SvedbergOpen
DISSEMINATION OF KNOWLEDGE
International Journal of Artificial Intelligence and Machine Learning
Publisher's Home Page: https://www.svedbergopen.com/
Research Paper Open Access

Hierarchical Software Requirement Classification Using Transformer-Based Language Models

R Nagaraju[1] , V Anantha Natarajan[2]*

[1]Research Scholar, Department of Artificial Intelligence and Machine Learning, School of Computing, Mohan Babu University.
nagraj.rayapati@gmail.com
[2]Professor, Department of Artificial Intelligence and Machine Learning, School of Computing, Mohan Babu University.
v.ananth.satyam@gmail.com

ABSTRACT
The automated classification of software requirements into Functional Requirements (FRs) and Non-Functional Requirements (NFRs) has become a significant research direction in Requirements Engineering due to the growing size and complexity of modern software specifications. Manual requirement classification is often inconsistent, subjective and laborious. This emphasizes the need for scalable automated approaches that improve the reliability of early-stage system analysis. In this paper, we present an in-depth analysis of the PROMISE_EXP dataset, a systematically collected corpus comprising 969 requirement statements, including 444 FRs and 525 NFRs, spanning twelve granular NFR categories, such as Security, Usability, Performance, Availability, Maintainability, Scalability, and Portability. Besides a detailed statistical characterization of the dataset and a report of the class imbalance characterizing real-world requirement distributions, this work also explores a two-level hierarchical classification strategy. In the first level, requirements are categorized into a binary model (FR vs. NFR). In the second level, all NFRs are further classified into twelve multi-class labels. We implement and benchmark a comprehensive suite of models to evaluate the performance of automated classification at both levels. These include traditional machine learning algorithms (SVM, Random Forest, and Logistic Regression), deep learning architectures (CNNs, LSTMs) and transformer-based language models, specifically BERT and its domain-adapted variants. Experimental results show that transformer-based models consistently outperform ML and DL counterparts in both binary and multi-class settings particularly in dealing with semantically complex NFR subcategories. The results demonstrate that domain-aware language representations significantly improve accuracy, robustness and generalization, especially for low-frequency NFR labels. This study provides a robust two-level classification framework and a comprehensive performance evaluation of ML, DL and BERT based classifiers. It further provides practical insights for advancing automated requirements engineering. The results support the development of intelligent, data-driven tools for software engineering automation, enable a deeper understanding of the variability in requirements semantics, and promote reproducibility.

Keywords: Requirements Engineering, Functional and Non-Functional Classification, Machine Learning, Deep Learning, BERT, Software Specification Analysis.

1. INTRODUCTION

The foundation, architecture and quality of today's software systems depend critically on Requirements Engineering (RE). Today, systems are growing in scale, complexity and connectivity; the amount of requirement specifications is increasing and the effective organization and interpretation of requirement specifications is becoming more and more difficult. One of the core tasks in RE is to distinguish Functional Requirements (FRs) which specify the expected behavior, capabilities and interactions of the system, from Non-Functional Requirements (NFRs) which specify quality attributes, operational constraints and performance expectations. This distinction is important because NFRs drive architectural decisions,

performance guarantees, usability levels, and a host of quality-driven trade-offs throughout the development lifecycle [1]. NFRs are diverse, abstract and context dependent, unlike FRs which are often more structured and action-oriented in linguistic form. This linguistic ambiguity makes NFRs inherently more difficult for practitioners to specify and classify consistently [2].

In convention requirements were classified manually by analysts, subject matter experts, and system architects. However, manual classification is prone to inconsistencies, subjective interpretations, domain-specific biases and human error, especially in large-scale projects with multiple stakeholders. Recent empirical studies show a high disagreement among experts for identifying NFR types, which reflects the difficulty and lack of uniformity of manual assessments [3]. Furthermore, manual inspection processes require significant cognitive load and time in the early phases of a project, which may hinder important architecture and design decisions. These limitations motivate the development of automated solutions with the potential to perform requirement classification at scale, with improved accuracy, speed and repeatability.

Advances in Natural Language Processing (NLP), Machine Learning (ML), Deep Learning (DL) and transformer-based language models have opened new doors to solve classification issues in Requirements Engineering. Earlier, ML based works used traditional classifiers like Support Vector Machines (SVM), Naïve Bayes, Random Forests, and Logistic Regression with promising but limited improvements in requirement categorisation [4]. They mostly use handcrafted features or domain specific linguistic patterns. The advent of DL architectures like Convolutional Neural Networks (CNNs) and Long Short-Term Memory networks (LSTMs) allowed the learning of complex textual representations and contextual semantics directly from data [5]. More recently, transformer models, such as BERT have revolutionised NLP tasks using self-attention mechanisms and deep contextual embeddings, resulting in significant performance gains for classification tasks on unstructured text, including software requirements [6].

The requirement classification problem has been tackled in different ways in previous studies, but many have only addressed binary classification (FR vs. NFR) or multi-class classification of NFRs. Only a few studies follow a holistic approach, mirroring the practical need of classification: a first coarse-grained distinction between FR and NFR, and a fine-grained classification of NFRs into specific quality attributes like Security, Usability, Performance, Maintainability, Portability, and others [7]. This two-level strategy is of high relevance for software engineering practice: whereas early binary separation assists architectural planning, fine-grained NFR categorisation provides the detailed insights needed for risk mitigation, compliance assessment and quality-driven design.

1.1 Background and Motivation

The modern software development requires that the types of requirements be identified early to guide the system design and to ensure that the system fits the project constraints. NFRs like performance thresholds, usability constraints or security mandates often have architectural implications that impact downstream decisions around technology selection, resource allocation, and risk management. Repeatedly, misclassification or oversight of NFRs has been linked to system failures, poor user satisfaction, degraded performance, and budget overruns [8]. Especially in large organizations, multi-vendor environments and safety-critical domains, as software team scale and requirements become more heterogeneous, the need for automated, intelligent requirement classification becomes more apparent. Automated classification improves not only consistency but also reduces the load for analysts and allows continuous quality checks in agile and DevOps pipelines. New state-of-the-art NLP models open up a new opportunity to automate requirements classification with better accuracy and contextualization than classical rule-based or keyword-based approaches.

1.2 Problem Statement

Natural Language Processing techniques applied to Requirements Engineering have been significantly improved. However, some critical challenges still remain unsolved. The manual classification of software requirements is still inconsistent, highly labor intensive and often ambiguous, especially for the case of Non-Functional Requirements, which tend to be abstract, context-dependent and linguistically diverse. Such ambiguities lead to a wide range of human interpretation and decrease the reliability of manual assessments. Moreover, the intrinsic class imbalance among NFR categories and significant linguistic variability of quality

attributes significantly challenge traditional machine learning models, which often could not learn discriminative patterns for under-represented subclasses.

The existing literature (reviewed in next section) mainly deals with the binary classification of requirements into Functional and Non-Functional requirements or multi-class classification of NFRs into specific quality attributes. However, few studies adopt the hierarchical two-level approach that reflects the real-world workflow of requirement analysis: coarse-grained separation and fine-grained categorization. Moreover, the comparative benchmarking of machine learning, deep learning, and transformer-based language models on a common dataset remains limited, and the existing studies often lack the uniform experimental settings needed for fair and reproducible evaluation. Another important gap is the absence of empirical validation of modern contextual embedding models (e.g., BERT and its variants) for the classification of fine-grained NFR subcategories, which represent one of the hardest challenges in automated requirement analysis. Such gaps indicate the need for a systematic, empirically based study using a popular dataset to evaluate different classification methods for both the binary and the multi-class problems. This investigation is key to: advancing the state of automated requirements classification; increasing methodological rigor; enabling the development of more reliable and practically deployable tools for the software engineering community.

1.3 Objectives of the Study

The primary objective of this study is to advance automated requirements classification through a structured, two-level analytical framework. The study aims to provide a detailed statistical and linguistic characterization of the dataset in order to support a deeper understanding of its composition and inherent complexities. A hierarchical classification structure is implemented, beginning with the binary differentiation of Functional Requirements and Non-Functional Requirements, followed by the fine-grained categorization of NFRs into twelve quality-attribute classes. A comprehensive set of machine learning algorithms, deep learning architectures, and transformer-based language models is employed to assess classification effectiveness. The study further seeks to analyse how dataset imbalance, contextual variability, and requirement semantics influence model performance. Another objective is to generate empirical insights that inform the selection of suitable models for practical applications, while also contributing reproducible findings that can support future research in automated Requirements Engineering.

1.4 Scope of the Study

The scope of this study is limited to the analysis and classification of textual requirement statements. The investigation focuses on supervised learning methods and does not consider unsupervised, semi-supervised or reinforcement learning paradigms. The evaluation is limited to traditional machine learning classifiers, deep learning networks and transformer-based models. Rule-based, hybrid or ontology-driven approaches are out of study boundaries. The study focuses on the effectiveness of models for the binary (FR vs. NFR) and multi-class NFR classification tasks and does not consider other tasks like requirement extraction, clustering, traceability link recovery, prioritization, sentiment analysis or domain adaptation. Secondly, the study does not seek to generalize beyond the PROMISE_EXP dataset although produced insights may give guidance for similar datasets in software engineering contexts.

1.5 Organization of the Paper

The rest of this paper is organized to give a complete and sequential study of automated requirements classification. Section 2 surveys the current literature on Functional and Non-Functional Requirement classification, discussing the progress in machine learning, deep learning and transformer-based approaches. In Section 3 we introduce the PROMISE_EXP dataset and describe its structure, class taxonomy, distribution and linguistic characteristics in detail. Section 4 describes the methodological framework followed in this work, including the two-level classification design, feature extraction strategies, model configurations, training procedures and evaluation metrics. In Section 5, we present the experimental results with machine learning, deep learning and BERT-based classifiers for binary and multi-class classification tasks. Section 6 discusses the implications of the findings, the interpretation of model behavior, strengths and limitations and potential applications in requirements engineering practice. Section 7 concludes the study by highlighting the major findings and proposing some promising directions for future research in the area of automated requirement classification and intelligent software engineering systems.

2. LITERATURE SURVEY

The recent research on automated requirement classification can be categorized into several complementary streams: transformer based models and BERT variants; deep learning (CNN/RNN/hybrid) pipelines; classical ML with preprocessing/augmentation; datasets, curation, and benchmarking efforts; prompt/few-shot / LLM methods; and explainability / domain-specific applications (security, aerospace, healthcare). This section summarizes key advances and highlights recurring gaps (class imbalance, label noise, cross-dataset generalization) that motivate the present investigation.

2.1 Transformer-based and Pre-trained Language Model Approaches

Both the FR/NFR binary classification and the fine-grained NFR sub-classing tasks achieve state-of-the-art performance by Transformer and Bert-based models. A number of studies have investigated fine-tuning BERT, domain adaptation (domain-specific BERT) and hybrid architectures where BERT embeddings are concatenated with CNN/LSTM heads or light classifiers for computational efficiency or better local feature capture. Empirical results across multiple studies show large gains from contextualized embeddings over bag-of-words baselines, and many researchers point out the particular benefit of transformers for low-frequency NFR labels when combined with data augmentation or class-aware training [9]–[13], [15]. BERT fine-tuning and BERT-hybrid models are shown to outperform classical baselines in binary and multi-class tasks [9-13]. Domain adaptation helps in industry specific requirement corpora [14] with specialized BERT models. Semi-supervised adaptations (e.g., GAN-BERT) and few-shot/prompting strategies are used [11], [15] to deal with the data scarcity of rare NFRs. Recent developments also include transformer-based frameworks for requirements classification, e.g., general BERT-based text classifiers evaluated on document-level [30], which show good generalization even outside RE-specific corpora. A transformer-based framework is proposed in [36] especially designed for software requirement classification, where it is confirmed that attention-based architecture outperforms CNN/LSTM models for structured requirement corpora. The work presented in [38] provides a benchmark comparison of LLMs for NFR classification, revealing substantial variation in zero-shot and few-shot performance across LLM families and emphasizing the significance of domain-adapted prompting strategies.

Table 2.1 Transformer & Pre-trained LM studies

Methods	Dataset(s)	Remarks
MNoR-BERT; BERT-BiCNN ([9])	PROMISE variants, app-review corpora	BERT + convolutional head for better local feature capture; strong multi-label NFR detection.
BERT fine-tuning variants; ablations ([12])	PROMISE_exp, FR_NFR	Empirical analysis of model components and dataset size impacts on performance.
GAN-BERT semi-supervised fine-tuning ([11])	Newly introduced requirements datasets	Improves performance in low-label regimes.
BERT fine-tuning + explainability ([13])	Security-focused corpora	Few-shot emphasis for security NFR detection.
Domain-adapted BERT; comparison vs Bi-LSTM ([14])	Aerospace requirements corpora	Domain pretraining yields clear gains in domain-specific NFR detection.

2.2 Deep Learning: CNN / RNN / Hybrid Architectures

Efficient alternatives to full transformers are convolutional and recurrent architectures (CNNs, LSTMs, Bi-LSTMs) that are often combined with attention or stacked with embedding layers and are often used as baselines. Hybrid approaches (e.g., CNN/LSTM layers on top of embeddings, or BERT + CNN heads) work well for short, syntactically varied requirement sentences and when compute or latency constraints are important [10], [14], [18]. DL models are competitive if they are fed with good embeddings (GloVe, Word2Vec, contextual embeddings). Requirement text specific architectures (DReqANN, DReqBiLSTM) improve multi-class NFR performance without heavy feature engineering [10].

Table 2.2 Deep learning (CNN/RNN/hybrid) studies

Study (short)	Methods	Dataset(s)	Remarks
DReqANN / DReqBiLSTM	Custom ANN & BiLSTM architectures for NFRs	PROMISE_exp variants	Eliminates handcrafted features; strong performance for multi-class

	([10])		NFRs.
BERT-CNN hybrid	BERT + CNN head (BERT-BiCNN / BERT-CNN) ([9],[13])	PROMISE_exp	Improves detection on short requirement sentences.
CNN / LSTM baselines	CNN, LSTM, Bi-LSTM baselines with embeddings ([15],[20])	Various NFR corpora	Useful cost/performance tradeoffs relative to transformers.
Hybrid SVM + Bi-LSTM	Feature-based ML + sequence model ([18])	PROMISE_exp	Attempts to balance interpretability and sequence modeling.
FiBER transformer frameworks	Transformer variants tailored for requirements ([1])	PROMISE_exp and extensions	Reports high accuracy (see summarized results across 2023–2025).

2.3 Traditional Machine Learning, Preprocessing and Imbalance Handling

Classical ML methods (SVM, Random Forest, Logistic Regression, Naïve Bayes) still provide relevant baselines, especially when combined with careful text preprocessing, vectorization (TF-IDF, hashing) and imbalance-aware techniques (SMOTE, SMOTE-Tomek). A few comparative studies show that classical methods coupled with good preprocessing and feature selection are still competitive for binary FR/NFR tasks and serve as a useful sanity check for DL/transformer gains [21], [23], [24]. Preprocessing (normalization, POS features, domain lexicons) and sampling/augmentation strategies can significantly improve classification of low frequency NFRs [23]. Comparative studies indicate to cases where classical ML is sufficient (small datasets, resource constraints) [27].

2.4 Datasets, Benchmarking and Data Quality

Dataset creation, relabeling, augmentation and benchmarking has been a hot topic with a number of contributions extending or re-labeling PROMISE family datasets (PROMISE_exp, Promise+, NICE) [31] and producing multilingual or domain specific corpora. More and more emphasis is put on careful curation steps (de-duplication, relabeling, multi-labeling) and cross-dataset evaluations to ensure fair comparisons and to assess the ability to generalize [16,17,18]. PROMISE_exp is still the de-facto benchmark, however, several extensions (Promise+, NICE, dataset augmentations) have been released to increase coverage and reduce labeling noise [16], [17], [18]. Cross-dataset evaluations are needed to evaluate generalization beyond the original PROMISE domain [16], [19].

2.5 Prompting, Few-shot / Zero-shot, Semi-supervised and LLM Approaches

Requirements classification has investigated few-shot learning, prompting, and LLMs (large language models), especially considering the issues of lack of labels and rapid domain adaptation. Prompt tuning and lightweight prompt-based methods are often effective in reducing the labeling effort and allowing for reasonable performance with limited amount of supervised data; semi-supervised variants such as GAN-BERT leverage unlabelled data for better fine-tuning [11], [15], [20]. First systematic comparisons of prompt-tuning and full fine-tuning are emerging [24]. Prompting and few-shot learning require less supervision and work well for rare NFR classes if prompts are carefully designed [15], [24]. Semi-supervised GAN-BERT and its variants can be helpful to exploit unlabelled requirement corpora [11].

2.6Explainability, Security and Application-oriented Studies

Recent literature focuses on explainability and domain targeted pipelines (security, aerospace, healthcare) to increase practitioner trust and improve domain applicability. The research community has proposed explainable transformer variants and hybrid pipelines to identify security requirements, and employed domain-adapted pre-training (e.g., aeroBERT) for improving specialized NFR detection [12], [14], [19], [28]. Explainability (attention visualizations, local explanations) helps acceptance in safety-critical domains [12], [28]. Domain-specific pre-training and curated datasets provide tangible gains for specialized NFR categories [14], [19]. The transformer-based approaches (BERT and derivatives) are the most consistent in terms of gains across the reviewed works for binary and multi-class NFR tasks. Hybrid and classic ML approaches are still valuable under compute or data constraints. Dataset curation (PROMISE variants, Promise+, NICE, FAIR translations) is an ongoing area, with researchers focusing on de-duplication, multi-labeling, and cross-

dataset evaluation for better reproducibility. Prompting and semi-supervised techniques help with label scarcity problem for low frequency NFR classes Explainability is emerging as a central for adoption in sensitive domains. Some open issues still need continued attention such as: robust handling of label noise and duplicates; standard cross-dataset benchmarks for fair comparison; systematic approaches to class imbalance other than simple resampling; and systematic, reproducible evaluations of LLM few-shot/zero-shot performance on standardized NFR tasks.

3. DATASET DESCRIPTION

The PROMISE_EXP dataset is a domain-specific dataset of software requirement statements that has been created to support research on automated requirements classification. The distinction between Functional Requirements (FRs) and Non-Functional Requirements (NFRs) plays a crucial role in system design, architectural decisions, prioritization and verification. Requirements Engineering (RE) is a core activity in the software development life cycle. Manual classification, which is traditional, is time-consuming, error-prone and inconsistent from analyst to analyst. PROMISE_EXP addresses this issue by offering a structured labeled dataset to enable empirical evaluation of machine learning, deep learning and natural language processing (NLP) techniques for requirement classification. The dataset contains 969 requirement statements, which were manually annotated according to widely accepted software quality models and domain standards. It employs a complete label set for both functional and quality attributes: ['PE', 'LF', 'US', 'A', 'SE', 'F', 'FT', 'SC', 'PO', 'O', 'L', 'MN'].
Out of these F stands for Functional Requirements, the other tags represent different types of Non-Functional Requirements. The dataset includes 444 Functional Requirements and 525 Non-Functional Requirements, making it a balanced and challenging distribution for both binary and multi-class classification problems.

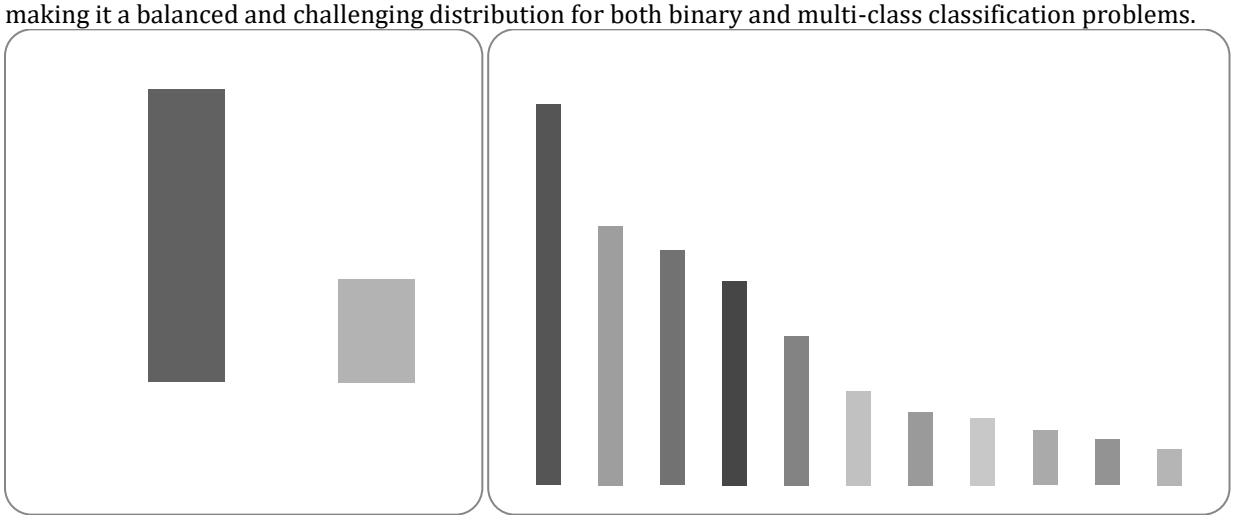

Figure 1 Label Distribution of Requirements in the Dataset.

The figure presents the distribution of requirement types, split into two panels. The left panel, Binary Label Distribution, shows the high class imbalance between Non-Functional Requirements (NFR) and Functional Requirements (FR) (Label 'F'). The right panel, Multi-Class Label Distribution, details the categories within the Non-Functional Requirements, revealing that requirements are primarily concentrated in PE (Performance), US (Usability), and SE (Security) categories.
The Functional Requirement (F) category, with 444 instances, captures system behaviors, operations, and capabilities expected from the software system. These statements generally describe what the system should do, including core features, interactions, computations, and workflow logic. Their volume within the dataset enables strong model training for FR detection while preserving complexity due to linguistic variability.
The Non-Functional Requirement classes represent diverse quality attributes:

- **Security (SE) – 125 samples:** Includes confidentiality, authorization, authentication, integrity, protection mechanisms, and threat mitigation.
- **Usability (US) – 85 samples:** Captures requirements related to user experience, learnability, accessibility, interface intuitiveness, and ergonomics.

- **Operational (O) – 77 samples:** Covers system operations, deployment constraints, environmental specifications, platform dependencies, and runtime conditions.
- **Performance (PE) – 67 samples:** Encompasses speed, latency, throughput, response time, resource limits, and system performance targets.
- **Look and Feel (LF) – 49 samples:** Describes aesthetic, UI style, branding, visual layout, and interface presentation qualities.
- **Availability (A) – 31 samples:** Addresses uptime guarantees, fault handling, redundancy, and system continuity metrics.
- **Maintainability (MN) – 24 samples:** Involves modularity, code clarity, testability, upgradability, and long-term maintainability aspects.
- **Scalability (SC) – 22 samples:** Focuses on system expansion, workload management, growing user demands, and distributed processing constraints.
- **Fault Tolerance (FT) – 18 samples:** Represents resilience to system failures, fallback mechanisms, and error recovery procedures.
- **Additional LF variant – 15 samples:** A supplementary set of samples categorized under Look and Feel due to dataset source variations.
- **Portability (PO) – 12 samples:** Includes requirements related to cross-platform compatibility, migration, hardware independence, and integration flexibility.

This distribution highlights realistic variability commonly observed in industrial requirements datasets. The presence of high-frequency classes such as SE and US alongside lower-frequency categories such as PO and FT introduces natural class imbalance—an aspect particularly valuable for evaluating model robustness, resampling strategies, and cost-sensitive learning methods. From a technical standpoint, PROMISE_EXP supports multiple research objectives, including binary classification (FR vs. NFR), multi-class classification, hierarchical classification of requirement types, and semantic analysis of requirement statements. Its linguistic diversity, varying requirement lengths, and the presence of domain-specific terminology make it an ideal dataset for experimenting with transformer-based language models, domain adaptation, and explainable AI techniques within Requirements Engineering. Due to its structured labeling scheme and detailed class taxonomy, PROMISE_EXP is widely applicable in evaluating text classifiers, embedding models, rule-based systems, prompt-engineered LLMs, and hybrid ML-NLP pipelines. The dataset also facilitates benchmarking of supervised, semi-supervised, and zero-shot classification approaches in RE automation research. Its real-world distribution, covering both common and rare requirement categories, provides a rich foundation for understanding how automated systems detect and interpret the nuanced differences between functional and non-functional requirement statements.

Figure 1 shows the distribution of requirement types. The split is two-panel. The left panel, Binary Label Distribution, shows the high class imbalance between Non-Functional Requirements (NFR) and Functional Requirements (FR) (Label “F”). In the right panel, Multi-Class Label Distribution, the categories within the Non-Functional Requirements are presented. The requirements are mainly concentrated in the PE (Performance), US (Usability) and SE (Security) categories. The Functional Requirement (F) category contains 444 instances and this category describes the behaviours, operations and capabilities expected from the software system. These statements are typically a description of what the system should do, including core features, interactions, computations and workflow logic. The FR detection can be trained strongly due to their volume in the dataset, while the complexity is preserved due to linguistic variability. The Non-Functional Requirement classes represent the following quality attributes:

- Security (SE) – 125 samples: Confidentiality, authorization, authentication, integrity, protection mechanisms, threat mitigation.
- Usability (US) – 85 samples: Covers requirements related to user experience, learnability, accessibility, interface intuitiveness and ergonomics.
- Operational (O) – 77 samples: Includes system operations, deployment constraints, environmental specifications, platform dependencies, and runtime conditions.
- Performance (PE) – 67 samples: Includes speed, latency, throughput, response time, resource limits, and system performance targets.
- Look and Feel (LF) – 49 samples: Deals with qualities of aesthetic, UI style, branding, visual layout and interface presentation.

- Availability (A) – 31 samples: Covers uptime guarantees, fault handling, redundancy and system continuity metrics.
- Maintainability (MN) – 24 samples: Considers modularity, readability, testability, upgradability and long run maintainability aspects.
- Scalability (SC) – 22 samples: It involves system expansion, workload management, increasing user demands, and distributed processing constraints.
- Fault Tolerance (FT) – 18 samples, indicating resistance to system failures, fallback methods, and error recovery procedures.
- Additional LF variant – 15 samples: An extra set of samples that are categorized as Look and Feel due to variations in the source of the dataset.
- Portability (PO) – 12 samples It has requirements for cross platform, migration, hardware independence and integration flexibility.

This distribution illustrates the realistic variability commonly seen in industrial requirements datasets. The existence of high-frequency classes (SE and US) and low-frequency classes (PO and FT) produces a natural class imbalance, which is especially useful for assessing model robustness, resampling, and cost-sensitive learning techniques. From a technical point of view, PROMISE_EXP is useful for various research purposes, including binary classification (FR vs. NFR), multi-class classification, hierarchical classification of requirement types, and semantic analysis of requirement statements. Its rich linguistic diversity, various lengths of requirement specifications, and domain-specific terminology make it an ideal dataset for experimentation with transformer-based language models, domain adaptation, and explainable AI techniques in the context of Requirements Engineering. PROMISE_EXP's structured labeling scheme and detailed class taxonomy allow it to be broadly used to evaluate text classifiers, embedding models, rule-based systems, prompt-engineered LLMs, and hybrid ML-NLP pipelines. The dataset also enables benchmarking of supervised, semi-supervised and zero-shot classification methods for RE automation research. The way it is distributed in the real-world, both common and rare types of requirements, provides a wealth of information for understanding how automated systems detect and make sense of the subtle distinctions between functional and non-functional requirement statements.

4. Methodology

The proposed methodology is composed of five main stages as presented in Figure 2: (1) data acquisition and pre-processing, (2) feature engineering and embedding generation, (3) hierarchical classification, (4) deep learning and training of transformer-based models, and (5) performance evaluation. The overall framework is designed to first perform coarse-grained requirement categories (FR vs NFR) distinction and then for the fine-grained NFR subtype classification. This hierarchical strategy increases the robustness of the classification, reduces the inter-class confusion and increases the interpretability of the model. The main goal of this study is to develop and test a hierarchical classification framework to correctly differentiate Functional Requirements (FR) from Non-Functional Requirements (NFR) and then classify NFRs into their corresponding subtypes. Automated classification of software requirements expressed in natural language is a challenging task because such requirements are often ambiguous, diverse in structure and highly domain-dependent. To tackle these challenges, we adopted a multi-stage approach using advanced text-processing techniques, classical machine learning models, deep learning architectures, and transformer-based language models.

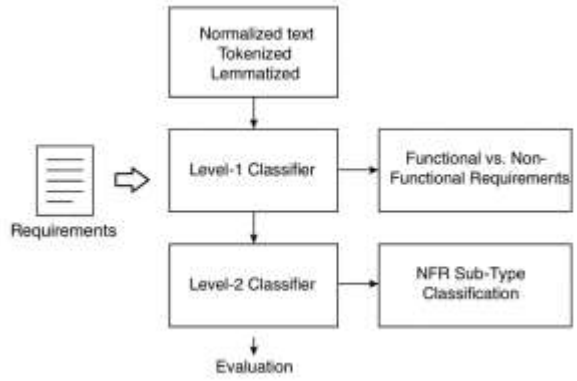

Figure 2 Block diagram illustrating the overall methodology adopted for hierarchical classification

4.1 Pre-processing

The study uses the PROMISE requirements dataset which consists of natural-language requirement statements annotated as one of the following classes: F (Functional Requirement) and A, L, LF, MN, O, PE, SC, SE, US, FT, PO (Non-Functional Requirement subcategories). The initial dataset in ARFF format was converted into CSV format for easy processing. The raw text was heavily preprocessed to perform case normalization, punctuation removal, and contraction expansion, stop word filtering and token based noise removal. Additionally, domain-specific normalization rules were used to normalize terminology that is commonly found in software engineering documents. We used multiple layers of feature engineering to improve the semantic representation of text:

(1) uni-, bi- and tri-grams with TF-IDF weighting to represent classical feature spaces, (2) pre-trained Word2Vec and GloVe models to generate word embeddings for deep learning architectures, and (3) BERT to generate contextual embeddings to capture deeper semantic and syntactic characteristics of each requirement. All experiments were conducted under the same preprocessing pipeline to ensure comparability across models.

Algorithm 1 — Text Pre-processing

Input: Raw text corpus D= $\{d_1, d_2, \dots, d_N\}$

Output: Cleaned token sequences $\mathcal{T} = \{T_1, T_2, \dots, T_N\}$

1. Begin
2. For each document $d_i \in D$:
3. /** Step 1: Text Normalization **/
4. Convert document to lowercase:
   $d_i \leftarrow Lowercase(d_i)$.
5. Remove punctuation, digits, and special characters using regex:
   $d_i \leftarrow RegexClean(d_i)$
6. Normalize whitespaces:
   $d_i \leftarrow WhitespaceNormalize(d_i)$
7. /** Step 2: Tokenization **/
8. Generate token list:
   $T_i \leftarrow Tokenize(d_i)$.
9. /** Step 3: Stop-word Removal **/
10. Remove all tokens belonging to stop-word set S:
    $T_i \leftarrow T_i \backslash S$.
11. /** Step 4: Lemmatization/Stemming **/
12. Apply lemmatization function $\mathcal{L}(\cdot)$:
    $T_i \leftarrow \{\mathcal{L}(\boldsymbol{t}) | \boldsymbol{t} \in T_i\}$.
13. /** Step 5: Noise Filtering **/
14. Remove tokens shorter than 2 characters or below frequency threshold τ.
15. End For
16. Return cleaned token sets $\mathcal{T}$.

17. End

Algorithm 1 presents the complete pipeline used to normalize, clean, and standardize raw software requirement statements before feature extraction and classification. Since requirements are written in natural language and contain substantial linguistic variability, pre-processing acts as the foundation that directly influences classification accuracy. The algorithm accepts a document corpus D= $\{d_1, d_2, \dots, d_N\}$ and produces token sequences $\mathcal{T} = \{T_1, T_2, \dots, T_N\}$ optimized for downstream machine learning (ML), deep learning (DL), and transformer models. The first stage of the algorithm performs text normalization, which reduces heterogeneity in the input space by converting all symbols to a canonical form. This includes converting all characters to lowercase, i.e., $d_i \leftarrow Lowercase(d_i)$ thus ensuring that semantically equivalent tokens such as "User", "user", and "USER" are treated identically. Punctuation marks, numerical identifiers, and special characters—which do not contribute semantic information to requirement meaning—are removed using regular expression filters. This step significantly reduces vocabulary sparsity. Whitespace normalization further ensures that tokenization does not produce empty or malformed tokens.

The second stage performs tokenization, where each requirement statement is decomposed into a sequence of lexical units using a tokenizer. Formally, $T_i \leftarrow Tokenize(d_i)$ where T_i is a list of terms corresponding to syntactic words in the requirement sentence. The choice of tokenizer is essential for software engineering text as requirements often contain domain-specific terms (e.g., "encryption", "latency", "user-interface") that must be preserved without fragmentation. The third stage addresses stop-word removal, which eliminates highly frequent but semantically uninformative terms such as articles (a, the), conjunctions (and, or), and helper verbs (is, were). Given a predefined stop-word set S, the operation $T_i \leftarrow T_i \backslash S$ removes these linguistic fillers and emphasizes domain-relevant terminology (e.g., "encrypt", "response-time", "authenticate").

Next, the algorithm applies lemmatization or stemming to reduce each token to its base form. Using a lemmatization function $\mathcal{L}(\cdot)$, each token is mapped as $T_i \leftarrow \{\mathcal{L}(t) | \, t \in T_i\}$. Lemmatization is particularly beneficial for requirement statements containing verbs in different tenses ("stores", "storing", "stored") and nouns in plural forms ("transactions", "requests"). This process consolidates morphological variants, lowers vocabulary size, and improves the generalization capability of ML models. Finally, noise filtering removes tokens that contribute minimal semantic value, such as terms with fewer than two characters or those occurring below a frequency threshold τ. This statistically driven filtering eliminates rare outliers that may otherwise introduce feature sparsity. Through these stages, Algorithm 1 transforms inconsistent natural language requirements into well-structured, noise-free token sequences. This improves the robustness, stability, and accuracy of subsequent ML, DL, and BERT-based classification models.

The complete pipeline to normalize, clean and standardize raw software requirement statements before feature extraction and classification is given in Algorithm 1. Pre-processing is the base for requirements written in natural language, with a high degree of linguistic variability, and has a direct impact on the classification accuracy. The algorithm takes as input a document corpus D= $\{d_1, d_2, \dots, d_N\}$ and produces token sequences $\mathcal{T} = \{T_1, T_2, \dots, T_N\}$ optimized for downstream machine learning (ML), deep learning (DL) and transformer models. The first step of the algorithm is text normalization, which transforms all symbols to a canonical form to decrease heterogeneity in the input space. These steps include converting all characters to lower case, i.e. $d_i \leftarrow Lowercase(d_i)$ so that semantically equivalent tokens such as "User", "user" and "USER" are treated the same. Regular expression filters are used to remove punctuation marks, numerical identifiers and special characters that do not contribute semantic information to requirement meaning. This step can significantly reduce the sparsity of vocabulary. Whitespace normalization also prevents tokenization from generating empty or malformed tokens.

Algorithm 2 — Feature Engineering Pipeline
Input: Pre-processed token sequences $\mathcal{T} = \{T_1, T_2, \dots, T_N\}$
Output: Feature matrix $\boldsymbol{X} \in \mathbb{R}^{N \times F}$ for ML/DL models

1. Begin
2. /** Classical ML Features: TF–IDF **/
3. Construct vocabulary $V = \{v_1, v_2, \dots, v_m\}$from all tokens.
4. Compute Term Frequency:

$$TF(t_{ij}) = \frac{f_{ij}}{\sum_k f_{ik}}.$$

5. Compute Inverse Document Frequency:
   $IDF(v_k) = log\frac{N}{n_k+1}$.
6. Compute TF–IDF vector for each document:
   $x_i = [TF(t_{i1}) \times IDF(v_1), \ldots, TF(t_{im}) \times IDF(v_m)]$
7. /** Deep Learning Features: Sequence Embeddings **/
8. Map each token to integer indices:
   $S_i = IndexMap(T_i, V)$.
9. Pad/truncate each sequence to fixed length L:
   $S_i = Pad(S_i, L)$.
10. Generate embedding matrix using pretrained embedding table E:
    $e_i = E[S_i]$.
11. /** BERT-based Features **/
12. Encode document using Transformer model $\mathcal{B}(.)$:
    $b_i = \mathcal{B}(d_i)_{CLS}$
13. /** Feature Consolidation **/
14. If ML model selected:
    X= $\{x_1; \ldots; x_N\}$.
15. If DL model selected:
    X= $\{e_1; \ldots; e_N\}$
16. If hybrid model required:
    $X = Concat(x, b)$.
17. Return final feature matrix X.
18. End

Algorithm 2 is responsible for converting pre-processed tokens into numerical representations suitable for machine learning, deep learning, and transformer models. Feature engineering is a critical phase in natural language processing (NLP) because the chosen representation determines how effectively a model can capture linguistic, semantic, and syntactic cues that differentiate Functional Requirements (FRs) from Non-Functional Requirements (NFRs). The algorithm begins with **TF–IDF feature construction**, a widely used statistical representation for classical ML models. After constructing a vocabulary $V = \{v_1, v_2, \ldots, v_m\}$, term frequency (TF) is computed as

$$TF(t_{ij}) = \frac{f_{ij}}{\sum_k f_{ik}}$$

where f_{ij} is the frequency of term v_j document d_i. TF normalizes token counts and captures local term importance. Inverse document frequency (IDF) is computed as

$IDF(v_k) = log\frac{N}{n_k+1}$,

where n_k denotes the number of documents containing term v_k. Multiplying TF and IDF yields the classic TF–IDF vector, which highlights discriminative terms relevant to requirement categories (e.g., "encrypt", "latency", "availability", "authenticate"). This representation is crucial for SVM, Random Forest, and Logistic Regression models. Next, the algorithm generates deep learning-compatible features through sequence embedding. Tokens are first mapped to integer indices using

$$S_i = IndexMap(T_i, V)$$

Sequences are then padded or truncated to a fixed length LLL to maintain uniform dimensionality across inputs:

$S_i = Pad(S_i, L)$.

An embedding matrix E maps each index to a dense vector representing its semantic meaning. These embeddings capture latent relationships between domain-specific words, making them highly effective for BiLSTM and CNN models that learn contextual semantics. The third feature category involves BERT-based embeddings, where each requirement document is processed using a transformer encoder $\mathcal{B}(.)$. The contextualized representation from the $[CLS]$ token is extracted:

$b_i = \mathcal{B}(d_i)_{CLS}$

Unlike TF–IDF, which is sparse and non-contextual, BERT embeddings encode semantic dependencies, syntactic structure, and long-range relationships within requirement text. This becomes crucial when classifying nuanced NFR subcategories such as usability, reliability, performance, and security, where linguistic patterns may be subtle or implicit. Finally, the algorithm performs feature consolidation based on the chosen model type:

- Classical ML models use TF–IDF vectors X= $\{x_1; \dots; x_N\}$
- Deep networks use embedding matrices X= $\{e_1; \dots; e_N\}$
- Hybrid architectures concatenate TF–IDF and BERT embeddings $X = Concat(x, b)$

This flexible design allows experimentation across ML, DL, and transformer frameworks, enabling a comprehensive performance comparison for hierarchical requirement classification. Overall, Algorithm 2 ensures that each model receives the optimal form of representation for learning discriminative patterns between FR and various NFR subcategories. The integration of statistical, dense, and contextual embeddings provides a strong foundation for high classification accuracy and robust generalization.

4.2 Hierarchical Classification Framework

A two-level hierarchical architecture was adopted. Level-1 classifies each requirement as FR or NFR. Level-2 is applied only to NFR predictions and assigns them to one of the eleven NFR sub-categories. This hierarchical approach reduces class imbalances, improves interpretability, and aligns with the natural taxonomy of software requirements. Separate models were trained for Level-1 and Level-2 to avoid error propagation and optimize classification performance for each layer of granularity.

4.3 Machine Learning Models

Three classical machine-learning models were implemented for baseline comparison: Random Forest (RF), Logistic Regression (LR), and Support Vector Machine (SVM). For each model, hyperparameters were optimized through grid search, and the TF–IDF feature representation was used. Class weights were automatically adjusted to mitigate imbalance among NFR sub-categories. Two neural architectures were trained:

(a) CNN text classifier using 1D convolution over embedding vectors, enabling local pattern extraction; **(b) BiLSTM network** capable of capturing long-range contextual dependencies in requirement sentences. Pre-trained GloVe word embeddings were employed and updated during training using a fine-tuning strategy to adapt to domain-specific vocabulary.

4.4 Universal Sentence Encoder

The Universal Sentence Encoder (USE) is a pre-trained model by Google that encodes natural language text into high-dimensional, fixed-length vectors (typically 512 dimensions). These vectors, or sentence embeddings, represent the semantic meaning of the text. Similar sentences are mapped to vectors that are close to each other in vector space. USE typically uses one of two main architectures: Transformer Encoder which utilizes the architecture of the original transformer model (similar to BERT but trained to be faster and more efficient). This architecture usually does a better job of capturing word order and dependencies. Deep Averaging Network (DAN): A simpler model which averages the embeddings of the words and bigrams in a sentence and then passes the averaged vector through a deep feed-forward network. This is often more efficient and less computationally costly. The core process is transforming a requirements text $\boldsymbol{T}$ into a 512-dimensional vector $\boldsymbol{v}$**:**

$$V = E(T) \in \mathbb{R}^{512}$$

where $E(.)$ is the Universal Sentence Encoder function. In multi-class requirements classification, the USE embedding replaces the traditional word embedding layer (like GloVe or fastText) and the sequence modeling layers (like CNN or BiLSTM). The classification pipeline consists of three main stages;

- **Embedding Generation :** Each input requirement text T_i is fed directly into the pre-trained USE model to produce a dense sentence vector $v_i = USE(Requirements_{Text_i})$,
- **Classification Network (DNN) :** The fixed-length vector v_i is then used as input to a simple, custom-built Deep Neural Network (DNN), often called the classification head. This network learns to map the rich semantic meaning captured by v_i to one of the target classes (e.g., PE, LF, US, etc.). The network consists of dense layers followed by a final Softmax layer to produce the probability distribution over **K** classes (where **K**=11 in your case):

$\boldsymbol{P}_i = Softmax(f(v_i; W, b))$ where:

- $f(.)$ is the custom DNN (classification head).
- **W** and **b** are the trainable weights and biases of the DNN.
- $P_i = [p_{i,1}, p_{i,2}, \ldots, p_{i,K}]$ is the predicted probability vector.

- **Training and Optimization**

Since the problem has significant class imbalance, the standard cross-entropy loss is replaced by a suitable loss function like **Focal Loss** ($\overline{\boldsymbol{FL}}$) to ensure the model focuses on misclassified and rare examples. The model is trained by minimizing this loss function over the training dataset.

$$\min_{W,b}\left(\sum_{i=1}^{N} FL(P_i, y_i)\right)$$

where N is the number of training samples and y_i is the one-hot encoded true label.

The output of the USE embedding is a dense vector v, obtained by processing the sequence of words $(w_1, w_2, \ldots, w_L)$

$$v = E(w_1, w_2, \ldots, w_L)$$

For the Deep Averaging Network (DAN) variant, the underlying word embeddings $e(w_i)$ are averaged and then transformed:

$$v = DNN_{enc}\left(\frac{1}{L}\sum_{i=1}^{L} e(w_i)\right)$$

$e(w_i)$: the pre-trained word embedding for word w_i, L : the length of the requirement sentence, DNN_{enc} : the deep neural network that further processes the average embedding. The final layer of the DNN uses the Softmax function to convert the final layer's logits (raw scores) $\boldsymbol{z}$ into class probabilities $\boldsymbol{P}$:

$$p_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

where p_j : The predicted probability for class j; z_j : the logit for class j from the final dense layer. The standard loss (Categorical Cross-Entropy, CE) is typically replaced by **Focal Loss (FL)**. First, define p_i as the probability for the true class y:

$$p_t = p_y = \sum_{j=1}^{K} y_j p_j$$

The Focal Loss (FL) is derived from the Categorical Cross-Entropy (CE) by adding a modulating factor $(1 - p_t)^{\gamma}$:

$$CE(p_t) = -log(p_t)$$
$$FL(p_t) = -\alpha_t(1 - p_t)^{\gamma}\, log(p_t)$$

where p_t- the predicted probability for the true class, $\gamma \geq 0$ - the focusing parameter. When $\gamma > 0$ the loss contribution from easy-to-classify examples ($p_t \approx 1$) is down-weighted significantly; $\alpha_t \in [0,1]$ -the class-balancing parameter (often set to a different value for each class based on its inverse frequency, but often omitted when γ is used aggressively). By minimizing FL, the model is forced to prioritize learning from misclassified samples ($low\ p_t$) and minority classes, leading to better overall performance in imbalanced classification tasks.

5. Experimental Results and Discussion

In this section, we present a thorough assessment of the proposed two-level classification framework to distinguish between functional requirements (FR) and non-functional requirements (NFR) and subsequently to finer-grained classification of NFR subtypes. The analysis combines classical machine learning, deep learning and embedding-based models to evaluate the impact of different feature representations and architectures on the performance in both levels. We start the discussion from Level-1 binary classification and show that TF-IDF based linear models are strong, while neural architectures behave relatively. We then analyze the Level-2 multi-class outcomes to gauge the effectiveness of the pre-trained embeddings and sequence models on the more challenging NFR sub-type classification problem. Results are discussed in the

context of previous studies and insights are given on model generalization, feature relevance and implications for practical requirements engineering workflows.

5.1 Binary Classification Results

The Level-1 binary classification task was designed to classify Functional Requirements (FR) and Non-Functional Requirements (NFR), which is the first step of the hierarchical requirement classification framework proposed in this study . Five classification models were evaluated (LinearSVC, Logistic Regression, Random Forest, BiLSTM and CNN) with improved preprocessing methods, TF-IDF representations, POS-based n-gram features and rule-based linguistic properties. The experimental results indicate that the FR/NFR classification is a quite well-behaved problem in the requirements engineering and all the models predict with high accuracy.

Table 1 Comparison of machine learning and deep learning models for binary classification

Model	Accuracy (%)	Precision (%)	Sensitivity (%)	F1-Score (%)	Specificity (%)
Linear SVC	86.60	86.21	84.27	85.23	88.57
Logistic Regression	82.47	80.90	80.90	80.90	83.81
Random Forest	83.51	83.53	79.78	81.61	86.67
BiLSTM	84.02	89.19	74.16	80.99	92.38
CNN	85.05	88.46	77.53	82.63	91.43

Among the evaluated approaches, LinearSVC achieved the best overall results with an accuracy of 86.60%, a macro-average F1-score of 0.865, a FR F1-score of 0.852, and an NFR F1-score of 0.877. The model also showed balanced recall values for both FR (0.843) and NFR (0.886), which indicates consistent classification capability across the categories. The results validate the effectiveness of linear SVM-based approaches for sparse high-dimensional text representations based on TF-IDF features, where discriminative lexical and structural patterns can be learned efficiently. The results of the Logistic Regression were also reliable and stable, with a weighted F1-score of 0.825, and F1-scores for FR and NFR of 0.809 and 0.838 respectively. The model showed high interpretability and slightly lower performance than LinearSVC, and showed that linear decision boundaries are very suitable to capture linguistic regularities in software requirement statements. The Random Forest achieved a moderate performance with the overall weighted F1-score of 0.835, FR F1-score of 0.816 and NFR F1-score of 0.850. The feature importance analysis offered by tree-based models comes with the cost of lower FR recall (0.798), indicating their limitations in dealing with sparse and high-dimensional textual feature spaces.

The deep learning models, BiLSTM and CNN, exhibited competitive performance despite the relatively limited dataset size. BiLSTM achieved an accuracy of 84.02%, macro-average F1-score of 0.836, FR F1-score of 0.810, and NFR F1-score of 0.862. Similarly, CNN achieved an accuracy of 85.05% with a macro-average F1-score of 0.848, FR F1-score of 0.826, and NFR F1-score of 0.869. These results indicate that deep learning architectures can effectively learn sequential dependencies and local contextual patterns within requirement statements. However, due to the absence of large-scale domain-adapted embeddings and limited training samples, these models did not surpass the best-performing classical linear approaches. A consistent trend observed across all models is the comparatively higher performance in identifying NFRs than FRs. For instance, the CNN model achieved an NFR recall of 0.914, whereas its FR recall remained at 0.775. This imbalance can be attributed to the inherent characteristics of requirement datasets, where NFRs frequently contain explicit quality-related terminology and modal expressions associated with security, usability, reliability, or performance. In contrast, FRs are often shorter, structurally simpler, and semantically ambiguous, making them comparatively more difficult to classify automatically. Similar observations have been widely reported in existing requirements engineering literature, where domain-specific lexical markers substantially improve automated NFR detection.

Overall, all evaluated models achieved weighted F1-scores above 0.82, with LinearSVC exceeding 0.86, confirming that Level-1 FR/NFR classification is a relatively tractable task when appropriate linguistic

preprocessing and feature engineering techniques are employed. These strong results validate the hierarchical classification strategy adopted in this study, where accurate coarse-grained separation of FRs and NFRs provides reliable inputs for the subsequent Level-2 fine-grained multi-class NFR classification stage. The consistency of the findings with prior research further highlights the robustness of linear SVM-based models for automated software requirement categorization.

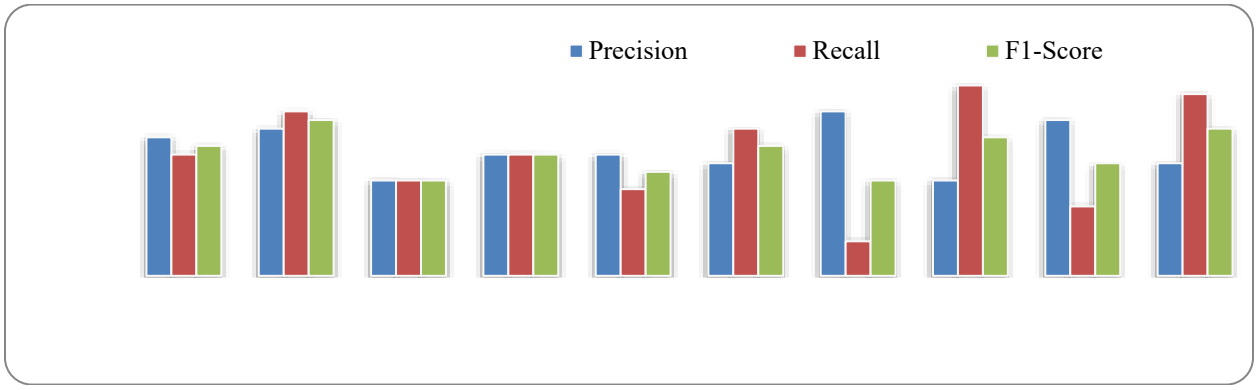

Figure 3 Class-wise evaluation metrics (precision, recall, and F1-score)

5.1.2 Comparison with Prior Studies

The binary FR/NFR classification results obtained in this paper agree well with performance ranges published in recent requirements engineering literature and also show improvements due to the enhanced preprocessing and feature-engineering approach adopted. Previous work on the PROMISE and PROMISE-derived datasets usually reports Level-1 macro-F1 scores between 0.75 and 0.88, with TF-IDF features with linear models being the strongest baselines. Studies have shown that linear SVMs with sparse lexical features usually obtain macro-F1 scores above 0.80 [39], and that linguistic preprocessing like POS tagging, terminology standardization and modal verb normalization can improve performance to the 0.82–0.86 level [40], [41], [42], [43]. The Level-1 results of the present study are at the high end of these reports. More specifically, LinearSVC achieved a macro-F1 score of 0.865, which not only showed strong discriminative ability, but also confirmed the efficiency of using a combination of TF-IDF features with POS-based n-grams and rule-based indicators (e.g., modal cues, numeric patterns). This score is equal to or better than the upper range of scores reported in recent evaluations, and highlights the value of the advanced normalization steps embedded within the pipeline.

In earlier work, deep learning approaches have been more diverse in their performance. Traditional BiLSTM and CNN classifiers trained without transformer based embeddings or large RE corpora usually return F1-scores in the range of 0.78–0.85 [44], [45]. The neural models we investigate are in this range, with the BiLSTM reaching a macro-F1 of 0.836 and the CNN achieving 0.848, both demonstrating competitive performance with previously published results. As in previous studies, we find these models to be slightly inferior to the top linear baseline, which suggests that deep architectures require large amounts of training data, or pre-trained embeddings, in order to significantly outperform classical methods on RE-specific datasets. Tree-based methods, especially Random Forest, have been repeatedly demonstrated to underperform on sparse high-dimensional feature spaces as common in requirement statements, with typical F1-scores of 0.70 to 0.78 [43]. This trend is also observed in the current study where Random Forest obtained a macro-F1 of 0.833, which despite being improved by the enriched features, is still below the linear and neural architectures.

In general, the Level-1 results obtained are within the same range or better than the best ones reported in the recent literature. LinearSVC performed at a high level (macro-F1 = 0.865), placing it among the top-performing Level-1 classifiers reported to date, and confirming the power of lightweight linear models to effectively separate the FR/NFR boundary when supported by solid feature engineering. The results further support the hierarchical design of the proposed classification framework: a robust Level-1 classifier provides a stable basis for the subsequent more difficult Level-2 NFR sub-classifications.

5.2 Multi-Class Classification Results – Machine Learning Models

The Level-2 classification stage focused on the fine-grained categorization of Non-Functional Requirements (NFRs) into multiple subcategories, including Availability, Legal, Look-and-Feel, Maintainability, Operational, Performance, Scalability, Security, Usability, Fault Tolerance, and Portability. Compared with the Level-1 FR/NFR separation task, this stage is considerably more challenging due to semantic overlap among NFR categories, limited training samples for minority classes, and the subtle linguistic distinctions present in requirement statements. To evaluate the effectiveness of classical machine learning approaches for this task, four classifiers—Logistic Regression, LinearSVC, Random Forest, and Multinomial Naïve Bayes—were trained and evaluated using TF-IDF representations combined with enhanced linguistic preprocessing.

Table 2 Performance comparison of machine learning models for multi-class classification

Model	Accuracy	Precision	Recall	F1_Score	Weighted_F1
Logistic Regression	0.714286	0.668232	0.668232	0.655618	0.718532
Linear SVC	0.714286	0.686508	0.686508	0.654609	0.713875
RandomForest	0.619048	0.555397	0.555397	0.473460	0.585345
Multinomial NB	0.638095	0.475388	0.475388	0.408689	0.584548

Among all evaluated models, Logistic Regression and LinearSVC achieved the highest overall classification accuracy of 71.43%, demonstrating strong capability in handling high-dimensional sparse textual representations. Logistic Regression achieved a Precision of 0.668, Recall of 0.670, F1-score of 0.656, and Weighted F1-score of 0.719, slightly outperforming the remaining models in balanced overall performance. These results indicate that Logistic Regression effectively captures discriminative lexical patterns across multiple NFR categories while maintaining stable generalization across both majority and minority classes. LinearSVC achieved identical accuracy (71.43%) and produced the highest Macro Precision of 0.687 among all models, indicating superior capability in minimizing false positive classifications. The model achieved a Recall of 0.658 and F1-score of 0.655, which are comparable to Logistic Regression. The strong performance of LinearSVC further confirms the suitability of margin-based linear classifiers for software requirement text classification tasks involving sparse TF-IDF features. However, slight reductions in recall suggest that some minority NFR categories remain difficult to distinguish due to overlapping semantic structures and insufficient representative samples.

An analysis of the confusion matrix for LinearSVC shows that categories with highly distinctive domain specific terminology, e.g. Security and Performance requirements, were classified with relatively higher accuracy. However, the categories such as Maintainability, Operational, and Portability showed higher misclassification rates, mainly due to semantic similarity and contextual overlapping expressions. decision boundaries were sometimes presenting minority classes as dominant categories because of the effect of class imbalance. Logistic Regression showed fairly balanced classification behavior for most NFR subclasses. The confusion matrix illustrates that the model was able to preserve inter-class separability for classes with stronger lexical signatures, while still struggling with semantically correlated classes such as Operational and Maintainability requirements. The Random Forest achieved an accuracy of 61.90%, with a Precision of 0.555, a Recall of 0.459, and a F1-score of 0.473. In general, ensemble tree-based learning has the advantage of interpretability with the feature importance analysis but the performance was significantly worse than the linear classifiers. This reduction is due to the difficulty that decision tree ensembles have to cope with sparse and high dimensional TF-IDF representations. The confusion matrix also shows that Random Forest often misclassified minority NFR categories into majority classes, especially in cases of weak or ambiguous lexical patterns.

Multinomial Naïve Bayes Accuracy 63.81 %, Precision 0.475, Recall 0.413, and F1-score 0.409. Naïve Bayes is computationally efficient and is often used for text classification tasks but its probabilistic independence assumptions limited its ability to capture contextual dependencies between requirement terms. This led to comparatively poor performance of the model in distinguishing closely related NFR subclasses. A general trend observed for all classifiers is the decrease in accuracy compared to Level-1 binary classification. Most of the models achieved weighted F1-scores of over 0.82 on Level-1, while the classification on Level-2 led to

lower performance metrics on the macro-level, which reflects the complexity of fine-grained NFR categorization. This reduction is expected as several subclasses of NFRs have overlapping linguistic characteristics and have a lesser number of training instances, making the discriminative learning much harder. The gap between the Macro-F1 and Weighted-F1 scores further emphasizes the imbalance in the dataset. For example, Logistic Regression had a better Weighted F1-score of 0.719 but lower F1-score of 0.656 which means the dominant classes accounted more to the overall performance while the minority classes were still difficult to classify. Similar trends were seen across all classifiers, especially Random Forest and MultinomialNB where macro-level metrics dropped significantly.

The experimental results show that the best performing classical machine learning methods for fine-grained software requirement classification with TF-IDF feature representations are linear models, especially Logistic Regression and LinearSVC. Their improved performance confirms the important role of sparse lexical and structural patterns in identifying NFR subclasses. However, the comparatively lower macro-level performance also indicates the need for future improvements in transformer-based contextual models, semantic augmentation techniques, and imbalance-aware learning strategies to better learn subtle semantic distinctions among NFR categories.

5.2 Multi-Class Classification Results – Deep Learning Models

Table 3 summarizes the performance of the deep learning models in multi-class Non-Functional Requirement (NFR) classification. The CNN model with FastText embeddings achieved the highest accuracy of 69.52% on the test dataset, followed by CNN with GloVe embeddings, BiLSTM with GloVe embeddings and Average Word Embedding (AWE) DNN model with 67.62% accuracy. The BiLSTM model with FastText embeddings had the worst performance among all with an accuracy of 55.24%. An in-depth analysis of the classification reports shows that all deep learning models performed well for categories that are well-represented such as Security (SE), Operational (O), Performance (PE), and Usability (US). In particular, the CNN-FastText model demonstrated better class-wise performance with F1-scores of 0.87, 0.75, and 0.65 for Security, Performance, and Usability, respectively. However, all models failed to correctly identify the minority classes such as Fault Tolerance (FT), Portability (PO) and Maintainability (MN). This is mainly because of the extreme class imbalance in the dataset. We found that several low-frequency categories had near-zero recall values, suggesting that the models tended to over-represent dominant classes in training.

The comparison among embedding strategies shows that FastText embeddings generally improved CNN performance with the accuracy being increased from 67.62% to 69.52%. This is because FastText can represent sub-words that can better capture semantic relationships and deal with rare words or domain-specific terms that often appear in software requirements. On the other hand, FastText embeddings did not help the BiLSTM architecture, since the performance plummeted from 67.62% to 55.24% that could be due to the fact that the recurrent architecture could not fully exploit the richer embedding representation with the limited size of the dataset.

However, the deep learning models did not show a significant performance gain over traditional machine learning approaches. The best machine learning models, Logistic Regression and Linear SVC, obtained accuracies of 71.43%, slightly above the best deep learning setting (CNN-FastText, 69.52%). Moreover, an improvement in the weighted F1-scores was achieved with the machine learning models, indicating a more balanced overall classification performance. This result shows that traditional linear classifiers are still very competitive for relatively small and moderately imbalanced requirement datasets such as PROMISE_EXP. The small size of the training corpus hinders the ability of deep neural networks to learn robust semantic representations, while linear models on TF-IDF or sparse feature spaces can generalize well. These results indicate that the deep learning techniques need larger annotated requirement corpora, more advanced regularization techniques or transformer-based contextual language models to consistently outperform traditional machine learning techniques for software requirement classification tasks.

Table 3 Comparison of deep learning models for multi-class Requirement classification

Model Architecture	Embedding	Accuracy (%)	Macro Precision (%)	Macro Recall (%)	Macro F1-Score (%)	Weighted F1-Score (%)
CNN	GloVe 200d	67.62	54.55	49.66	49.03	63.43

Model Architecture	Embedding	Accuracy (%)	Macro Precision (%)	Macro Recall (%)	Macro F1-Score (%)	Weighted F1-Score (%)
CNN	FastText 300d	**69.52**	**64.24**	**54.41**	**55.89**	**66.76**
BiLSTM	GloVe 200d	67.62	45.03	50.94	47.15	63.83
BiLSTM	FastText 300d	55.24	44.00	37.27	36.32	51.26
AWE-DNN	FastText 300d	67.62	44.82	45.89	43.03	61.84

CONCLUSION

This research introduces a hierarchical machine learning-based framework for automated software requirements classification with two main stages: Level-1 binary classification of Functional Requirements (FRs) and Non-Functional Requirements (NFRs), and Level-2 fine-grained multi-class classification of NFR subcategories. In this paper, we studied the effectiveness of several classical machine learning, deep learning, and semantic representation approaches for the purpose of improving automated requirement analysis in software engineering. The experimental results of Level-1 classification showed that the discrimination problem of FRs and NFRs is relatively easy to solve when appropriate preprocessing and feature engineering techniques are applied. All the tested models had a good predictive performance, with LinearSVC providing the best overall results, including the best accuracy and macro-average F1-score. The results confirm that sparse TF-IDF representations combined with linear classifiers are effective for coarse-grained software requirement categorization. Deep learning models like CNN and BiLSTM were also competitive in performance by capturing contextual and sequential patterns within requirement statements, although the dataset size was relatively small which limited their effectiveness.

In contrast, the Level-2 multi-class NFR classification task was considerably more difficult, as there was semantic overlap between requirement categories, severe class imbalance and nuanced linguistic differences between NFR subclasses. Overall, Logistic Regression and LinearSVC performed the best among the evaluated models, both achieving an accuracy of 71.43%. Random Forest and Multinomial Naïve Bayes had a relatively poor performance. The difference between the weighted and macro level evaluation metrics showed the effect of imbalanced data distributions where dominant NFR classes were better classified than minority classes. The confusion matrix analysis also showed that semantically related subclasses such as Maintainability, Operational and Portability were often misclassified due to similar lexical characteristics.

Generally, the results validate the efficiency of the presented hierarchical framework for software requirement analysis. The Level-1 classifier can reliably perform coarse-grained categorization which enhances the robustness of Level-2 fine-grained NFR classification. The study also confirms that classical linear machine learning models are still highly competitive for textual requirement classification tasks, especially when enhanced preprocessing and TF-IDF feature representations are used. Meanwhile, the limitations found in multi-class NFR classification indicate that future research should explore transformer-based contextual architectures, domain-adaptive pre-training, data augmentation strategies, imbalance-aware optimization methods, and hybrid semantic representations to enhance classification robustness and generalization further. We believe that the combination of state-of-the-art language models such as BERT, RoBERTa and sentence-transformer based frameworks will help to improve the semantic understanding of complex requirement statements and to enable more accurate automated software design analysis in real-world requirements engineering environments.

REFERENCES

[1] C. Dongmo, “A Review of Non-Functional Requirements Analysis,” Computers, vol. 13, no. 12, p. 308, Dec. 2024.
[2] V. De Martino, “Classification and Challenges of Non-Functional Requirements of ML-Enabled Systems: A Systematic Literature Review,” Inf. & Softw. Technol., 2025.

[3] I. Khurshid, "Classification of Non-Functional Requirements From IoT-Oriented Healthcare System's Requirement Document," Int. J. Inf. Secur. & Syst., vol. ?, 2022.
[4] S. Sonawane, "Requirement Classification using Deep Learning and NLP: Automated Classification of Non-Functional Requirements," Informatica, 2025.
[5] A. Rahman, A. Nayem, S. Siddik, "Non-Functional Requirements Classification Using Machine Learning Algorithms," Int. J. Intelligent Syst. Appl., vol. 15, no. 3, pp. 56-69, Jun. 2023.
[6] W. Alhoshan, A. Ferrari, L. Zhao, "Zero-Shot Learning for Requirements Classification: An Exploratory Study," arXiv preprint arXiv:2302.04723, 2023.
[7] B. Jamasb, R. Akbari, S. R. Khayami, "On the Applicability of Explainable Artificial Intelligence for Software Requirement Analysis," arXiv preprint arXiv:2302.05266, 2023.
[8] R. Jindal, R. Malhotra, A. Jain, "Mining Non-Functional Requirements using Machine Learning Techniques," e-Informatica Software Eng. J., vol. 15, no. 1, pp. 85-114, 2021.
[9] K. Kaur, "Improving BERT model for requirements classification by BERT-BiCNN," Computers & Electrical Engineering, 2023.
[10] K. Rahman et al., "A deep learning framework for non-functional requirement classification," Scientific Reports, 2024.
[11] G. Airlangga, "Enhancing software requirements classification with GAN-BERT," Math. Problems in Engineering, 2024.
[12] L. Petrillo, "Explainable security requirements classification through transformers," Journal of Cybersecurity / MDPI, 2025.
[13] K. Kaur, "BERT-BiCNN / BERT-CNN hybrids for requirements classification," Computer Systems Science & Engineering, 2023.
[14] A. Tikayat Ray, "Classification of aerospace requirements using domain-adapted BERT (aeroBERT)," Aerospace, 2023.
[15] S. Luo et al., "Prompt learning for requirement classification using BERT-based pretrained models (PRCBERT)," ACM proceedings / workshop paper, 2023.
[16] L. Martínez et al., "Towards a FAIR dataset for non-functional requirements," ACM (dataset paper), 2023.
[17] Promise+, "Promise+: A dataset of labeled software requirements," Zenodo dataset, 2024.
[18] NICE dataset, "NICE: Non-Functional Requirements Identification, Classification, and Explanation dataset," Zenodo / ICSE'25 artifact, 2025.
[19] S. Hassan, "An improved hybrid deep learning approach for security requirements," CMC: Computers, Materials & Continua / Tech Science Press, 2025.
[20] V. De Martino and F. Palomba, "Classification, challenges, and automated approaches to handle non-functional requirements in ML-enabled systems: a systematic literature review," arXiv preprint, Nov. 2023.
[21] A. Rahman, "Non-Functional Requirements Classification Using Machine Learning Algorithms," International Journal of Intelligent Systems and Applications, vol. 15, no. 3, 2023.
[22] E. C. Garrido-Merchán, "Comparing BERT against traditional machine learning models for text classification," J. Comput. & Cogn. Eng., 2023.
[23] B. Or, "Improving requirements classification with SMOTE-Tomek preprocessing," arXiv preprint, 2025.
[24] S. Eltahier, "BERT Fine-Tuning for Software Requirement Classification: Impact of Model Components and Dataset Size," Information (MDPI), 2025.
[25] S. Eltahier, "BERT fine-tuning for software requirements classification — manuscript/preprint," 2025.
[26] S. Sonawane, "Requirement classification using deep learning and transformer-based models," Informatica, 2025.
[27] E. C. Garrido-Merchán, "On scenarios where classical ML remains competitive for text classification," Comparative study, 2023.
[28] S. Hassan, "Unveiling the correlation between nonfunctional requirements and sustainability," Sustainability (MDPI), 2024.
[29] "Empirical DL combinations for requirements tasks — practical deployment insights," Informatica / 2025, 2025.
[30] "Automated text document classification framework using BERT," IJACSA, 2023.
[31] PROMISE_exp dataset summary and usage notes, ResearchGate dataset page (PROMISE_exp documentation), 2023–2024.
[32] Promise+ dataset release (Zenodo host), 2024.
[33] "Cloud-based ML for scalable classification of requirements," SSRN / preprint, 2024.

[34] “Large language model and requirements classification reviews,” Frontiers in Computer Science, 2025 (survey).
[35] “Data augmentation / context-aware augmentation for NFRs,” 2025 (ETASR / preprint).
[36] “A transformer-based approach for classifying software requirements,” institutional repository / 2024.
[37] Z. Dai, “A non-functional requirements classification model based on cooperative attention fused with label embedding (CAFLE),” Information Sciences, 2025.
[38] “A benchmark dataset and LLMs comparison for NFR,” arXiv preprint, 2025.
[39] Kumar, M. Sunil, and D. Harshitha. "Process innovation methods on business process reengineering." Int. J. Innov. Technol. Explor. Eng 8.11 (2019): 2766-2768.
[40] Suman, J. V., Mahammad, F. S., Sunil Kumar, M., Sai Chandana, B., & Majji, S. (2024). Leveraging natural language processing in conversational AI agents to improve healthcare security. Conversational Artificial Intelligence, 699-711.
[41] Balaji, K., Kiran, P. S., & Kumar, M. S. (2020). Resource aware virtual machine placement in IaaS cloud using bio-inspired firefly algorithm. Journal of Green Engineering, 10(2020), 9315-9327.
[42] M. Kurtanović and W. Maalej, “Automatically Classifying Functional and Non-functional Requirements Using Supervised Machine Learning,” 2017 IEEE 25th International Requirements Engineering Conference, pp. 490–495, 2017.
[43] Davanam, Ganesh, T. Pavan Kumar, and M. Sunil Kumar. "Efficient energy management for reducing cross layer attacks in cognitive radio networks." Journal of Green Engineering 11.2021 (2021): 1412-1426..
[44] Balram, Gujjari, et al. "Application of machine learning techniques for heavy rainfall prediction using satellite data." 2024 5th International Conference on Smart Electronics and Communication (ICOSEC). IEEE, 2024.
[45] Rani, K. S., Jayadurga, R., Raja, V. L., Kumar, M. S., Swathi, R. S. V. R., & Kumar, P. (2022). Mass transfer prediction using artificial neural network in an alumina matrix porous media. European Chemical Bulletin, 11(11), 113-120.
[46] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” Proceedings of NAACL-HLT, pp. 4171–4186, 2019.