Torino, 2024. [Online]. Available: https://webthesis.biblio.polito.it/31177/

[19] E. A. Abdelnabi, A. M. Maatuk, and M. Hagal, "Generating uml class diagram from natural language requirements: A survey of approaches and techniques," in Proc. 2021 Int. Maghreb Meeting Conf. Sciences and Techniques of Automatic Control and Computer Engineering (MI-STA), 2021, pp. 288–293. [Online]. Available: https://doi.org/10.1109/MI-STA52233.2021.9464433

[20] T. Ahmad, F. Khan, Z. Malik, and S. Ahmed, "Model-based testing using uml activity diagrams: A systematic mapping study," Computer Science Review, vol. 33, pp. 98–112, 2019. [Online]. Available: https://doi.org/10.1016/j.cosrev.2019.07.001

[21] J. A. Rozo, R. Mazo, and H. Sahraoui, "Review of automated test case generation, optimization, and prioritization using uml diagrams: Trends, limitations, and future directions," Scalable Computing: Practice and Experience, vol. 25, no. 4, pp. 3030–3045, 2024. [Online]. Available: https://www.scpe.org/index.php/scpe/article/view/3030

[22] M. A. Zadenoori, F. Dalpiaz, and S. Brinkkemper, "Large language models (llms) for requirements engineering: A systematic literature review," arXiv preprint, 2025. [Online]. Available: https://arxiv.org/abs/2509.11446

[23] S. Ronanki, A. Ferrari, and X. Franch, "Generative ai for requirements engineering: A systematic literature review," Software and Systems Modeling, 2025, in press. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/spe.70029

[24] N. J. Waragde and V. K. Sharma, "Automated generation of software design diagrams from textual requirements using natural language processing," International Journal of Advanced Research in Science, Communication and Technology, vol. 5, no. 6, pp. 69–73, 2025. [Online]. Available: https://ijarsct.co.in/Paper30213.pdf

[25] N. Bashir et al., "Modeling class diagram using nlp in object-oriented designing," in Proc. 2021 National Computing Colleges Conf. (NCCC), 2021, pp. 1–6. [Online]. Available: https://doi.org/10.1109/NCCC49330.2021.9428817

[26] S. Isotani and I. I. Bittencourt, "Ontology driven software engineering: A review of challenges and opportunities," IEEE Software, vol. 32, no. 1, pp. 62–70, 2015. [Online]. Available: https://ieeexplore.ieee.org/document/7069116/

[27] F. Dalpiaz, A. Ferrari, X. Franch, and C. Palomares, "Natural language processing for requirements engineering: A systematic mapping study," ACM Computing Surveys, vol. 54, no. 3, pp. 1–41, 2018. [Online]. Available: https://dl.acm.org/doi/10.1145/3266708

[28] M. Kahani, M. Bagherzadeh, J. Dingel, and J. R. Cordy, "Model transformation languages under a magnifying glass: A controlled experiment with xtend, atl, and qvt," in Proc. 2018 ACM Joint Meeting European Software Engineering Conf. and Symp. Foundations of Software Engineering (ESEC/FSE), 2018, pp. 445–455. [Online]. Available: https://dl.acm.org/doi/10.1145/3236024.3236046

[29] C. Ponsard and P. Michot, "Survey of automation practices in model-driven development," in Proc. 10th Int. Conf. Model-Driven Engineering and Software Development (MODELSWARD), 2022, pp. 70–81. [Online]. Available: https://ieeexplore.ieee.org/document/9814525/

[30] A. Gonzalez, M. Pinto, and L. Fuentes, "Towards an automatic model transformation mechanism from uml state machines to devs models," Computación y Sistemas, vol. 19, no. 2, pp. 331–344, 2015. [Online]. Available: http://www.scielo.edu.uy/scielo.php?script=sci_arttext&pid=S0717-50002015000200004

[31] M. Goulao, F. Araújo, and F. Brito e Abreu, "A survey of traceability in requirements engineering and model-driven development," Software and Systems Modeling, vol. 10, no. 4, pp. 529–565, 2011. [Online]. Available: https://link.springer.com/article/10.1007/s10270-009-0145-0

ISSN: 2961-5992

[32] J. A. García-García, E. Cutilla, F. J. Rodríguez-Perez, I. Ramos, and M. J. Escalona, "Lean requirements traceability automation enabled by model-driven engineering," PeerJ Computer Science, vol. 8, p. e817, 2022. [Online]. Available: https://peerj.com/articles/cs-817

[33] A. Ferrari, S. Abualhaija, and C. Arora, "Model generation with llms: From requirements to uml sequence diagrams," in Proc. 2024 IEEE 32nd Int. Requirements Engineering Conf. Workshops (REW), 2024, pp. 291–300. [Online]. Available: https://doi.org/10.1109/REW61692.2024.00044

[34] G. B. Herwanto, "Automating data flow diagram generation from user stories using large language models," in CEUR Workshop Proc., vol. 3672, 2024. [Online]. Available: https://ceur-ws.org/Vol-3672/NLP4RE-paper3.pdf

[35] N. Bouali, M. Gerhold, T. U. Rehman, and F. Ahmed, "Toward automated uml diagram assessment: Comparing llm-generated scores with teaching assistants," in Proc. 17th Int. Conf. Computer Supported Education (CSEDU), vol. 1, 2025, pp. 158–169. [Online]. Available: https://www.scitepress.org/Papers/2025/134819/134819.pdf

[36] A. Rouabhia, A. Ferrari, and A. Hadjadj, "Behavioral augmentation of uml class diagrams: An empirical study of large language models for method generation," arXiv preprint, 2025. [Online]. Available: https://arxiv.org/abs/2506.00788

[37] S. Ronanki, A. Ferrari, X. Franch, and F. Dalpiaz, "Nomad: A multi-agent llm system for uml class diagram generation from natural language requirements," arXiv preprint, 2025. [Online]. Available: https://arxiv.org/abs/2511.22409

[38] M. J. Page et al., "The prisma 2020 statement: An updated guideline for reporting systematic reviews," BMJ, vol. 372, p. n71, 2021. [Online]. Available: https://doi.org/10.1136/bmj.n71

[39] M. Jahan, Z. S. H. Abad, and B. Far, "Generating sequence diagram from natural language requirements," in Proc. 2021 IEEE 29th Int. Requirements Engineering Conf. Workshops (REW), 2021, pp. 39–48. [Online]. Available: https://doi.org/10.1109/REW53955.2021.00012

[40] S. Yang and H. Sahraoui, "Towards automatically extracting uml class diagrams from natural language specifications," in Proc. 25th Int. Conf. Model Driven Engineering Languages and Systems: Companion Proc., 2022, pp. 396–403. [Online]. Available: https://dl.acm.org/doi/10.1145/3550356.3561592

[41] Shweta, S. Mittal, and S. Chauhan, "Advancing class diagram extraction from requirement text using transformer-based approach," in Proc. 19th Int. Conf. Natural Language Processing (ICON), 2023, pp. 319–329. [Online]. Available: https://aclanthology.org/2023.icon-1.36

[42] J. Navarro, S. d. R. S. Lopes de Souza, and M. E. Delamaro, "Automatic test case generation using natural language processing: A systematic literature review," Journal of Systems and Software, vol. 220, p. 112303, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S095058492500268X

[43] Shweta, R. Sanyal, and B. Ghoshal, "A hybrid approach to extract conceptual diagram from individual use cases," Science of Computer Programming, vol. 239, p. 103186, 2025. [Online]. Available: https://doi.org/10.1016/j.scico.2024.103186

[44] K. A. D. O. K. Arachchi, "Ai based uml diagrams generator," Master's thesis, University of Colombo School of Computing, 2021. [Online].Available: https://dl.ucsc.cmb.ac.lk/jspui/handle/123456789/4634

[45] S. Nair and P. Thushara, "Hybrid machine learning approach for automatic actor identification in use case modeling," in Proc. 2024 Int. Conf. Emerging Trends in Engineering, Science and Technology (ICETEST), 2024, pp. 553–560. [Online]. Available: https://doi.org/10.1109/ICETEST59021.2024.10456789

[46] O. S. D. Omer and S. Eltyeb, "Towards an automatic generation of uml class diagrams from textual requirements using case-based reasoning approach," in Proc. 2022 4th Int. Conf. Applied Automation and Industrial Diagnostics (ICAAID), 2022, pp. 1–5. [Online]. Available: https://doi.org/10.1109/ICAAID51067.2022.9799502

[47] Z. Babaalla et al., "Extraction of uml class diagrams using deep learning: Comparative study and critical analysis," Procedia Computer Science, vol. 236, pp. 452–459, 2024. [Online]. Available: https://doi.org/10.1016/j.procs.2024.05.053

[48] Z. Babaalla et al., "Towards an automatic extracting uml class diagram from system's textual specification," in Proc. 7th Int. Conf. Networking, Intelligent Systems and Security, 2024, pp. 1–5. [Online]. Available: https://doi.org/10.1145/3659677.3659742

[49] E. Alwanain, "Automated composition of sequence diagrams," Ph.D. dissertation, University of Birmingham, 2016. [Online]. Available: https://etheses.bham.ac.uk/6919/1/Alwanain16PhD.pdf

[50] J. Shivamurthy, T. Uppal, and D. Vidyarthi, "Nlp-based auto generation of graph database from textual requirements," in Proc. 2024 IEEE Int. Conf. Electronics, Computing and Communication Technologies (CONECCT), 2024, pp. 1–6.

ISSN: 2961-5992