TYPE Original Research
PUBLISHED 26 February 2025
DOI 10.3389/fcomp.2025.1537100

OPEN ACCESS

EDITED BY
Huai Liu,
Swinburne University of Technology, Australia

REVIEWED BY
Reyes Juárez-Ramírez,
Universidad Autónoma de Baja California, Mexico
Tanjila Kanij,
Swinburne University of Technology, Australia

*CORRESPONDENCE
Muhammad Aminu Umar
aminu.umar@kcl.ac.uk

RECEIVED 29 November 2024
ACCEPTED 27 January 2025
PUBLISHED 26 February 2025

CITATION
Umar MA, Lano K and Abubakar AK (2025) Automated requirements engineering framework for agile model-driven development. *Front. Comput. Sci.* 7:1537100. doi: 10.3389/fcomp.2025.1537100

Automated requirements engineering framework for agile model-driven development

Muhammad Aminu Umar[1,2]*, Kevin Lano[1] and Abdullahi Kutiriko Abubakar[3]

[1]Department of Informatics, King's College London, London, United Kingdom, [2]Department of Computer Science, Ahmadu Bello University, Zaria, Nigeria, [3]Department of Computer Science, University of Surrey, Guildford, United Kingdom

Introduction: Advances in requirements engineering, driven by various paradigms and methodologies, have significantly influenced software development practices. The integration of agile methodologies and model-driven development (MDE) has become increasingly critical in modern software engineering. MDE emphasizes the use of models throughout the development process, necessitating structured approaches for handling requirements written in natural language.

Methods: This paper proposes an automated requirements engineering framework for agile model-driven development to enhance the formalization and analysis of textual requirements. The framework employs machine learning models to extract essential components from requirements specifications, focusing specifically on class diagrams. A comprehensive dataset of requirements specification problems was developed to train and validate the framework's effectiveness.

Results: The framework was evaluated using comparative evaluation and two real-world experimental studies in the medical and information systems domains. The results demonstrated its applicability in diverse and complex software development environments, highlighting its ability to enhance requirements formalization.

Discussion: The findings contribute to the advancement of automated requirements engineering and agile model-driven development, reinforcing the role of machine learning in improving software requirements analysis. The framework's success underscores its potential for widespread adoption in software development practices.

KEYWORDS

requirements engineering, model-driven engineering, model-driven development, agile development, machine learning, NLP

1 Introduction

The model-driven engineering (MDE) approach to software development advocates the construction of software models, often in the form of Unified Modeling Language (UML) diagrams, that depict how the software system should work, and the production of executable code. The construction of such models constitutes a key requirements engineering activity. Requirements Engineering (RE) focuses on identifying and specifying the real-world objectives, functions, and constraints of software systems (Abdouli et al., 2016). Within the context of functional requirements, RE plays a pivotal role in defining the specific actions and operations that a system must execute to achieve its intended objectives. RE is widely recognized as a fundamental process for ensuring the quality of software products. Despite its inherently complex and interdisciplinary nature, which presents challenges in software and systems development, it remains essential for the successful delivery of software projects (Aguirre, 2017). Moreover, the significance of requirements engineering extends beyond traditional development models, as it is also integral to agile development methodologies, where it ensures iterative refinement and alignment with evolving project goals.

Requirements engineering plays a crucial role in agile development by ensuring that the evolving needs and expectations of stakeholders are accurately captured, documented, and addressed throughout the development process. Thus, requirements engineering serves as the bridge between stakeholders and development teams, focusing on the elicitation, analysis and specification, validation, and management of software requirements (Inayat et al., 2015); which ensures the end product aligns precisely with the envisioned functionality and user expectations. The synergy between Agile Development and requirements engineering is rooted in their complementary nature. Agile's iterative cycles accommodate changing requirements, and requirements engineering provides the structured process for defining, prioritizing, and managing those requirements within each iteration. Therefore, regular collaboration between developers and stakeholders ensures a continuous feedback loop, refining and clarifying requirements as the project evolves (Elallaoui et al., 2018). Requirements engineering practices such as observations, interviews, workshops, and strong team collaboration are integrated into iteration-based agile methods, alongside customer involvement, requirements prioritization, modeling, and documentation. However, the software development community still lacks a deep understanding of how these RE practices function within agile environments and how they address common issues found in traditional RE processes, despite their perceived benefits and potential challenges (Bjarnason et al., 2011; Inayat et al., 2015).

The growing demand for more effective requirements engineering processes has prompted the introduction and adoption of automated requirements engineering to overcome the limitations of traditional requirements engineering. Automated requirements engineering refers to using software tools and techniques to support and automate eliciting, analysing, specifying, validating, and managing software requirements. These tools can help streamline and optimize the requirements engineering process, which can be complex and time-consuming. The goal of automation in requirements engineering is to minimize the time, effort, and cost involved in the RE process (Nassar and Khamayseh, 2015; Saini et al., 2020; Vemuri et al., 2017; Ibrahim and Ahmad, 2010) by leveraging the power of NLP tools and technologies (Letsholo et al., 2013; Harmain and Gaizauskas, 2003; Yue et al., 2015; Elallaoui et al., 2018; Mu et al., 2009). This approach aims to avoid or minimize human mistakes in reading and analyzing large volumes of natural language text (Thakur and Gupta, 2014; Seresht et al., 2008; Deeptimahanti and Sanyal, 2011), as well as in software development in general, while simultaneously enhancing the quality and accuracy of the requirements.

More recently, machine learning (ML) and deep learning (DL) approaches have been used for requirements engineering, including the use of Large Language Models (LLMs) (Abukhalaf et al., 2023, 2024; Camara et al., 2023; Fill et al., 2023; Saini et al., 2022; Wang et al., 2024).

In the context of Agile methodology, where adaptability and rapid response to change are paramount, automated requirements engineering emerges as a crucial asset. By automating the elicitation, analysis, and validation of requirements, agile teams can significantly enhance their ability to respond swiftly to evolving project needs. Automated requirements engineering not only accelerates the initial phase of requirements gathering but also ensures continuous alignment with changing project dynamics. Furthermore, the iterative and collaborative nature of agile development can benefit from the efficiency and consistency offered by automated processes.

Our proposed automated requirements engineering framework for agile development using NLP and ML is described in Figure 1 focusing primarily on functional requirements. In the figure, we have added an automation component to allow for automated RE because RE is a continuous process throughout the development life cycle in agile development. We acknowledge that RE is part of agile development; however, usually in agile development the requirements analysis, especially generating UML models, is done in the traditional (manual) agile manner. Thus, the integration of an automated framework will enhance the existing benefits of the synergy between agile development and requirements engineering. In particular, it achieves:

I. Adaptability: The iterative nature of Agile allows for the seamless integration of new requirements, enabling adaptation to changing project landscapes.
II. User-Centric Development: Requirements Engineering ensures that development is tightly aligned with user needs, promoting user satisfaction and product success.
III. Early and Continuous Delivery: Agile's emphasis on incremental development and rigorous requirements analysis enables early and continuous delivery of valuable software features.

In object-oriented design, a fundamental aspect is to represent and understand the domain of interest accurately. Object-oriented analysis focuses on creating a description of the domain from the perspective of classification by objects. Decomposing the domain involves identifying the concepts, attributes, and associations that are considered significant. The result can be expressed in a domain model, which is represented by a set of diagrams showing domain concepts or objects. A domain model visualizes conceptual classes or real-world objects within a domain of interest (Fowler, 1996). Using UML notation, this domain model is described using class diagrams that do not include operations but depict domain objects or conceptual classes, their associations, and their attributes (Larman, 2004). Traditional requirements elicitation and analysis often rely on textual descriptions, which can be ambiguous, inconsistent, and prone to misinterpretation. To address these challenges, formal methods have emerged as a promising approach to enhance the precision and rigour of requirements engineering. Among the various formalization techniques, class diagrams, a visual modeling language, have gained significant attention due to their intuitive nature and ability to capture static structural aspects of a system. According to Li and Liu (2014), the core models employed in system requirement analysis are conceptual and use-case models. A conceptual model serves as a static representation of the application domain, delineating domain concepts as classes and their interconnections as associations.

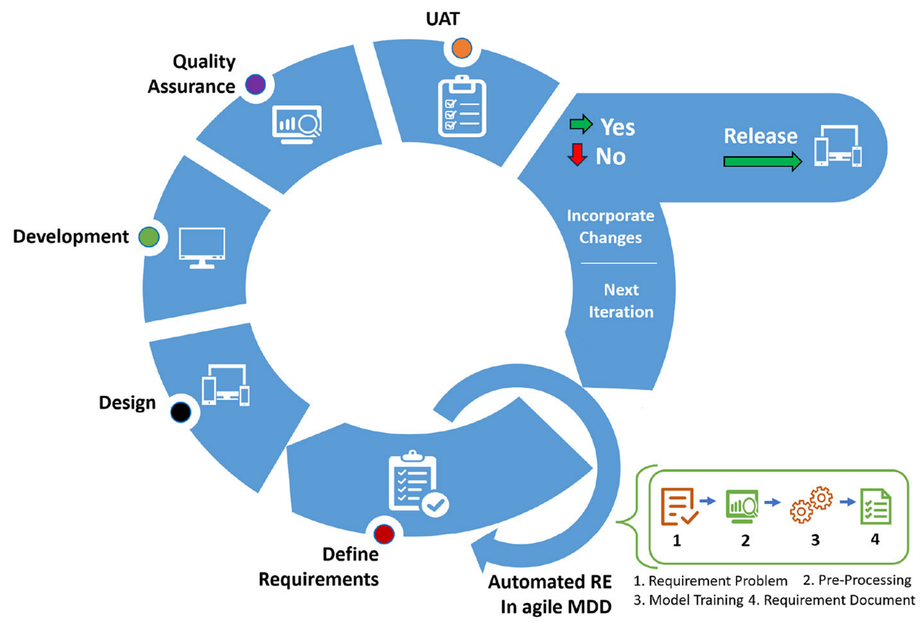
UAT
Quality Assurance
Development
Design
Define Requirements
Yes
No
Release
Incorporate Changes
Next Iteration
Automated RE In agile MDD
1
2
3
4
1. Requirement Problem 2. Pre-Processing
3. Model Training 4. Requirement Document

FIGURE 1
Automated agile requirements engineering in agile iterations.

Model-driven development and class diagrams are often used together. MDD promotes the use of models as primary artifacts in software development, emphasizing their essential role in system specification, design, and implementation. Class diagrams, a fundamental UML construct, excel at capturing static structural aspects of a system, aligning seamlessly with MDD's model-centric approach. By serving as a visual representation of classes, attributes, and relationships, class diagrams provide a concrete foundation for model-driven transformations, allowing automated code generation and system evolution. Consequently, the integration of class diagrams within an MDD framework enhances development efficiency, reduces errors, and improves system maintainability. Thus, MDD can help verify the accuracy of models and generate source codes, minimizing the development time needed to evaluate the software and allowing for greater focus on the modeling process (Akayama et al., 2013). Class diagrams are versatile tools in the software development lifecycle. Early in a project, they serve as exploratory domain models to visualize domain concepts and relationships, aiding in understanding the problem domain (Larman, 2004). Later, they can act as system domain models, specifying aspects of the domain relevant to the system, or as system design models, detailing operations and classes for user interfaces and architecture. Additionally, class diagrams can specify metamodels, providing higher abstraction for modeling. This paper focuses on exploratory domain models developed during the early stages of software projects.

The remainder of this paper is organized as follows: Section 2 discusses the literature survey by highlighting related works. Section 3 describes the framework architecture. Section 4 presents the model training and testing of the proposed framework. Section 5 discusses evaluation comprising comparative evaluation and two experimental studies evaluation. In Section 6, we discuss the results and their implications. Section 7 describes the threats to the validity of our study. Finally, Section 8 concludes and highlights prospects for future research directions.

2 Literature review and analysis

With the rapid advancement of technology, Artificial Intelligence (AI) has emerged as a powerful assistant in enhancing various aspects of software engineering, including Requirements Engineering. There has been considerable adoption of artificial intelligence techniques and technologies in requirements engineering, ranging from natural language processing techniques that enable machines to understand and process human language efficiently (Nazir et al., 2017) to requirement analysis (Letsholo et al., 2013; Yue et al., 2015) and prioritization (Duan et al., 2009) that allows AI algorithms to analyse large datasets of requirements and user feedback to identify trends, extract valuable insights, and aid in requirement prioritization.

Despite the advances in AI in software engineering, specifically in requirements engineering, a key challenge persists in the requirements analysis phase. During this phase, data is often unstructured and requires careful analysis to extract valuable information for use in the subsequent development phases (Al-Hroob et al., 2018). This challenge not only underscores the need for further innovation in AI-driven techniques for effective requirements analysis but also implies significant research opportunities in developing methodologies and tools that can better handle unstructured data in this critical phase of software development. Therefore, this section presents related work on research efforts in requirements analysis with support tools with a specific focus on datasets, model extraction from text and tool evaluation techniques/approaches. We also identify the research gaps and the contributions of our approach to address these gaps.

2.1 Datasets

In automated requirements engineering, support tools play a crucial role in streamlining the process of eliciting, analyzing, and managing requirements for software systems. These tools rely heavily on datasets, which are collections of data used for various purposes such as training machine learning models, testing algorithms, and validating system functionality.

The availability of high-quality datasets is essential for the development and evaluation of automated requirements engineering tools. These datasets typically consist of real-world or simulated examples of software requirements, user stories, use cases, and other artifacts relevant to the requirements engineering process. By using diverse and representative datasets, developers can ensure that their tools are robust, accurate, and applicable to a wide range of scenarios (Zhao et al., 2021).

Despite the proliferation of automated tools in this field, a common challenge is the lack of availability of comprehensive datasets. Many existing tools are developed and evaluated using proprietary or limited datasets, making it difficult for researchers and practitioners to assess their performance objectively (Umar and Lano, 2023). This limitation hinders the advancement of the field and may lead to tools that are not fully effective or generalizable across different contexts (Umar and Lano, 2024). For example, Nasiri et al. (2020) and Jabbarin and Arman (2014) utilized only one requirement problem, while Al-Hroob et al. (2018) employed two, Vemuri et al. (2017); Vidya Sagar and Abirami (2014), and Letsholo et al. (2013) used three, Yue et al. (2015) used 7 problems, Saini et al. (2020) included 18, and Deeptimahanti and Sanyal (2011) used 26 problems, constituting the highest number. Overall, this suggests the need for greater diversity and depth in the datasets used to develop and evaluate tools for addressing requirements problems. Thus, this would enhance the robustness, reliability, and applicability of these tools in real-world settings.

Addressing this gap in dataset availability is crucial for advancing the state-of-the-art in automated requirements engineering. Researchers and tool developers need access to large, diverse, and well-curated datasets that reflect the complexities and nuances of real-world software projects. By openly sharing datasets and establishing standard benchmarks for tool evaluation, the community can foster innovation, collaboration, and continuous improvement in automated requirements engineering practices. In order to address this research gap, our work establishes a working dataset of 105 requirements problems, which is publicly available to researchers and practitioners in automated RE tool development. Details on how the data for the dataset were sourced and collected are discussed in Section 3.

2.2 Extracting models from text

Over the years, researchers have proposed various approaches to extract UML models from natural language automatically. This was prompted by the recognition within both industry practitioners and the research community of the need for tool support to generate UML diagrams automatically. In doing so, several methods and techniques were employed, including those based on grammatical rules and heuristics (Lee and Bryant, 2002), ontology and knowledge representation (Shibaoka et al., 2007), and, more recently, on NLP and ML techniques (Saini et al., 2020). The recent techniques leverage artificial intelligence technologies, especially generative artificial intelligence. Therefore, we present related works in extracting models from requirements text written in natural language.

Using NLP techniques, Ibrahim and Ahmad (2010) introduced a tool called Requirements Analysis and Class Diagram (RACE), designed to streamline the analysis of requirements for extracting class diagrams through NLP techniques. The tool consists of three key components: the first conducts syntactic and lexical analysis of the requirements, and the second focuses on extracting and refining diagram elements. At the same time, the third manages the extracted UML concepts. Despite its functionality, the tool has limitations, as it may produce class diagrams that are incomplete and missing attributes.

Vemuri et al. (2017) proposed an automated tool to generate UML use case diagram components (actors and use cases) from requirements documents leveraging natural language processing and machine learning using a probabilistic classification. The approach used three short requirements documents in a single domain to evaluate the model developed. However, a case study evaluation of the proposed approach was conducted. This is similar to the approach proposed in Deeptimahanti and Babar (2009) but used syntactic reconstruction rules and NLP tools to extract required Object-oriented artifacts like use cases, actors, classes, and attributes. Neither approach defines rules for generalization.

Narawita and Vidanage (2016) developed a web-based application called UML Generator, designed to efficiently and cost-effectively produce both UML use cases and class diagrams from informal software specifications. The SharpNLP library powers the application's NLP capabilities. To enhance extraction accuracy, the approach applies a set of XML rules to filter out noise words from the identified entities. A classification model, Weka, is then used to categorize these entities as elements of either a class diagram or a use case diagram. Users can review the generated diagrams and either accept or reject them using the Visual Studio Modeling module. However, there is a lack of quantitative evaluation of the accuracy of the extracted UML models and the dataset used for evaluation.

Yang and Sahraoui (2022) introduced an approach for automatically generating UML class diagrams from English-language specifications, leveraging machine learning algorithms to build language models. The process begins by pre-processing the requirements and replacing pronouns to reduce sentence interdependency. The SpaCy library and an English language model are then used to break down the pre-processed text into individual sentences. A Naive Bayes classifier is subsequently applied to categorize each sentence as either describing a "class" or a "relation." For sentence vectorization during classifier training, the TF-IDF method is employed. Following classification, analysis and extraction methods are used to generate UML fragments from the labeled sentences, which are ultimately combined to construct the final class diagram.

Another work by Elallaoui et al. (2018) used a model-driven development approach to propose automatic transformation of user stories into UML use case diagrams using NLP techniques. The approach takes semi-structured user stories as input and applies POS tagging as its main NLP technique. The input text sentences are POS-tagged, and the words of each sentence are filtered to remove adjectives and auxiliary words, retaining only nouns and verbs. The approach was compared with the manually extracted, and case study evaluation results showed precisions between 87% and 98%.

Nasiri et al. (2020) developed an approach for generating UML class diagrams from user stories using Stanford CoreNLP. The method transforms user stories (CIM) into a Platform-Independent Model (PIM) by creating an XMI file that outlines classes, attributes, relationships, and operations. NLP techniques such as tokenization, POS tagging, coreference resolution, and stemming (via WordNet) are applied to process user stories and remove redundancy. Design elements are extracted through predefined rules analysing typed dependencies. Actors and classes are identified first, followed by relationships and attributes based on composition. An XMI file is then generated using PyEcore API, and the class diagram is visualized through PlantUML. Although 98% accuracy was reported, the approach lacks case study evaluation and addresses only one requirement specification.

Saini et al. (2022) proposed a tool named DoMoBOT, which leverages the capabilities of natural language processing (NLP) and machine learning to automate the generation of domain models, specifically class diagrams, from natural language textual descriptions. The tool emphasizes user interaction and maintains traceability between different artifacts produced during the model development process. The first component of DoMoBOT utilizes the SpaCy library to pre-process the problem description. The second component, called descriptive, identifies relevant domain concepts such as classes and attributes by applying NLP techniques based on extraction rules. The third component incorporates a pre-trained predictive model using machine learning and deep learning methods to enhance the accuracy of identifying concepts and relationships. Finally, the fourth component applies two algorithms to integrate the results from the NLP and machine learning modules, constructing a complete domain model with a median F1 score of 86%. Even though the approach was supported with an interactive user interface, the resulting output accuracy can be improved with additional datasets and case study evaluation.

The studies reviewed indicate that larger training datasets improve model accuracy and reliability in capturing user requirements. Building on this, our work aims to extract domain models, specifically class diagrams, using a broader dataset. We developed three models: one extracts classes and attributes, another identifies their relationships, and the third recognizes relationships among classes. Our models were trained and tested on diverse requirement problems from various sources, as detailed in Section 4. By using labeled words from realistic requirement scenarios, our approach ensures contextually relevant diagrams, unlike earlier studies, such as Saini et al. (2022), which used rule-based NLP methods and noun-based corpora. The latter approach lacked the pattern recognition essential for complex requirements analysis, positioning our work as a comparative advancement over this baseline.

2.3 Tool evaluation approaches

Evaluation is a critical stage in every software development project, to determine whether a tool has attained its objectives. Various evaluation approaches are employed in assessing automated requirements engineering tools, including controlled experiments, simulations, proof of concept (prototypes), case studies, and user studies (Meth et al., 2013) . These methods collectively contribute to a comprehensive understanding of the tool's effectiveness and suitability for practical application. Thus, the two studies extensively reported and discussed evaluation approaches for automated requirements engineering tools as employed by empirically published works. According to the findings, controlled experiments were the most commonly utilized evaluation approach. However, the industry practitioners favored the case study evaluation approach (Umar and Lano, 2023). This preference stems from the fact that the case study evaluation approach is more appropriate, as it allows for real-world use and provides valuable feedback on a particular tool. Thus, this suggests that conducting in-depth case studies to assess the effectiveness of automated RE tools is a valuable approach. This can lead to the development of more robust evaluation frameworks that mimic real-world usage scenarios and gather feedback from actual tool users, enhancing the credibility and practical applicability of the assessment.

Therefore, considering the important role of case study evaluation in validating theoretical frameworks and assessing their practical applicability, our study embarked on a comprehensive exploration. This is achieved through analysis of two distinct case studies, we aimed to scrutinize the efficacy and versatility of the proposed framework in real-world scenarios. Furthermore, we augmented our evaluation by soliciting expert insights into the process of extracting class diagram components from the requirements outlined in the case study texts. Through the combination of case study evaluation and expert assessment, our objective was to obtain a comprehensive understanding of the framework's performance and its alignment with industry standards and best practices.

3 Framework architecture

Our work is motivated by the growing need for more intuitive and efficient methods to translate textual descriptions, often contained in software requirements or documentation, into visual representations that may be seamlessly integrated into the software development lifecycle. The UML class diagram is one of the essential artifacts extracted for requirements analysis. A class diagram is a powerful visualization tool in this context, offering insights into the structure and relationships within a software system. This diagram helps in requirements analysis and provides the basis for the software design and subsequent development phases (Sharma et al., 2015). Providing automated requirement formalization reduces development effort, cost and time. Therefore, the proposed automated tool will be able to analyse text written in natural language to extract various components of a class diagram, including classes, Attributes, and Relationships. The structure of a requirements sentence typically includes several key parts that ensure clarity and completeness (Wang and Zhang, 2016). These parts provide a standardized way to communicate system needs, making them suitable for analysis and implementation. The main components include the subject, which identifies who or what is performing the action (e.g., the user); the action or behavior, which specifies the functionality to be performed (e.g., book); the target or object, which defines what the action applies to (e.g., an appointment); the conditions or context, which provide details about how or under what circumstances the action occurs (e.g., by selecting an available time slot from a calendar interface); and, optionally, the outcome or goal, which describes the expected result (e.g., ensuring the appointment is scheduled and confirmed). For example, a well-structured requirement sentence might state: “The user should be able to book an appointment by selecting an available time slot from a calendar interface.” This structure ensures the requirement is actionable, precise, and ready for further analysis or implementation. This aligns closely with a typical user story format in requirements, which often follows the structure of “As a [user], I want to [perform an action] so that [I achieve a specific outcome].” Lucassen et al. (2017) For instance, the example sentence could be reframed as a user story: “As a user, I want to book an appointment by selecting a time slot so that my booking is confirmed.”

For functional requirements, specific elements or attributes considered as part of a requirement sentence are critical for accurately assessing the machine learning algorithm’s capability to extract components essential for the model. These elements include: Entities (Classes): Objects referenced in requirements, such as “user” or “appointment,” are modeled as classes central to system operations. For example, in the requirement sentence “The user should be able to book an appointment,” the entities include user and appointment. Attributes of Entities: Properties of entities, such as a “user” with attributes like name or email and an “appointment” with attributes like time or date. Relationships Between Classes: Connections between entities, such as “A user can book multiple appointments,” which define a one-to-many relationship.

The process of extracting and understanding requirements from textual data is a crucial step in the software development lifecycle. Machine learning models are increasingly employed to enhance accuracy and automate this process. This section outlines the architecture of a model designed for extracting classes, attributes, and relationships from textual requirements. To facilitate the development of this automated support tool, we have developed three distinct machine-learning models: (1) A model to extract classes and attributes, (2) A model to extract attributes of a given class, and (3) A model to depict relationships between extracted classes.

These models represent a significant step forward in automating requirement analysis, a foundation for effective software development. Figure 2 shows an overview of the architecture used to develop the class diagram from data pre-processing to model evaluation for the automated requirement engineering framework for model-driven agile development. Thus, the following constitute the various components of the framework architecture:

3.1 Dataset development

Data collection is a critical phase that shapes the model’s effectiveness in applying learned patterns to real-world scenarios, particularly for generating class diagram components. The quality and relevance of collected data are essential for enabling the model to accurately interpret requirement specifications. Figure 3 shows the data collection process used for this work. The collected dataset is publicly available on GitHub[1] to support transparency, reproducibility, and research collaboration.

1. **Data collection:** This component discusses the creation of the dataset used to train the machine learning models for generating class diagrams from requirement texts. This dataset comprises 105 requirement problems covering various domains sourced from academic literature (Bozyigit et al., 2023; Saini et al., 2020; Alessio Ferrari et al., 2017), open educational resources (GitHub and Kaggle). The quality and appropriateness of this dataset are vital for the project’s success, forming the foundation of our research and model development.

   Creating a well-structured dataset is critical for developing a machine learning model that can effectively extract classes, attributes, and relationships from textual requirements. This process is essential for training and evaluating the model’s performance, ensuring accurate, and robust results. Data annotations provide the necessary ground truth labels that guide the model during training, enhancing its ability to generalize effectively to new and unseen data. Consequently, we employed a combination of automated and manual data annotations to create the machine learning dataset. In the absence of a standard RE dataset suitable for training our model to extract class diagram components, we developed a customized dataset specifically aimed at enabling the automatic extraction of class diagrams from textual requirements. The dataset, derived primarily from academic literature, presented challenges due to its limited availability. To address these challenges, we focused on ensuring the dataset provided realistic and unbiased information—an essential factor for the model’s effectiveness in practical applications. In its development, we carefully considered several concerns highlighted by Paullada et al. (2021) regarding dataset creation and use in machine learning research:

   - Avoiding biases in dataset annotation: A dataset characterized by subjective values, judgments, and biases may result in undesirable or unintended dataset bias. To mitigate this, our dataset was carefully annotated objectively.
   - Dataset documentation practices: Proper documentation is essential to avoid data under-specification and

1 https://github.com/amiinuumar/AutomatedRE

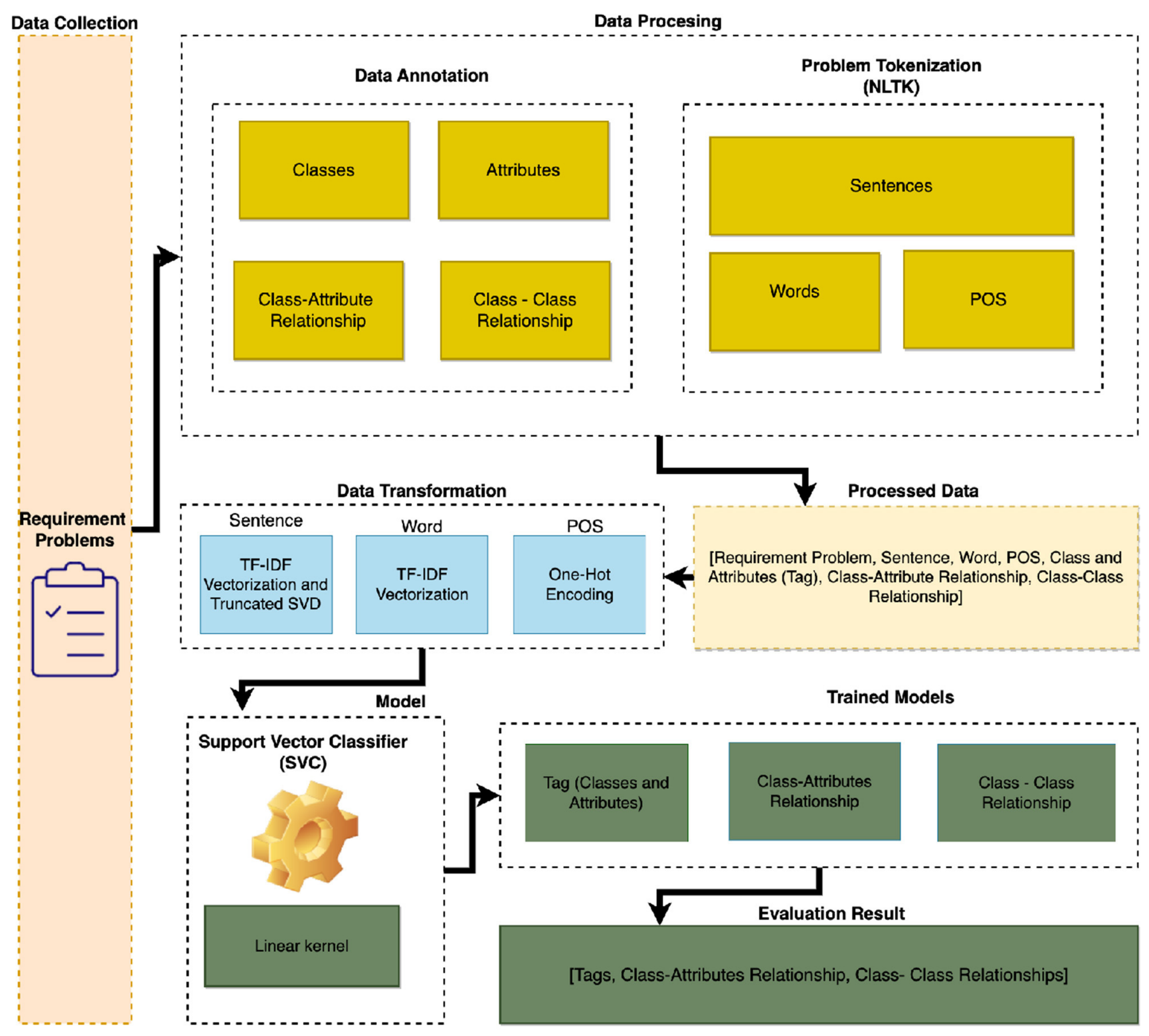
Data Collection
Data Procesing
Data Annotation
Classes
Attributes
Class-Attribute Relationship
Class - Class Relationship
Problem Tokenization (NLTK)
Sentences
Words
POS
Requirement Problems
Processed Data
[Requirement Problem, Sentence, Word, POS, Class and Attributes (Tag), Class-Attribute Relationship, Class-Class Relationship]
Data Transformation
Sentence
Word
POS
TF-IDF Vectorization and Truncated SVD
TF-IDF Vectorization
One-Hot Encoding
Model
Support Vector Classifier (SVC)
Linear kernel
Trained Models
Tag (Classes and Attributes)
Class-Attributes Relationship
Class - Class Relationship
Evaluation Result
[Tags, Class-Attributes Relationship, Class- Class Relationships]

FIGURE 2
An overview of the requirements engineering ML architecture from data collection to evaluation of the model.

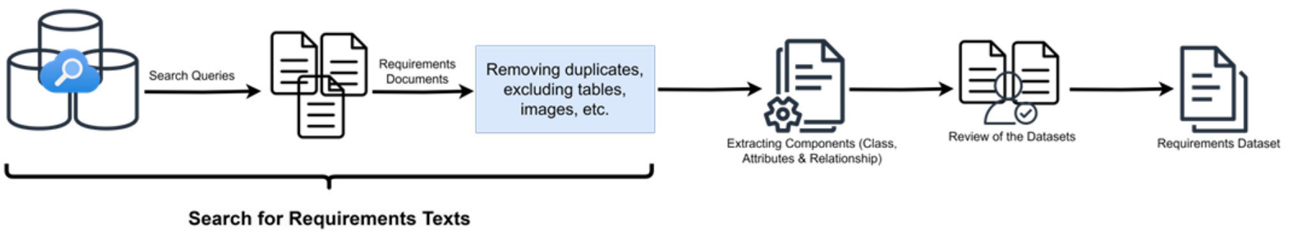
Search Queries
Requirements Documents
Removing duplicates, excluding tables, images, etc.
Extracting Components (Class, Attributes & Relationship)
Review of the Datasets
Requirements Dataset
Search for Requirements Texts

FIGURE 3
Data collection and extraction steps.

inconsistency. We ensured that our data reflects the ground truth by careful annotation to allow for applicability.

- Reproducibility concerns: The absence of rigorous and standardized documentation processes contributes to reproducibility issues. We addressed this by ensuring detailed documentation to support other researchers in replicating and reproducing our work.

Additionally, we followed the three dataset creation elements by Sim et al. (2003): a motivating comparison, a representative sample task, and performance measures.

- **Motivating comparison**: Combining motivation with comparison, this element provides both the technical comparison and the research agenda, aiming to support dataset development for software engineering research. Despite limited requirements texts available for model generation, a literature search was conducted to address this need.
- **Sample task**: Benchmark tests should reflect tasks that the tool or technique will solve in practice. Given the impracticality of including all problem domain cases, a subset of tasks was selected based on criteria established through a thorough search.
- **Performance measures**: Quantitative or qualitative, performance measures are used to evaluate features, showcase tool capabilities, and compare technologies. These measures are detailed in Section 4.

In summary, we created the dataset with careful attention to data collection, annotation, and documentation to ensure replicability. Table 1 presents part of the statistics description of the 105 requirements problems used in this study.

2. **Data annotation and pre-processing:** This is an essential process to ensure that the data is well-annotated and formatted correctly for use in training and evaluating the machine learning model. Effective data preprocessing is crucial for model development, as inaccurately preprocessed data can lead to errors and unreliable predictions from the ML model, potentially trained on these erroneous inputs. The following are details of the specific preprocessing techniques we utilized to prepare the datasets for model development.

   (a) *Data annotation*: A key part of data collection is annotating the dataset with ground truth labels, where class diagram components (classes, attributes, and relationships) act as reference points for model training. This annotation process enables the model to learn linguistic patterns and understand the semantics of class diagram elements. To create these labels, we manually review and categorize the requirements data, identifying classes, attributes, and relationships within each requirement problem. The dataset includes five columns: problem number, requirement text, and labeled class, attribute, and relationship entries.

   (b) *Problem tokenization*: To facilitate this process, we employed the Natural Language Toolkit (NLTK) library.[2] The primary use of NLTK was to tokenize the requirement problems into sentences, words, and parts of speech (POS). Each requirement problem is broken down into sentences, the sentences are broken down into words, and the POS of each word is labeled using the NLTK library. This ensures that the model has efficient features to learn from.

   (c) *Processed data*: Here, we combine the annotated data with the tokenized data to form the final dataset used in training the model. In the class/attribute column, we label each class and attribute as identified in the annotated data. The class/attribute/relationship column is specifically used to label attributes that belong to a class, while the class/class/relationship column is used to identify relationships between classes. Excluding stop words, which are discarded, any remaining words in the sentence are tagged as "Other." The resulting dataset contains the following columns:

   - **Problem-number**: A unique identifier for each requirement problem.
   - **Sentence**: Each sentence that makes up the problem specification.
   - **Word**: Each word comprising a sentence.
   - **POS**: Parts of Speech.
   - **Class and attribute**: Each word is identified as either a class or an attribute, while the remaining words are categorized as "other."
   - **Class—Attribute relationship**: The relationship of each word (attribute) to a Class.
   - **Class—Class relationship**: The relationship between two classes.

Table 2 presents a sample requirement problem that has been preprocessed for training. It illustrates how sentences are tokenized into individual words and annotated with linguistic and semantic information. Each word is assigned its Part of Speech (POS), a tag indicating its role (e.g., "Class," "Attributes," or "Other"), and additional classifications related to domain concepts: "Class_Related," which defines the relationship between a class and its attributes, and "Class_R," which depicts relationships among classes. The example highlights relationships between domain-specific entities, such as "Event" and "Venue," within the context of a software requirement. Therefore, the complete table for the preprocessed data of the 105 requirements problems has 19,194 rows, which constitute the dataset used for the study.

3. **Data transformation:** In our data transformation process, we employ a `ColumnTransformer` to apply distinct preprocessing techniques to various columns of our dataset, tailoring the treatment according to the nature of the data in each column. This approach ensures that each feature is optimally prepared for the machine learning model.

   Firstly, for the "Word" column, we use a `TfidfVectorizer` with an n-gram range of (1, 2). This means that the transformer not only considers individual words (unigrams) but also pairs of consecutive words (bigrams) in its analysis. This technique converts the text data into a TF-IDF (Term Frequency-Inverse Document Frequency) format, which is effective in capturing the importance of words in relation to the corpus of documents, thereby enhancing the model's ability to understand and leverage textual data.

2 https://www.nltk.org/

TABLE 1 Statistics of the requirements problems.

No.	Requirements name	Source	Domain	Sentences	Words	Unique words	Stop words removed	Classes	Attributes	Relationships
R1	Movie database	Academic	Entertainment	14	188	81	90	6	13	8
R2	Restaurant management	Academic	Business	18	211	104	130	10	28	8
R3	Veterinary clinic	Academic	Health	13	176	93	90	6	18	6
R4	Company database	Saini et al., 2020	Business	10	107	61	67	8	12	8
R5	Food delivery system	Saini et al., 2020	Business	8	84	48	50	3	5	3
R6	Event management system	Saini et al., 2020	Business	9	76	44	40	5	5	4
R7	Course registration	Bozyigit et al., 2023	Education	16	253	121	148	7	28	7
R8	Banking system	Bozyigit et al., 2023	Finance	10	173	95	102	9	14	8
R9	Conference system	Saini et al., 2020	Education	9	124	74	56	5	5	2
R10	Online auction	Academic	Business	15	266	110	143	8	23	7
R11	Online library system	Saini et al., 2020	Education	12	120	59	69	4	4	3
R12	Course enrolment	Saini et al., 2020	Education	7	82	51	45	3	4	2
...	...	...	...	...	...	...	...	...	...	...
R36	Estate agency	Academic	Construction	11	222	108	149	5	18	5
R37	Health and fitness tracker	Academic	Health	12	207	115	143	6	20	7
R38	Customer support ticketing	Academic	Business	13	232	123	157	4	14	3
R39	Car rental system	Bozyigit et al., 2023	Business	9	167	87	113	6	20	7
R40	Library management	Academic	Education	10	170	106	114	4	16	5
...	...	...	...	...	...	...	...	...	...	...
R101	Finance management	Academic	Finance	10	128	76	81	3	14	3
R102	Smart agriculture	Academic	Agriculture	9	199	113	140	4	13	3
R103	Movie store	Academic	Entertainment	7	108	73	58	6	18	5
R104	Team management	Academic	Sports	16	171	89	97	8	13	7
R105	Pizza ordering system	Academic	Business	17	136	67	71	9	15	8

Next, the "Sentence" column undergoes a two-step transformation within a `Pipeline`. Initially, it is processed by a `TfidfVectorizer`, similar to the "Word" column, but it only considers unigrams (individual words). This is followed by a dimensionality reduction step using `TruncatedSVD` with 100 components. Truncated Singular Value Decomposition (SVD) is particularly useful for handling the sparse matrices that result from TF-IDF, enabling the model to capture the essence of the data with reduced complexity.

Lastly, for the Part of Speech column, an `OneHotEncoder` is applied. This transformer converts the categorical data of parts of speech into a binary matrix — a format that is more suitable for machine learning algorithms. It is set to ignore unknown categories, ensuring robustness in handling a variety of inputs. By using these tailored transformations, our data preprocessing pipeline effectively prepares each aspect of the textual data, enhancing the subsequent model's performance and accuracy.

4. **Data splitting:** After pre-processing, data splitting divides the dataset into training, validation, and test sets to enable accurate model evaluation, hyperparameter tuning, and reduce overfitting. The training set helps the model learn patterns in the data, the validation set fine-tunes and evaluates the model during training, and the test set assesses the final performance on unseen data, simulating real-world scenarios. In this work, we adopted the common 80:20 split ratio, widely used in machine

TABLE 2 A sample preprocessed requirements problem description.

Sentence	Word	POS	Tag	Class_Related	Class_R
An event is organized in a venue.	An	DT	Other	Other	Other
An event is organized in a venue.	event	NN	Class	Event	Venue
An event is organized in a venue.	is	VBZ	Other	Other	Other
An event is organized in a venue.	organized	VBN	Other	Other	Other
An event is organized in a venue.	in	IN	Other	Other	Other
An event is organized in a venue.	a	DT	Other	Other	Other
An event is organized in a venue.	venue	NN	Class	Venue	Event
An event can be a concert event or a theatrical event.	An	DT	Other	Other	Other
An event can be a concert event or a theatrical event.	event	NN	Class	Event	Venue
An event can be a concert event or a theatrical event.	can	MD	Other	Other	Other
An event can be a concert event or a theatrical event.	be	VB	Other	Other	Other

learning, to dedicate 80% of data for training and 20% for testing, following the Pareto principle for balancing learning and evaluation. Increasing the training set size typically enhances model performance (Neto et al., 2024).

3.2 Machine learning algorithm

We choose the Support Vector Machines (SVMs) algorithm, which is a powerful supervised learning method used for text classification. The SVM is particularly favored in text classification due to its effectiveness in high-dimensional spaces, which is typical of text data. It utilizes text vector representations like TF-IDF (Term Frequency-Inverse Document Frequency) and can create a feature space with thousands of dimensions. SVMs excel in such settings by not only handling the high dimensionality but also by focusing on the most critical features that define the decision boundary. At its core, SVM works by finding the hyperplane that best separates different categories in the data. This separation is achieved by maximizing the margin between data points of different categories, which is key to SVM's robustness and generalization ability in text classification.

SVM also has different kernel functions, which allow the algorithm to fit the maximum-margin hyperplane in a transformed feature space. We choose the linear kernel for our text classification after we find that the kernel accuracy is better than that of the other kernels [Polynomial and Radial Basis Function (RBF)]. We leveraged the grid search to test all the kernels and hyperparameters and found that the accuracy of the linear kernel was better in the developed models.

3.2.1 Model evaluation

The evaluation phase in our machine learning framework ensures the model performs well on both training and unseen data. This involves using precision, recall, and F1 scores to measure accuracy in categorizing data, particularly in classification tasks. These metrics highlight strengths and areas for improvement, offering insights into performance aspects like error rates and balancing false positives and negatives. For tasks with imbalanced datasets or specific outcomes, selecting appropriate metrics is essential to provide a fair, comprehensive assessment of the model's capability to identify relevant instances effectively.

Precision: Precision measures the proportion of correctly predicted positive instances (e.g., correctly identified classes or attributes) out of all instances that were predicted as positive. High precision indicates that the model makes fewer false positive errors.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \tag{1}$$

Recall: Recall assesses the model's ability to identify relevant components by measuring the proportion of correctly predicted positive instances out of all actual positive instances. High recall means the model misses fewer relevant components (e.g., classes or attributes).

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \tag{2}$$

F1 score: The F1 score provides a harmonic mean of precision and recall, offering a balanced metric that considers both false positives and false negatives. This is particularly important in our case, as it ensures that the model performs well in both accuracy (precision) and completeness (recall).

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3}$$

These metrics collectively offer a robust framework for evaluating the machine learning model's capability in accurately and comprehensively extracting class diagram components from natural language requirements. Precision focuses on the quality of the positive predictions, recall emphasizes capturing all true instances, and the F1 score balances these two aspects to provide an overall performance measure.

4 Model training and testing

The training of the machine learning model was structured using a `Pipeline`, which streamlined the process by combining data preprocessing and classification into a single step. The pipeline included a `preprocessor` for data transformation and an `SVC` (Support Vector Classifier) with `probability=True` for classification. To optimize the SVM classifier, we employed a grid search approach, systematically exploring a range of hyperparameters for each SVM kernel: linear, RBF, polynomial, and sigmoid. For each kernel type, we varied the regularization parameter `C`, and for the RBF, polynomial, and sigmoid kernels, we also varied the `gamma` parameter. Additionally, for the polynomial kernel, the degree of the polynomial was varied. This comprehensive grid search was conducted using `GridSearchCV` with 5-fold cross-validation, ensuring a robust search for the most effective model parameters.

The grid search process was executed separately for each kernel type, with the results of each search being carefully recorded. For each kernel, key metrics like mean test score, standard deviation of the test score, and rank based on test score were analyzed. Additional parameters such as `C`, `gamma`, and `degree` were also displayed where relevant. This detailed analysis allowed for a nuanced understanding of each kernel's performance. After identifying the best-performing model for each kernel, we evaluated them on the test set. The performance of these models was then summarized in a classification report, providing a comprehensive overview of metrics such as precision, recall, and F1-score. The results from each kernel were compiled into a summary table, offering a clear comparison of the efficacy of different kernels and their respective hyperparameter configurations in our specific classification task.

The model testing phase assesses the performance of machine learning models in extracting class diagram components (classes, attributes, and relationships). Following standard methodology, the models were tested on unseen data to evaluate their generalization ability, using precision, recall, and F1 score as key metrics. These metrics, commonly applied in NLP and requirements engineering, provide a balanced view of the model's accuracy, completeness, and effectiveness in correctly identifying relevant components from natural language requirements, ensuring comparability with existing studies.

To apply this framework, we developed and tested three distinct models, each designed to predict one of the key features required for generating class diagrams: `class_attribute`, `class_attribute_relationship`, and `class_class_relationship`. The following section presents the results of testing these models, providing insights into their performance and effectiveness in addressing requirement problems.

In our evaluation of the `class_attribute` prediction model, we employed a comprehensive grid search approach to identify the optimal kernel and hyperparameter settings for our SVM classifier. The grid search revealed that the linear kernel provided the best accuracy, approximately 93.5%. The identified hyperparameters were `kernel='linear'` and `C=1`, which were used to develop the final model. The overall accuracy of the model stands at 92%, indicating its reliability in predicting correct classes and attributes, which is critical for reducing errors in real-world applications.

For the `class_attribute_relationship` model, we similarly conducted a grid search to determine the optimal configurations for the SVM classifier. The results indicated that the linear kernel provided the best performance, with a mean test score of approximately 93.4%. The chosen hyperparameters were `kernel='linear'` and `C=10`. This model achieved an overall accuracy of 94% and high weighted average precision and F1-score of 0.97 and 0.94, respectively.

Lastly, the `class_class_relationship` model was also evaluated through a grid search, where the linear kernel achieved a high mean test score of approximately 96.3%. The final hyperparameters selected were `kernel='linear'` and `C=100`. This model reached an impressive accuracy of 97%, with weighted averages for precision, recall, and F1-score all exceeding 0.97.

The comprehensive results of these evaluations are summarized in Table 3. The high precision, recall, and F1-scores across models indicate robust performance in extracting relevant components for class diagram generation. While the macro averages suggest some inconsistency across classes, particularly in the `class_attribute_relationship` model, the overall high weighted averages demonstrate the models' effectiveness and reliability in practice. These findings support the utility of the selected linear kernels and their respective hyperparameters, confirming their suitability for the task of extracting class diagram components from natural language requirements.

4.1 Model deployment

In the model deployment phase, we used the NLTK library to preprocess the text and then used the three trained models to process and analyse the structured text data. These models are `class_attribute` model, `class_attribute_relationship` model, and `class_class_relationship` model, each dedicated to a specific aspect of text classification. The models are loaded using the `load` function from the `joblib` library. For demonstration, a sample text representing a problem statement is introduced for processing. This text undergoes tokenization and POS tagging using NLTK's sentence and word tokenization methods, along with POS tagging. The tokenization process breaks the text into sentences and words, while POS tagging assigns grammatical categories to each word.

After preprocessing, the data is structured into a DataFrame, comprising columns for sentences, words, and their respective POS tags. The `class_attribute` model is first applied to predict tags for each word, categorizing them as potential classes or attributes. Following this, the `class_attribute_relationship` model predicts the relationships of these words to classes, indicating

TABLE 3 Precision, recall, F1-score, and accuracy results for different models.

Models	Precision	Recall	F1-score	Accuracy
`class_attribute`	0.85	0.82	0.83	0.92
`class_attribute_relationship`	0.97	0.66	0.65	0.94
`class_class_relationship`	0.98	0.97	0.97	0.97
Macro avg	0.87	0.74	0.72	-
Weighted avg	0.92	0.92	0.92	-

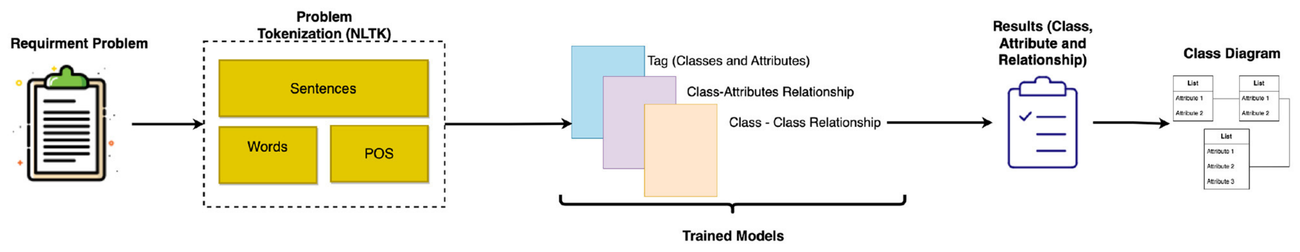
Requirment Problem
Problem Tokenization (NLTK)
Sentences
Words
POS
Tag (Classes and Attributes)
Class-Attributes Relationship
Class - Class Relationship
Trained Models
Results (Class, Attribute and Relationship)
Class Diagram

FIGURE 4
Model deployment workflow.

whether each word is related to a class or not. Finally, the `class_class_relationship` model uses the `Class and Attribute` and `class-Attribute Relations` predictions to determine relationships between classes. Figure 4 shows an overview of the model flow. This figure illustrates deploying the trained models to analyze new text data. Starting from tokenization and tagging the sample problem text, it follows through the prediction stages using the three models, culminating in extracting key information like tags, class-related words, and class relationships. The predictions from these models provide comprehensive insights into the structure and components of the text, identifying classes, attributes, and their interrelations. This information is crucial for understanding and extracting meaningful patterns from the text.

5 Evaluation

In the previous section, we tested the models using validation data from the requirements dataset. Therefore, to assess the applicability of the proposed automated requirements engineering framework, we conducted two forms of evaluation: comparative evaluation and experimental study with real-world requirements specification. The importance of rigorous evaluation in machine learning cannot be overstated, especially in requirements engineering, where the accuracy and relevance of extracted information directly impact the quality of the resulting software models. Given the inherent variability and complexity of natural language, especially in software requirements, a more robust and comparative evaluation is required to ensure that the models can generalize effectively beyond the dataset used for training. Additionally, experiments conducted with real-world software requirements will help determine whether the models can handle diverse and complex scenarios, providing valuable insights into their practical applicability.

5.1 Evaluation data and methodology

This section details the evaluation techniques and metrics employed to assess the machine learning models' performance for extracting class diagram components from natural language requirements. The approach adopted follows the standard methodology commonly used in evaluating machine learning models, ensuring consistency and comparability with existing studies in the field. Specifically, the evaluation consists of three key metrics: precision, recall, and F1 score. These metrics provide a comprehensive understanding of the model's performance, particularly in accuracy, completeness, and balance between precision and recall.

5.1.1 Comparative evaluation data

The dataset used for the evaluation consists of 12 requirement problems, which were sourced from the work of Saini et al. (2022). The problems (1–12) consist of software requirements described in natural language (NL). Each requirement description comprises sentences mostly in the format described earlier, i.e., subject, action, object, context, etc., with a focus on functional requirements. Each problem provides a detailed description of a specific requirement for the design and development of software across various domains. These problems facilitate the extraction of components for class diagrams, namely classes, attributes, and relationships, which form the primary focus of this study. Based on the three models developed, the components are systematically analyzed and extracted for further modeling and implementation. This dataset was selected based on its alignment with the nature of the evaluation problem and for the sake of maintaining fairness and consistency in comparisons with previous approaches. The selection of this dataset ensures that the performance of our model can be directly compared to the results reported by Saini et al. (2022), thus allowing for a meaningful evaluation and benchmarking. The dataset contains a diverse range of software requirements, covering different domains which were originally used to evaluate their tool for extracting domain models, particularly class diagrams, from natural language requirements. By using the same dataset, we ensure that the complexity and variety of the requirements remain consistent with the prior study, thereby enhancing the rigour and comparability of our evaluation—an aspect notably lacking in previous research. Moreover, the use of a common dataset mitigates potential biases or inconsistencies that might arise from using different datasets across studies.

A comparative assessment of our machine learning models and existing approaches for extracting class diagrams from natural language requirements is presented, highlighting both strengths and limitations. The evaluation combines quantitative and qualitative analyses using metrics such as accuracy, precision, recall, and F1 score. Models were selected for comparison based on their relevance to class diagram extraction tasks. Specifically, our models are compared to those in prior studies, such as Saini et al. (2022), which used a rule-based and machine learning approach. The work was chosen because we were able to obtain the requirements problems used in their evaluation, ensuring a fair, direct baseline for comparison.

Table 4 shows the comparative results of our model vs. DoMoBot, highlighting F1, precision, and recall across 12 requirement problems. Our model consistently outperforms DoMoBot in most cases; for instance, in Problem 1, it achieves an F1 score of 87% compared to DoMoBot's 71%, with similarly higher recall (91% vs. 67%), reflecting better true positive detection. In some problems, the difference is minimal; for example, in Problem 10, both models perform well, with our model at an F1 of 91% and DoMoBot slightly higher at 94%. The trend shows our model's balanced precision and recall performance, particularly for class and attribute extraction. This comparison uses DoMoBot's "Found Configurations" (pre-interactive adjustments), providing a fair baseline of initial output capabilities.

The comparison highlights our model's robustness across diverse requirements while also noting areas where DoMoBot remains competitive, particularly in precision for certain problems (e.g., Problems 10 and 11). Overall, our model's consistently high recall and F1 scores demonstrate its strength in balancing precision and recall, effectively managing false positives and negatives in class diagram extraction. In summary, this analysis confirms our model's high accuracy and strong performance across key metrics like F1 score and recall in processing natural language requirements, showcasing its potential for reliable class diagram extraction and identifying areas for further refinement.

5.1.2 Analysis of the experimental studies

We evaluated our machine learning models on two real-world software requirements: a stroke recovery assistant system (medical domain) and an archive space project (information management domain). These longer, more complex requirements test the model's ability to handle domain-specific language and ambiguity. Statistical data, presented in Table 5, highlights key metrics such as vocabulary diversity and class relationships, informing the evaluation of the model's precision and robustness. Thus, demonstrating the model's adaptability and effectiveness in extracting class diagram components across diverse domains.

The results of testing the proposed machine learning models on two real-world requirements problems are summarized. These experiments evaluate the models' performance in extracting class diagram components and their generalization ability with diverse data. The outcomes include quantitative metrics such as precision, recall, and F1 score, offering insight into the models' effectiveness in identifying key components from natural language requirements. Table 6 presents the performance of our class diagram extraction model on two real-world requirements problems. While accuracy is high for both systems (96% and 93%, respectively), a significant discrepancy exists between precision and recall, impacting the overall F1-score.

For System 1, both precision and recall are low (32% and 33%, respectively), resulting in a low F1-score of 32.5%. This indicates a difficulty in correctly identifying relevant components while minimizing false positives. For System 2, the model achieves high recall (97%), demonstrating its ability to capture most relevant components. However, lower precision (46%) leads to a moderate F1-score of 62.6%, suggesting a higher rate of false positives. These results highlight two key challenges:

- **Compound words and implicit relationships:** The presence of compound words in the ground truth and implicit relationships between classes posed a significant challenge for the model. For example, System 1's requirements included terms like "MobileInterface," "WebInterface," "MedicalEvent," and "DataCollection," which the model sometimes failed to correctly segment into their constituent parts (e.g., separating "Mobile" and "Interface"). This difficulty in recognizing the individual components of compound terms contributed to lower precision. Furthermore, when relationships between classes were implied rather than explicitly stated, the models struggled to consistently extract accurate relationships.
- **Data imbalance:** The software requirement problems exhibit an imbalance, with high-frequency words like "the" contributing to high accuracy but not necessarily reflecting true performance in extracting meaningful class diagram components. This imbalance also likely contributes to the divergence between precision and recall, as the model may be biased toward predicting frequent terms.

To address the model's weaknesses in precision, increasing the amount of training data is essential, as larger datasets enhance generalization and reduce false positives. Future work could involve

TABLE 4 Comparison between our model and Base Model (DoMoBot) using the same requirement problems.

Requirements	F1 (%)		Precision (%)		Recall (%)	
	Our model	Base model	Our model	Base model	Our model	Base model
Problem 1	87	71	84	75	91	67
Problem 2	83	77	77	83	93	71
Problem 3	81	67	76	75	90	60
Problem 4	81	80	76	86	92	75
Problem 5	74	82	77	87	71	78
Problem 6	95	93	93	100	98	87
Problem 7	96	80	93	80	100	80
Problem 8	73	90	76	93	79	87
Problem 9	81	94	77	96	91	92
Problem 10	91	94	90	100	92	89
Problem 11	80	95	77	100	85	91
Problem 12	98	89	97	100	99	80

TABLE 5 Statistical overview of the experimental studies.

No	Requirements name	Domain	Sentences	Words	Unique words	Stop words removed	Classes	Attributes	Relationships
System 1	Stroke recovery assistant	Health	12	256	131	174	13	10	5
System 2	Archive space project	Information system	11	187	123	132	11	2	6

TABLE 6 Results of the two studies.

Requirements	Accuracy	Precision	Recall	F1
System 1	96	32	33	32.5
System 2	93	46	97	62.6

fine-tuning large language models (LLMs) such as GPT or BERT, which excel at understanding complex language patterns, on domain-specific datasets for class diagram extraction.

In addition to the quantitative results, a qualitative evaluation compared the class diagrams generated by the machine learning models with those created by human experts to assess alignment with human understanding of the requirements. To establish the ground truth, a human expert extracted the components of the class diagrams based on the problem descriptions, producing a manual sketch that identified relevant classes, attributes, and relationships. A second expert reviewed these diagrams to ensure accuracy and reliability, providing a trustworthy benchmark for comparison with machine-generated outputs. The comparison showed that the machine learning models successfully extracted most classes and attributes identified by human experts. The automated models often recognized attributes and relationships similar to the human interpretations, demonstrating thorough parsing of the natural language descriptions. In some cases, the models identified additional classes and attributes, although these were technically functions of the system. However, when relationships were implied rather than explicitly stated, the models struggled to extract them accurately, reflecting the lower F1 scores in this area. Overall, the machine learning model extracted the most relevant classes, attributes, and relationships for the Stroke Recovery Assistant System and the archive space system, though some differences in naming conventions and relationship identification were noted between the machine-generated and manually crafted diagrams. The results indicate that the framework can extract class diagram components—classes, attributes, and relationships—providing requirements analysts with a quick overview to facilitate model-driven engineering. Figure 5 summarizes the evaluation of the extracted class diagrams for the two case studies.

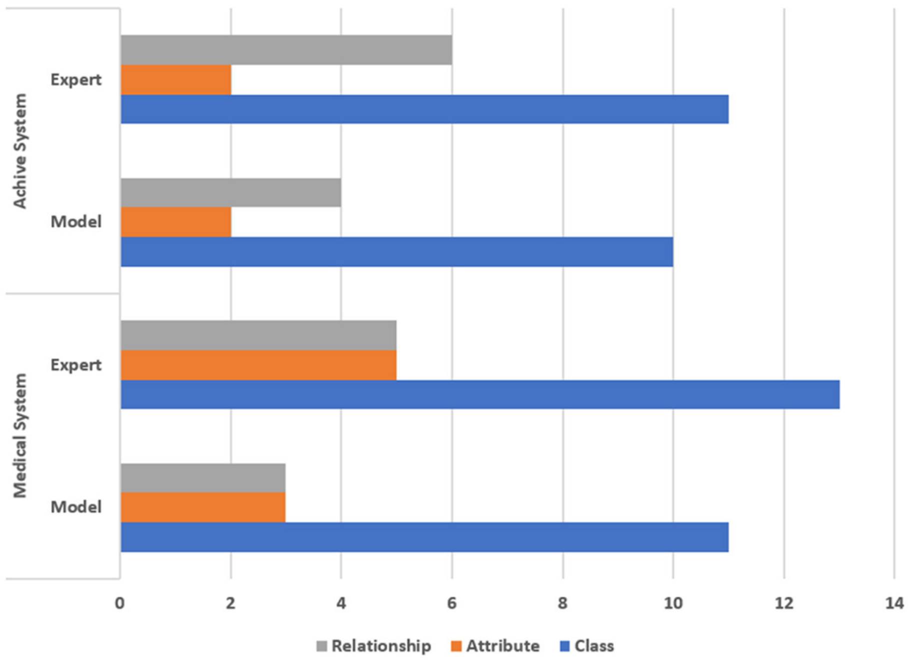

FIGURE 5
Evaluation of extracted class diagram for the studies.

6 Discussion

The comparative evaluation of our models against existing approaches, particularly Saini et al. (2022), reveals several key insights on the validation and evaluation of an automated RE tool. Across the 12 natural language requirements problems, our models demonstrated better performance in most tasks, especially in terms of recall and F1 score. For example, in Problem 1, our model outperformed DoMoBot with an F1 score of 87% compared to DoMoBot's 71%, and a recall of 91% vs. DoMoBot's 67%. This trend highlights the model's ability to capture a broader range of relevant class and attribute components, effectively reducing false negatives. However, the performance comparison also exposes areas for improvement, particularly in precision. While the model excels at recall—detecting most of the correct elements—it occasionally identifies irrelevant components, leading to false positives. In Problem 10, for instance, DoMoBot narrowly surpasses our model in precision (100% vs. 90%), indicating that DoMoBot's rule-based approach may handle certain edge cases better. This demonstrates that while our model maintains a more balanced trade-off between precision and recall, further refinement is needed in cases where precision is critical.

In the real-world evaluation using two larger problems (System 1 and System 2), the model achieved high accuracy (96% and 93%, respectively). However, a significant discrepancy between precision and recall impacted the F1-score. For System 1, both precision and recall were low (32% and 33%, respectively), resulting in a low F1-score of 32.5%. This indicates difficulty in correctly identifying relevant components while minimizing false positives. This difficulty was exacerbated by the presence of compound words within the requirements, which the model struggled to segment correctly. The high accuracy, despite low precision and recall, suggests the model may be influenced by factors like data imbalance, rather than accurately extracting class diagram components. While the results from both comparative and real-world evaluations indicate strong potential, several limitations need to be addressed. One major limitation is the model's precision, especially when dealing with real-world, complex requirements where ambiguities are more frequent. The model often misclassifies irrelevant components as part of the class diagrams, contributing to false positives. This issue was especially pronounced in System 2 of the real-world evaluation, where the precision was only 46%, despite the recall being high at 97%. This imbalance suggests that the model's ability to generalize may be limited by the complexity of the requirements language. Another performance gap is the model's struggle to infer implied relationships between classes. In the qualitative evaluation against human-expert-generated diagrams, the model successfully identified the most explicit classes and attributes but faltered when relationships were implied rather than clearly stated. This limitation indicates that the model still lacks the deep contextual understanding required to capture implicit details, which often form the basis of more complex class relationships in real-world systems.

This study highlights significant implications for automating class diagram extraction from natural language in requirements engineering. The model's high recall across diverse specifications demonstrates its potential to streamline processes by generating system models directly from requirements, reducing the workload on human analysts. However, precision issues limit its reliability for fully replacing manual efforts, especially in safety-critical systems. While it performs well in identifying components, its limitations in handling complex, ambiguous requirements necessitate human oversight to ensure accuracy. Despite these challenges, the model shows promise as a supplementary tool, particularly in agile environments, enhancing productivity by automating initial diagram extraction. Academically, it offers a foundation for research and education in automated requirements engineering, bridging theoretical concepts with practical applications.

In conclusion, the proposed model demonstrates strong potential, particularly in recall, but further refinement is required to improve precision and address the complexities of real-world requirements. The insights gained from this research provide a solid foundation for future advancements, paving the way for more practical, reliable, and efficient automated requirements engineering solutions.

7 Threats to validity

While the framework developed and evaluated in this paper shows promising results, several threats to its validity must be considered. These limitations arise from the methodology, experimental design, and the inherent constraints of automated approaches, and they affect both the internal and external validity of the study.

Internal validity: A key limitation of this study lies in the ground truth used for evaluation, as the manually created class diagrams by human experts introduce subjectivity, which can vary in interpreting requirements. Although these diagrams were verified by a second expert to reduce bias, differences in expertise and understanding can lead to inconsistencies, affecting the internal validity of the comparison with the model's output. Additionally, the model's inability to extract compound words for class names or attributes limits its accuracy; missing terms like "user profile" or "student type" results in incomplete representations of system components. Furthermore, the model struggles to capture implicit concepts within requirements, often failing to infer meanings that human experts would readily understand, such as the notion of "status" or "type" associated with student classifications. These limitations collectively challenge the model's ability to accurately represent requirements and compromise the validity of the results.

External validity: The generalizability of the framework raises significant concerns, primarily because the evaluation was conducted using software requirements exclusively in English, limiting its applicability to non-English-speaking environments. Given that software development projects are often multinational, the framework's performance in other languages remains untested, which threatens its external validity. Additionally, while the two real-world requirements specifications evaluated reflect typical software engineering scenarios, they may not encompass the full diversity and complexity of real-world projects. The limited scope of these case studies makes it challenging to generalize the findings across various software development contexts. Therefore, a broader evaluation using a more diverse range of requirements from different industries and project sizes is necessary to enhance the framework's robustness and applicability.

Construct validity: The model's reliance on natural language processing (NLP) and machine learning techniques presents inherent limitations that may affect the construct validity of the study. NLP models often struggle with domain-specific language and ambiguity, which are prevalent in software requirements. Although the framework attempts to mitigate these challenges, its capability to interpret ambiguous terms and specialized terminology remains constrained, threatening its effectiveness in accurately capturing all relevant class diagram components. Furthermore, the framework currently lacks integration with Large Language Models (LLMs), which could enhance context comprehension and enable more informed inferences about implicit concepts. Incorporating these advanced techniques could significantly improve the model's ability to interpret complex requirements, thereby bolstering the overall construct validity of the approach.

Conclusion validity: Finally, the comparison between the model's output and the manually created ground truth raises potential risks to conclusion validity. The subjective nature of human-generated class diagrams means that observed differences between the model's performance and the ground truth may not only indicate model errors but also reflect discrepancies in human interpretation. Although human validation was conducted to address this issue, any biases or inconsistencies in the manual process could still impact the reliability of the evaluation results. Additionally, evaluation metrics such as precision, recall, and F1 score provide valuable quantitative insights but may not fully capture the model's performance nuances in real-world scenarios. While these metrics indicate accuracy, they do not consider factors like usability or the model's ability to manage complex, dynamic requirements. Consequently, the recorded accuracy may not completely represent the model's effectiveness in practical, less-controlled environments, posing a potential threat to conclusion validity.

8 Conclusion and future work

In conclusion, our work on developing an automated requirements engineering framework for agile model-driven development, aimed at analysing natural language text and extracting class diagrams, has resulted in a solid foundational solution. The framework incorporates three distinct machine learning models, which were specifically developed for this purpose: the `Class_Attribute` model for extracting classes, the `Class_Attribute_Relationship` model for identifying class attributes, and the `Class_Class_Relationship` model for identifying relationships between extracted classes. Trained and evaluated on a carefully curated dataset of requirements specification problems across various domains, these models achieved high accuracy. We also conducted a comparative evaluation against an existing approach using the same 12 requirements problems and standard evaluation metrics (precision, recall, and F1 score). Our framework achieved an average F1 score of 85.00%, precision of 82.75%, and recall of 90.08%, demonstrating its effectiveness in capturing relevant components across diverse scenarios. Additionally, we tested the framework on two real-world requirements descriptions—one from the medical field and another from information systems—yielding promising accuracy rates of 96% and 93%, respectively. Overall, these findings highlight the potential for widespread adoption of automated requirements engineering frameworks, leading to improved accuracy, reduced manual effort, and more reliable software development practices. However, challenges remain, particularly in handling requirements where the model occasionally misidentifies irrelevant elements. Addressing these limitations will be crucial for further refining the framework's capabilities and ensuring its practical utility.

In the future, we plan to enhance the framework's usability and adoption by exploring multilingual representations using large language models (LLMs). Key priorities include fine-tuning LLMs for improved extraction of class diagrams, better handling of compound words, and integrating an interactive user interface. The potential of LLMs, such as ChatGPT, is substantial. Furthermore, recognising the importance of non-functional we will explore machine learning techniques for their extraction and analysis from textual descriptions. To extend these capabilities, we will also consider integrating requirements management activities, such as tracking and implementing changes throughout the software development life cycle, ensuring alignment with agile methodologies.

Data availability statement

The datasets presented in this study can be found in online repositories. The names of the repository/repositories and accession number(s) can be found below: https://github.com/amiinuumar/AutomatedRE.

Author contributions

MU: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Project administration, Software, Writing – original draft, Writing – review & editing. KL: Supervision, Writing – review & editing. AA: Formal analysis, Software, Writing – review & editing.

Funding

The author(s) declare financial support was received for the research, authorship, and/or publication of this article. The authors acknowledge the support of the EPSRC MDENet project (Grant No. EP/T030747/1). Muhammad Aminu Umar also acknowledges funding support from the Petroleum Technology Development Fund (PTDF), Federal Government of Nigeria.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Generative AI statement

The author(s) declare that no Gen AI was used in the creation of this manuscript.

Publisher's note

References

Abdouli, M., Ben Abdessalem Karaa, W., and Ghezala, H. B. (2016). "Survey of works that transform requirements into UML diagrams," in *2016 IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA)* (Towson, MD: IEEE), 117–123. doi: 10.1109/SERA.2016.7516136

Abukhalaf, S., Hamdaqa, M., and Khomh, F. (2023). On Codex prompt engineering for OCL generation: an empirical study. *arXiv:2303.16244v161*.

Abukhalaf, S., Hamdaqa, M., and Khomh, F. (2024). PathOCL: path-based prompt augmentation for OCL generation with GPT-4. *arXiv:2405.12450v121*.

Aguirre, N. (2017). "Efficient SAT-based software analysis: from automated testing to automated verification and repair," in *2017 IEEE/ACM 5th International FME Workshop on Formal Methods in Software Engineering (FormaliSE)* (Buenos Aires, Argentina: IEEE). doi: 10.1109/FormaliSE.2017.21

Akayama, S., Hisazumi, K., Hiya, S., and Fukuda, A. (2013). "Using model-driven development tools for object-oriented modeling education," in *EduSymp@ MoDELS 2013*.

Alessio, F., Giorgio Oronzo, S., and Stefania, G. (2017). "PURE: a dataset of public requirements documents," in *2017 IEEE 25th International Requirements Engineering Conference (RE)* (Lisbon, Portugal: IEEE), 502–505. doi: 10.1109/RE.2017.29

Al-Hroob, A., Imam, A. T., and Al-Heisa, R. (2018). The use of artificial neural networks for extracting actions and actors from requirements document. *Inf. Softw. Technol.* 101, 1–15. doi: 10.1016/j.infsof.2018.04.010

Bjarnason, E., Wnuk, K., and Regnell, B. (2011). "A case study on benefits and side-effects of agile practices in large-scale requirements engineering," in *Proceedings of the 1st Workshop on Agile Requirements Engineering, AREW '11, New York, NY, USA* (Association for Computing Machinery). doi: 10.1145/2068783.2068786

Bozyigit, F., Bardakci, T., Khalilipour, A., Challenger, M., Ramackers, G., Babur, O., et al. (2023). *Dataset for: Text Requirements to Models*. IEEE Dataport.

Camara, J., Troya, J., Burgueno, L., and Vallecillo, A. (2023). On the assessment of generative AI in modeling tasks. *Softw. Syst. Model* 22, 781–793. doi: 10.1007/s10270-023-01105-5

Deeptimahanti, D. K., and Babar, M. A. (2009). "An automated tool for generating UML models from natural language requirements," in *2009 IEEE/ACM International Conference on Automated Software Engineering* (Auckland, New Zealand: IEEE), 680–682. doi: 10.1109/ASE.2009.48

Deeptimahanti, D. K., and Sanyal, R. (2011). "Semi-automatic generation of UML models from natural language requirements," in *Proceedings of the 4th India Software Engineering Conference on - ISEC '11* (Thiruvananthapuram, Kerala, India: ACM Press), 165–174. doi: 10.1145/1953355.1953378

Duan, C., Laurent, P., Cleland-Huang, J., and Kwiatkowski, C. (2009). Towards automated requirements prioritization and triage. *Requir. Eng.* 14, 73–89. doi: 10.1007/s00766-009-0079-7

Elallaoui, M., Nafil, K., and Touahni, R. (2018). Automatic transformation of user stories into UML use case diagrams using NLP techniques. *Procedia Comput. Sci.* 130, 42–49. doi: 10.1016/j.procs.2018.04.010

Fill, H.-G., Fettke, P., and Kopke, J. (2023). Conceptual modeling and large language models: Impressions from first experiments with ChatGPT. *Enterpr. Model. Inf. Syst. Archit.* 18:318. doi: 10.18417/emisa.18.3

Fowler, M. (1996). *Analysis Patterns: Reusable Objects Models*. Boston: Addison-Wesley Longman Publishing Co., Inc.

Harmain, H. M., and Gaizauskas, R. (2003). CM-builder: a natural language-based CASE tool for object-oriented analysis. *Autom. Softw. Eng.* 10, 157–181. doi: 10.1023/A:1022916028950

Ibrahim, M., and Ahmad, R. (2010). "Class diagram extraction from textual requirements using natural language processing (NLP) techniques," in *2010 Second International Conference on Computer Research and Development* (Kuala Lumpur, Malaysia: IEEE), 200–204. doi: 10.1109/ICCRD.2010.71

Inayat, I., Salim, S. S., Marczak, S., Daneva, M., and Shamshirband, S. (2015). A systematic literature review on agile requirements engineering practices and challenges. *Comput. Hum. Behav.* 51, 915–929. doi: 10.1016/j.chb.2014.10.046

Jabbarin, S., and Arman, N. (2014). "Constructing use case models from Arabic user requirements in a semi-automated approach," in *2014 World Congress on Computer Applications and Information Systems (WCCAIS)* (Hammamet, Tunisia: IEEE), 1–4. doi: 10.1109/WCCAIS.2014.6916558

Larman, C. (2004). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Unified Process*. London: Pearson Education (US), third edition.

Lee, B.-S., and Bryant, B. R. (2002). "Automation of software system development using natural language processing and two-level grammar," in *Radical Innovations of Software and Systems Engineering in the Future. RISSEF 2002* (Berlin, Heidelberg: Springer), 219–233. doi: 10.1007/978-3-540-24626-8_15

Letsholo, K. J., Zhao, L., and Chioasca, E.-V. (2013). "TRAM: a tool for transforming textual requirements into analysis models," in *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (Silicon Valley, CA: IEEE), 738–741. doi: 10.1109/ASE.2013.6693146

Li, X., and Liu, Z. (2014). *Formal Specification of UML Requirement Models*. Available at: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=4f51660ddda6b9c609ed303ca490979468c211a5

Lucassen, G., Robeer, M., Dalpiaz, F., van der Werf, J. M. E. M., and Brinkkemper, S. (2017). Extracting conceptual models from user stories with Visual Narrator. *Requir. Eng.* 22, 339–358. doi: 10.1007/s00766-017-0270-1

Meth, H., Brhel, M., and Maedche, A. (2013). The state of the art in automated requirements elicitation. *Inf. Softw. Technol.* 55, 1695–1709. doi: 10.1016/j.infsof.2013.03.008

Mu, Y., Wang, Y., and Guo, J. (2009). "Extracting software functional requirements from free text documents," in *2009 International Conference on Information and Multimedia Technology* (Jeju Island, Korea (South): IEEE), 194–198. doi: 10.1109/ICIMT.2009.47

Narawita, C. R., and Vidanage, K. (2016). "UML generator - an automated system for model driven development," in *2016 Sixteenth International Conference on Advances in ICT for Emerging Regions (ICTer)* (Negombo, Sri Lanka: IEEE), 250–256. doi: 10.1109/ICTER.2016.7829928

Nasiri, S., Rhazali, Y., Lahmer, M., and Chenfour, N. (2020). Towards a generation of class diagram from user stories in agile methods. *Procedia Comput. Sci.* 170, 831–837. doi: 10.1016/j.procs.2020.03.148

Nassar, I. N., and Khamayseh, F. T. (2015). "Constructing activity diagrams from Arabic user requirements using Natural Language Processing tool," in *2015 6th International Conference on Information and Communication Systems (ICICS)* (Amman, Jordan: IEEE), 50–54. doi: 10.1109/IACS.2015.7103200

Nazir, F., Butt, W. H., Anwar, M. W., and Khattak, M. A. K. (2017). "The applications of natural language processing (NLP) for software requirement engineering: a systematic literature review," in *International Conference on Information Science and Applications*, 485–493. doi: 10.1007/978-981-10-4154-9_56

Neto, E. C. P., Dadkhah, S., Sadeghi, S., Molyneaux, H., and Ghorbani, A. A. (2024). A review of Machine Learning (ML)-based IoT security in healthcare: a dataset perspective. *Comput. Commun.* 213, 61–77. doi: 10.1016/j.comcom.2023.11.002

Paullada, A., Raji, I. D., Bender, E. M., Denton, E., and Hanna, A. (2021). Data and its (dis)contents: a survey of dataset development and use in machine learning research. *Patterns* 2:100336. doi: 10.1016/j.patter.2021.100336

Saini, R., Mussbacher, G., Guo, J. L. C., and Kienzle, J. (2020). "DoMoBOT: a bot for automated and interactive domain modelling," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings* (Virtual Event Canada: ACM), 1–10. doi: 10.1145/3417990.3421385

Saini, R., Mussbacher, G., Guo, J. L. C., and Kienzle, J. (2022). Automated, interactive, and traceable domain modelling empowered by artificial intelligence. *Softw. Syst. Model.* 21, 1015–1045. doi: 10.1007/s10270-021-00942-6

Seresht, S. M., Ormandjieva, O., and Sabra, S. (2008). "Automatic conceptual analysis of user requirements with the requirements engineering assistance diagnostic (READ) tool," in *2008 Sixth International Conference on Software Engineering Research, Management and Applications* (Prague, Czech Republic: IEEE), 133–142. doi: 10.1109/SERA.2008.34

Sharma, R., Srivastava, P. K., and Biswas, K. K. (2015). "From natural language requirements to UML class diagrams," in *2015 IEEE Second International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)* (Ottawa, ON: IEEE), 1–8. doi: 10.1109/AIRE.2015.7337625

Shibaoka, M., Kaiya, H., and Saeki, M. (2007). "GOORE: goal-oriented and ontology driven requirements elicitation method," in *Advances in Conceptual Modeling – Foundations and Applications. ER 2007* (Berlin, Heidelberg: Springer), 225–234. doi: 10.1007/978-3-540-76292-8_28

Sim, S., Easterbrook, S., and Holt, R. (2003). "Using benchmarking to advance research: a challenge to software engineering," in *25th International Conference on Software Engineering, 2003. Proceedings* (Portland, OR, USA: IEEE), 74–83. doi: 10.1109/ICSE.2003.1201189

Thakur, J. S., and Gupta, A. (2014). "Automatic generation of sequence diagram from use case specification," in *Proceedings of the 7th India Software Engineering Conference on - ISEC '14* (Chennai, India: ACM Press), 1–6. doi: 10.1145/2590748.2590768

Umar, M. A., and Lano, K. (2023). 'Automated requirements engineering in agile development: a practitioners survey," in *2023 3rd International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, 1–7. doi: 10.1109/ICECCME57830.2023.10253030

Umar, M. A., and Lano, K. (2024). Advances in automated support for requirements engineering: a systematic literature review. *Requir. Eng.* 29, 177–207. doi: 10.1007/s00766-023-00411-0

Vemuri, S., Chala, S., and Fathi, M. (2017). "Automated use case diagram generation from textual user requirement documents," in *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)* (Windsor, ON: IEEE), 1–4. doi: 10.1109/CCECE.2017.7946792

Vidya Sagar, V. B. R., and Abirami, S. (2014). Conceptual modeling of natural language functional requirements. *J. Syst. Softw.* 88, 25–41. doi: 10.1016/j.jss.2013.08.036

Wang, B., Wang, C., Liang, P., Li, B., and Zeng, C. (2024). How LLMs aid in UML modeling: an exploratory study with novice analysts. *arXiv:2404.17739.*

Wang, Y. and Zhang, J. (2016). "Experiment on automatic functional requirements analysis with the efrf's semantic cases," in *2016 International Conference on Progress in Informatics and Computing (PIC)*, 636–642. doi: 10.1109/PIC.2016.7949577

Yang, S., and Sahraoui, H. (2022). "Towards automatically extracting UML class diagrams from natural language specifications," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, MODELS '22* (Montreal, Quebec, Canada: Association for Computing Machinery), 396–403. doi: 10.1145/3550356.3561592

Yue, T., Briand, L. C., and Labiche, Y. (2015). aToucan: an automated framework to derive UML analysis models from use case models. *ACM Trans. Softw. Eng. Methodol.* 24, 1–52. doi: 10.1145/2699697

Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K. J., Ajagbe, M. A., Chioasca, E.-V., et al. (2021). Natural language processing for requirements engineering: a systematic mapping study. *ACM Comput. Surv.* 54:75. doi: 10.1145/3444689