Volume: 13 Issue: XI Month of publication: November 2025

DOI: https://doi.org/10.22214/ijraset.2025.75418

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 XI Nov 2025- Available at www.ijraset.com

Requirement Specification Semantic Pruning: An NLP Approach for Redundancy Identification

Naimisha Soni[1], Jay Singh[2], Shivendra Singh[3], Asst. Prof. Himani Purohit[4]
Artificial Intelligence & Data Science, Thakur College of Engineering and Technology Mumbai, India

Abstract: Software requirement specifications (SRS) often contain repetitive, ambiguous, or inconsistent requirements, which drives up costs and delays project timelines. Because they mainly rely on syntactic similarity, traditional redundancy detection methods like TF-IDF and Word2Vec have trouble detecting semantic overlaps. This study proposes a semantic pruning framework that uses advanced NLP techniques, with a focus on transformer-based models like BERT, to find and eliminate superfluous requirements from SRS documents. Precision, recall, F1-score, and runtime were used as evaluation criteria to compare several methods, including CountVectorizer, TF-IDF, Word2Vec, and BERT. The findings show that deep learning models outperform conventional methods, which yield high precision but poor recall. Despite having a longer runtime, BERT outperformed Word2Vec with F1 = 0.87 and recall = 0.77.
The outcomes show the effectiveness of transformer-based embeddings for re-dundancy detection and provide a scalable approach to improve SRS quality while reducing the amount of manual review effort.
Keywords: Software Requirement Specifications (SRS), Redundancy Detection, Semantic Pruning, NLP, BERT.

I. INTRODUCTION TO SRS AND NLP

Software Requirement Specifications (SRS) serve as the foundation for software development by defining both a system's functional and non-functional needs. They ensure that the finished product complies with technological limitations and user expectations by fostering a shared understanding among stakeholders, developers, testers, and clients. However, the quality of an SRS document significantly impacts the efficiency and accuracy of the software development lifecycle. Issues such as redundancy, ambiguity, and inconsistency within these documents often result in miscommunication, increased development costs, and delays in project delivery. Redundancy in SRS arises when the same requirement is expressed multiple times using different wording or phrasing. Traditional redundancy detection methods, including manual reviews and keyword-based techniques like TF-IDF and Word2Vec, primarily depend on syntactic similarity. These techniques have difficulty in detecting semantically similar needs with different structures, rendering them unsuitable for real-world, large-scale documents. Recent advances in Natural Language Processing (NLP), notably transformer-based models like BERT, have enabled more detailed contextual interpretation of text. Such models are ideal for capturing semantic equivalence and can significantly enhance redundancy identification in requirements engineering.

This paper provides three main contributions:
1) It proposes a semantic pruning system that uses BERT embeddings to discover redundancy in SRS in an accurate and scalable manner.
2) It compares classical, machine learning, and deep learning methodologies, emphasizing the tradeoffs between accuracy and runtime.
3) It demonstrates how contextual embeddings can reduce manual review time, improving SRS quality and enabling more efficient software development.
4) This work intends to improve the maintainability of SRS documents by removing redundancy through semantic pruning, while also contributing to the larger goal of automating requirement engineering processes.

II. LITERATURE REVIEW

Numerous approaches, including advanced deep learning models and conventional text similarity techniques, have been used to study Software Requirement Specifications (SRS) redundancy identification. The most common kinds are hybrid approaches, machine learning embeddings, deep learning models and conventional similarity-based techniques. The comparative summary of these approaches used for redundancy detection in requirement specifications is presented in Table 1.

Category	Approach / Reference	Core Technique	Strengths	Limitations
Traditional	Bag of Words, CountVectorizer (Choudhary et al., 2019)	Frequency-based vectorization + cosine similarity	Simple, fast, interpretable	Ignores semantics; surface-level overlap only
	TF-IDF (Jha & Verma, 2021)	Weighted term vectors	Well-established, stable	Misses paraphrased or contextually similar sentences
	Jaccard / Levenshtein / Euclidean (Tarawneh, 2020)	Token overlap / edit distance	Lightweight; handles spelling variations	Sensitive to wording changes; fails on semantics
Machine Learning	Word2Vec (Mikolov et al., 2013)	Word embeddings from co-occurrence	Captures semantic proximity	Context-independent; sentence meaning diluted
Deep Learning	BERT (Devlin et al., 2019)	Transformer-based contextual embeddings	Captures deep semantic relationships	Computationally expensive; slower runtime
	Sentence-BERT (Li et al., 2021)	Optimized sentence embeddings	High accuracy, faster than BERT	Domain adaptation required
Hybrid	TF-IDF + BERT (Anand et al., 2024)	Combination of shallow + deep features	Balances speed and accuracy	Added pipeline complexity
	Clustering + BERT (Kici et al., 2021)	Embedding-based clustering	Groups redundant requirements effectively	Requires threshold tuning

Table 1. Literature Review

A. *Traditional Techniques*

Early techniques included statistical and lexical similarity measurements like Bag of Words, TF-IDF and CountVectorizer. These methods use texts as sparse vectors and assess similarity using measures such as cosine similarity or Jaccard similarity (Choudhary et al., 2019). While computationally efficient, these methods are restricted to syntactic overlap and do not capture semantic content. String-based approaches like Levenshtein Distance and Euclidean Distance were also investigated (Tarawneh, 2020). However, these techniques do not identify paraphrased or contextually similar criteria.

B. *Machine Learning Models*

The use of word embeddings revolutionized similarity detection. Word2Vec (Mikolov et al., 2013) maps words into dense vector spaces that preserve semantic links. In requirement engineering, averaging word embeddings for sentences enhanced redundancy detection over TF-IDF. However, Word2Vec's context-independence limits its use for complex SRS statements because a word is given a single embedding regardless of the surrounding text.

C. *Deep Learning Models*

The change to contextual embeddings resulted in models like BERT (Devlin et al., 2019), that captures bidirectional dependencies in text. BERT surpasses Word2Vec and TF-IDF in terms of sentence-level context, making it ideal for SRS redundancy detection. Recent research (Kici et al., 2021; Necula et al., 2023) has demonstrated its advantage in recognizing semantically comparable requirements and contradictions. Variants like Sentence-BERT (SBERT) (Li et al., 2021) improve sentence-level similarity computation.

D. *Hybrid Approache*

Recent research investigates hybrid pipelines that combine traditional efficiency with deep learning accuracy (Anand et al., 2024). For example, clustering approaches (K-Means, Hierarchical Clustering) applied to BERT embeddings improve redundancy grouping, but TF-IDF-based filtering reduces computational cost. Hybrid models strike a mix between accuracy, scalability, and runtime performance, making them intriguing for large-scale SRS analysis.

E. Gaps Identified

Despite significant progress in redundancy detection for Software Requirement Specifications, existing approaches exhibit notable gaps. Low recall is the result of traditional methods like Bag of Words, TF-IDF, Jaccard and Levenshtein, which rely on surface-level lexical similarity and miss paraphrased or contextually equivalent requirements. Word2Vec improved semantic representation but remained context-independent, limiting its effectiveness for complex requirement sentences. Although transformer-based models like BERT have demonstrated strong performance in NLP tasks, their application in requirement engineering is still underexplored, with most studies lacking large-scale comparative evaluations. Furthermore, many works are constrained by small or domain-specific datasets, with limited focus on scalability, runtime efficiency, and real-world integration. Very few studies provide direct head-to-head comparisons across traditional, embedding-based, and deep learning models, leaving trade-offs between accuracy and efficiency largely unaddressed. These gaps highlight the need for a comprehensive, context-aware, and scalable framework that can effectively detect semantic redundancies while balancing performance and computational cost.

III. PROLEM STATEMENT

Software Requirement Specifications (SRS) are critical for defining system behavior and guiding software development. However, these documents often contain semantically redundant requirements—statements that differ in wording but convey the same meaning. Such redundancies lead to ambiguity, inflated effort, and potential implementation errors.
Traditional methods like manual reviews, TF-IDF, or Word2Vec are inadequate for detecting these redundancies due to their focus on surface-level or syntactic similarity. Manual methods are time-consuming and unscalable, while automated ones often miss deeper contextual overlaps.

1) Definition: Semantic pruning refers to the automated identification and removal of redundant requirements based on contextual meaning rather than exact wording.
2) Purpose: The major objective is to make sure every need is distinct and significant in order to improve the quality and maintainability of SRS papers.

IV. METHODOLOGY

The proposed framework introduces an NLP-based semantic pruning system for automatically detecting and eliminating redundant requirements from Software Requirement Specification (SRS) documents. The design emphasizes both accuracy (capturing semantic equivalence beyond surface-level similarity) and scalability (applicable to large industrial SRS repositories). The methodology integrates modern deep learning techniques with traditional similarity measures to allow comparative analysis of their effectiveness.

A. System Architecture

The framework operates in five main stages:
1) Data Gathering
2) Pre-processing
3) Feature Extraction
4) Similarity Computation
5) Redundancy Detection & Pruning
6) Evaluation and Comparative Analysis

Each stage is designed to handle specific aspects of the redundancy detection pipeline, from raw text to actionable insights.

B. Data Gathering

A dataset of requirement specifications was curated from:
1) Open-source repositories (e.g., PROMISE and NASA SRS datasets).
2) Academic case studies from software engineering literature.
3) Industry-provided SRS documents (where available).

Each requirement is treated as an atomic unit. For instance, a sentence such as *"The system should authenticate users before granting access"* is stored as a discrete requirement for analysis. The dataset includes both functional requirements (describing system behaviour) and non-functional requirements (performance, reliability, etc.), since redundancy can appear in both categories.

C. Preprocessing Layer

SRS documents typically contain formatting inconsistencies, abbreviations, and domain-specific terms. To standardize them for NLP models, the following steps are performed:

1) Sentence Splitting: Each SRS document is segmented into requirement-level sentences.
2) Tokenization: Sentences are broken into tokens (words or sub-words) using spaCy/NLTK.
3) Stopword Removal: Common but non-informative words such as "is", "the", and "in" are eliminated.
4) Lemmatization: Words are reduced to base forms (e.g., "computes" → "compute").
5) Normalization: Lowercasing and removal of punctuation, digits, and special characters.
6) Consistency Check: Domain-specific abbreviations are expanded (e.g., "UI" → "User Interface") for improved semantic understanding.

This preprocessing ensures uniform text input, reduces noise, and improves embedding quality.

D. Feature Extraction

The pre-processed requirements are transformed into numerical representations to enable similarity analysis. Three approaches are compared:

1) TF-IDF (Term Frequency–Inverse Document Frequency): Captures the relative importance of the words in the document by representing each requirement as a sparse high-dimensional vector. The scope of TF-IDF is restricted to lexical similarity, despite its interpretability and speed.
2) Word2Vec: Generates dense vector embeddings for words by learning co-occurrence patterns. Sentence vectors are created by averaging word embeddings, preserving word-level semantics better than TF-IDF. However, it lacks sentence-level context.
3) BERT (Bidirectional Encoder Representations from Transformers): A model based on transformers that generates contextualized embeddings for entire sentences. Unlike TF-IDF or Word2Vec, BERT accounts for surrounding context, enabling detection of paraphrased requirements. For example, *"The system shall validate input"* and *"Input must be verified by the system"* receive similar embeddings.

E. Similarity Computation

Once embeddings are generated, pairwise similarity between requirements is calculated.

1) Cosine Similarity: Applied to TF-IDF, Word2Vec, and BERT embeddings to measure angular distance.

$$Cosine\ Similarity(A,B) = cos(\theta)\frac{\parallel A \parallel\parallel B \parallel}{A \cdot B}$$

2) Jaccard Similarity: Between two requirement statements, it calculates the proportion of common tokens to all unique tokens.

$$J(A,B) = \frac{|\ A \cup B\ |}{|\ A \cap B\ |}$$

3) Euclidean Distance: Used as an alternative metric for BERT embeddings.

$$d(A,B) = i = \sqrt{\sum_{i=1}^{n}(Ai - Bi)^2}$$

4) Thresholding: A similarity threshold (e.g., 0.80) is applied; pairs exceeding this threshold are flagged as potentially redundant.

For instance, if two requirements achieve a cosine similarity of 0.85, they are treated as duplicates. Threshold tuning is critical: a high threshold risks missing subtle redundancies, while a low threshold risks false positives.

F. Redundancy Detection and Pruning

The redundancy detection phase integrates clustering and pruning:

- *Pair Identification*: Requirement pairs with similarity scores above the threshold are recognized.
- *Clustering:* Algorithms such as K-Means or Agglomerative Clustering group requirements into redundancy clusters.
- *Pruning Strategy:* Within each cluster, the most complete or unambiguous requirement is retained, while others are pruned. This preserves clarity while reducing duplication.

- *Redundancy Report:* A detailed report is generated, including:
  - Redundant requirement pairs.
  - Their similarity scores.
  - Clustered groups.
  - Recommendations on which requirements to retain.

This stage not only flags redundancy but also provides actionable suggestions for improving SRS quality.

G. Evaluation and Comparative Analysis

The framework's performance is evaluated across multiple algorithms (CountVectorizer, TF-IDF, Word2Vec and BERT) using precision, recall, F1-score and runtime as metrics.

- *Precision:* The percentage of correctly identified redundancies.
- *Recall:* The percentage of real redundancies that are found and identified.
- *F1-score:* A balanced, harmonic mean of memory and precision.
- *Execution Time:* Evaluates computing effectiveness, which is essential for industrial implementation.

Results (from Table 1):

- Traditional methods (e.g., CountVectorizer, TF-IDF) achieved perfect precision (1.0) but low recall (0.25–0.45), missing many semantic redundancies.
- Word2Vec improved recall (0.69) and F1-score (0.81) but remained context-independent.
- BERT achieved the best results with F1-score = 0.87 and recall = 0.77, successfully capturing semantic overlaps. However, it required higher runtime (6 minutes vs <1 minute for TF-IDF).

These findings confirm that while traditional methods are faster, deep learning models provide superior semantic detection, making them better suited for real-world redundancy pruning in large SRS datasets.

H. Technology Stack

The following tools and libraries are used to build the system:

- *Python:* Core programming language
- *Hugging Face Transformers:* For BERT model implementation
- *Gensim:* For Word2Vec embeddings
- *Scikit-learn:* For TF-IDF, cosine similarity, and clustering
- *spaCy / NLTK:* For preprocessing tasks
- *Pandas & NumPy:* For data manipulation
- *Matplotlib & Seaborn:* For visualizing clusters and similarity scores
- *Flask/FastAPI:* For frontend and UI/UX.

V. TECHNOLOGY, TOOLS AND DATASET

The implementation of the semantic pruning framework leverages a variety of open-source tools and libraries across multiple stages—from data preprocessing to advanced semantic analysis and visualization. These tools enable efficient, scalable, and accurate detection of redundant requirements in SRS documents.

A. Programming Language

- Python: Chosen due to its broad support in data analysis, machine learning, and natural language processing. Python is perfect for quick prototyping and deployment because of its extensive developer community, simple syntax and strong libraries.

B. NLP and Text Processing

- spaCy: Used for Tokenization, Lemmatization, Part-of-Speech (POS) tagging and Named Entity Recognition (NER). Efficient for processing large volumes of text.
- NLTK (Natural Language Toolkit): Useful for pre-processing tasks such as stopword removal, stemming, and sentence segmentation.
- Regex (Regular Expressions): For cleaning raw text by removing special characters, digits, or formatting artifacts.

C. Feature Extraction and Embedding Models
- Scikit-learn: Implements TF-IDF Vectorizer, cosine similarity metrics, and clustering algorithms like K-Means. Also supports preprocessing functions like vector normalization and dimensionality reduction (e.g., PCA).
- Gensim: Used to train or load Word2Vec models, allowing vector representation of words based on semantic proximity.
- Hugging Face Transformers: Provides easy access to BERT and its variants for generating deep contextual sentence embeddings.

D. Similarity and Clustering
- NumPy / SciPy: For numerical computations, distance metrics, and matrix operations.
- Scikit-learn: Used for cosine similarity, Euclidean distance, and clustering techniques (e.g., K-Means, Agglomerative Clustering) to group similar requirements.

E. Data Handling and Processing
- Pandas: Enables efficient data manipulation and tabular representation of SRS entries, similarity scores, and redundancy reports.

F. Visualization
- Matplotlib & Seaborn: For plotting similarity score distributions, cluster visualizations, and redundancy graphs.
- Plotly: Interactive visualization for web-based dashboards, enabling dynamic exploration of redundant requirement clusters.

G. Deployment & Scalability
- Flask / FastAPI: For developing a lightweight web interface to upload SRS documents and display redundancy results.
- Docker: Containerizes the NLP framework for consistent deployment across different environments.
- Kubernetes: For orchestrating large-scale deployments with load balancing and auto-scaling support (useful in enterprise settings).

H. Reporting
- Jupyter Notebooks / PDF Generators: Useful for generating formatted reports that include similarity scores, redundancy clusters, and recommendations for pruning.

VI. WORKFLOW OF THE PROPOSED SYSTEM
The overall workflow of the proposed semantic pruning framework for redundancy detection in Software Requirement Specifications (SRS) is illustrated in Figure 1.

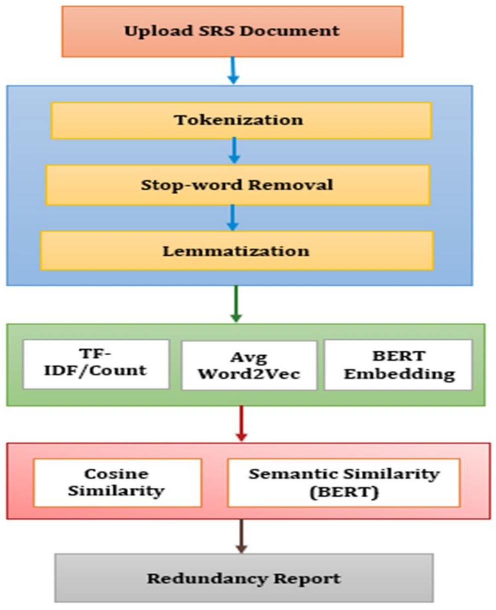


Fig. 1. Redundant Requirements Detection in SRS Document

VII. RESULTS

The proposed semantic pruning framework was evaluated on curated SRS datasets using multiple approaches, including CountVectorizer, Jaccard Similarity, Euclidean Distance, Levenshtein Distance, TF-IDF, Word2Vec and BERT. Each approach was assessed using precision, recall, F1-score and runtime as performance metrics and shown in Table 2.

Algorithm	Precision	Recall	F1-score	Runtime (min)
CountVectorizer	1.00	0.37	0.54	<1
Jaccard Similarity	1.00	0.25	0.40	<1
Euclidean Distance	1.00	0.06	0.10	<1
Levenshtein Distance	1.00	0.03	0.06	<1
TF-IDF	1.00	0.45	0.62	<1
Word2Vec (Avg.)	1.00	0.69	0.81	4
BERT	1.00	0.77	0.87	6

Table 2. Performance Metrics of Algorithms

The number of redundant requirements identified using various similarity algorithms is compared in Figure 2.

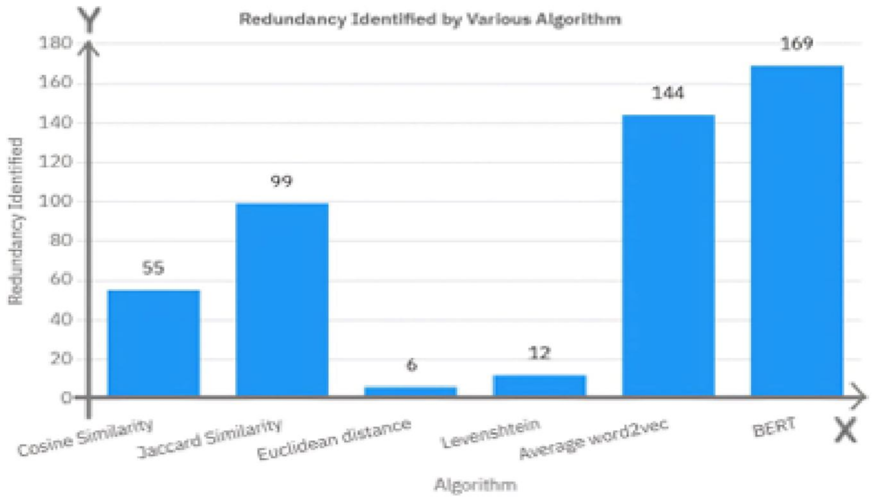


Fig. 2. Redundancy Identified by various Algorithm

The computation time required by different similarity algorithms to identify redundancy is shown in Figure 3.

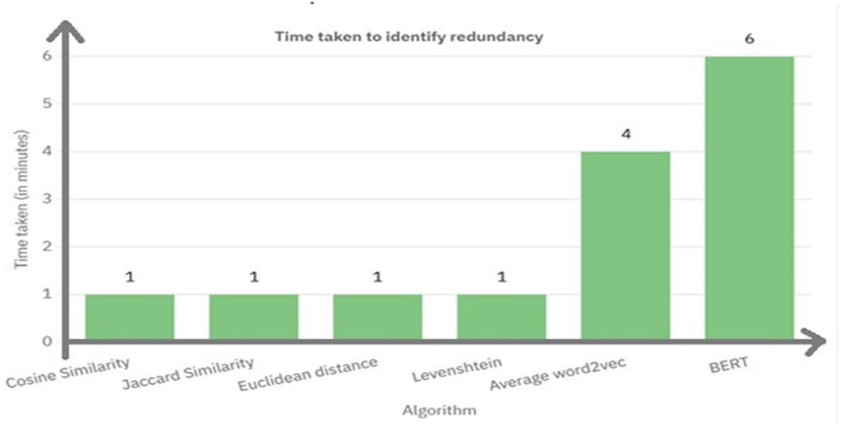
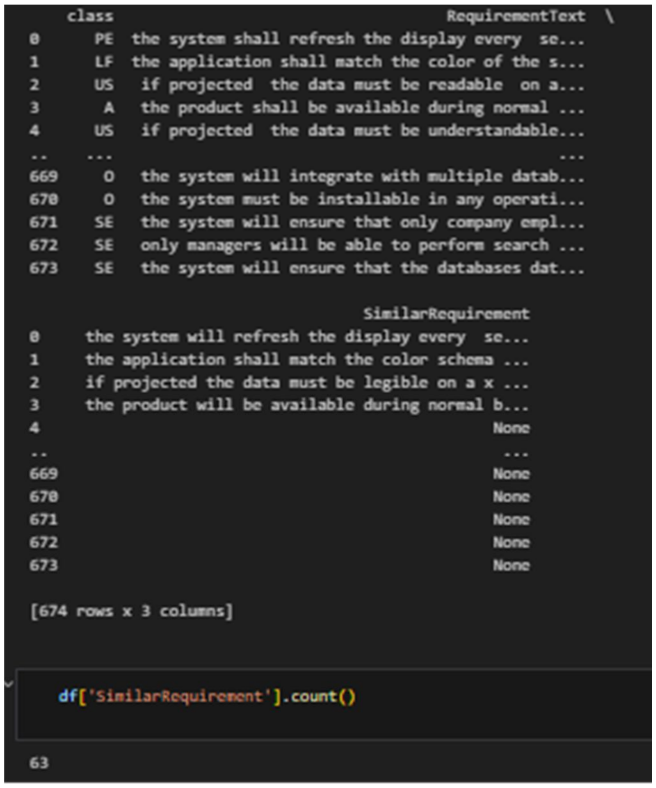


Fig. 3. Time Taken to Identified Redundancy

The sample output showing the identification of redundant requirement pairs in the dataset using Cosine Similarity is presented in Figure 4.



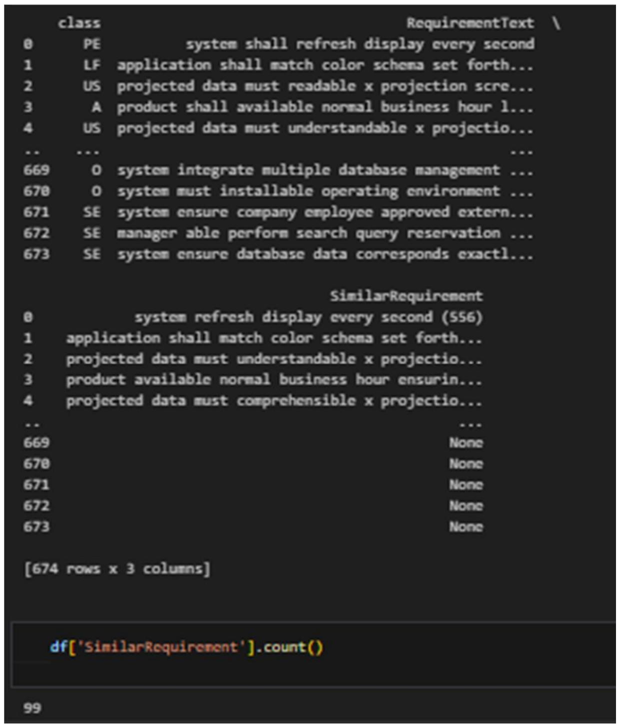
```
    class                                      RequirementText  \
0      PE  the system shall refresh the display every  se...
1      LF  the application shall match the color of the s...
2      US   if projected  the data must be readable  on a...
3       A   the product shall be available during normal ...
4      US   if projected  the data must be understandable...
..    ...                                                ...
669     O   the system will integrate with multiple datab...
670     O   the system must be installable in any operati...
671    SE   the system will ensure that only company empl...
672    SE   only managers will be able to perform search ...
673    SE   the system will ensure that the databases dat...

                               SimilarRequirement
0     the system will refresh the display every  se...
1     the application shall match the color schema ...
2     if projected the data must be legible on a x ...
3     the product will be available during normal b...
4                                               None
..                                              ...
669                                             None
670                                             None
671                                             None
672                                             None
673                                             None

[674 rows x 3 columns]
```

```
df['SimilarRequirement'].count()
```

```
63
```

Fig. 4. Output from Cosine Similarity

The output showing redundancy identification using the Jaccard Similarity algorithm is illustrated in Figure 5.



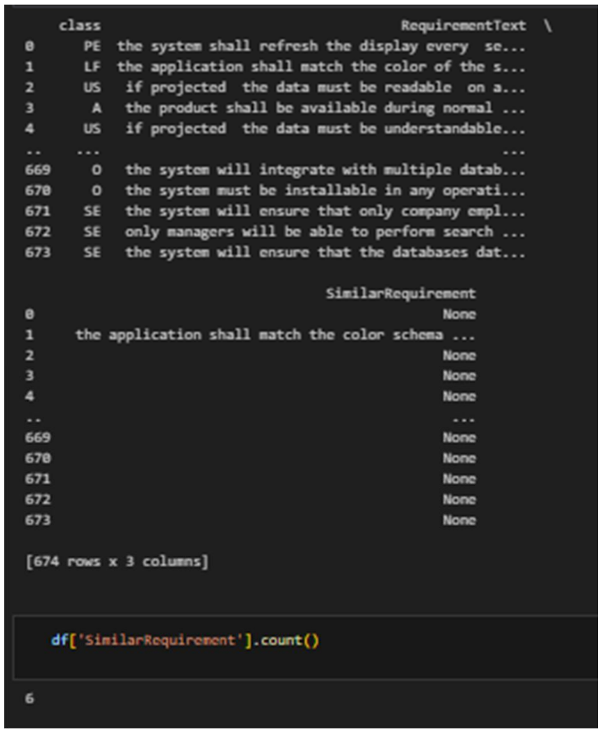
```
    class                                   RequirementText  \
0      PE          system shall refresh display every second
1      LF  application shall match color schema set forth...
2      US  projected data must readable x projection scre...
3       A  product shall available normal business hour l...
4      US  projected data must understandable x projectio...
..    ...                                               ...
669     O  system integrate multiple database management ...
670     O  system must installable operating environment ...
671    SE  system ensure company employee approved extern...
672    SE  manager able perform search query reservation ...
673    SE  system ensure database data corresponds exactl...

                                SimilarRequirement
0           system refresh display every second (556)
1    application shall match color schema set forth...
2    projected data must understandable x projectio...
3    product available normal business hour ensurin...
4    projected data must comprehensible x projectio...
..                                               ...
669                                             None
670                                             None
671                                             None
672                                             None
673                                             None

[674 rows x 3 columns]
```

```
df['SimilarRequirement'].count()
```

```
99
```

Fig. 5. Output from Jaccard Similarity

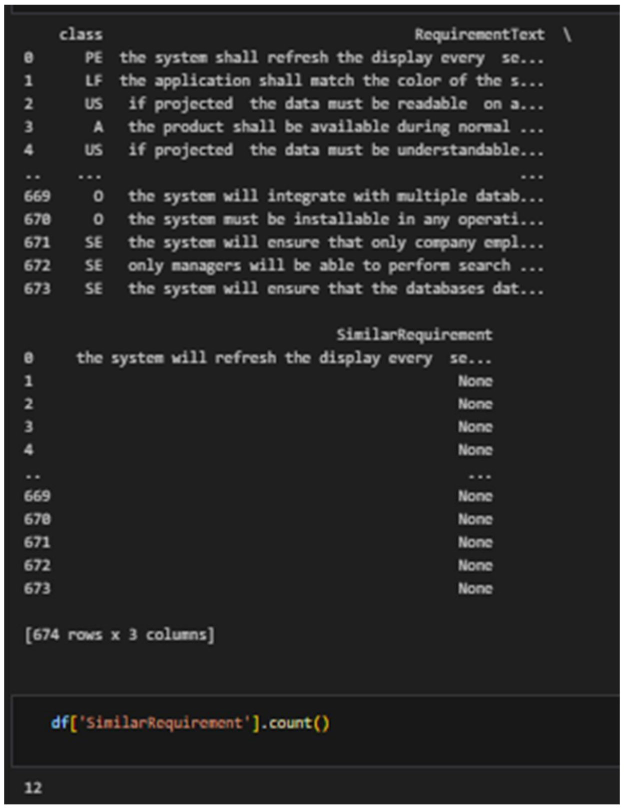
The results obtained through the Euclidean Distance-based similarity measure are presented in Figure 6.



```
    class                                        RequirementText  \
0      PE  the system shall refresh the display every  se...
1      LF  the application shall match the color of the s...
2      US   if projected  the data must be readable  on a...
3       A   the product shall be available during normal ...
4      US   if projected  the data must be understandable...
..    ...                                                 ...
669     O   the system will integrate with multiple datab...
670     O   the system must be installable in any operati...
671    SE   the system will ensure that only company empl...
672    SE   only managers will be able to perform search ...
673    SE   the system will ensure that the databases dat...

                                     SimilarRequirement
0                                                  None
1    the application shall match the color schema ...
2                                                  None
3                                                  None
4                                                  None
..                                                  ...
669                                                None
670                                                None
671                                                None
672                                                None
673                                                None

[674 rows x 3 columns]

df['SimilarRequirement'].count()

6
```

Fig. 6. Output from Euclidean Distance

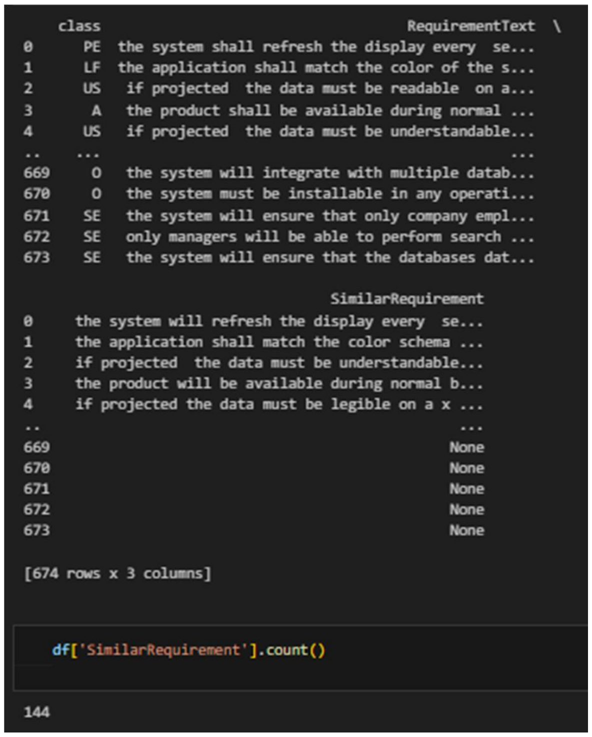
The redundancy detection process using the Levenshtein edit-distance algorithm is depicted in Figure 7.



```
    class                                        RequirementText  \
0      PE  the system shall refresh the display every  se...
1      LF  the application shall match the color of the s...
2      US   if projected  the data must be readable  on a...
3       A   the product shall be available during normal ...
4      US   if projected  the data must be understandable...
..    ...                                                 ...
669     O   the system will integrate with multiple datab...
670     O   the system must be installable in any operati...
671    SE   the system will ensure that only company empl...
672    SE   only managers will be able to perform search ...
673    SE   the system will ensure that the databases dat...

                                    SimilarRequirement
0    the system will refresh the display every  se...
1                                                 None
2                                                 None
3                                                 None
4                                                 None
..                                                 ...
669                                               None
670                                               None
671                                               None
672                                               None
673                                               None

[674 rows x 3 columns]

df['SimilarRequirement'].count()

12
```

Fig. 7. Output from Levenshtein Algorithm

The performance of the Average Word2Vec-based semantic similarity approach is shown in Figure 8.



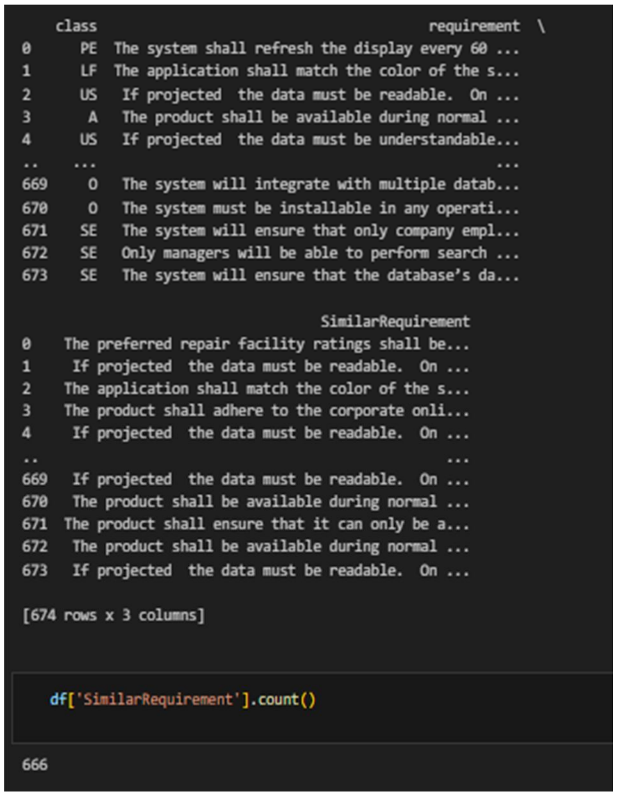
```
    class                                    RequirementText  \
0      PE  the system shall refresh the display every  se...
1      LF  the application shall match the color of the s...
2      US   if projected  the data must be readable  on a...
3       A   the product shall be available during normal ...
4      US   if projected  the data must be understandable...
..    ...                                                ...
669     O   the system will integrate with multiple datab...
670     O   the system must be installable in any operati...
671    SE   the system will ensure that only company empl...
672    SE   only managers will be able to perform search ...
673    SE   the system will ensure that the databases dat...

                                   SimilarRequirement
0    the system will refresh the display every  se...
1    the application shall match the color schema ...
2    if projected  the data must be understandable...
3    the product will be available during normal b...
4    if projected the data must be legible on a x ...
..                                                ...
669                                              None
670                                              None
671                                              None
672                                              None
673                                              None

[674 rows x 3 columns]
```

```
df['SimilarRequirement'].count()
```

```
144
```

Fig. 8. Output from Avg Word2Vec

The implementation output generated using the Gensim library for word embeddings is displayed in Figure 9.



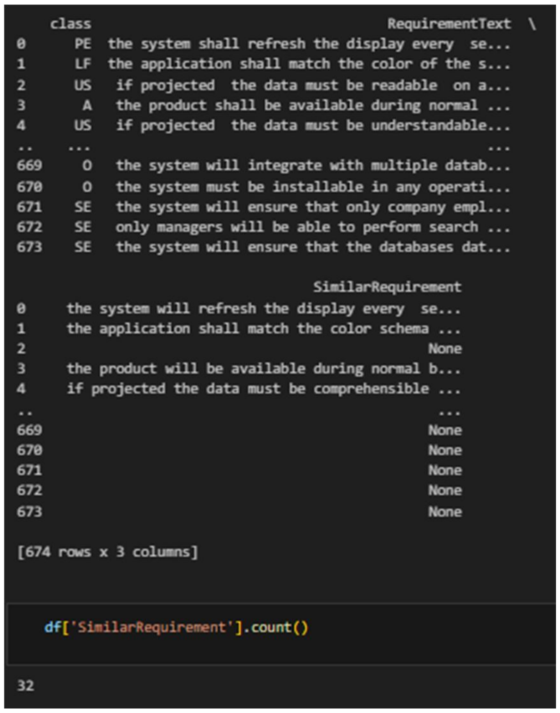
```
    class                                        requirement  \
0      PE  The system shall refresh the display every 60 ...
1      LF  The application shall match the color of the s...
2      US   If projected  the data must be readable.  On ...
3       A   The product shall be available during normal ...
4      US   If projected  the data must be understandable...
..    ...                                                ...
669     O   The system will integrate with multiple datab...
670     O   The system must be installable in any operati...
671    SE   The system will ensure that only company empl...
672    SE   Only managers will be able to perform search ...
673    SE   The system will ensure that the database's da...

                                   SimilarRequirement
0    The preferred repair facility ratings shall be...
1     If projected  the data must be readable.  On ...
2    The application shall match the color of the s...
3    The product shall adhere to the corporate onli...
4     If projected  the data must be readable.  On ...
..                                                ...
669   If projected  the data must be readable.  On ...
670   The product shall be available during normal ...
671  The product shall ensure that it can only be a...
672   The product shall be available during normal ...
673   If projected  the data must be readable.  On ...

[674 rows x 3 columns]
```

```
df['SimilarRequirement'].count()
```

```
666
```

Fig. 9. Output from Gensim (Doc2Vec)

The redundancy detection results obtained using the Universal Sentence Encoder model are presented in Figure 10.



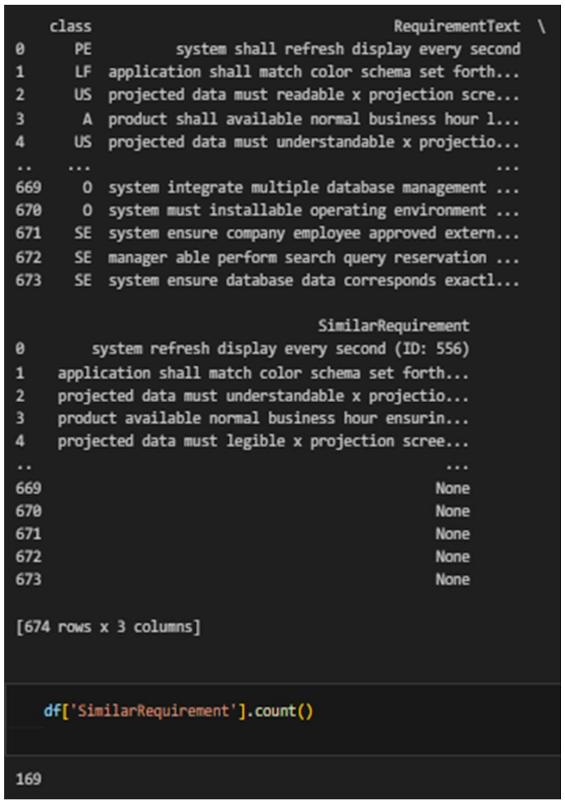
```
    class                                    RequirementText  \
0      PE  the system shall refresh the display every  se...
1      LF  the application shall match the color of the s...
2      US   if projected  the data must be readable  on a...
3       A   the product shall be available during normal ...
4      US   if projected  the data must be understandable...
..    ...                                                ...
669     O   the system will integrate with multiple datab...
670     O   the system must be installable in any operati...
671    SE   the system will ensure that only company empl...
672    SE   only managers will be able to perform search ...
673    SE   the system will ensure that the databases dat...

                                SimilarRequirement
0     the system will refresh the display every  se...
1     the application shall match the color schema ...
2                                              None
3     the product will be available during normal b...
4     if projected the data must be comprehensible ...
..                                               ...
669                                            None
670                                            None
671                                            None
672                                            None
673                                            None

[674 rows x 3 columns]
```

```
df['SimilarRequirement'].count()
```

```
32
```

Fig. 10. Output from Universal Sentence Encoder

The semantic similarity analysis based on BERT contextual embeddings is illustrated in Figure 11



```
    class                                    RequirementText  \
0      PE           system shall refresh display every second
1      LF  application shall match color schema set forth...
2      US  projected data must readable x projection scre...
3       A  product shall available normal business hour 1...
4      US  projected data must understandable x projectio...
..    ...                                                ...
669     O  system integrate multiple database management ...
670     O  system must installable operating environment ...
671    SE  system ensure company employee approved extern...
672    SE  manager able perform search query reservation ...
673    SE  system ensure database data corresponds exactl...

                                SimilarRequirement
0         system refresh display every second (ID: 556)
1     application shall match color schema set forth...
2     projected data must understandable x projectio...
3     product available normal business hour ensurin...
4     projected data must legible x projection scree...
..                                               ...
669                                            None
670                                            None
671                                            None
672                                            None
673                                            None

[674 rows x 3 columns]
```

```
df['SimilarRequirement'].count()
```

```
169
```

Fig. 11. Output from BERT Embedding

The clustering of semantically similar requirements using the K-Means algorithm is shown in Figure 12.

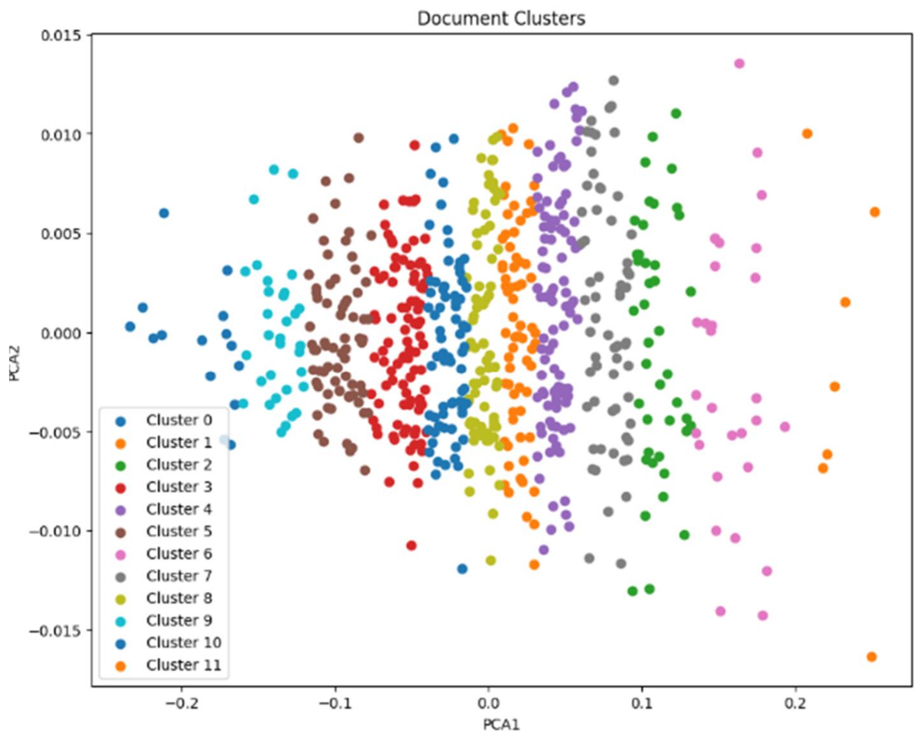
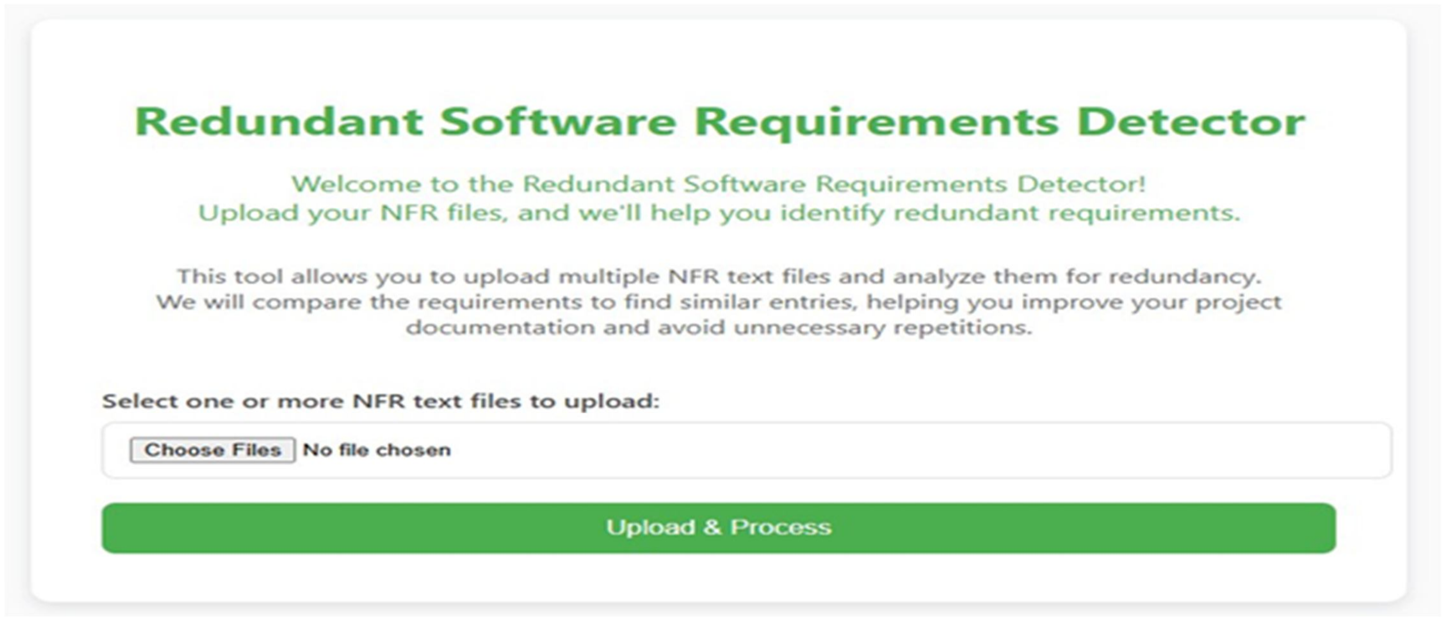


Fig. 12. Clustering using K Means

Traditional methods achieved perfect precision but low recall, indicating that many semantic redundancies were missed. TF-IDF performed moderately (F1 = 0.62), while Word2Vec captured more semantic similarity (F1 = 0.81). BERT achieved the highest level of performance (F1 = 0.87, recall = 0.77), effectively detecting semantic overlaps, though at higher runtime (6 minutes).

In practice, BERT provides the most accurate results for redundancy detection, while Word2Vec offers a strong balance between performance and efficiency. Traditional methods remain fast but are limited to surface-level similarity.

The user interface of the developed web application for uploading SRS documents is presented in Figure 13.


Redundant Software Requirements Detector

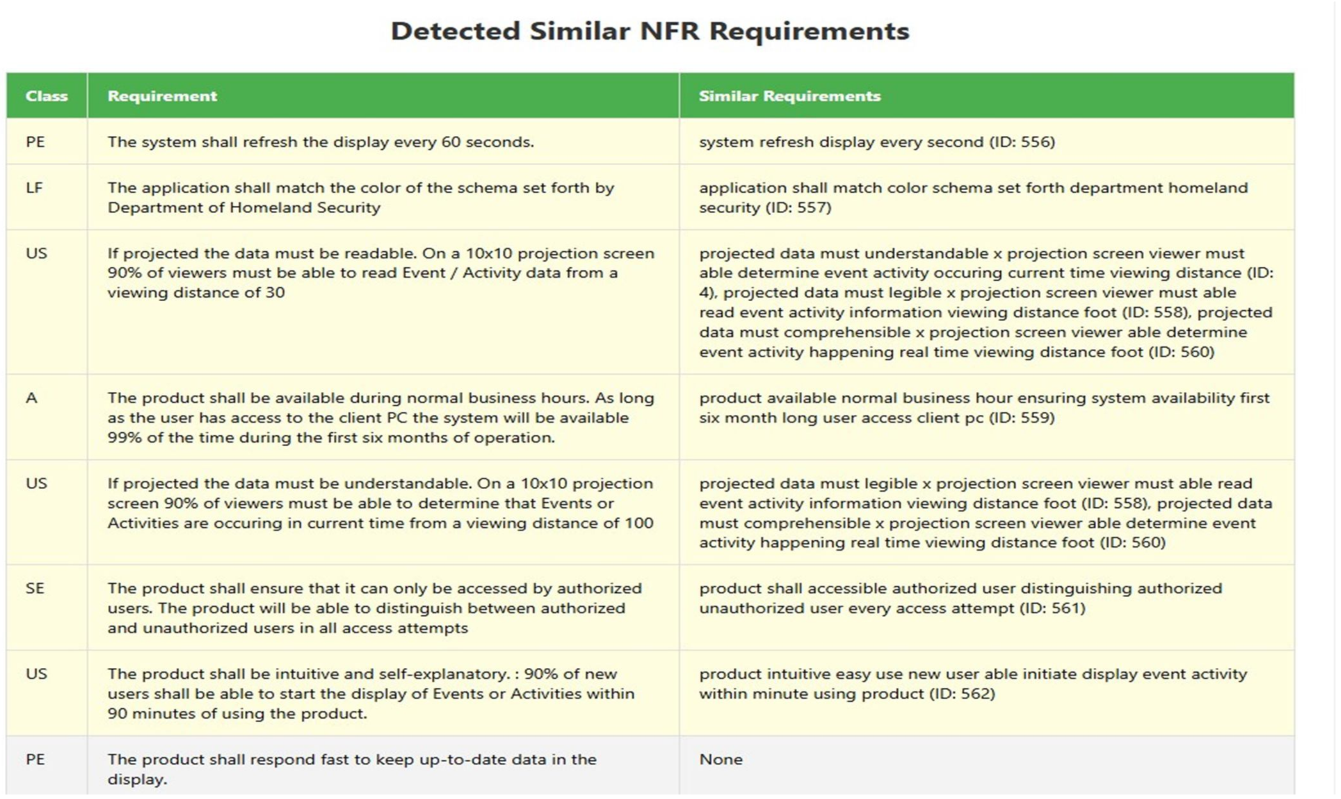
Welcome to the Redundant Software Requirements Detector!
Upload your NFR files, and we'll help you identify redundant requirements.

This tool allows you to upload multiple NFR text files and analyze them for redundancy. We will compare the requirements to find similar entries, helping you improve your project documentation and avoid unnecessary repetitions.

Select one or more NFR text files to upload:

Fig. 13. Frontend used to Upload SRS document

The output screen of the web interface displaying the identified similar requirements is shown in Figure 14.



Detected Similar NFR Requirements

Class	Requirement	Similar Requirements
PE	The system shall refresh the display every 60 seconds.	system refresh display every second (ID: 556)
LF	The application shall match the color of the schema set forth by Department of Homeland Security	application shall match color schema set forth department homeland security (ID: 557)
US	If projected the data must be readable. On a 10x10 projection screen 90% of viewers must be able to read Event / Activity data from a viewing distance of 30	projected data must understandable x projection screen viewer must able determine event activity occuring current time viewing distance (ID: 4), projected data must legible x projection screen viewer must able read event activity information viewing distance foot (ID: 558), projected data must comprehensible x projection screen viewer able determine event activity happening real time viewing distance foot (ID: 560)
A	The product shall be available during normal business hours. As long as the user has access to the client PC the system will be available 99% of the time during the first six months of operation.	product available normal business hour ensuring system availability first six month long user access client pc (ID: 559)
US	If projected the data must be understandable. On a 10x10 projection screen 90% of viewers must be able to determine that Events or Activities are occuring in current time from a viewing distance of 100	projected data must legible x projection screen viewer must able read event activity information viewing distance foot (ID: 558), projected data must comprehensible x projection screen viewer able determine event activity happening real time viewing distance foot (ID: 560)
SE	The product shall ensure that it can only be accessed by authorized users. The product will be able to distinguish between authorized and unauthorized users in all access attempts	product shall accessible authorized user distinguishing authorized unauthorized user every access attempt (ID: 561)
US	The product shall be intuitive and self-explanatory. : 90% of new users shall be able to start the display of Events or Activities within 90 minutes of using the product.	product intuitive easy use new user able initiate display event activity within minute using product (ID: 562)
PE	The product shall respond fast to keep up-to-date data in the display.	None

Fig. 14. Final Results showing Similar Requirements

VIII. FUTURE TRENDS

The proposed semantic pruning framework enables the automatic detection of redundancy in Software Requirement Specifications (SRS) using transformer-based models and natural language processing (NLP). There are still a number of possibilities for further study and improvement, though.

1) *Transformer Models Adapted by Domain:* To enhance contextual understanding within technical domains like healthcare, automotive, and aerospace, future research can investigate refining BERT and its variations (like RoBERTa, DistilBERT, or domain-specific models) on sizable, annotated requirement engineering datasets.
2) *Using Requirement Management Tools:* To facilitate real-time redundancy detection and suggestion during requirement drafting and updates, the framework can be integrated into already-existing requirement management platforms (such as Jira, IBM D OORS, or ReqView).
3) *Hybrid and Ensemble Architectures:* Accuracy and computational efficiency can be balanced, especially for large-scale industrial applications, by combining transformer embeddings with clustering-based techniques or traditional similarity metrics.
4) *Optimization of Scalability and Deployment:* For enterprise-level deployment, future research can use distributed architectures and model compression methods like quantization or knowledge distillation to shorten runtime without sacrificing semantic accuracy.
5) *Multimodal and Cross-Lingual SRS Analysis:* The model's applicability in international software projects will be further improved by expanding it to handle multilingual requirements or multimodal inputs (such as textual, diagrammatic, and tabular specifications).
6) *Human-in-the-Loop Verification:* Model accuracy can be continuously improved and false positives can be decreased by incorporating human review feedback into the pruning pipeline via active learning or reinforcement learning.
7) *Automated Visualization and Quality Metrics:* A thorough SRS quality dashboard that offers real-time visual analytics of redundancy levels, semantic clusters and document quality scores may be a feature of future implementations.

Reliability and efficiency in software development can be greatly increased by extending this research toward domain-specific, explainable, and adaptive redundancy detection systems.

IX. CONCLUSION

This study demonstrates how semantic pruning can improve the quality of Software Requirement Specifications (SRS). While classic approaches like TF-IDF and Word2Vec produce quick and understandable results, they are confined to surface-level similarity and frequently fail to uncover semantically identical requirements. In contrast, transformer-based models, such as BERT, improve redundancy detection by collecting more contextual information.

With an F1-score of 0.87 and recall of 0.77, the experimental evaluation showed that BERT performed better than both conventional and embedding-based methods. This is a feasible option for real-world SRS analysis, especially in large and complex projects, despite requiring more runtime due to its increased accuracy. Word2Vec strikes a decent mix between efficiency and semantic detection, whereas purely statistical techniques are more appropriate for lightweight or small-scale use cases.

The findings emphasize the importance of including contextual embeddings into requirement engineering to save manual review time, eliminate ambiguity, and increase stakeholder communication. Future study will look into domain adaption of transformer models, large-scale industrial validation, and integration with hybrid pipelines to improve accuracy-runtime trade-offs. This study advances redundancy identification, which helps to develop more dependable, maintainable and efficient software systems.

REFERENCES

[1] N. Soni, S. Singh and J. Singh, “Semantic Pruning of Requirement Specifications: An NLP Framework for Redundancy Detection,” TechRxiv, Oct. 2025. [Online]. Available: https://www.techrxiv.org/users/971427/articles/1340011-semantic-pruning-of-requirement-specifications-an-nlp-framework-for-redundancy-detection
[2] Patil, Omkar Sanjay. "Filtering Redundant Requirements in software requirement specifications (SRS) using machine learning (ML)." (2025).
[3] Anand, Aayush and Pandey, Sushank and Shah, Kirtan and Mohnot, Chetan, Optimizing Redundancy Detection in Software Requirement Specifications Using BERT Embeddings (December 02, 2024).
[4] Necula, C., Popescu, M., & Ionescu, L. (2023). NLP Applications in Software Engineering. IEEE Software.
[5] Jha, S., & Verma, P. (2021). Machine Learning for Redundancy Detection in SRS. Journal of Systems and Software.
[6] Tarawneh, M. M. (2020). Using NLP and SVD for Classifying Software Requirements. TERA-PROMISE Dataset Study.
[7] Choudhary, S. et al. (2019). Automated Requirement Extraction Using NLP Techniques. Journal of Software Engineering.
[8] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL.
[9] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv preprint arXiv:1301.3781.
[10] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is All You Need. Advances in Neural Information Processing Systems (NeurIPS).
[11] Jurafsky, D., & Martin, J. H. (2023). Speech and Language Processing (3rd ed.). Draft, Stanford University.
[12] Zhang, Y., & Yang, Q. (2022). A Survey on Multi-task Learning. IEEE Transactions on Knowledge and Data Engineering.
[13] Li, Y., Wang, S., & Liang, P. (2021). Sentence-BERT for Semantic Textual Similarity. arXiv preprint arXiv:2104.09542.
[14] Mavin, A., Wilkinson, P., & Harwood, A. (2020). Big Data Analytics in Requirements Engineering. Requirements Engineering Journal.
[15] Bhatia, A., & Kumar, M. (2019). A Comparative Study of Redundancy Detection in Text Mining. International Journal of Computer Applications.
[16] Kici, D., Baloglu, U., & Aydin, R. (2021). Automatic Detection of Duplicate and Contradictory Requirements Using BERT. Empirical Software Engineering.
[17] Kitchenham, B., & Charters, S. (2007). Guidelines for Performing Systematic Literature Reviews in Software Engineering. Technical Report, EBSE.
[18] Lutz, R. R. (1993). Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems. IEEE Transactions on Software Engineering.
[19] Jaitly, N., Hinton, G.: Vocal Tract Length Perturbation (VTLP) Improves Speech Recognition. In: Proc. ICML Workshop on Deep Learning for Audio, Speech and Language (2013).
[20] Amodei, D., et al.: Deep Speech 2: End-to-End Speech Recognition in English and Mandarin. In: Proc. ICML (2016).
[21] Sharma, R.K., Yadav, S.K.: A Review on Real-Time Speech Translation Systems. In: IEEE International Conference on Computing, Communication and Automation (ICCCA), pp. xx–yy (2021).
[22] Lu, Y., Jiang, Y., Zhang, Z.: Real-Time Multilingual Speech Translation with Deep Learning. IEEE Transactions on Audio, Speech and Language Processing 28, 1123–1135 (2020).
[23] Garg, R., Joshi, P.: Offline Speech Recognition and Translation using Transformer Models. In: IEEE Conference on Computational Intelligence and Communication Networks (CICN), pp. 303–308 (2023).