Title of the Final Year Undergraduate Project

An undergraduate project proposal report submitted to the

Department of Electrical and Information Engineering
Faculty of Engineering
University of Ruhuna
Sri Lanka

in partial fulfillment of the requirements for the

Degree of the Bachelor of the Science of Engineering Honours

by

R. Lareef - EG/2021/4644
A. Abekon - EG/2021/4645
B. Safeer - EG/2021/4771

.....	.....
Prof. A.B.C. Dee	Prof. A.B.C. Dee
(Supervisor)	(Co-Supervisor)

Abstract

The growing complexity of modern software systems and the need for effective communication among stakeholders have made Unified Modeling Language (UML) diagrams a vital component of software engineering. However, manually creating UML diagrams from textual requirements is a time-consuming, error-prone process that requires considerable expertise. To address this challenge, this research proposes an AI-driven framework for automatically generating UML class and use case diagrams from natural language software requirements.

The proposed system leverages advanced Natural Language Processing (NLP) and Deep Learning (DL) techniques to interpret unrestricted, ambiguous, and incomplete textual inputs more accurately. By improving grammatical and semantic understanding, the framework aims to produce more reliable and comprehensive UML representations.

The methodology includes both quantitative and qualitative evaluation. Accuracy is measured using standard metrics such as precision, recall, and F1-score, while usability is assessed through feedback from students and industry professionals. A comparative analysis with existing tools demonstrates that the proposed approach offers improved scalability, robustness, and efficiency.

This research contributes to the field of software engineering by reducing manual effort, enhancing design accuracy, and improving collaboration among stakeholders. By automating UML diagram generation, the proposed framework supports more efficient and adaptive software development workflows, benefiting both academic research and industrial practice.

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence

DL Deep Learning

ML Machine Learning

NLP Natural Language Processing

UML Unified Modeling Language

Chapter 1

Introduction

1.1 Background

The rapid growth of technology and the expanding influence of the software and IT industry have significantly transformed how individuals and organizations operate. Software applications now play a central role in driving innovation across domains such as healthcare, education, finance, and entertainment. As software systems become more complex, the demand for efficient, accurate, and collaborative software development processes has increased. Effective communication tools and clear design methodologies are essential to ensure a shared understanding among all stakeholders involved in software projects.

Unified Modeling Language (UML) diagrams are widely used to represent software designs in a standardized and visual manner. They help illustrate system components, interactions, and workflows, making them understandable to both technical and non-technical users. However, creating UML diagrams manually from textual software requirements is a time-consuming and expertise-dependent task. This process often leads to inconsistencies, misinterpretations, and inefficiencies, especially in large-scale projects where speed and accuracy are critical.

Recent advancements in Artificial Intelligence (AI), particularly in Natural Language Processing (NLP) and Deep Learning (DL), provide a promising solution to this problem. NLP enables machines to understand and analyze human language, while DL models can learn complex patterns and generate meaningful outputs. By leveraging these technologies, our project aims to automate the generation of UML diagrams—specifically class diagrams and use case diagrams—from natural language software requirements. This approach seeks to provide a more comprehensive and practical solution for software design automation.

Existing automated systems face several limitations, including the inability to handle incomplete requirement documents, restrictions on sentence structure, and difficulties in understanding grammatical nuances. These issues often result in inaccurate or incomplete UML diagrams. Our proposed system addresses these challenges by supporting unrestricted natural language input, improving grammatical analysis, and handling missing information more effectively.

From a practical and commercial perspective, automating UML diagram generation can significantly reduce development time and cost. It can improve team productivity, enhance collaboration among distributed teams, and ensure consistency in design documentation. By improving efficiency, accuracy, and scalability,

our solution can help organizations gain a competitive advantage in today's fast-paced software industry.

This project proposes a robust AI-driven framework that converts textual software requirements into accurate UML diagrams. By bridging the gap between natural language and visual system models, our work contributes to the integration of AI technologies into the software development lifecycle.

1.2 Problem Statement

Despite the significant role that UML diagrams play in software development, existing tools for generating them from natural language requirements have several limitations. Most current solutions focus primarily on class diagrams, while neglecting other important diagram types such as use case diagrams. Furthermore, these tools struggle with incomplete requirement documents, restricted input formats, and grammatical inconsistencies, which often result in errors and inefficiencies. As a result, considerable manual intervention is required, reducing the scalability, accuracy, and practicality of these systems in real-world software development environments.

1.3 Aim

The aim of this project is to develop a comprehensive deep learning–based framework that automatically generates accurate UML class and use case diagrams from natural language software requirements. The framework will address key challenges such as ambiguity, incomplete or unrestricted textual inputs, and grammatical inconsistencies. By improving automation in software design, the project seeks to enhance efficiency, accuracy, and consistency in software development practices.

1.4 Objectives

To achieve the above aim, the following objectives are defined:

- **RO1:** To review and summarize preprocessing techniques and key features used in existing literature for automated UML diagram generation.
- **RO2:** To develop methods for handling incomplete, ambiguous, and unrestricted textual inputs, with improved grammatical processing for more precise UML diagrams.
- **RO3:** To design and implement a deep learning–based system capable of generating UML class and use case diagrams from natural language software requirements.
- **RO4:** To evaluate the adaptability, efficiency, and performance of the proposed system across different domains and types of software requirements.

1.5 Scope of the Project

The scope of this project includes:

- Automatic generation of UML Class Diagrams and Use Case Diagrams
- Processing of unrestricted natural language software requirements
- Use of Natural Language Processing (NLP) and Deep Learning techniques
- Handling ambiguous, incomplete, and grammatically inconsistent inputs
- Evaluation using accuracy metrics such as precision, recall, and F1-score
- Basic usability testing with students or software practitioners

The project does not cover:

- Other UML diagrams such as sequence, activity, or state diagrams
- Full industrial-scale deployment
- Domain-specific customization for every software field

1.6 Research Questions

- **RQ1:** What preprocessing techniques and key features are commonly used in existing literature for automating UML diagram generation? *(Addresses RO1)*
- **RQ2:** How can methods be developed to handle incomplete, ambiguous, and unrestricted textual inputs while improving grammatical processing for accurate UML diagram generation? *(Addresses RO2)*
- **RQ3:** How can a deep learning–based framework be implemented to generate accurate UML class and use case diagrams from natural language software requirements? *(Addresses RO3)*
- **RQ4:** How adaptable and efficient is the proposed framework across different domains and varying software requirements? *(Addresses RO4)*

Chapter 2

Literature Review

2.1 Previous Work

The extraction of UML class diagrams from informal software requirements has traditionally been addressed using Natural Language Processing (NLP) techniques combined with heuristic rules or domain ontologies. Over time, researchers have explored more advanced approaches, including machine learning (ML) and deep learning (DL), to improve automation, accuracy, and efficiency. Existing studies mainly fall into three categories: NLP-based, ML-based, and DL-based approaches [**?**].

NLP-Based Approaches

Ibrahim and Ahmad introduced the RACE tool, which uses NLP techniques to analyze software requirements and extract UML class diagrams. The process includes structural analysis, refinement of diagram elements, and validation of UML concepts. However, the generated diagrams often lack complete information [**?**].

Kumar and Sanyal developed the SUGAR tool using the Rational Unified Process (RUP) methodology. It generates UML class diagrams from natural language requirements by first creating use case diagrams and then linking them to class diagram components. This approach supports object-oriented analysis but still relies heavily on predefined structures [**?**].

Sharma et al. proposed the Functional Design Creation Tool (FDCT), which preprocesses and analyzes requirements to identify model entities and integrate them into high-level class diagrams [**?**].

Deeptimahanti and Sanyal introduced UMGAR, a semi-automatic system that uses syntactic rules and analysis trees to extract UML elements. Although it can generate multiple UML diagram types, it requires human intervention to refine relationships and remove irrelevant elements [**?**].

Similarly, the RAPID tool by More and Phalnikar can generate use case, sequence, and collaboration diagrams using an architecture similar to RACE [**?**].

Shinde et al. developed a system that applies NLP preprocessing and rule-based extraction to generate UML class and sequence diagrams, and even produce Java source code from the extracted models [**?**].

The UDRA tool by Amune and Naik combines NLP with domain ontologies to

generate various UML diagrams such as class, object, package, and deployment diagrams. It supports relationships like association, generalization, and composition [**?**].

Other language-specific approaches include the work of Jaiwai and Sammapun, who developed a system for Thai-language requirements, and Utama and Jang, who used SpaCy with linguistic rules to extract UML class diagrams from unrestricted natural language [**?**].

Abdelnabi et al. proposed an NLP and heuristic-based method for generating class diagrams from informal requirements [**?**].

Bashir et al. developed the READ tool, which uses the NLTK library and rule-based extraction to automatically identify classes, attributes, and operations. Their results showed improved accuracy compared to earlier tools [**?** **?**].

Machine Learning and Deep Learning Approaches

Although NLP-based systems improved automation, they still struggle with ambiguity, incompleteness, and uncertainty in natural language requirements [**?**].

Arachchi proposed a hybrid system that combines NLP, machine learning, and deep learning to generate UML class and use case diagrams. The system uses NLP for text processing, Naive Bayes for classification, and RNNs for identifying actors, classes, and relationships [**?**].

Saini et al. introduced DoMoBOT, which integrates NLP with ML and DL models to generate domain models as UML class diagrams. The system preprocesses text using SpaCy, extracts domain concepts using NLP rules, refines them using trained ML/DL models, and integrates the results into a final UML model [**?**].

Rigou and Khriss proposed a deep learning-based framework using Bi-LSTM, BERT, and GPT-2 models to extract UML class diagrams from natural language requirements. Their system identifies entities, removes redundant references, and classifies relationships using neural networks. Results showed that larger training datasets improved diagram accuracy [**?** **?**].

2.2 Gaps in Literature

Despite significant progress, several limitations remain in existing UML diagram extraction systems:

- **Limited Diagram Coverage:** Most tools primarily focus on generating UML class diagrams, while other important diagram types such as use case diagrams, sequence diagrams, and activity diagrams receive less attention.
- **Dependence on Rule-Based Systems:** Many approaches rely on fixed linguistic rules and heuristics, making them less flexible when dealing with diverse writing styles, complex sentence structures, or informal requirements.
- **Handling of Incomplete Requirements:** Existing systems struggle to generate accurate diagrams when requirement documents are missing key information, leading to incomplete or incorrect UML models.

- **Grammatical and Semantic Challenges:** Ambiguity, grammatical inconsistencies, and contextual misunderstandings in natural language often cause extraction errors, even in advanced NLP-based tools.
- **High Manual Intervention:** Several systems require human input to refine relationships, remove irrelevant elements, or correct extracted models, reducing scalability and automation.
- **Limited Real-World Scalability:** Many tools are tested only on small datasets or controlled inputs, making their performance in large-scale, real-world projects uncertain.
- **Restricted Input Formats:** Some systems require standardized or structured requirements, limiting their usability with freely written natural language documents.

Chapter 3

Methodology

3.1 Research Design

In addition to the initial literature review and problem analysis, this research adopts a **constructive research methodology** to design, develop, and evaluate an AI-driven framework for automating UML diagram generation from natural language software requirements. The methodology is structured into several sequential stages, as illustrated in Figure 1 and Figure 2.

3.1.1 Requirement Analysis

The process begins with the collection and analysis of natural language software requirements from academic datasets, industry documentation, and open-source repositories. This stage focuses on identifying ambiguities, inconsistencies, and missing information within the textual requirements to understand real-world challenges in requirement interpretation.

3.1.2 Preprocessing using NLP

The collected text is preprocessed using Natural Language Processing (NLP) techniques such as tokenization, part-of-speech (POS) tagging, dependency parsing, and coreference resolution. These steps help normalize the input text, resolve pronoun references, and improve grammatical clarity for accurate information extraction.

3.1.3 Sentence Classification

Each sentence is classified into UML-related categories such as *class definitions, relationships, use cases, and actors*. Transformer-based deep learning models, including BERT and GPT, are used to capture semantic meaning and contextual dependencies in the text.

3.1.4 UML Element Extraction

Relevant UML components such as classes, attributes, associations, and use cases are extracted from the classified sentences. Linguistic features and semantic pat-

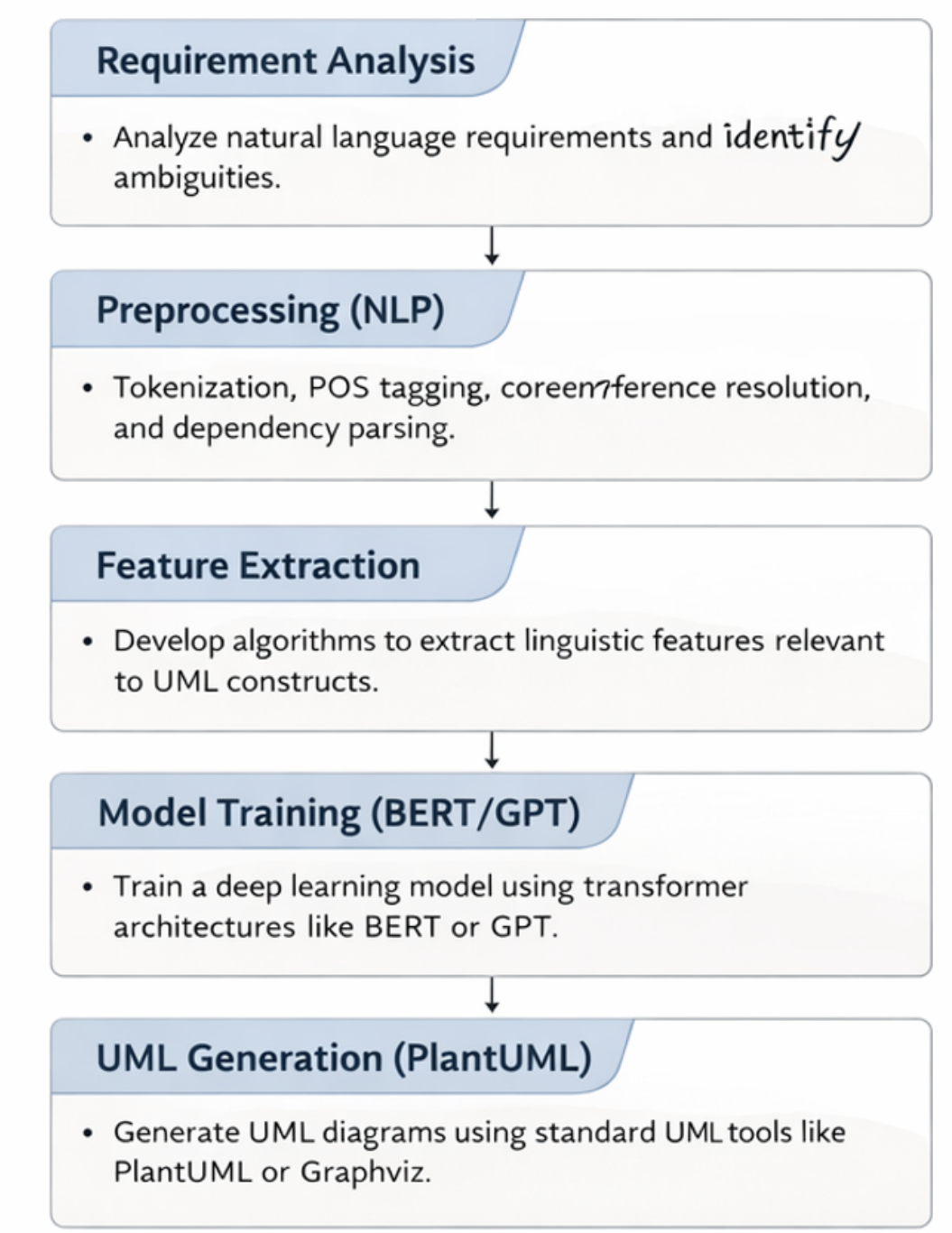

Figure 3.1: proposed Methodology

terns are analyzed to ensure accurate identification of structural and behavioral elements.

3.1.5 UML Diagram Generation

The extracted elements are converted into UML diagrams using tools such as **PlantUML** and **Graphviz**. The framework supports the generation of both *class diagrams* and *use case diagrams*, ensuring coverage of system structure and functionality.

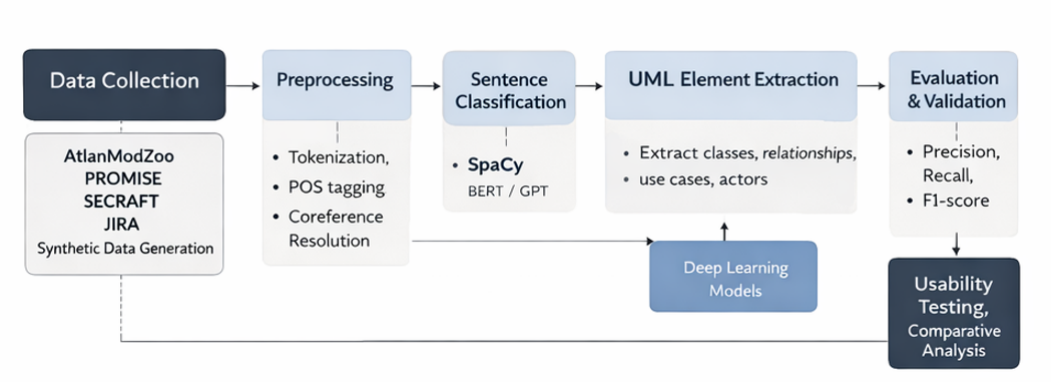

Figure 3.2: proposed Model

3.1.6 Evaluation and Validation

The generated UML diagrams are evaluated using standard metrics such as **precision, recall, and F1-score** to measure accuracy. Usability testing is conducted with software engineering students and professionals to assess clarity, usefulness, and practical applicability. Comparative analysis with existing tools highlights improvements in efficiency, scalability, and automation.

3.2 Application of the Proposed Model

The proposed framework has wide applicability across multiple domains in software engineering and education.

3.2.1 Software Development Projects

The system enables automatic generation of UML diagrams from textual requirements, reducing manual effort and improving consistency. This is especially useful in *Agile environments*, where rapid documentation is essential.

3.2.2 Education and Training

Students can focus on understanding system design rather than manual diagram creation. Educators can use the tool to assess projects efficiently.

3.2.3 Requirements Engineering

Analysts can visualize stakeholder requirements early in the development lifecycle, reducing ambiguities and improving client validation.

3.2.4 Industry Domains

The framework is suitable for domains such as *healthcare, finance, and education*, where complex workflows and regulatory compliance require precise system modeling.

3.3 Evaluation Plan

The evaluation strategy ensures the reliability, scalability, and effectiveness of the proposed framework.

3.3.1 Dataset Selection

Datasets include academic sources, industry case studies, and open-source documentation to ensure diversity and linguistic complexity.

3.3.2 Evaluation Metrics

- **Accuracy:** Precision, Recall, F1-score
- **Completeness:** Comparison with ground-truth UML diagrams
- **Scalability:** Runtime and resource usage

3.3.3 Experimental Validation

The system is compared with existing UML automation tools using benchmark datasets and real-world case studies.

3.3.4 Usability Testing

Surveys and interviews gather feedback on diagram clarity, usability, and practical value.

3.3.5 Error Analysis

Misclassifications are analyzed to improve handling of ambiguous and incomplete inputs.

3.3.6 Discussion

The proposed framework introduces a robust AI-based solution for automating UML diagram generation using advanced NLP and deep learning techniques. By supporting both *class* and *use case diagrams*, the model addresses structural and behavioral system representations. Its ability to process *unrestricted and ambiguous text* makes it suitable for real-world requirements.

The use of transformer models such as BERT and GPT enables deep semantic understanding, while integration with PlantUML ensures industry-standard outputs. Although challenges such as computational cost and domain adaptability remain, the framework significantly improves efficiency, reduces manual effort, and enhances collaboration among stakeholders.

3.4 Data Collection

To train and evaluate the proposed AI-driven UML generation framework, a diverse and high-quality dataset of natural language software requirements will be collected from multiple reliable sources. The data collection process will focus on ensuring linguistic diversity, domain variety, and real-world relevance.

3.4.1 Data Sources

The dataset will be compiled from the following sources:

- **Open-source Software Repositories:** Requirement documents, user stories, and project descriptions will be extracted from platforms such as GitHub, SourceForge, and GitLab.
- **Academic Datasets:** Publicly available datasets used in software engineering and NLP research will be utilized to ensure benchmark comparability.
- **Industry Case Studies:** Sample requirement documents from industry projects (where permitted) will be included to reflect real-world complexity.
- **Synthetic Data Generation:** Artificial requirement texts will be generated to simulate incomplete, ambiguous, and unstructured requirements.

3.4.2 Data Types

The collected data will include:

- Functional and non-functional requirements
- User stories
- Use case descriptions
- Software specification documents

3.4.3 Data Annotation

The dataset will be manually and semi-automatically annotated with:

- UML classes
- Attributes
- Relationships
- Use cases
- Actors

These annotations will serve as ground truth for training and evaluating the deep learning model.

3.4.4 Data Preprocessing

Before model training, the collected data will be cleaned and standardized by:

- Removing noise and irrelevant text
- Resolving pronouns using coreference tools
- Correcting grammatical inconsistencies
- Normalizing sentence structures

3.4.5 Ethical Considerations

All data will be collected from publicly available or authorized sources. Any sensitive or proprietary information will be anonymized to ensure privacy and compliance with ethical research standards.

Chapter 4

Timeline and Resource Required

4.1 Timeline

4.2 Resource Required

Time Period	Activities	Deliverables
Month 1–2	• Conduct literature review on UML automation • Study existing NLP and AI-based tools • Identify challenges and refine problem statement	• Literature Review Report • Refined Problem Statement
Month 3–4	• Prepare training and testing dataset • Design preprocessing pipeline (tokenization, POS tagging, parsing) • Explore NLP and DL models (BERT, GPT)	• Prepared Dataset • Preprocessing Pipeline Design
Month 5–6	• Implement AI-based UML extraction model • Train and fine-tune the model • Generate UML class and use case diagrams • Integrate PlantUML / Graphviz	• Working UML Generation System • Generated UML Diagrams
Month 7	• Evaluate using precision, recall, F1-score • Compare with existing tools • Conduct usability testing	• Evaluation Report • Performance Comparison Results
Month 8	• Finalize system • Write final report • Prepare presentation and documentation	• Final Project Report • Presentation Slides • System Documentation

Table 4.1: 8-Month Project Work Plan

Resource Category	Description
NLP Tools	SpaCy, NLTK, and Coreferee will be used for text preprocessing tasks such as tokenization, part-of-speech tagging, dependency parsing, and coreference resolution. These tools help extract linguistic features required for UML diagram generation.
Deep Learning Frameworks	TensorFlow and PyTorch will be used to develop and train deep learning models. Pretrained transformer models such as BERT, GPT, or RoBERTa will be used for feature extraction and classification.
Diagram Generation Tools	PlantUML and Graphviz will be used to automatically generate UML class and use case diagrams. These tools support multiple UML diagram types and allow customization according to UML standards.
Data Storage and Management	PostgreSQL or MongoDB will be used to store datasets, training results, and generated UML models in an organized manner.
Hardware Resources	A computer with sufficient processing power (GPU support preferred), at least 8GB RAM, and stable internet access for model training and testing.
Software Tools	Python, VS Code, GitHub for version control, and LaTeX for documentation and report writing.

Table 4.2: Resource Requirements for the Project

Chapter 5

Conclusion

This research proposes a robust AI-driven framework for automating the generation of UML diagrams from natural language software requirements. By addressing key challenges such as ambiguity, incomplete inputs, and grammatical inconsistencies, the proposed system enhances the accuracy and efficiency of software design processes. The integration of advanced Natural Language Processing (NLP) and Deep Learning (DL) techniques enables scalable automation and supports multiple UML diagram types, thereby improving collaboration among stakeholders.

The proposed methodology, supported by rigorous evaluation and validation, ensures that the framework is practical and suitable for real-world software engineering applications. Although certain limitations, including scalability and domain-specific adaptability, require further investigation, this research establishes a strong foundation for incorporating AI-based solutions into the software development lifecycle.

The outcomes of this study have the potential to transform how software designs are documented and communicated, offering significant benefits to both academia and industry. Future work will focus on extending support for additional UML diagram types, enhancing domain adaptability, and refining evaluation methods to further improve the effectiveness of AI-driven software design automation.

References

[1] A. Jones, *Machine Learning: Theory and Practice*. Tech Publications, 2019.