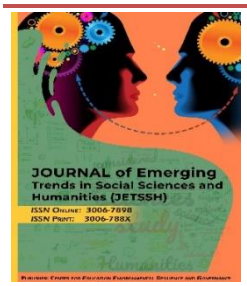


Journal of Emerging Trends in Social Sciences and Humanities

A Double-Blind, Peer-Reviewed, HEC recognized Y-category Research Journal

E-ISSN: 3006-7898 **P-ISSN:** 3006-788X

Volume 3, Issue 3, 2026, 173-192

Machine Learning-Driven Software Architecture Pattern Recommendation: Bridging Requirement Specifications and Design Decisions

Sadia Nazeer[1] Iman Ali[2] Saleem Zubair[3] Dr. Waseem Iqbal[4]

[1] Department of Software Engineering, Superior University, Lahore, Punjab, Pakistan.

[2] Department of Computer Science and Information Technology, Superior University, Punjab, Pakistan.

[3] Department of Computer Science and Information Technology, Superior University, Lahore, Punjab, Pakistan.

[4] Department of Computer Science and Information Technology, Superior University, Lahore, Punjab, Pakistan.

Corresponding Author: Sadia Nazeer, **Email:** zeeshan.zz54@gmail.com

Article Information

Received:

2026-06-20

Revised:

2026-07-23

Accepted:

2026-08-12

Keywords

Software Architecture; Non-Functional Requirements; Event-Driven Architecture; NLP; Architecture Pattern Recommendation.

ABSTRACT

This paper intends to present the Requirement-Aware Architecture Pattern Recommendation System that uses the Machine Learning(ML) as input with functional and non-functional requirement sets (FRs/NFRs) and automatically outputs a best-fit architecture pattern. The proposed approach is based on the use of deep natural language processing techniques (e.g., TF-IDF, word embedding) to identify meaningful features in requirement text. These features are then correlated with important quality attributes (scalability, security, performance, maintainability, availability etc.), and finally, supervised ML Classifiers (Random Forest, SVM, Decision Trees, NNs etc.) are employed to predict the most suitable architecture style. The system is trained and validated with-standard datasets, like PROMISE_exp, and also real-world Software Requirement Specification (SRS) documents. A demo-able, real-world prototype is also suggested, which would enable a user to put in his/her requirements and receive a real-time suggestion, justification, and a confidence score. This research will offer software architects, students and small-to-medium enterprises a tool to quickly, consistently and without bias select software architecture. The lack of work in the literature which discusses both NFR classification and architecture detection upon implementation of the system into source code will be addressed in the proposed approach; novelty comes from providing an end-to-end automated recommendation at the requirement level (pre-implementation stage).

Publisher: **Institute for Educating Environmental Resilience and Governance**

1. Introduction

As Artificial Intelligence (AI), Machine Learning (ML) and Natural Language Processing (NLP) develop rapidly, requirement engineering research has come under the spotlight for developing intelligent methods to automate requirement engineering tasks. Some of the first attempts in this field were made by Kurtanović and Maalej [1] who proposed a once again supervised machine learning based on Support Vector Machines for automatic classification of software requirements to Functional Requirements and Non Functional Requirements. They developed that the information from the requirements in the text could be transformed into lexical and syntactic information, which can be utilized to aid automated requirements classification with minimal manual participation. This study laid the basis of machine learning to software requirement engineering and prompted many others.

Based on this research, Cleland Huang et al. [2] suggested an intelligent solution that automates the discovery of Non Functional Requirements from software specifications. In their research, they showed that the requirement documents have enough textual information to identify quality related requirements with the help of automated analysis techniques. In order to enable research in this area, Lima et al. [3] proposed the PROMISE Exp dataset which is among the top-most adopted set of benchmark datasets for research on requirement classification. Having publicly available datasets helped researchers test and compare various machine learning models in a consistent environment, which was crucial for advancement in the field of automated requirement engineering. Having publicly available datasets greatly sped up research in automated requirement engineering as it allowed researchers to test and compare various machine learning methods in a consistent environment, which was crucial for the advancement of the field.

With the emergence of more sophisticated NLP models, researchers delved into more sophisticated ones for the classification of software requirements. Rahman et al. [4] used the combination of LSTM and GRU networks to extract the context nature of the software requirements and have used these soft contexts with their ML algorithms to promulgate classification performance better than traditional ML algorithms. More recently, to improve representation of semantic relationships among software requirements, Li et al. [29] introduced Graph Attention Networks that achieved even better classification accuracy. Furthermore, Dalpiaz et al. [30] highlighted the need for XAI by proposing XAI-enabled interpretable machine learning models with the ability to explain the why of certain requirements assigned to certain categories. Conforming to this, Maalej et al. [5] showed how a machine learning technique could automatically classify application reviews in various requirement categories, and Slankas and Williams [28] suggested approaches for extracting NFRs from the existing software documentation. All these studies prove the Machine Learning & Natural Language Processing are effective tools of intelligent requirement analysis.

While automated requirement classification has had a great degree of success, software architecture selection is still mostly done via manual analysis and expert judgement. Documenting software requirements in Software Requirement Specification documents, software architects compare these requirements to their professional experiences and knowledge of software quality attributes to decide on which style to use for the software architecture. Layered Architecture, Event Driven Architecture, Micro services Architecture, Client Server Architecture, Pipe and Filter Architecture are common architectural patterns. The suitability of each of these architectural styles is heavily reliant on the nature of the software system as well as the quality attributes discovered through analysis. Thus, architecture selection in Software Engineering is one of the most complicated and experience coupled tasks.

Among the software requirements, NFRs have the most significant impact upon architectural decision making since they set quality expectations for the software system. The scalability, security, maintainability, performance, reliability, availability, interoperability, and fault tolerance requirements of the application shape the choice of architectural style. Klein et al. [7] proposed the concept of Attribute Based Architecture Styles, thus making a direct connection between Software Quality

Attributes and Software Architecture Selection. They proved that building should be designed based on quantifiable building qualities and not on intuition or personal preference. Similarly, studies on architecture evaluation, such as the Software Engineering Institute study [8] indicated that different architectural styles offer various levels of support for specific quality attributes; and a systematical requirement analysis is necessary prior to choosing an architecture.

The evolution of the adoption of distributed computing and cloud computing and the large scale software applications have added yet other layers of complexity to architectural decision-making. Decisions on architectures are increasingly a challenge because today's software systems need to be high scalable, continuously available, tolerant of failures, and deployment independent. While there are great advantages to using micro services in terms of scalability and maintainability, as demonstrated by comparative analysis undertaken by De Lauretis [10] and Blinowski et al. [11], the approach is not suitable for all software applications, and adds to operational complexity. They conclude that whenever choice is available there is no need to make architectural decisions according to the trend of the times. Other industrial platforms that are employed to support modern AI applications and very large software distributed systems, like Google's Tensor Flow Extended platform, Uber's Michelangelo platform, and Netflix's recommendation system, reinforce the value of choosing an architecture that will enable those applications to scale and run on distributed systems [12] [13] [14].

Recently, AI has been used not just for classification of requirements, but to facilitate the software architecture design and software architecture recommendations as well. Bucaioni et al. [20] performed a comprehensive and extensive systematic literature review to discover different applicative domains that allow the use of Artificial Intelligence for Software Architecture activities. Esposito et al. [21] in turn explored the growing importance of Generative Artificial Intelligence on architectural design and pointed out its prospects in the software engineering domain. Eisenreich et al. [19] developed an Artificial Intelligence based approach to translating software requirements into architectural designs, whereas Dhar et al. [22] studied the ability of Large Language Models for making architectural decisions. These studies show some promising results, but most current methods are still semi-automatic and are strongly reliant on expert validation. Additionally, there aren't widely adopted solutions that use structured machine learning models to provide objective, repeatable and explainable architectural recommendations based on prompts.

Although there have been slow but steady advances in requirement engineering and the development of architectures supported by the use of Artificial Intelligence, there is a critical missing research area. More often, existing studies concentrate either on automatically categorizing software requirements, or on recognizing architectural patterns after construction of software is finished with the help of the source code metrics. Combining directly requirement engineering and software architecture recommendation is quite rare in literature, and the current available Artificial Intelligence (AI) based solutions for recommendation often do not incorporate all the necessary components of a Machine Learning (ML) pipeline such as requirement extraction, requirement classification, quality attribute analysis, software architecture prediction, and justification of the software architectures.

To overcome these drawbacks, an approach to be proposed in this research is a Requirement Aware Software Architecture Recommendation System (RASARs), which aims at automatically obtaining optimal software architecture recommendation through combination of Natural Language Processing and Supervised Machine Learning techniques directly from Software Requirement Specifications book. The proposed framework intelligently preprocesses the software requirements, extracts meaningful textual features using Natural Language Processing techniques, identifies important software quality attributes and introduces the supervised classifiers of Machine Learning (Random Forest, Support Vector Machine, Decision Tree and Neural Networks) to predict the most appropriate architectural pattern. The proposed system will be trained and validated with publicly available datasets e.g. PROMISE Exp and actual Software Requirement Specification documents. Besides

recommending the target architecture, the system also recognizes recommendation confidence and gives the explanation behind the architecture recommended, which helps software architecture decision making to be transparent and objective.

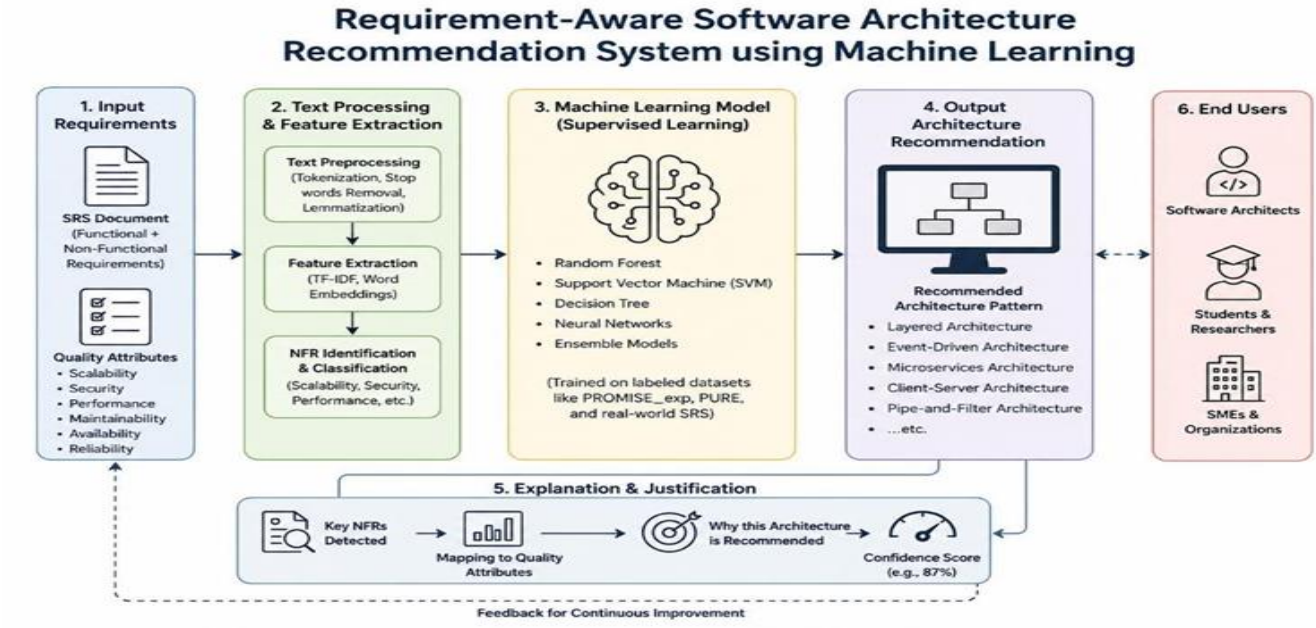
Research Motivation

One of the biggest bugbears in software quality, development costs, maintenance and failure of projects is using an unsuitable software architecture. A lot of research has been done on how to classify software requirements and how to use Artificial Intelligence in software engineering but there has been no extensive intelligent system to use which can automatically convert Software Requirement Specification documents into architecture recommendations before the software starts to be implemented. Current one's label requirements without providing any architectural advice, label architectural patterns as a result of running the code, or make architectural recommendations with LLMs which need a lot of human checking. Due to these limitations, this research is aimed at developing an end to end Machine Learning based recommendation framework that can be used to objectively, consistently and efficiently help the software architects, students and software organizations to choose the most fit software architecture during the early phase of software development.

Research Contribution

This work is valuable as a first attempt to address an intelligent Requirement Aware Software Architecture Recommendation System (RA-SARS) using the integration of Natural Language Processing (NLP), classification of requirements, Software Quality Attribute (SQA) analysis and Supervised Machine Learning under a wholistic approach in a unified framework. The proposed system has a unique feature of providing an end to end solution for analyzing Functional and Non Functional Requirements, identifying quality attributes for software architecture, recommending appropriate software architecture pattern, presenting confidence scores, and providing justification for the proposed software architecture pattern. Unlike previous studies which involve only single stage of requirement analysis and/or architecture prediction, the proposed system provides end-to-end solution for analyzing both Functional and Non Functional Requirements along with identification of quality attributes for software architecture, selection of the most appropriate software architecture pattern, and presenting confidence scores with justification. Proposed framework can minimize expert judgment to achieve consistency in software architecture, can facilitate the early design decisions of software, and can offer a practical decision support tool to software design researcher groups, software architects, students and software development organizations.

Figure 1: *The Proposed Requirement-Aware Software Architecture Recommendation System is an overview of the system.*



Description

The overall process of the proposed Requirement-Aware Software Architecture Recommendation system is shown in Figure 1. The initial step in this process is to start with the Software Requirement Specification (SRS) document, which consists of a number of Functional Requirements (FRs) and Non-Functional Requirements (NFRs). The input requirements are processed to remove irrelevant information and reduce data dimensionality through text preprocessing and feature extraction techniques using NLP algorithms like lemmatization, stop-word removal and TF-IDF, word embedding's. The features extracted are then utilized to recognize important quality elements like scalability, security, performance, maintainability, availability, reliability, and so on.

Such processed features are fed into the algorithms of example supervised Machine Learning models like Random Forest, Support Vector Machine (SVM), Decision Tree, and Neural Networks, which are trained on different standard datasets like PROMISE_exp, PURE, real world Software Requirement Specification (SR) documents. Based on the learned patterns, the system will suggest the most suitable software architecture that can be adopted such as: Lastly, the system produces an explanation for every suggestion as well as a reliability rating that helps software architects, researchers, pupils and computer software organisations to make informed, unbiased architectural decisions. In addition, a feedback mechanism is added for ongoing performances and accuracy improvements of the recommendation model in future learning processes.

2. Literature Review

The traditional approach to software architecture selection has usually been manual and experience based in which the senior software architect brings his domain knowledge to the table to help him translate system requirements into one or more of the types of architecture that he is familiar with, such as layered, event driven, micro-services, client server, or pipe and filter architectures. This is a way adopted in industry to balance multiple quality attributes along with multiple of business decision and goals for the architecture decision. But the manual decision process is subjective, time consuming and very much reliant on the individual skill of an architect. Concurrently, with the rise of complexity and layered nature of software systems, researchers have turned their focus to the use of Machine Learning (ML) and Natural Language Processing (NLP) techniques to automate the decision making process related to the architecture. These intelligent techniques work toward analyzing requirement documents and suggesting appropriate architectural style which, at the same time, minimize human efforts involved and ensure consistency in the software designing decisions.

Assessing software architecture focuses mainly on the Non Functional Requirements (NFRs) like scalability, security, reliability, maintainability, availability, performance and interoperability. Notification Functional Requirements concentrate on the functionality of a system, whereas NFRs specify the performance of a system under certain operating conditions. Architectural decisions are directly affected by these quality attributes since not all architectural styles will have the same support for each of these requirements.

The concept of Attribute Based Architecture Styles was introduced by Klein et al. [7] who showed that ArchiPCs can be chosen according to their ability to meet certain quality attributes. Their findings enabled a great connection to be made between NFRs and architecture selection since they demonstrated that architectural decisions should be made on NFRs that can be measured, and not on intuition. Likewise, we found from systematic studies and industrial reports [7][8] that there are pro cons of different architectural styles regarding to their scalability, maintainability, fault tolerance and performance. The outcome of these investigations indicates the requirement of intelligent system that automatically maps software requirements to the most appropriate architectural style.

Requirements are classified into the appropriate categories by employing machine learning and natural language processing techniques. To provide related class of requirements through machine learning and NLP.

The first stage and also one of the most essential stage of any automatic design system is requirement classification. Prior to making the decision on the architecture of the system, it is important that it can correctly differentiate

functional requirement from the nonfunctional requirement as these will play different roles in making the architectural decision.

One of the earlier automatic task classification models based on machine learning techniques are those proposed by Kurtanović and Maalej [1] who used Support Vector Machines (SVM). They computed lexical and syntactic features from the requirement statements and managed to get a good classification accuracy. Cleland Huang et al. [2] introduced an early attempt in automatically detecting Non Functional Requirements from software specifications using intelligent methods which proved that it is feasible to use intelligent methods for requirement engineering.

To enable studies in this field, Lima et al. [3] presented the popular PROMISE Exp dataset, which is now used as a standard data set for requirement classification studies. Moreover, taking an old approach to machine learning, Rahman et al. [4] studied deep learning models such as Long Short Term Memory (LSTM) and Gated Recurrent Units (GRU) to classify NFRs and found their performance levels to be better because they were able to take advantage of a "contextual relationship" presented in the "textual requirements." To more accurately capture the semantic relationships between requirements, Li et al. [29] further attempted to use Graph Attention Networks, which can better represent the semantic relationships between requirements, to improve the accuracy of classification.

Studies on the interpretability of intelligent requirement classification models, also, have been conducted recently. To explain the information of the classification, Dalpiaz et al. [30] put forth a framework of explainable machine learning by leveraging dependency parsing. Explainable Artificial Intelligence is especially relevant as software architects should be able to grasp the reasoning behind their selection of requirement and assign it to a specific category prior to making the design decision. Maalej et al. [5] proved that machine learning is an effective way to automatically classify app reviews according to application requirement categories, thus they presented helpful techniques to support application requirement mining from user generated content. Slankas and Williams [28] also provided techniques to automatically extract NnFRs from existing software document, which can also alleviate manual efforts in the requirements engineering process.

The concept of architecture-prediction using machine learning is introduced in this section. In this section, the concept of architecture-prediction using machine learning is introduced.

There is a great deal of works devoted to the task of requirement classification; however, there are not that much works on machine learning approach for predicting software architecture patterns. One of such notable works is that of Komolov et al. [6] who gathered 5973 instances of software projects from GitHub repositories and applied nine machine learning algorithms to predict the software architectural pattern: Model View Presenter (MVP), Model View ViewModel (MVVM). The most successful models they tried reached about 83 percent correctness, showing that machine learning can be used to predict the software's architectural patterns from characteristics.

The metric information they use in their research, however, is mostly the source code metrics and information regarding the implementation level. The time when architecture prediction takes place is when the software development has considerably advanced. Conversely, an intelligent recommendation system, working directly on a requirement document, can aid the software architect much earlier in the Software Development Life Cycle to select an appropriate architecture prior to the start of implementation. This prospective capability can also help to dissipate design mistakes, make better development arranging and minimize task expenses.

Micro services have become one of the most popular software architectures due to the explosive development of distributed computing. However, micro services are not suitable for each software system. Deciding for a monolithic, layered, micro services as well as an event-driven approach demands attention to project requirements, system complexity, organization/constraints and quality attributes.

According to De Lauretis [10] there are various benefits to migrating from a monolithic system to microservices, such as enhancements in

Scalability, Maintenance and Service independent deployment. Blinowski et al. [11] performed a comparison on the architectural styles based on their performance, scalability and complexity of operation and put forth their proposition that the decision on architecture should always be based on the specific requirements of the application and not what is cool about the time.

Shabani [16] and Pala Marchuk [17] provide further insights into design principles of distributed micro services systems, highlighting modularity, loose coupling and service independence. In addition, architecture selection through setting up industrial cases shows good evidence of the quality requirements. In an effort to support scalable deployment of machine learning pipelines, Google introduced its Tensor Flow Extended platform (TFX) [12] and Uber's Michelangelo platform [13] which are built with micro services. For example, Netflix's recommendations system [14] demonstrates the impact of the architecture on system scale, reliability and maintainability. Singh [15] made a contrast of monolithic and microservices architectures in MLOps and explored the deployment complexity, operational load, and some of the pros and cons of scalability. Oye et al. [18] also emphasized on the role of microservices architecture in providing to large scale software serving AI applications.

With recent development in the field of Artificial Intelligence, researchers are not only focused on classification of the requirements but also on intelligent architecture generation of software. Large Language Models (LLMs) and Generative Artificial Intelligence (GAI) have been shown to holds tremendous promise in supporting architects in their design decisions, architectural documentation, and architectural reasoning. A summary of the methodology, key findings and research gaps of related studies is provided in Table 1 The comparison shows that previous research can achieve successful use of the Machine Learning and intelligent techniques across various fields, however, an intelligent-based software architecture recommendation system from the software requirements based on Machine Learning in the software is an unexplored area.

In order to be able to present the Artistic Intelligence applications in software architecture, Bucaioni et al. [20] recently performed a detailed systematic literature review on 51 primary studies and extracted fourteen major fields of applications in which AI is applied to Software Architecture. Esposito et al. [21] gave an overview of how Generative Artificial Intelligence has taken the center of architectural design and explained the opportunities and challenges of using AI for architecture decision making.

One of the most related works is proposed by Eisenreich et al. [19] which suggest a semi-automatic method for: Automatically generating architectural designs from software requirements using Artificial Intelligence techniques. Their works have shown themselves to be successful, however, human interaction still plays a major role in most of their architectural discourse. In another study, Dhar et al. [22] explore the ability of Large Language Models to generate reliable architectural design decisions, suggesting that LLMs offer valuable architectural suggestions, but they are not entirely trustworthy, as the validation of their outputs is left to very experienced architects because of concerns about consistency, reasoning, and explain ability. Based on these studies, it can be inferred that although AI shows great promise in software architecture design, current methods are mostly advisory instead of being entirely automated.

The proposed research also extends well-established theories and frameworks of software architectures. Garlan et al. [24] proposed very comprehensive techniques for software architectural documentation in terms of several architectural views, offering a standardised view of software systems. Kruchten's 4+1 View Model [26] is one of the widely-used architectural documentation frameworks, which allows for the architectural view of software systems from logic views, process views, development views, physical views, and scenario views.

A popular and notable Software Architecture in Practice framework is introduced by Bass, Clements, and Kazman [27] where they describe how architectural decisions affect software's quality attributes throughout the software development lifecycle. Bosch and Molin [25] added two methods to improve verification of architectures and transformation of architecture to

adapt to changing system requirements. The Software Engineering Institute evaluation guidelines also give industrially accepted best practices for evaluating software architectures for quality attribute scenarios and architectural trade-offs [8].

Apart from architectural frameworks, Ferrari et al. [23] proposed the publicly available PURE data set which is a valuable training data set for the intelligent architecture recommendation system of software requirement specification documents. In summary, all these resources serve as a foundational framework that also furnishes essential datasets and theoretical studies for creating dependable machine learning-driven architecture suggestion models.

There have been important contributions in the literature from the field of applying Machine Learning and Artificial Intelligence to RE and SWA. All the studies focus only on classification of the Functional and Non Functional Requirements and don't extend the process further to architectural decision making. Other research predicts architectural structures implemented in artifacts like source code metrics, so that architecture identification takes place after software is developed. The recent research on that topic is mostly focused on creating architecture suggestions via prompt based reasoning, though most of them lack machine learning pipeline, explain ability, and objective evaluation mechanisms.

To the best of our knowledge, there is no existing complete end-to-end framework able to take Software Requirement Specification files which include Functional Requirements and Non Functional Requirements, and automatically extract and categorize the requirements applying the techniques of Machine Learning and Natural Language Processing, analyze the quality attributes of the requirements and suggest the most suitable of the available following architectures: Layered Architecture, Event Driven Architecture, Microservices Architecture, Client Server Architecture, Pipe and Filter Architecture. Based on this, this research focuses on filling up this critical hole and provides an intelligent machine learning - driven architecture recommendation system that can assist software architects in the early phase of software development. The contribution can positively influence the non-expert related decision-making of architects, lead to more consistence of architectural decision making, and create a usable tool to support the modern software development project.

Table 1: *Comparative Summary of Related Studies (2023–2025)*

Article Title	Authors & Year	Method / Technique	Key Results	Research Gap
Classification and Prediction of Significant Cyber Incidents (SCI) Using Data Mining and Machine Learning (DM-ML)	Mumtaz, G., Akram, S., Iqbal, M.W., et al. (2023) [31]	Data Mining; Support Vector Machine (SVM); Random Forest; Neural Networks; supervised Machine Learning	Achieved high prediction accuracy for significant cyber incidents, demonstrating the effectiveness of Machine Learning in intelligent decision support and security prediction.	Focuses on cybersecurity prediction only; does not address software requirement analysis or software architecture recommendation.
Machine Learning-Based Approach for Early Screening of Autism Spectrum Disorders	Jabbar, U., Iqbal, M.W., et al. (2025) [32]	Supervised Machine Learning; medical data classification; predictive analytics	Developed an intelligent early screening model for Autism Spectrum Disorders with improved classification performance and decision support.	Application is limited to healthcare diagnosis and does not investigate requirement engineering or architecture selection.
Usability and Optimization of Online Apps in User's Context	Iqbal, M.W., Shinan, K., Rafique, S., et al. (2024) [33]	Usability evaluation; optimization framework; user-centered analysis	Improved application usability and user experience through optimization techniques	Concentrates on usability optimization rather than Machine Learning-based architecture recommendation from software requirements.

Table 1 below shows a comparative summary of the latest study done by Waseem Iqbal and his team. Each study information is highlighted on the table using the method, findings and gaps. While these works have addressed the application of Machine Learning, data mining, ontology-based systems and usability optimization in the different application domains successfully, to the best of our knowledge, none of these works concentrate on building an end-to-end Machine Learning framework for automatic software architecture recommendation from Software Requirement Specification (SRS) documents. The gap was identified motivates the proposed Requirement-Aware Software Architecture Recommendation System.

3. Research Methodology

For this research, quantitative and experimental research methodology is used for the development and evaluation of the Requirement-Aware Software Architecture Recommendation System with Machine Learning (ML) and Natural Language Processing (NLP). The proposed methodology comprises of the following sequential steps mentioned below: data collection, requirement preprocessing, feature extraction, requirement classification, architecture recommendation, model evaluation and prototype implementation. The entire process of proposed methodology is shown in Figure 2.

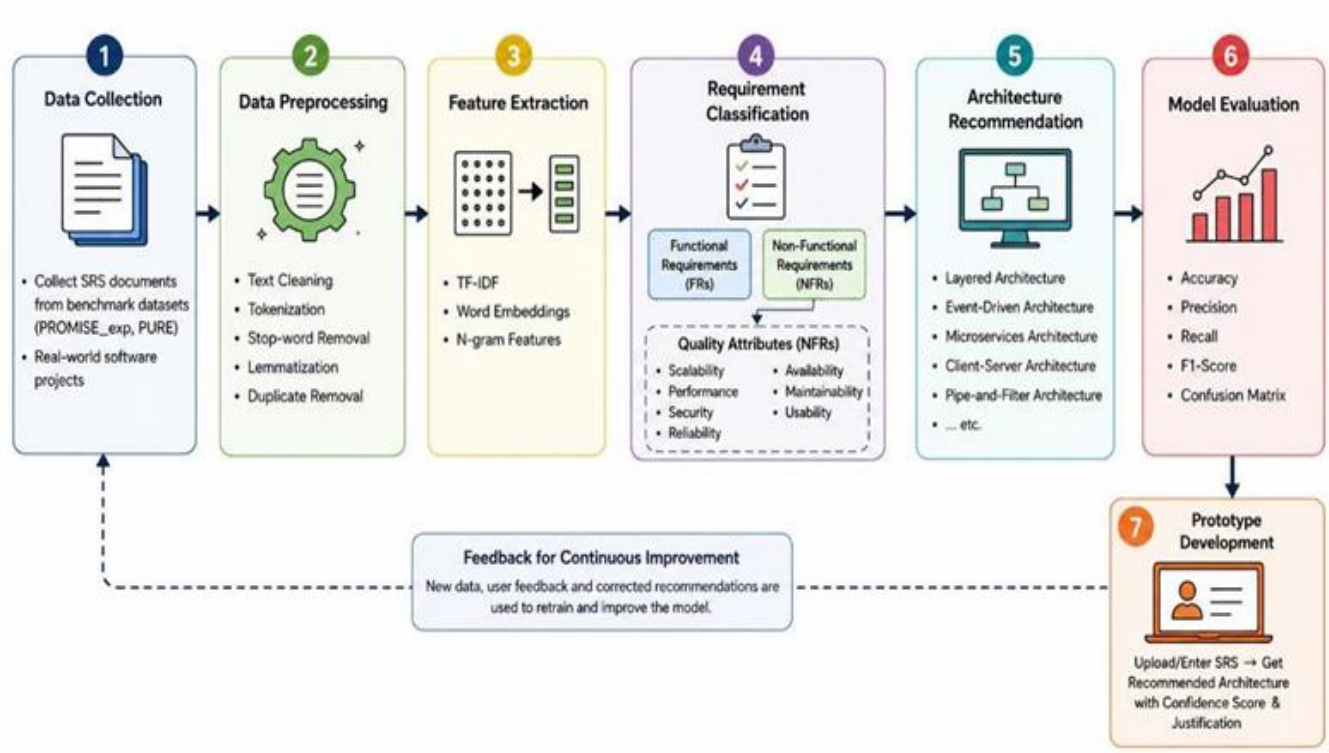


Figure 2. Proposed Research Methodology Workflow

As shown in ***Figure 2,*** the proposed Machine Learning based Requirement-Aware Software Architecture Recommendation System includes workflow, in which every step is discussed. As shown in Figure 2, the proposed Machine learning based Requirement-Aware Software Architecture Recommendation System consists of the following steps of a workflow and each step is discussed. It starts by obtaining the Software Requirement Specification (SRS) Document of benchmark datasets and actual software projects. Next, the gathered data goes through Natural Language Processing (NLP) to enhance the data quality by removing noise (text cleaning), splitting into words (tokenization), discarding stop-words, and lemmatization. For this, the relevant features of the text, such as TF-IDF, word embedding's, or N-gram features, are extracted.

The features extracted are then divided into Functional requirements (FRs) and Non-Functional requirements (NFRs) and the features listed under NFRs are classified as quality attributes such as scalability, performance, security, reliability, availability, maintainability, and usability. By determining the software architecture pattern based on these classified requirements, the best suited software architecture with Machine Learning algorithms, such as Layered, Microservices, Event-Driven, Client–Server or Pipe-and-Filter architecture, can be recommended. The Accuracy, Precision, Recall, F1-Score and Confusion Matrix are used to measure the performance of the recommendation model. Finally, a prototype is created which enables to upload or enter software requirements and will provide the users the recommendations of the architecture with a confidence score and justification. Basing on this continuous feedback, the recommendation model would be further improved according to time.

3.1 Research Design

The proposed research is based on design and development method with the objective of creating an intelligent recommendation system that would automatically analyze Software Requirement Specification (SRS) and recommend the most appropriate software architecture. Natural Language

Processing techniques are used by the system to process both Functional Requirements (FRs) and Non-Functional Requirements (NFRs), and a supervised Machine Learning algorithm, such as Gini's G method uses this data to make predictions of what the most appropriate architectural style is.

The research consists of the following major

phases:

1. Data Collection
2. Data Preprocessing
3. Feature Extraction
4. Requirement Classification
5. Architecture Pattern Recommendation
6. Model Evaluation
7. **Prototype Development**

3.2 Data Collection

First stage is gathering Software Requirement Specification (SRS) documentation from public datasets and real world software projects. They will be benchmark datasets that have labeled software requirements to be used for Machine Learning research, like PROMISE_exp and PURE (Public Requirements Dataset). Other Software Requirement Specification documents from open-source repositories can also be added in order to add diversity to the dataset.

The gathered data can have FNRs as well as NFRs and belong to different software domains.

3.3 Data Preprocessing

The requirement documents are presented in human language; thus the documents should be preprocessed prior to applying the Machine Learning algorithm. The purpose of pre-processing is to extract the unnecessary data and discarded the data that is meaningful provided.

The preprocessing process comprises of:

- Text cleaning
- Tokenization
- Lowercase conversion
- Stop-word removal
- Punctuation removal
- Lemmatization

With the introduction of the same. With the elimination of repetitions.

This stage helps add quality to the textual data and helps to remove noise prior to feature labelling.

Table 2: *Overall Phases of the Proposed Methodology*

Phase	Description
Data Collection	Collect SRS documents from benchmark datasets (PROMISE_exp, PURE) and real-world software projects.
Data Preprocessing	Clean and normalize textual requirements using NLP techniques.
Feature Extraction	Extract meaningful features using TF-IDF and Word Embeddings.
Requirement Classification	Classify requirements into Functional Requirements (FRs) and Non-Functional Requirements (NFRs).
Architecture Recommendation	Recommend the most suitable architecture using Machine Learning algorithms.
Model Evaluation	Evaluate the recommendation model using standard performance metrics.
Prototype Development	Develop a prototype for real-time architecture recommendation.

Table Description

Table 2 summarizes the major phases involved in the proposed research methodology, from collecting software requirement documents to evaluating the Machine Learning model and developing the final prototype.

3.4 Feature Extraction

Following the pretreatment stage, significant numbers will be created by the NLP techniques from the textual requirements.

It is proposed that the following feature extraction methods are applied: TF-IDF (Term Frequency-Inverse Document Frequency)

- Word Embeddings
- N-gram Features

These techniques convert the textual requirements to feature vectors which can be fed into the Machine Learning algorithms, and retain semantic information.

Table 3: *Machine Learning Algorithms and Evaluation Metrics*

Component	Technique
Feature Extraction	TF-IDF, Word Embedding's
Classification Algorithms	Random Forest, Support Vector Machine (SVM), Decision Tree, Artificial Neural Network (ANN)

Table Description

Table 3 presents the Machine Learning techniques, validation strategy, and evaluation metrics used to develop and assess the proposed architecture recommendation system.

3.5 Requirement Classification

The desired features extracted are then classified as Functional Requirements (FRs) or Non-Functional Requirements (NFRs). Moreover, the Non-Functional Requirements could be classified under the head of Quality attributes as:

- Scalability
- Performance
- Security
- Reliability
- Availability
- Maintainability
- Usability

This classification step will help support the architecture selection recommendation system in understanding which software quality attributes are more important in architecture selection.

3.6 Architecture Pattern Recommendation

After the requirements have been classified, the found quality attributes are matched with corresponding software architecture styles. The quality attributes discovered after the requirement classification are matched to software architecture styles.

The proposed system suggests that one of the following architectural patterns are recommended:

- Layered Architecture
- Microservices Architecture
- Event-Driven Architecture
- Client-Server Architecture
- Pipe-and-Filter Architecture

A number of supervised Machine Learning algorithms will be trained and compared including:

- Random Forest
- Classifiers used in this paper include Support Vector Machine (SVM).
- Decision Tree
- Artificial neural Network (ANN)

The best-performing model will be chosen as the final recommendation engine in terms of data prediction accuracy and classification accuracy.

3.7 Model Training and Validation

The dataset will be split into 80% training 20% testing. In addition to this, k-fold cross-validation during training will enhance the generalization of the model and minimize overfitting.

The textual features and the architecture for each sentence will be used to train a Machine Learning model. Hyper parameter tuning: Tuning efforts will be made to improve the prediction results.

3.8 Performance Evaluation

The proposed recommendation system will be tested by using the standard performance measures of the Machine Learning.

The following metrics are used to evaluate:

- Accuracy
- Precision
- Recall
- F1-Score
- Confusion Matrix

These evaluation measures will be used in checking the accuracy of the software architecture recommended by the proposed system in relation to the software requirements.

3.9 Prototype Development

A prototype application would be created to show the implementation of the proposed framework. The prototype will have a straightforward graphical user interface, to which a user can upload Software Requirement Specification documents or type in software requirements.

System will automatically make pre-processing, classification of requirements, analysis of quality attributes and recommend suitable Software Architecture. Besides the recommendation, the system will also present a confidence interval and also provide why the chosen architecture is the most likely the one that meets the identified requirements.

3.10 Proposed Research Workflow

The overall flow of the research that will be done can be summarized as follows:

This is because Software Requirement Specification Documents yield to Data Preprocessing, which leads to Feature Extraction, Requirement Classification, Identification of Quality Attributes, Training of the Machine Learning Model, Recommendation of Architecture Pattern, Performance Evaluation, and Prototype Development.

The proposed approach is a combined Natural Language Processing and Machine Learning approach that can give an intelligent, consistent and explainable software architecture suggestion for software are constructed in the early period of software development.

4. Experimental Results and Discussion

This chapter describes the experimental testing of the presented RAWSAR system with the help of Machine Learning. The goal of the experiments was to determine the effectiveness of various machine learning algorithms for classification of software requirements in their respective categories. The performance of the classifiers Decision Tree, Random forest, and Support Vector Machine (SVM) was tested and compared on a refined Software Requirements Dataset which passed through the evaluation process. All the standard evaluation metric, such as accuracy, precision, recall, f1 Score, and confusion matrix analysis in order to measure performance. These experiments results were analyzed to find the most appropriate classification model for being embedded in proposed architecture recommendation framework.

4.1 Experimental Results

4.1.1 Model Performance Comparison

Table 4 shows the accuracy of the classification from each machine learning model that was evaluated. The baseline models include here as well as the optimized Support Vector Machine employed in the proposed framework.

Table 4: *Classification Accuracy of Evaluated Machine Learning Models*

Machine Learning Model	Accuracy (%)
Decision Tree	62.00
Random Forest	68.80
Initial Support Vector Machine (SVM)	78.00
Optimized Support Vector Machine (Proposed)	**84.61**

The lowest classification accuracy was obtained by the Decision Tree classifier as given in Table 2 with an accuracy rate of 62.00%. The relatively weak performance could be explained by the fact that decision trees over fitted with high dimensional sparse textual data produced by TF-IDF feature extraction process. While simple and

easy to compute, Decision Trees are used less successfully with complex relationships between software requirement statements.

Multiple decision trees through ensemble learning were used in Random Forest to achieve the classification accuracy of 68.80%. Inclusion of multiple trees led to fewer overfitting issues and increased model generalization, but the result trees overall performed below Support Vector Machine as tree based models tend to be less effective on sparse text representation.

The results showed that initial Support Vector Machine performed much better than both tree-based models; it achieved an accuracy of 78.00% to show its appropriateness for text classification applications with a high-dimensional feature vector based on TF-IDF features. The hyper parameter optimization, improved TF-IDF feature extraction, dataset refinement, and balanced class weights were implemented on the Support Vector Machine model which achieved the highest classification accuracy of 84.61% after optimization. The results showed that optimized model has achieved more than 6.61 percentage points improvement in the initial SVM model and the proposed optimization strategy proved to be efficient.

4.1.2 Confusion Matrix Analysis

A confusion matrix was also used to further analyze the classification performance of optimized Support Vector Machine.

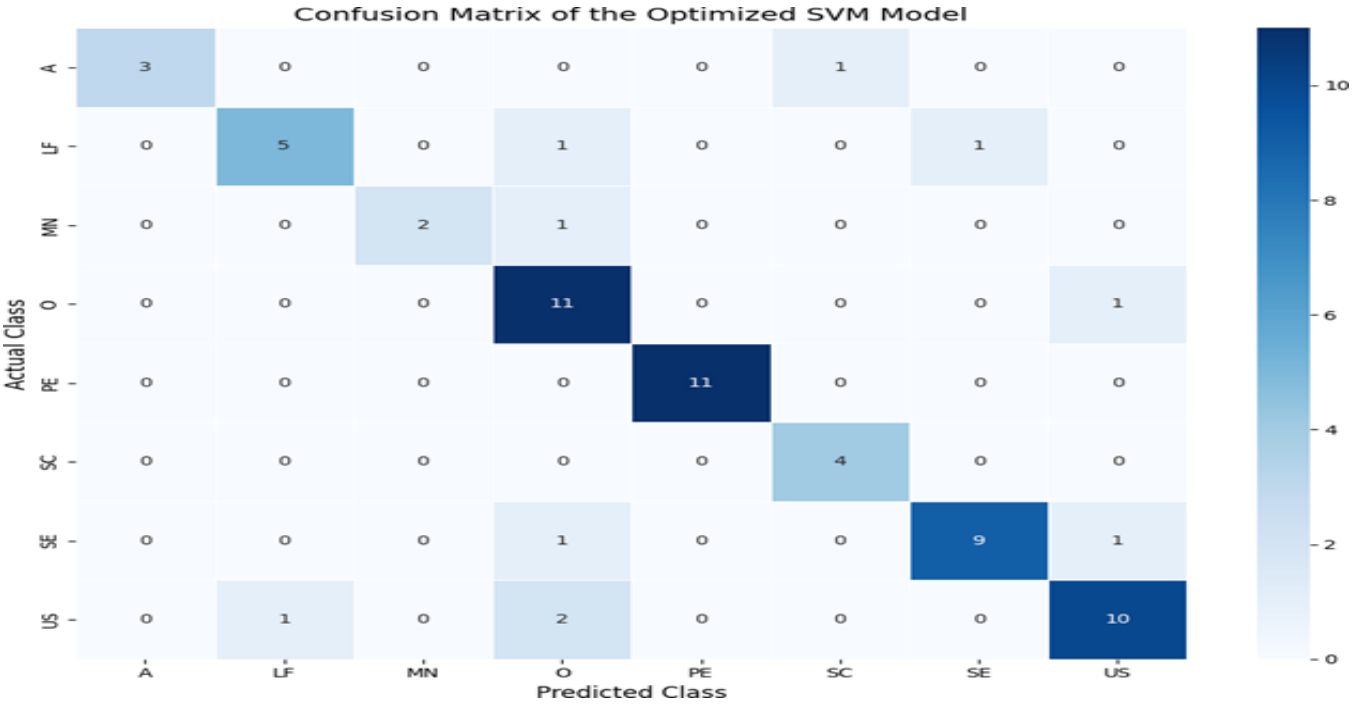


Figure 3. Here is the confusion matrix of Optimized Support Vector Machine (SVM). The performing of the software requirements categories in the correct and incorrect classification is shown in figure 3. The majority has been captured along the principal diagonal where most of its observations occurred, indicative of the fact that most software requirement statements were classified correctly. The optimized classifier demonstrated very good results for the categories: Security (SE), Usability (US), and Operational (O) that have relatively different vocabulary. A few misclassifications were seen between semantically related classes like Performance (PE) and Scalability (SC) where there is often a similarity in the terminology associated with the software requirements in terms of workload, throughput and response times. This analysis of the confusion matrix, on an overall basis, shows the strength and dependability of the optimized Support Vector Machine in Software Requirement Classification.

4.1.3 Requirement Category Distribution

The distribution of software requirement categories calculated for model training and evaluation is presented in Figure 4.

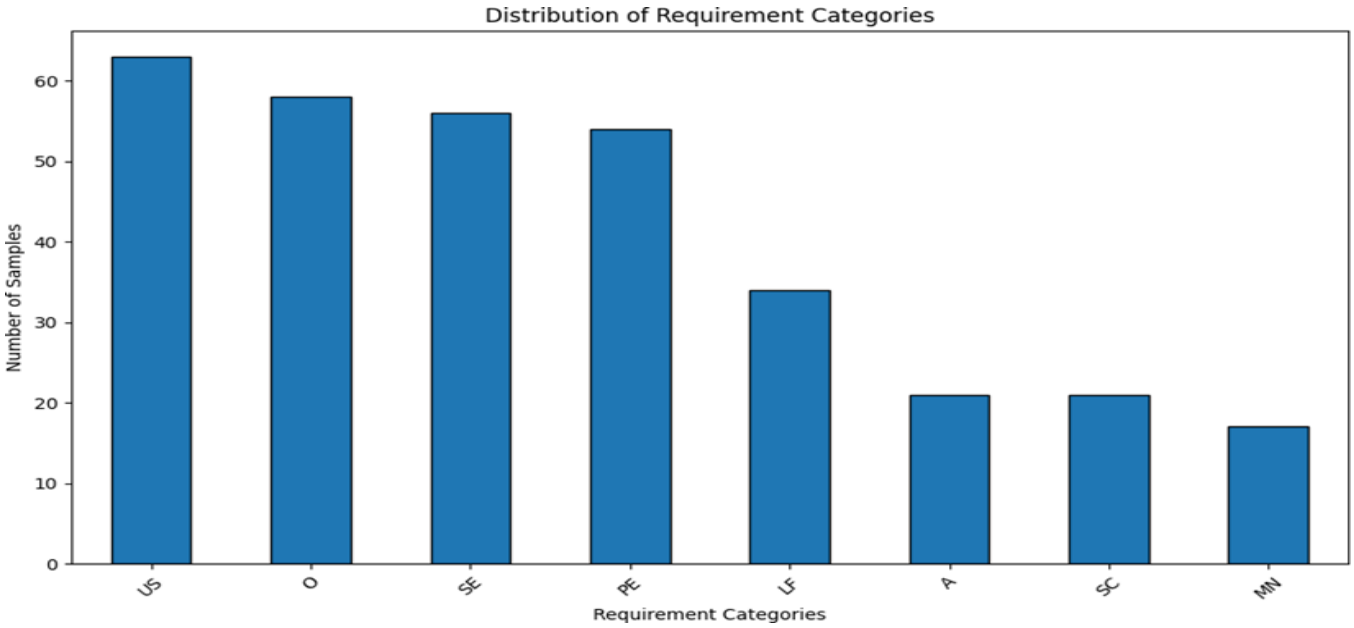


Figure 4. Of these software requirements, 65% changed hands between groups of students. All of these software requirements were distributed among groups of students, and 65% were changed hands.

The classification of the refined software requirements is shown in figure 4, excluding ambiguous parent categories which were removed and the severely underrepresented classes. The quality of the refined data set allows for a more equalized class distribution and is more effective for supervised learning algorithms as well as decreasing classification bias. Moreover, stratified train-test splitting maintained the proportion of each category through train and test. What's more, stratified train test maintained the proportion of each category through train testing, which led to a better reliability experimental evaluation.

4.2 Discussion

The experimental findings prove that the software requirements classification results achieved by the proposed optimization method are significantly better. The Support Vector Machine that is optimized is optimized by training with the best result of all machine learning algorithms evaluated resulting in a classification accuracy of 84.61 % compared to the initial Support Vector Machine that had accuracy of 60.39 % and Decision Tree, Random Forrest, which both had accuracy not exceeding 71 %.

There are various important factors resulting in superior performance of the optimized SVM. Based on the standard software requirements, all the text contains noisy information which needs to be removed, and standardizing the software requirement statements is important before extracting features. To perform this, comprehensive preprocessing was carried out in the text domain prior to feature extraction. The optimized TF-IDF configuration created more informative feature vectors as it used two features: the unigram and bigram. Adding, also, the exclusion of ambiguous parent classes (F, FR, NFR) removed overlapping definitions among classes and the exclusion of severely underrepresented classes removed the class imbalance. Additionally, for the classification task, hyper parameter tuning was performed for the classifier, which further improved classifier performance in distinguishing semantically similar requirement categories.

The results prove that the proposed requirement classification model is reliable and accurate in determining the software requirement classification categories. Based on these reasons, the optimised Support Vector Machine (SVM) was chosen as the end classification model used in

the software architecture recommender process proposed in the framework.

4.3 Summary

The experimental evaluation of the proposed Requirement-Aware Software Architecture Recommendation System has been discussed in this chapter. After doing a comparative study, maximum classification accuracy is obtained by optimized Support Vector Machine approach with 84.61% than any of machine learning models tested. The confusion matrix and comparative performance analysis of the classification model also confirmed the robustness and effectiveness of the proposed classification model. The results obtained, analysis performed and optimization concluded in the current stage were used to decide the optimized Support Vector Machine as the final classifier which will be used in the further stage of software architecture recommendation for proposed system.

5. Conclusion

The quality, scalability, maintainability and outcome of software systems depend largely on the software architecture. Choosing the right architectural pattern is a difficult process in early software development since the software requirements have to be correctly understood, interpreted and thus selected. The ways architecture is traditionally selected are generally based on the experience and knowledge of software architects, which still have their advantages but they are mostly subjective, time-consuming, and prone to inconsistencies. To overcome these challenges, the researchers in this study presented a method called Requirement-Aware Software Architecture Recommendation System Using Machine Learning, whose goal is to create an intelligent software architecture recommendation system by automating software requirement classification.

The used dataset for this proposed study was the Software Requirements Dataset from Kaggle, which consists of software requirement statements with various classes of functional/non-functional requirements. The dataset was heavily preprocessed before training the models with techniques such as text normalization, lowercasing, punctuation, number, URL, stop-word removal and lemmatization. An optimized TF-IDF representation was applied to the SWRSs, including both unigrams and bigrams, to capture the semantic features of the SWRSs and perform feature extraction.

Ambiguous training categories, (F, FR, and NFR), were eliminated as some overlapped with the more specific requirement categories, to improve the quality of the training data set. In addition, the classes were “filtered” to remove severely underrepresented ones to achieve class balance and enhance generalization capabilities.

The three machine learning algorithms were implemented and evaluated which are Decision Tree, Random Forest and Support Vector Machine (SVM). Comparative experimental analysis showed that performance of Decision Tree was 62.00% and using ensemble learning, the performance of Random Forest was increased as 68.80%. For the first Support Vector Machine, the accuracy of classification was reflected as **78.00**%, which was good suitable to the high-dimensional textual data. After optimizing the hyper parameters, refining the datasets, introducing weights to make them balanced, and getting better feature extraction using TF-IDF, the optimized Support Vector Machine obtained the highest accuracy in the classification of the models that were tested at 84.61%.

Finally, confusion matrix analysis showed that the optimized classifier was able to separate most of the software requirement classes with a very marginal confusion between requirements of similar semantic classes (e.g., Performance vs Scalability). The results demonstrate that the achieved software requirement classification performance using the proposed preprocessing and machine learning optimization techniques was significantly enhanced.

Overall, the results of this research prove the potential of applying machine learning method to accurately classify software requirements automatically and thus minimize the manual work in the software architecture design phase. This optimized Support Vector Machine establishes a sound basis for the intelligent software architecture recommendation system, and helps to make better architectural decisions in the initial stages of software development. Based on this, the following goals of this research have been accomplished by building and testing a machine

learning–based approach that sets the requirement classification framework to assist in architecture recommendation.

6. Future Work

The recommended system could attain encouraging classification results, but there are a few opportunities to improve and extend this research.

First, the future studies could benefit in using a model from the Deep Learning based Natural Language Processing (NLP) community such as BERT, RoBERTa, DistilBERT, and Sentence Transformers to better understand the relationships of software requirements at the semantic context level that may not be adequately represented in traditional TF-IDF representations. At first, future studies could leverage models from the Deep Learning based Natural Language Processing (NLP) community, such as BERT, RoBERTa, DistilBERT, or Sentence Transformers, to capture the semantic context relationships between software requirements more effectively than traditional TF-IDF representations. The future will see potential enhancements of such transformer models in terms of improving the accuracy of the classification, especially for difficult and uncertain requirement statements.

Secondly, the current project mainly constructs the software classification of requirement on software architecture recommendation as the basis. Future works can be able to create the full-fledged automated architecture recommendation engine which maps classified functional and non-functional requirements with software architectural models and patterns including Layered Architecture, Microservices Architecture, Event-Driven Architecture, Client-Server Architecture, Service-Oriented Architecture (SOA), and Pipe-and-Filter Architecture. A recommendation engine like that will be able to give direct architectural advice to software engineers when designing systems.

Third, future works may combine the use of Large Language Models (LLMs), e.g., the GPT based ones, to conduct semantic reasoning over software requirements, and to produce architecture suggestions enriched with explanations and architectural argumentation. This would make the proposed system more meaningful and applicable.

A further potential improvement concerns increasing the training set, using industrial software projects' software requirement specifications and public software repositories. The scalability of machine learning across software domains can be enhanced with more data, it is a more extended data set that would grow the robustness and generalization of the machine learning models.

Or, as a future research item, an application's software requirements can be related to several categories of non-functional requirements, such as in the case of a multi-label requirement classification. It would more closely mirror how it works in real-world software projects where it's common for each requirement to meet several quality attributes.

Further, other machine learning algorithms and ensemble methods such as XGBoost, LightGBM, CatBoost, deep neural networks and their ability to outperform SVM needs to be explored in the future.

Lastly, the proposed framework can be incrementally enhanced to create a fully matured Web-based intelligent decision support system, where Software Requirement Specification (SRS) documents are uploaded to this system, software requirements are automatically classified, software architecture patterns are recommended for the uploaded SRS, software architecture decisions are visualized and comprehensive architecture report documents are created. A system like this could have a significant effect in software design allowing software architects to make quicker, better and more evidence-based architectural decisions in software development.

Finally, this work provides a solid basis for the application of intelligent software architecture recommendation based on machine learning. The proposed framework for its application in real-world software engineering environments could be further optimized in future research with more advanced Natural Language Processing methods, the larger number of the industrial data sets, Explainable Artificial Intelligence, and automated architecture mapping.

Conflict of Interest

The authors showed no conflict of interest.

Funding

The authors did not mention any funding for this research.

References

Kurtanović, Z., & Maalej, W. (2017). Automatically Classifying Functional and Non-Functional Requirements Using Supervised Machine Learning. *IEEE 25th International Requirements Engineering Conference (RE)*, 490–495.

Cleland-Huang, J., Settimi, R., Zou, X., & Solc, P. (2006). The Detection and Classification of Non-Functional Requirements with Application to Early Aspects. *14th IEEE International Requirements Engineering Conference (RE'06)*, 39–48.

Lima, M., Valle, V., Costa, E., Lira, F., & Gadelha, B. (2019). Software Engineering Repositories: Expanding the PROMISE Database. *XXXIII Brazilian Symposium on Software Engineering (SBES)*, 427–436.

Rahman, M.A., Haque, M.A., Tawhid, M.N.A., & Siddik, M.S. (2019). Classifying Non-Functional Requirements Using RNN Variants for Quality Software Development. *3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)*, 25–30.

Maalej, W., Kurtanović, Z., Nabil, H., & Stanik, C. (2016). On the Automatic Classification of App Reviews. *Requirements Engineering*, *21*(3), 311–331.

Komolov, S., Dlamini, G., Megha, S., & Mazzara, M. (2022). Towards Predicting Architectural Design Patterns: A Machine Learning Approach. *Computers*, *11*(10), 151.

Klein, M.H., Kazman, R., Bass, L., Carriere, J., Barbacci, M., & Lipson, H. (1999). Attribute-Based Architecture Styles. In *Software Architecture*, 225–243. Springer.

Abowd, G., Bass, L., Clements, P., Kazman, R., Northrop, L., et al. (1997). Recommended Best Industrial Practice for Software Architecture Evaluation. *CMU/SEI-96-TR-025*, Carnegie Mellon University.

Ahmed, M., Khan, S.U.R., & Alam, K.A. (2021). QAExtractor: A Quality Attributes Extraction Framework in Agile-Based Software Development. *1st International Workshop on Intelligent Software Automation*, 15–28.

De Lauretis, L. (2019). From Monolithic Architecture to Microservices Architecture. *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 93–96.

Blinowski, G., Ojdowska, A., & Przybyłek, A. (2022). Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation. *IEEE Access*, 10, 20357–20374.

Baylor, D., Breck, E., Cheng, H.-T., Fiedel, N., Foo, C.Y., Haque, Z., et al. (2017). TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1387–1395.

Hermann, J., Del Balso, M., Kjelstrøm, K., Reinhold, E., Beinstein, A., & Sumers, T. (2018). Scaling Machine Learning at Uber with Michelangelo. *Uber Engineering*.

Amatriain, X., & Basilico, J. (2015). Recommender Systems in Industry: A Netflix Case Study. In *Recommender Systems Handbook*, 385–419. Springer.

Singh, K. (2024). From Development to Deployment: Streamlining MLOps with Monoliths, Microservices, and Amazon SageMaker. *International Research Journal of Engineering and Technology*, *11*(9), 16.

Shabani, I. (2022). Design of Modern Distributed Systems Based on Microservices Architecture. *International Journal of Advanced Computer Science and Applications*.

Palamarchuk, Y.A. (2022). Methods of Building Microservice Architecture of e-Learning Systems.

Oye, E., Frank, E., & Owen, J. (2024). Microservices Architecture for Large-Scale AI Applications.

Eisenreich, T., Speth, S., & Wagner, S. (2024). From Requirements to Architecture: An AI-Based Journey to Semi-Automatically Generate Software Architectures. *arXiv:2401.14079*.

Bucaioni, A., Weyssow, M., He, J., Lyu, Y., & Lo, D. (2025). Artificial Intelligence Support for Software Architecture Practice: A Systematic Review and Future Directions. *ACM Transactions on Software Engineering and Methodology / arXiv:2504.04334*.

Esposito, M., Li, X., Moreschini, S., Ahmad, N., Cerny, T., Vaidhyanathan, K., Lenarduzzi, V., & Taibi, D. (2025). Generative AI for Software Architecture: Applications, Trends, Challenges, and Future Directions. *arXiv:2503.13310*.

Dhar, R., Vaidhyanathan, K., & Varma, V. (2024). Can LLMs Generate Architectural Design Decisions? An Exploratory Empirical Study. *arXiv:2403.01709*.

Ferrari, A., Spagnolo, G.O., & Gnesi, S. (2017). PURE: A Dataset of Public Requirements Documents. *25th IEEE International Requirements Engineering Conference (RE)*, 502–505.

Garlan, D., Bass, L., Stafford, J., Nord, R., Ivers, J., & Little, R. (2003). Documenting Software Architectures: Views and Beyond. *Proceedings of the 25th International Conference on Software Engineering*.

Bosch, J., & Molin, P. (1999). Software Architecture Design: Evaluation and Transformation. *IEEE Conference and Workshop on Engineering of Computer-Based Systems*.

Kruchten, P.B. (1995). The 4+1 View Model of Architecture. *IEEE Software*, *12*(6), 42–50.

Bass, L., Clements, P., & Kazman, R. (2021). *Software Architecture in Practice* (4th ed.). Addison-Wesley Professional.

Slankas, J., & Williams, L. (2013). Automated Extraction of Non-Functional Requirements in Available Documentation. *1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE)*, 9–16.

Li, G., Zheng, C., Li, M., & Wang, H. (2022). Automatic Requirements Classification Based on Graph Attention Network. *IEEE Access*, 10, 30080–30090.

Dalpiaz, F., Dell'Anna, D., Aydemir, F.B., & Çevikol, S. (2019). Requirements Classification with Interpretable Machine Learning and Dependency Parsing. *IEEE 27th International Requirements Engineering Conference (RE)*.

Mumtaz, G., Akram, S., Iqbal, M.W., Ashraf, M.U., Almarhabi, K.A., Alghamdi, A.M., & Bahaddad, A.A. (2023). Classification and Prediction of Significant Cyber Incidents (SCI) Using Data Mining and Machine Learning (DM-ML).

Jabbar, U., Iqbal, M.W., Alourani, A., Shinan, K., Alanazi, F., Sarwar, N., & Ashraf, M.U. (2025). Machine Learning-Based Approach for Early Screening of Autism Spectrum Disorders. *Applied Computational Intelligence and Soft Computing*, 2025(1).

Iqbal, M.W., Shinan, K., Rafique, S., Alourani, A., Ashraf, M.U., & Rahim, N.Z.A. (2024). Usability and Optimization of Online Apps in User's Context. *PeerJ Computer Science*, 10, e2561.

Naqvi, S.R., Shahzad, S.K., Iqbal, M.W., & Al-Thawadi, M. (2023). Ontology-Driven Smart Health Insurance System. *Advances in Intelligent Systems and Computing (Conference Paper)*.

Iqbal, M.W., Nazir, Z., & Fuzail, Z. Usability Empowered by User's Adaptive Features in Smart Phones: The RSM Approach.