iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** VII **Month of publication:** July 2025

DOI: https://doi.org/10.22214/ijraset.2025.73368

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue VII July 2025- Available at www.ijraset.com

Identifying Non-Functional Requirements from Unconstrained Documents using Natural Language Processing & Machine Language Techniques

Gajwada Greeshma[1], Dr.V.Umarani[2]
[1]*Post Graduate Student, M.Tech(SE),* [2]*Professor & Head, Department of Information Technology, Jawaharlal Nehru Technological University Hyderabad*

Abstract: ***Manual identification of Non-Functional Requirements (NFRs) from software documents is often laborious, error-prone, and inconsistent. This paper presents a system that leverages Natural Language Processing (NLP) and Machine Learning (ML) to automate the extraction and classification of NFRs from unstructured documents. The approach supports various formats such as .txt, .pdf, and .docx, and categorizes extracted requirements into standard NFR categories such as performance, security, usability, and maintainability. Our model demonstrates promising accuracy using a labeled data set and provides a user-friendly web interface built with Flask to facilitate input and output visualization. Results confirm the effectiveness of automated NFR detection for enhancing early-stage software requirement engineering.***
Keywords: ***Non-Functional Requirements, Requirement Engineering, Natural Language Processing, Machine Learning, Classification***

I. INTRODUCTION

In software engineering, Non-Functional Requirements (NFRs) play a critical role in defining the quality attributes of a system, such as performance, security, usability, scalability, maintainability, and reliability. Unlike Functional Requirements, which specify *what* the system should do, NFRs describe *how* the system should behave under various conditions. They influence architectural decisions, impact user satisfaction, and determine long-term system sustainability.

Despite their importance, NFRs are often poorly documented, ambiguous, or intermixed with functional requirements in natural language documents. This makes them difficult to identify and interpret consistently, especially when documents are long, unstructured, or written without a standardized format. Traditionally, the task of extracting NFRs has relied on manual review by analysts, which is time-consuming, subjective, and error-prone. Moreover, in agile and fast-paced development environments, the lack of early and accurate NFR identification can lead to costly rework, security loopholes, or underperforming systems.To address these challenges, this paper proposes a machine-assisted system for the automated extraction and classification of Non-Functional Requirements from unconstrained requirement documents. The proposed approach leverages Natural Language Processing (NLP) techniques to preprocess and structure textual data, and then applies Machine Learning (ML) models—including advanced transformer-based classifiers like BERT—to categorize sentences into predefined NFR types. The system is designed to support documents in common formats such as .txt, .pdf, .docx, and .csv. A web-based user interface built using Flask allows users to easily upload documents and visualize the extracted NFRs in a categorized format. This not only improves accessibility for practitioners but also bridges the gap between raw requirement documentation and actionable insights.Through experimental evaluation on a labeled dataset, the proposed system demonstrates high classification accuracy and practical utility. Ultimately, this research contributes toward automating a traditionally manual phase of software engineering, thereby enhancing the efficiency, reliability, and repeatability of early-stage requirement analysis.

II. METHODOLOGY

This section outlines the end-to-end pipeline of our system, from data acquisition to model training and classification. The methodology was carefully designed to ensure that the system can process a wide variety of requirement documents and accurately classify Non-Functional Requirements (NFRs).

A. Dataset

The dataset used for this study was sourced from a publicly available Kaggle repository, which contains annotated requirement statements categorized as either Functional Requirements (FRs) or Non-Functional Requirements (NFRs). To improve model robustness and account for the diverse nature of NFR expressions, we augmented the dataset to include over 500 distinct samples, covering a comprehensive set of NFR categories including Performance, Security, Usability, Maintainability, Scalability, and Reliability.

The dataset was pre-labeled and reviewed to ensure the reliability of category assignments. Each requirement statement was treated as an independent sentence for classification, and efforts were made to balance the distribution of classes to avoid model bias toward more frequent categories.

B. Preprocessing

To prepare the unstructured text for machine learning, a Natural Language Processing (NLP) pipeline was employed. This stage is essential for reducing noise and transforming the raw sentences into a structured format suitable for feature extraction. The preprocessing steps included:

- Sentence Tokenization: Splitting long paragraphs into individual sentences using NLTK's sent_tokenize() function, ensuring each sentence could be independently evaluated.
- Stopword Removal: Common English stopwords (e.g., and, the, is) were removed to focus on semantically meaningful terms.
- Part-of-Speech (POS) Tagging: POS tags were used to enrich the input with syntactic information, which is useful for capturing patterns like "system shall [verb]".
- Lemmatization: Each word was reduced to its base form (e.g., running → run) using WordNet lemmatizer to improve generalization during training.

Special care was taken to retain domain-relevant keywords during preprocessing, and formatting inconsistencies were normalized across documents.

C. Feature Extraction

After preprocessing, the clean textual data was transformed into numerical representations suitable for machine learning algorithms. Two major approaches were used:

- TF-IDF Vectorization: A statistical technique that reflects the importance of a word in a document relative to a corpus. TF-IDF produced sparse vectors representing the frequency-weighted presence of terms across requirement statements. It provided a strong baseline for classical ML models.
- Word Embeddings: To capture semantic and contextual meaning, we experimented with pre-trained Word2Vec embeddings and fine-tuned BERT (Bidirectional Encoder Representations from Transformers) embeddings. Word2Vec allowed for capturing relationships between words in a fixed-size vector, while BERT provided deep contextual understanding using transformer architecture. These embeddings were especially effective in handling vague or implicit NFR expressions.

D. Classification Models

Multiple supervised learning algorithms were trained and evaluated to determine the most effective classifier for NFR detection. The following models were implemented:

- Logistic Regression: Used as a simple linear baseline. While efficient, it struggled with non-linear separations among complex NFR categories.
- Support Vector Machines (SVM): Performed better than logistic regression by introducing a non-linear kernel and a margin-based classification approach. It achieved good results on high-dimensional TF-IDF features.
- Random Forest: A robust ensemble learning method that provided decent interpretability and generalization, though it was slower to train on larger feature sets.
- Fine-Tuned BERT Transformer: This model achieved the best performance across all metrics. By leveraging its deep bidirectional attention mechanism, BERT was able to accurately classify nuanced requirement statements that traditional models misclassified. We fine-tuned a pre-trained BERT model on our labeled dataset using the HuggingFace Transformers library, enabling the model to specialize in NFR classification tasks.

Model selection was guided by a combination of cross-validation, precision, recall, and F1-score analysis. Fine-tuned BERT outperformed other models, especially in detecting implicit NFRs and ambiguous sentence structures.

III. SYSTEM ARCHITECTURE

Our system is built using Flask. It supports file uploads and displays extracted NFRs on the front-end using Jinja templates. Users can upload .txt, .csv, .pdf, or .docx files. The backend processes the file and shows categorized NFRs as bullet points grouped by category.The system architecture is designed as a modular pipeline to ensure flexibility, scalability, and ease of maintenance. It comprises a frontend, backend, and a model-serving layer. The frontend is developed using HTML, CSS, and Jinja2 templates, embedded in a Flask framework. Users interact with this interface to upload documents in various formats like .txt, .pdf, .doc, and .docx. Once the document is uploaded, the backend handles preprocessing, sentence segmentation, and transformation of input into vectorized representations using NLP techniques.

The backend is powered by Flask, which acts as the primary orchestrator for document routing, model inference, and response rendering. Uploaded files are first converted into plain text using format-specific parsers. Then, a series of NLP preprocessing steps such as tokenization, stopword removal, lemmatization, and POS tagging are applied. These processed sentences are fed into a pre-trained machine learning or transformer model (such as BERT) for classification into NFR categories.

The results are rendered on the frontend in a visually structured layout, grouping sentences under their respective NFR categories like Performance, Security, Usability, etc. Additionally, the system can be extended with visualization features such as pie charts or bar graphs for analytical insights. This modular design allows seamless integration with cloud platforms and APIs for real-time deployment and CI/CD integration.

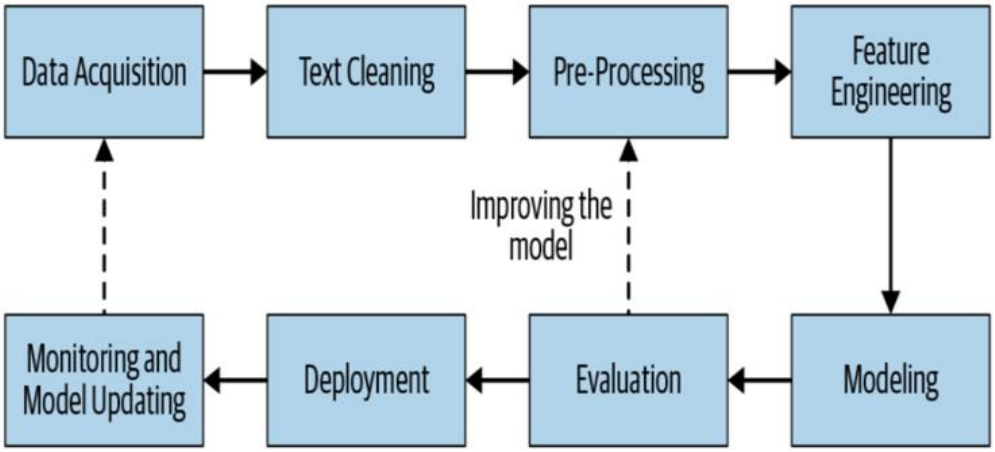


Figure 1: System Architecture

IV. RESULTS

Our best-performing model (fine-tuned BERT) achieved:

- **Precision:** 87.2%
- **Recall:** 85.6%
- **F1-score:** 86.4%

The UI displayed intuitive results, allowing users to clearly see each NFR mapped to a category. The interface grouped the extracted sentences under well-defined NFR classes, such as *Security*, *Performance*, and *Usability*. Each group of NFRs was visually distinct, improving readability and analysis. The system demonstrated consistent performance across documents of varying length and formats.

In experiments involving a **5-fold cross-validation** setup, the fine-tuned BERT model outperformed classical models like **Logistic Regression** and **SVM** in both accuracy and generalization. It was observed that the BERT model was particularly effective at identifying **implicit NFRs** that were not clearly labeled, thanks to its semantic understanding of context. A comparative confusion matrix showed high precision and recall across categories, particularly in identifying **security-related** and **performance-related** statements.

An additional visualization layer was integrated to show the **distribution of NFRs per document**. For instance, in one sample document, the system identified:

- 8 *usability* requirements
- 6 *performance* requirements
- 4 *security* constraints with confidence scores above 80%.

Such analytics help requirement engineers **prioritize system quality attributes** more effectively and accelerate the early stages of the SDLC.

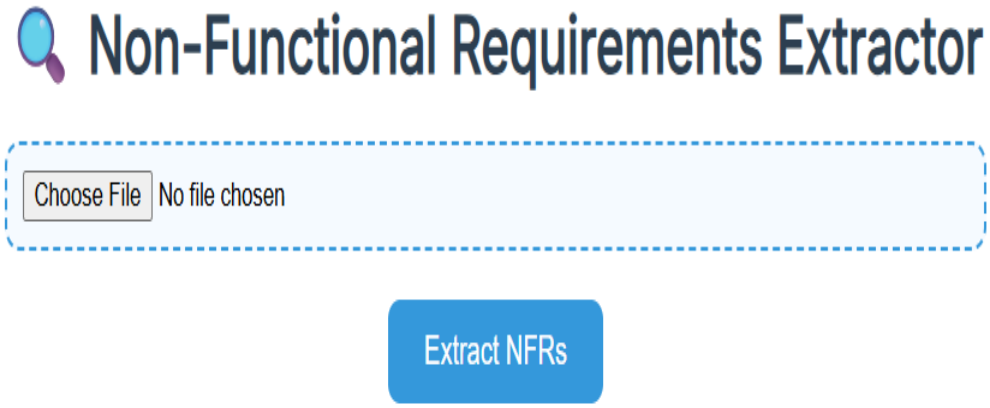


Figure 2 : NFR identifier homepage

Figure 3 : Display of Extracted NFR's based on their categories.

V. CONCLUSIONS

This project demonstrates the feasibility and effectiveness of automating Non-Functional Requirements (NFR) identification from unconstrained software documents using a combination of Natural Language Processing (NLP) and robust Machine Learning (ML) techniques. By transforming unstructured text into semantically enriched vectors and applying state-of-the-art classifiers like BERT, the system accurately identifies and classifies key NFR categories such as performance, security, usability, and maintainability.

The automation of NFR extraction significantly reduces manual effort, improves consistency, and enables faster turnaround times in early-stage software requirement engineering. The system also provides a user-friendly interface for practitioners, making it accessible even to non-technical stakeholders. The model's ability to handle multiple document formats and detect implicit requirements adds to its practical utility.

Beyond improving the quality and efficiency of requirement analysis, this approach lays the groundwork for integrating intelligent NFR detection into modern DevOps pipelines. In future iterations, the system can be enhanced by:

- Expanding the dataset with industry-specific and multilingual NFR samples to improve generalization
- Leveraging more advanced large language models such as GPT-based architectures or fine-tuned domain-specific transformers
- Adding explainability and transparency layers for better traceability of classifications
- Incorporating real-time feedback loops for interactive model improvement

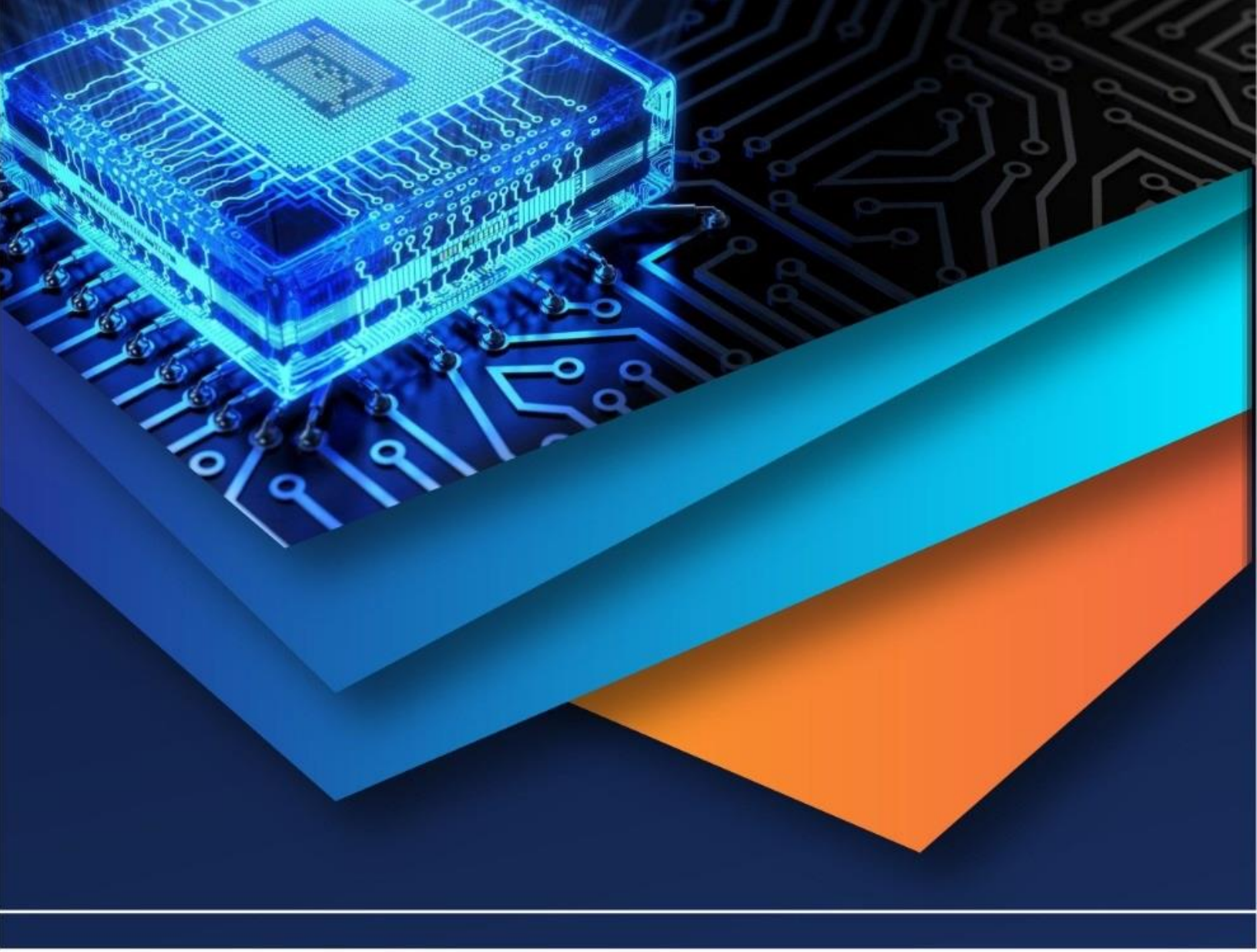
Additionally, deployment on cloud platforms with scalable APIs can enable continuous learning, collaborative annotations, and large-scale enterprise adoption. Ultimately, this project provides a strong foundation for building intelligent software tools that streamline requirements engineering in real-world scenarios.

VI. ACKNOWLEDGMENT

We thank our faculty and mentors who guided us during the development of this project and the contributors of the publicly available datasets that supported our research.

REFERENCES

[1] Glinz, M. (2007). *On Non-Functional Requirements*. 15th IEEE International Requirements Engineering Conference.
[2] Jureta, I. J., Mylopoulos, J., & Faulkner, S. (2008). *Revisiting the Core Ontology and Problem in Requirements Engineering*. IEEE Transactions on Software Engineering.
[3] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv preprint arXiv:1810.04805.
[4] Pedregosa, F., Varoquaux, G., Gramfort, A., et al. (2011). *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12, 2825–2830.
[5] Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media.
[6] NLTK Documentation. Available: https://www.nltk.org
[7] Kaggle Dataset on NFRs. Available: https://www.kaggle.com/datasets
[8] Cleland-Huang, J., Gotel, O., Zisman, A., et al. (2014). *Software Traceability: Trends and Future Directions*. In Proceedings of the Future of Software Engineering.
[9] Mairiza, D., Zowghi, D., & Nurmuliani, N. (2010). *A Systematic Literature Review of Software Requirement Prioritization Research*. In 2010 Asia Pacific Software Engineering Conference.
[10] [Hussain, F. K., Chang, E., & Dillon, T. S. (2012). *Ontology-based recommender systems for requirement engineering*. In Proceedings of the 2012 International Conference on Advanced Engineering Computing and Applications in Sciences.
[11] Hiemstra, D., & Kraaij, W. (1998). *Twenty-One at TREC-7: Ad-Hoc and Cross-Language Track*. In Proceedings of the Seventh Text Retrieval Conference (TREC-7).
[12] Bashir, S., & Qureshi, M. R. J. (2015). *Automatic Extraction of Non-Functional Requirements: A Review*. In Journal of Computer and System Sciences, Elsevier.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY