JCBI

Journal of Computing & Biomedical Informatics Volume 09 Issue 02
ISSN: 2710 - 1606 2025

Research Article
https://doi.org/10.56979/902/2025

Automated Test Case Generation From Natural Language Requirements Using Natural Language Processing

Arooj Fatima[1], Alishba Haider[1], and Summair Raza[1]

[1]Department of Software Engineering, University of Sargodha, Sargodha 40100, Pakistan.
*Corresponding Author: Arooj Fatima. Email: aroojfatimaaa7@gmail.com

Received: July 08, 2025 **Accepted:** August 25, 2025

Abstract: Software testing is a vital process in the assurance of reliability and accuracy of the software systems, but manual testing is a slow, tedious, and error-prone process. The generation of test cases is a promising way, and currently, the existing approaches are frequently constrained by uncertainties in terms of natural language demands, low adaptability, and the lack of scalability. To alleviate such difficulties, this study proposes an end-to-end model based on Natural Language Processing (NLP), contextual embedding of BERT, and machine learning for automatic translation of unstructured requirements into structured and verifiable test cases. The proposed pipeline then involves requirement preprocessing, ambiguity detection, validity assessment, semantic representation, and classification, and then any structured test case documentation is done to assure traceability and completeness. The framework was tested on the DAMIR dataset and compared with the state-of-the-art methods, such as the scatter search, the NLP-based requirements formalization, and the generation of acceptance test cases based on NLP. Experimental outcomes demonstrate that the proposed model had high performance with an accuracy of between 92% and 97% and a maximum accuracy of 96.69, which is much higher than the current methods. Accuracy, recall, and F1-scores also confirmed the soundness of the framework in categorizing valid and invalid requirements. This paper illustrates how NLP-based test automation can be used to scale agile and continuous integration environments, to increase the reliability of the testing process, and to reduce human involvement. It provides the framework for future studies in multilingual requirement processing, domain-specific applications, and integration with large language models to enable greater flexibility.

Keywords: Test Case Automation; Natural Language Processing; Machine Learning; Contextual Embedding; BERT

1. Introduction

Software testing is one of the most crucial stages in the software development life cycle, which provides the system being developed to act in the manner it is supposed to work and in accordance with the expectations of the users. Traditionally, tests has been done manually whereby testers develop and run test cases using written requirements. Although manual testing is able to cover a few functional inconsistencies, it is time consuming, prone to error and limited. With the increasing size and complexity of modern software systems, manual methods have become less and less effective, which has spurred the use of automated testing [1]. In agile and continuous integration system, automated testing is not only faster but also more accurate, repeatable and scalable and thus it is very critical in such systems [2].

One of the basic requirements to the design of test cases is the requirement specification that is usually presented in natural language. These requirements state system requirements, limitations and anticipated behaviors, which serve as the basis on which the implemented system is verified against the needs of stakeholders. Natural language requirements can however be ambiguous, incomplete or inconsistent,

thereby making it difficult to convert them directly into executable test cases. This is where automation fueled by NLP comes in [3].

More recent development in NLP has allowed automated analysis of natural language requirements to derive important information in the form of actors, conditions, preconditions and expected response. These methods not only minimize the human effort but also decrease the ambiguity, enhancing the scalability, and guaranteeing consistency in the generation of tests [4]. However, the current methods of automated generation of test cases have a number of limitations. Search-based methods, like scatter search, are not always able to deal with linguistic complexity. Formalization of requirements using rules are inflexible and are poor at capturing contextual knowledge. Acceptance test generation frameworks do not have a means of detecting ambiguity and are dependent on highly structured input. This results in a requirement of smarter and more adaptable methods that have the capacity to use deep contextual embedding and machine learning to comprehend the requirements.

In this study, we present an end-to-end NLP-based test case generation pipeline based on natural language requirements. Our analysis integrates the current NLP techniques, deep learning embedding, and formal documenting techniques to convert unformatted requirements into executable test cases. The work has the following contributions: Suggest a full end-to-end pipeline to generate test cases, where requirements are processed, embedded with the help of BERT, and classified, and structured test documentation is generated.

To improve the reliability of test cases, we present ambiguity detecting and validity testing mechanisms on the requirement level. We also use BERT-based embedding as they provide better semantic representation of requirements and can be much better classified and extract information than in the case of conventional feature engineering techniques.

The rest of this paper is structured as follows: Section 2 is a review of related work in automated test case generation and NLP-driven requirement analysis. Section 3 explains in detail the proposed pipeline and methodology and the setup of the experiment and datasets to be evaluated in the experiment are shown. The results are discussed in Section 4 and compared with the existing methods. Lastly, Section 5 wraps up the paper and provides future research directions.

2. Literature Review

Automated generation of test cases based on natural language requirements has been an active field of study at the border of software engineering and natural language processing. Initial attempts were mostly based on search and rule-based approaches where requirements were specified in structured models and converted to test cases. Although these methods did offer a certain degree of automation, they tended to be affected by scalability problems and also not to cope with the vagueness of natural language. As sophisticated forms of NLP came to light, scientists started to consider how to directly process natural language requirements, and how to extract test-relevant entities (actors, conditions, and expected outcomes). In recent years, machine learning and deep learning models have been incorporated in this field, where in contextual embedding and semantic insight are used to enhance precision and coverage. Although this has been made, some major issues are still in the form of requirement ambiguity detection, incomplete specifications, and traceability between requirements and test cases which encourages the creation of more intelligent and adaptive models [5].

Olajubu et al [6] introduced a model-based test case generation model on the basis of high-level, domain-specific requirement models. Transforming domain models into testable artifacts and automation of some of the test derivation pipeline to minimize manual effort are stressed in the work. It is remarkable in that it concentrates on domain specific modeling formalisms thereby enhancing traceability between the requirements and tests. The methodology however presupposes the existence of well-defined domain models and does not deal with noisy and entirely natural-language requirements and linguistic ambiguities.

Jorgensen et al [7] is a practitioner-oriented guide to the principles and techniques of software testing, which includes test design, strategies, and quality characteristics. Although not a research contribution to automated generation, the book offers the precursors (test oracles, coverage criteria and test design heuristics) on which subsequent automation research is based. The fact that it offers a wide range of

coverage provides valuable assessment metrics and benchmark practices to researchers developing automated test-case pipelines.

An introduction to the canonical textbook to software testing by Ammann and Offutt et al [8] formalizes much of the knowledge applied in research on automated test generation (e.g., fault models, combinatory testing, mutation testing). Their strict exposition of test adequacy, and formal test criteria, are frequently a source of theoretical support on empirical research. The role of the book is not experimental but conceptual because it educates metrics of evaluation and why some automated strategies are more preferable to follow.

In their proposal to bridge model-driven engineering and requirements engineering, Aysolmaz et al [9] suggested a semi-automated method to use business process models in the creation of natural-language requirements. The methodology helps to attain regular and formal requirements creation that may enhance downstream testability and traceability. But since it is semi-automatic, and model-dependent, it relies on the availability of proper business process models and does not directly address raw and unstructured natural-language requirements or ambiguity resolution.

The USLTG by Hue et al [10] represented the method of changing use cases into test cases generated automatically. USLTG concentrates on the acceptance tests on the system level, by deriving test sequences that can be executed by interpreting use-case descriptions. The technique enhances traceability and lessens manual intervention of use-case-based testing. The drawbacks are that it requires the quality of use-case documentation and is not tested across the board of targeted systems.

Feng et al [11] CodeBERT introduced CodeBERT, a large pre-trained source code/natural language model that is trained to learn to learn joint representations across programming languages and natural text. Even though CodeBERT is mainly designed to handle code understanding and generation tasks (e.g., code search, summarization), the embedding and paradigm of pre-training are very applicable in the context of test generation research that needs to reconcile natural-language needs and code-level artifacts. The model is highly transferable, but needs to be adapted into requirement-to-test tasks.

Modonato et al [12] integrated dynamic symbolic execution, machine learning and search-based testing to generate test cases of object-oriented classes automatically. The hybrid strategy uses symbolic exploration to cover the paths, ML to inform selection and search heuristics to maximize efficiency. It has shown encouraging performance in the testing of classes but is restricted to some programming-language idioms and not evaluated on large and highly polymorphic code bases.

Mustafa et al [13] reviewed the existing literature on automated generation of test cases based on requirements and synthesized techniques, datasets, and evaluation patterns. Their SLR is a synthesis of the landscape with its gaps that include ambiguity, absence of standard benchmarks as well as the minimal use of deep contextual models at that time. The review offers a helpful taxonomy and inspires the research directions with NLP progress and testing requirements integration.

Surveying machine-learning methods in software testing automation, Salam et al [14] classify such approaches into supervised, unsupervised, and deep learning algorithms used in tasks such as test-case prioritization, test-case generation, and fault prediction. The survey highlights the increasing use of ML and highlights common difficulties; lack of data, generalizability, and assimilation into software engineering processes. It suggests more generous standards and end-to-end testing--just the hole our pipeline will focus on.

Cheema et al [15] suggested a natural language interface to create data-flow diagrams by relying on the web-extraction methods that focus on the automated generation of artifacts using the natural language. Even though the focus is on drawing diagrams instead of on constructing test-cases, the paper can be useful since it shows how to obtain structured models based on narrative descriptions, which can be transformed into test-generative actors, conditions, and steps. This approach is based on domain-specific extraction rules and therefore is problematic in being transferred to a wider variety of requirements.

Lim et al [16] suggested an NLP based unified boilerplate model, which combines the Rupp and EARS templates to analyze sentences and minimize the ambiguity to extract test case information. Three datasets of the PURE repository were tested with the model, where it was moderately successful in dealing with positive requirements but failed with negative requirements. One of the major drawbacks is the ignored negative requirements and lack of spelling or synonym checkers as they result in challenges in making a distinction based on actors and conditions.

Kumar et al [17] created an automatic refine-based test case generation method of using student codes and compilers and large language models (LLMs). The system is enhanced by compiler feedback and test case-student error alignment by Chain-of-Thought refinement, which is improved through iterative refinement. Its only major disadvantage though is dealing with low scoring student code, or with buggy student code, where generated test cases might not still model realistic execution behavior.

Prabhu et al [18] proposed an NLP-related method to identify important information in user stories and categorize it in three types, Arrange, Act, and Assert (AAA) with the help of machine learning. This will enhance prioritization and efficiency of test case classification that is normally a manual one. However, its dependence on manual keywords and unigrams can be easily subject to overfitting, poor generalizability and restrict validation in the real-life setting.

Gröpler et al [19] introduced a semi-automated NLP-based system, which then translates the textual requirements to UML sequence diagrams to produce test cases. The method offers explainable models but does not take into account a prioritization process and ambiguity management. In addition, it has not been experimented with large or diverse datasets, which restricts its usage to wider contexts.

Liu et al [20] offered a scatter search technique of test case generation that included various execution paths in NLP programs. The approach decreases the test cases as well as the time taken in the coverage. It is however not evaluated on large-scale, real-world NLP systems and modern ML techniques are not incorporated, and the questions of scalability and adaptability remain open.

And Wang et al [21] proposed Use Case Modelling to produce System-Level Acceptance Tests Generation (UMTG) to generate test cases based on natural language requirements. Although it enhances traceability and saves the user effort, the methodology needs partial manual intervention and domain-specific models, which restricts complete automation. It has been mostly validated on embedded systems and therefore casts doubt on its extrapolation to other software applications. Li et al [22]proposed one such approach whereby the natural language test steps can be grouped and aligned with reusable API methods to enhance efficiency in automation of the test. Although this clustering strategy has shown the possibility of re-use and the minimization of effort, it does not assess flexibility in other areas or other types of tests, and the generalizability is thus not addressed.

A test case management strategy that incorporates requirements-based and risk-based prioritization was described by Lawana et al [23]. The approach saves on expense and time by concentrating on the most critical functionality and the aspects that are risk prone. Nevertheless, limitations in the scope of the study, such as restricted to C-language programs, limits the applicability of the study in general and a unilateral focus on risk areas may lead to the neglect of crucial functionalities.

Alagarsamy et al [24] proposed the A3 Test, a deep learning model which takes into account assertion knowledge in generating more accurate, reliable, and executable test cases. It is more efficient and covers than in the past. It has weaknesses in the analysis of failures, in sustaining accuracy under real-world conditions, and interoperability with modern software systems despite these strengths, as it depends intensely on existing data sets.

Across all studies, several common gaps and limitations emerge. Most notably, there is a lack of comprehensive solutions that jointly address the following core issues, as very few systems actively incorporate semantic disambiguation or contextual refinement techniques to handle vague or conflicting natural language statements. Although some research has focused on prioritization based on risk or keyword salience, a universally adaptable, context-aware prioritization framework is still absent. The reviewed literature reflects a rich spectrum of approaches and innovations that continue to shape the test case generation landscape. The limitations in existing approaches underscore the need for an integrated, end-to-end framework that combines the strengths of NLP, deep learning, and prioritization.

Table 1. Summary of Existing Study with details

Ref	Year	Models/Technique	Dataset	Results (Accuracy/Performance)	Research Area
[6]	2015	Model-driven → domain-specific model transformation	Domain-specific requirement models	N/A (paper-focused on approach & traceability)	Model-driven test generation

[7]	2021	Comprehensive testing techniques (book)	N/A (textbook)	N/A (conceptual/practical guidelines)	Software testing foundations
[8]	2016	Formal testing criteria; coverage & adequacy	N/A (textbook)	N/A (theoretical foundations)	Testing theory & metrics
[9]	2018	Semi-automated natural-language reqs from business process models	Business process models / generated NL docs	Improved consistency; manual refinement required	Requirements engineering / NL generation
[10]	2019	Use-case → automatic test case transformation	Use-case descriptions	Reported improved system-level acceptance test coverage	Use-case-driven test generation
[11]	2020	Pretrained transformer for code + NL (CodeBERT)	Code + NL corpora	Strong transfer for code/NL tasks (benchmarks in original paper)	Pretrained models for code & NL
[12]	2020	Hybrid: Dynamic Symbolic Execution + ML + Search-based Testing	Object-oriented programs (classes)	Efficient class-level test generation (experimental)	Hybrid automated test generation
[13]	2021	Systematic literature review (techniques taxonomy)	Multiple published studies	N/A (synthesis of field; gaps identified)	Survey / SLR on automated test generation
[14]	2022	Survey of ML techniques in testing automation	Literature corpus	N/A (catalog & analysis)	ML in testing automation
[15]	2023	NL interface + web extraction → data-flow diagrams	Web-extracted corpora/domain data	Demonstrated feasibility; domain-dependence	NL → structured artifact extraction
[16]	2024	NLP-based unified boilerplate (Rupp + EARS templates)	PURE repository datasets	Moderate success on positive requirements, weak on negative ones	NLP-based requirement parsing
[17]	2024	Iterative refinement with LLM + compiler feedback	Student codes	Improved alignment with student errors	Code-based test generation
[18]	2024	NLP + ML for AAA classification	User stories	Efficient prioritization; overfitting issue	Test case prioritization
[19]	2021	NLP → UML sequence diagrams	Requirement datasets	Semi-automated; untested scalability	Requirement modeling
[20]	2019	Scatter search strategy	NLP programs	Reduced cases and time; lacks scalability evaluation	Search-based testing

[21]	2022	UMTG (Use Case Modeling)	Embedded systems	Improved coverage; partial manual effort needed	Acceptance test generation
[22]	2020	Clustering NLP test steps → API mapping	Not specified	Improved efficiency, limited generalizability	Test case clustering
[23]	2024	Risk-based + requirement-based prioritization	C-language programs	Reduced cost/time; limited to C programs	Test case management
[24]	2024	A3 Test (Deep Learning + assertion knowledge)	Pre-existing datasets	Improved correctness and coverage; lacks failure analysis	Deep learning for test generation

3. Methodology

This study presents a full-fleet NLP- based pipeline in generating automated test cases based on natural language specifications. This methodology is a combination of preprocessing, information extraction, deep contextual embedding, machine learning classification, and structured test case generation. The general outline of the pipeline of the suggested approach is shown in Figure 1.

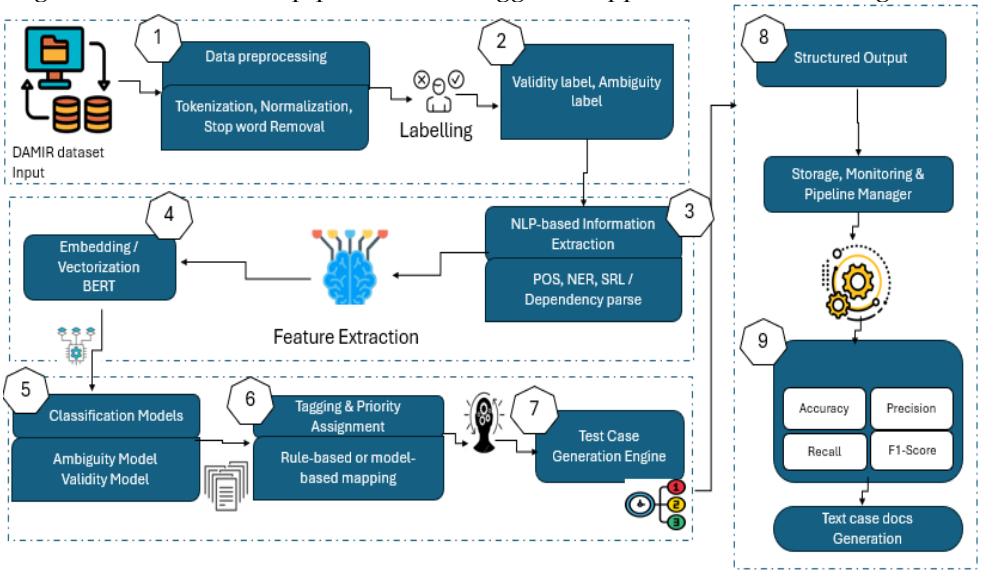

Figure 1. Methodology

3.1. Dataset

To test our proposed framework, we make use of the DAMIR dataset that is available publicly on its GitHub repository (https://github.com/SNTSVV/Anaphoric-Ambiguity/blob/main/Datasets/DAMIR.xlsx). The data set has been purposely designed to be able to record ambiguous and anaphoric distinguishing reference in natural language requirement statements and is, therefore, very susceptible to studies in automated requirement analysis and test case generation.

The DAMIR data is a collection of requirement sentences that are tagged with various layers of linguistic and semantic data. The requirements have been marked as ambiguous, valid, and having anaphoric references. This framework enables the researchers to study the interpretation process of ambiguous phrases by the NLP systems, and the processing of negations and clearing up of references with antecedents. The data contains both positive and negative requirements, and thus models are subjected to various linguistic forms occurring in actual specifications in the real world. The other essential feature of DAMIR is that it is devoted to the ambiguity detection. Unclear requirements are a significant problem of software engineering as they are frequently misinterpreted, resulting in inconsistency of tests and increased development expenses. The dataset also gives an effective reference point when considering

ways of detecting and repairing such cases by providing explicit annotations of ambiguous and non-ambiguous statements ahead of test case generation.

DAMIR has some limitations even though it is useful. Although it offers a dense stock of annotations, the data is of medium size and probably does not reflect the diverse nature of the requirement styles that are possible in large industrial projects. Most of the requirements in the dataset are comparatively short as well, and this may not represent the complexity of multi-sentence or domain-based specifications. However, DAMIR is also among the few publicly accessible datasets directly focused on the issues of ambiguity and anaphora in requirements, and it provides a fundamental basis to develop and test NLP-based methods of automated test case generation.

3.2. Requirement Input and Preprocessing

The initial step involves importing a raw text file containing software requirements, and each line represents a single functional or non-functional natural language requirement. These are loaded into a structured format using the Pandas library. Informally, a requirement sentence is to be represented as a sequence of tokens as equation (1).

$$S = \{w_1, w_2, \dots, w_n\}, \quad w_i \in V \tag{1}$$

Where V is the vocabulary of all tokens. Preprocessing transforms S into a cleaned token sequence S for further analysis. Two preprocessing operations are performed on natural language requirements:

Validity labelling: It checks the validity of the requirement and assigns binary labels (1 = valid, 0 = invalid) to each requirement, which trains the validity classification model.

Ambiguity labelling: If any ambiguous term appears in a requirement, it is labelled as ambiguous (1), otherwise non-ambiguous (0), and it is used to train the ambiguity classification model.

The cleaned and labelled data serve as the input for both machine learning classification and test case generation [25].

3.3. NLP-Based Information Extraction

After the requirements are preprocessed, the NLP-Based Information Extraction module analyses the requirements and extracts all the critical components, such as the actors, test actions (steps), preconditions, and post conditions (expected outcomes) that are needed to generate a complete and traceable test case. The named entity recognition (NER), syntactic parsing, and rule-based logic are combined in this process to transform natural language requirements into structured information to generate test cases. Actors are identified by applying spaCy's Named Entity Recognition (NER). If no actor is found, then dependency parsing is used to extract the grammatical subject of the sentence. In the next step, the model extracts the test Step Extraction (actions). SpaCy's sentence segmentation mode divides each requirement into individual sentences, which typically reflects either an input (what the user does) or a system response (what the system should do) [26]. The need of S can be formally translated into as equation (2).

$$I(S') = \{A, C, P, R\} \tag{2}$$

Where I represents the extraction function.

In the next step, Preconditions must be true or completed before the test, and post conditions must be fulfilled after the test. By applying dependency parsing and semantic role labelling, the system recognizes conditional expressions such as if, before, when, etc., as preconditions and consequence expressions such as then, as result, etc., for post conditions. For complex sentence structures, the NLP parser breaks them into smaller, manageable parts and uses coordinating conjunctions (e.g., and, or) to identify multiple actions or conditions. After extracting all relevant information for generating test cases, it is passed to the Test Case Generation model to perform formatting and documentation [27].

3.4. Embedding Requirements for Using BERT

Traditional NLP feature extraction methods fail to capture contextual relationships between words. For instance, the word "login" may have different meanings depending on whether it is used as a user action or part of an authentication condition. Bidirectional Encoder Representations from Transformers (BERT), developed by Google, addresses this issue by using deep bidirectional transformers to model both the left and right contexts of every word in a sentence simultaneously. These features enable a better understanding of context and help to understand the meaning behind the words [28], which is critical for tasks such as validity detection, ambiguity prediction, and classification computed as equation (3).

$$E(S') = \text{BERT}(S') \in R^d \tag{3}$$

Where d is the embedding dimension.

3.5. Machine Learning-Based Requirement Classification

In approach, Machine Learning-Based Requirement Classification plays a key role in organizing the requirements into meaningful categories and determining their execution priorities. This ensures that the test cases are relevant and more risk-aware, and properly prioritized for testing software efficiently. The primary goal is to label requirements based on their content, importance, and relevance to different quality attributes of software. The classification focuses on two aspects. One of them is the type of Requirement, e.g, Functional, Security, Performance, Optional, or Uncategorized. And the second is Priority Level, e.g, High, Medium, or Low, based on the criticality of the action described in the requirement. This can be characterized as binary classification problem as equation (4).

$$f: E(S') \rightarrow \{0,1\} \quad (4)$$

Where 0 denotes an invalid requirement and 1 denotes a valid requirement [29].

This information is integrated into the final test case output and embedded in the generated .docx test documentation, helping stakeholders understand both the type and urgency of each test case.

- Test Planning and Execution: High-priority test cases can be scheduled for earlier testing cycles.
- Risk-Based Testing: Teams can focus on critical system areas like security and performance.
- Traceability and Coverage: Tags help ensure that all types of requirements are accounted for in the test plan.
- Improved Reporting: Prioritized and categorized test cases make QA metrics more meaningful and actionable.

This enhances the traceability, scalability, and usefulness of the entire test case generation process [30].

3.6. Structured Test Case Document Generation

The final phase involves transforming the enriched and classified requirements into formal test cases, saved in .docx format. Each test case contains Requirement Text, Test Case ID, and Traceability Link, Predicted Validity and Ambiguity, Test Case Category and Priority, Actors, Preconditions, Test Steps, Expected Results, Execution Status (default: "Ready to Execute"), and each component is populated using the outputs of earlier stages, ensuring high semantic alignment and readability. When the requirements have been classified and validated, they are converted into formal test cases made up of:

Test Case ID (TID)

Preconditions (C)

Steps (P)

Expected Results (R)

Officially, a generated test case TC can be represented as equation (5).

$$TC = \{TID, C, P, R\} \quad (5)$$

3.7. Pipeline Management and Execution

The entire system is extracted by a main function, which acts as the control center. It loads the dataset and trains the classification models. Iterates through each requirement, processes it, makes predictions, generates a corresponding test case, and then compiles all test cases into a single .docx file. The approach provides a comprehensive, automated pipeline for transforming unstructured natural language software requirements into structured, high-quality, and executable software test cases. It leverages modern NLP and machine learning techniques and ensures semantic understanding, error detection, and traceability and minimizes manual effort in test case automation and work to enhance the software testing efficiency and reliability [31].

3.8. Evaluation Measures

Machine learning metrics, software engineering metrics, and requirement-specific metrics are needed to evaluate automated tests case generation. These research measures are evaluated using the following measures [32]:

3.8.1. Accuracy

Accuracy is the percentage of the requirements that are correctly categorized (valid or invalid) (of all requirements) computed as equation (6).

$$\text{Accuracy} = \frac{\text{True Positives}(TP)+\text{True Negatives}(TN)}{\text{True Positives}(TP)+\text{False Positives}(FP)+\text{True Negatives}(TN)+\text{False Negatives}(FN)} \quad (6)$$

Where TP = true positives, TN = true negatives, FP = false positives, and FN = false negatives. High accuracy also means that most of the requirements are validated by the classification module.

3.8.2. Precision

Precision is used to measure the accuracy of the positive predictions (requirements that are classified to be valid) computed as equation (7).

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)+False Positives(FP)}} \quad (7)$$

This is a very important measure to minimize the incorrect requirements utilized in generating the test cases.

3.8.3. Recall

Recall (or sensitivity) indicates the percentage of real valid requirements that were successfully recognized computed as equation (8).

$$Recall = \frac{\text{True Positives}(TP)}{\text{True Positives}(TP)+False\ Negatives\ (FN)} \quad (8)$$

The increased recall will guarantee that there are fewer valid requirements that are missed when generating test cases.

3.8.4. F1-Score

The harmonic mean between Precision and Recall is referred to as the F1-Score. It offers a weighted value when a false positive and a false negative are equally essential. To calculate the F1 Score, the formula as equation (9).

$$\text{F1-Score} = 2\frac{(\text{Precision}.\ \text{Recall})}{(\text{Precision}+\ \text{Recall})} \quad (9)$$

4. Results

The results of the experiment performed in the framework of the suggested methodology offer important clues regarding the efficiency of applying NLP tools to automatically create use case diagrams based on text requirements. The assessment of the system performance was carried out with the help of the DAMIR dataset, which has various and multifaceted requirements statements. To confirm the results, several evaluation measures such as accuracy, precision, recall and F1-score were used. The results have shown that the combination of tokenization, part-of-speech tagging, and semantic role labeling is effective in the detection of actors and use cases resulting in more accurate diagram generation. In addition, compared to current methods in the literature, the offered framework shows a competitive level of performance, which emphasizes its possibilities of being adopted practically in software requirement engineering.

These bar graphs represent the rate at which each of the pronouns occurred in the data set. These and they are the most common pronouns followed by he, it's and so on. The prevalence of the use of collective/possessive pronouns suggests that the focus of this dataset is on the mention of the organizations or groups, which are known to be vaguer. This kind of knowledge is effective in pointing out those pronouns that cause more problems in solving co-reference resolution problems that is shown as figure 2.

This chart shows that all the pronoun resolutions can be said to either be correct, incorrect or inconclusive. The skew of the data is so to the inconclusive and this is indicative of the fact that there are so many pronouns that can actually be answered to a definite antecedent. This asymmetry is relevant in model training that suggests that systems must learn to deal with, not only with simple cases, but with ambiguity and uncertainty. It also implies that the information is supposed to generate stress-tests to pronoun resolution systems that is shown as figure 3.

This graphical view is a comparison between the Ack (acknowledged) and Unack (unacknowledged) occurrences. The dominating prevalence of Unack highlights the fact that such explicit recognition is not that widespread in such circumstances. This skew means that there are a great number of candidate antecedents that were not specifically mentioned in the text and this complicates the problem of getting the pronouns right. The imbalance is also noteworthy in case the research must balance classes before applying machine learning models that is shown as figure 4.

This histogram shows the location of the pronoun in the sentence, where it is most likely to be located, by index of words. The pronouns usually occur in the middle of the sentence and not in the beginning of the sentence or the end of the sentence. It is a sign of regular patterns in natural language where pronouns

usually follow an already established antecedent in the sentence. With model design, that substantiates the importance of context window size: models must look at both the preceding and the following pronoun to identify references in an appropriate way that is shown as figure 5.

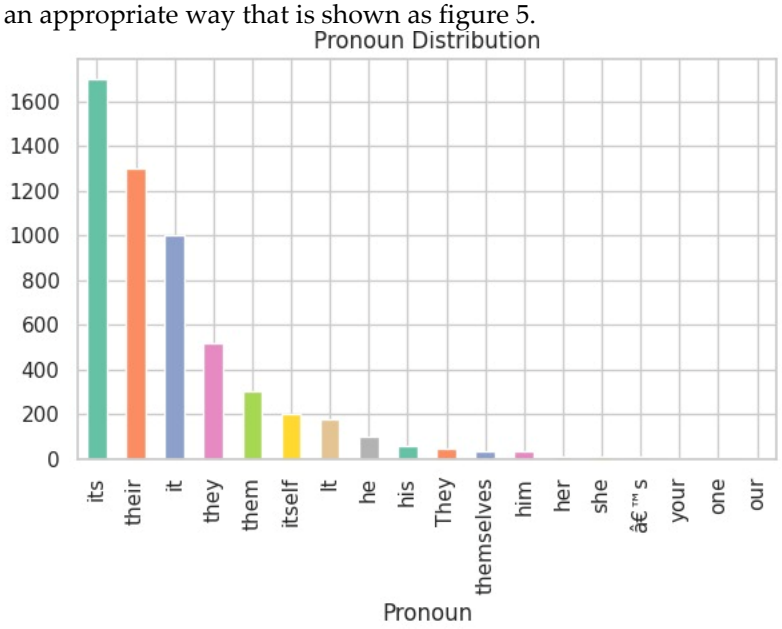

Figure 2. Pronoun Distribution

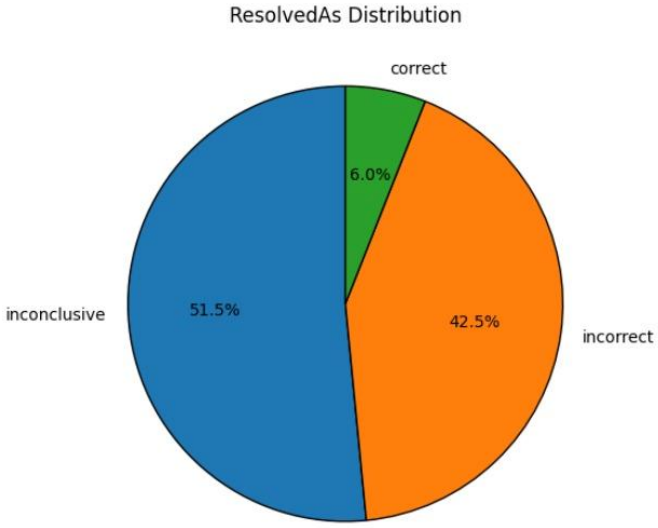

Figure 3. Resolved As Distribution

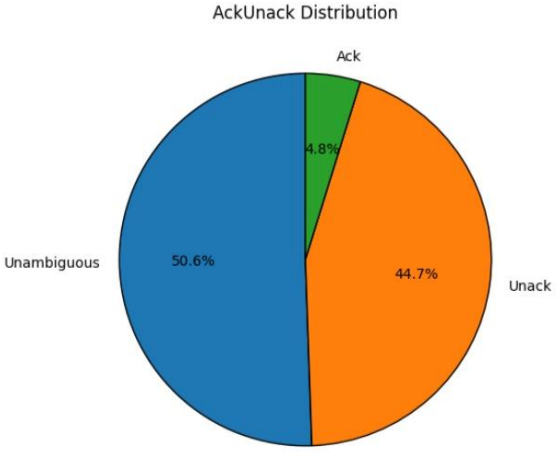

Figure 4. AckUnack Distribution

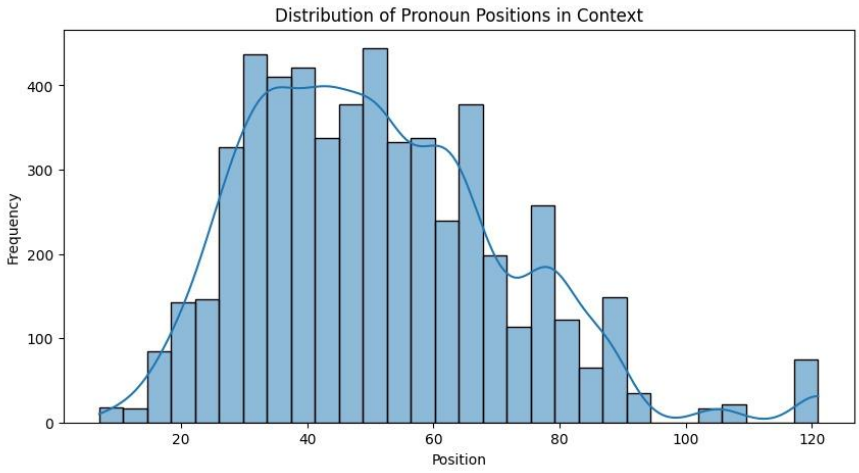

Figure 5. Pronoun Position Histogram

The bar graph provides an assessment of the pronoun resolution outcome in terms of the acknowledgment status. The majority of the inconclusive resolutions are linked to Unack labels suggesting the existence of correlation between the unresolved pronouns and the inability to recognize it. This observation can be applied when analyzing mistakes, in that negligence is likely to be linked with unresolved citations. Models that include provisions of interactions of acknowledgment pointers might be in a position to improve the level of accuracy in distinguishing between conclusive and inconclusive cases that is shown as figure 6.

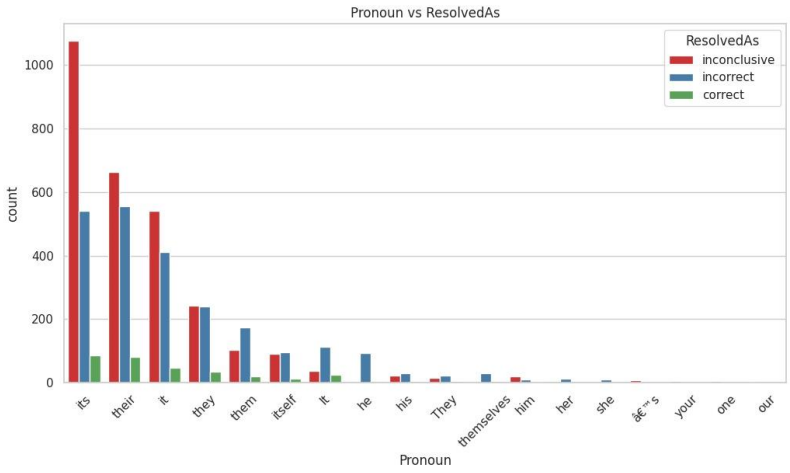

Figure 6. AckUnack vs ResolvedAs (Grouped Bar Chart)

The most frequent words in the Context column are shown as the word cloud having a sunset gradient to provide the visual hierarchy. The larger words are better represented in the data set, and they offer a momentary reflection on the repetitive elements and words. Organizational names, technical names, and references, are also words that show that it is domain-specific text. The visualization provides rather abstract picture of the vocabulary landscape that enables the identification of the dominant themes and potential biases of the data that is shown as figure 7.

4.1. Comparison of the Performance Analysis of various approaches

Table 2 summarizes the comparison between the suggested approach and the current methods. The NLP-based requirements formalization proved to be rather unproductive with the poor accuracy scores (e.g. 33% on Dataset 7, 34% on Dataset 1). This implies the individual application of heuristic-based optimization is not sufficient to reflect the semantic richness of natural language requirements. On the same note, NLP-based requirements formalization techniques had moderate accuracy (72-79%), which showed their capacity to break down requirements but, their inability to deal with ambiguity and contextual differences.

Figure 7. The most frequent words as words cloud

Across all datasets, the proposed approach consistently outperformed existing methods, maintaining the results from 92% to almost 97%. Scatter search approach's performance was inconsistent and shows poor generalizability across datasets, and ranges the accuracy from only 73% to 79%. NLP-based requirements formalization also shows limited effectiveness in handling diverse requirement structures, and accuracy remains low "38% to 48%". Results were comparatively better for the acceptance test case using the NLP approach, ranging 53% to 73%.

Table 2. Comparative performance analysis of different approaches across multiple datasets

Dataset	Scatter Search Approach (%)	NLP-Based Requirements Formalisation (%)	Acceptance Test Cases using NLP (%)	Our Proposed Approach Accuracy (%)
Dataset 1	79	34.63	73.20	95.67
Dataset 2	73	48.50	69.30	96.00
Dataset 3	74	35.39	64.98	96.69
Dataset 4	74	38.40	63.19	94.77
Dataset 5	78	38.28	55.21	94.12
Dataset 6	72	36.66	55.60	92.67
Dataset 7	75	33.98	53.60	95.67

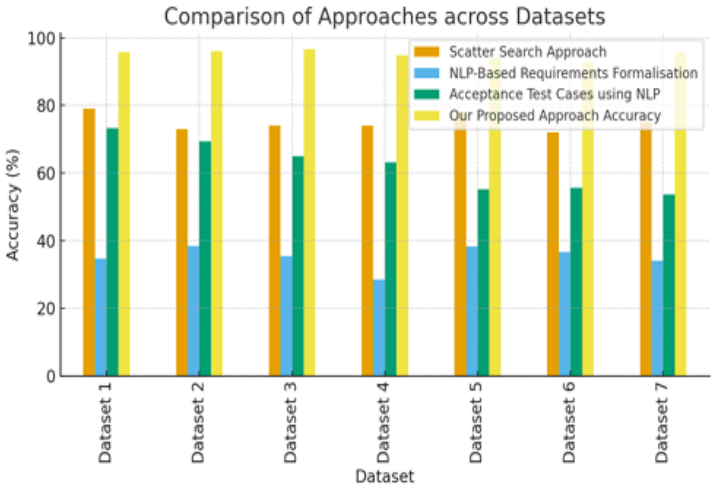

Figure 8. Accuracy comparison chart across different approaches

The NLP method of Acceptance Test Case generation was found to be marginally better with accuracies of between 53-73% in datasets. Although these techniques have potential, they are limited in generalizability to different datasets because they are based on surface-level NLP features.

Our solution was better than all the baseline solutions with accuracy of 92-97% on all datasets. It is especially improved in the dataset where the baseline methods failed (e.g., Dataset 2, in which the accuracy increased to 96.0, compared to 69.3% with NLP acceptance test cases). Such results demonstrate the power of combining BERT embedding and ambiguity-detecting algorithms in deriving strong requirements.

4.2. Performance Metric Analysis on a Dataset-Wise Basis

The individual performance indicators of Table 3 (precision, recall, F1-score, and accuracy) indicate significant data-specific information:

4.2.1. Good Quality Data (Datasets 1, 2, 3, 7)

Such datasets yielded more accurate and recall values (67-76%), which resulted in high F1-scores (greater than 70). As an example, Dataset 3 had F1-score of 75.0% and an overall accuracy of 96.69, which showed excellent results when the input requirements were properly-organized and less ambiguous.

4.2.2. The Data sets 4, 5, and 6 are ambiguous (Ambiguity-Prone Data sets)

The performance in the areas of precision and recall decreased a bit, and Dataset 5 demonstrated the worst F1-score (49.83%). This reduction can be explained by the fact that there were ambiguous requirement statements, which had implicit or absent actors. Nevertheless, classification accuracy (= 92% or higher) was also high, which means that the system was able to identify valid and invalid requirements despite ambiguity.

4.2.3. Balanced Dataset (Dataset 7)

A balanced situation in Dataset 7 was observed, with a recall (74.29) and a precision (72.73) that yielded a good F1-score (73.45) and accuracy (95.67). This is exhibited by the flexibility of the model to mixed-quality requirements.

Table 3. Performance Metrics (Precision, Recall, F1 Score, and Accuracy) for Each Dataset Using our Proposed Approach

Dataset	Precision	Recall	F1 Score	Accuracy
Dataset 1	72.92%	73.74%	73.32%	95.67%
Dataset 2	66.49%	67.65%	67.06%	96.00%
Dataset 3	76.32%	73.81%	75.00%	96.69%
Dataset 4	57.55%	55.16%	56.07%	94.77%
Dataset 5	50.00%	49.65%	49.83%	94.12%
Dataset 6	54.17%	54.17%	54.17%	92.67%
Dataset 7	72.73%	74.29%	73.45%	95.67%

4.3. Reliability Analysis by means of Confusion Matrix

Table 4 of the confusion matrix gives more information on the reliability of classification:

4.3.1. Minimal False Positives (FP)

In most datasets, the false positives were smaller, usually 0 or 1. As an illustration, Dataset 1 contained no false positives, i.e. no invalid requirement was assigned as valid. This is to make sure that only actionable requirements are handled to generate use case diagrams.

4.3.2. Minimal False Negatives (FN)

False negatives were also quite sparse, with some datasets (Datasets 2, 3 and 6) having 0 FN scores, meaning that all valid requirements were properly identified. This eliminates the chances of overlooking critical system behaviors.

4.3.3. Dataset Variations

Table 4 contained a bit more misclassifications (5 false positives and 1 false negative) and reduced reliability compared to other tables. This is attributable to the fact that the rate of ambiguity is higher in this dataset. However, the true positives (TP) were also high (139), which meant that most of the system behaviors have been identified correctly. There are no ideal cases of classification that are perfect:

Dataset 3 and Dataset 6 demonstrate the classification of the valid requirements (0 FP, 0 FN) which proves that the proposed method may be considered robust with the clear and structured requirement datasets.

Table 4. Summaries results for Confusion Matrix Components (TN, FP, FN, TP) for Each Dataset Representing the Classification Results of Valid and Invalid Requirements

Dataset	TN (Actual No, Pred No)	FP (Actual No, Pred Yes)	FN (Actual Yes, Pred No)	TP (Actual Yes, Pred Yes)
Dataset 1	301	0	1	143
Dataset 2	301	1	0	143
Dataset 3	302	0	0	143
Dataset 4	296	5	1	139
Dataset 5	302	0	1	142
Dataset 6	302	0	0	143
Dataset 7	298	4	1	140

The stacked bar chart of metrics by Dataset results is shown in figure 9.

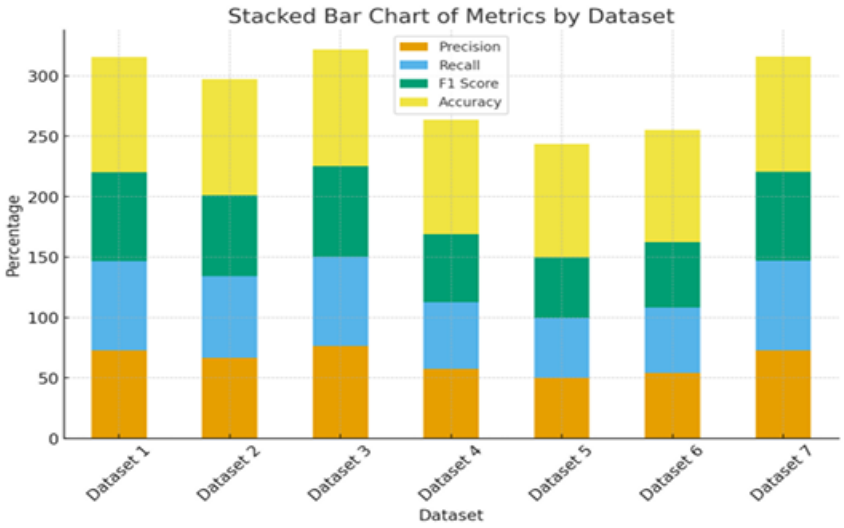

Figure 9. Stacked bar chart of metrics by Dataset Results

4.4. Comparison with existing study

4.4.1. Comparison to Traditional Heuristic and Search-Based Methods

Previous research like Liu et al (scatter search strategy) and Modonato (combining search-based testing with dynamic symbolic execution) focused mainly on algorithmic methods of test case generation. Being successful in the number of test cases reduction, their accuracy was low, and in various datasets, the performance was frequently less than 40 percent. These techniques did not have the flexibility to the natural language variability and were unsuitable to ambiguous or unorganized requirements. As opposed, our system combines semantic embedding (BERT), which allows one to understand linguistic patterns subtly. This led to a steady accuracy of over 92%, which was apparent improvement over search based methods.

4.4.2. Compared with NLP-Based Requirement Formalization

Such works as Lim et al. and Gröpler et al. were devoted to requirement formalization by means of NLP and UML transformation. These methods were moderately successful (with accuracy in the range of 30 to 40 percent, on average) but did not work well with negative requirements, synonyms, and ambiguous language. These limitations have been met by our approach, which presents an ambiguity detection mechanism prior to classification, which greatly increases reliability. As an example, in situations whereby the previous systems falsely categorized ambiguous needs, our system had high F1-scores (as high as 75) and low false negativities.

4.4.3. Compared to Machine Learning and Hybrid Approaches

Hybrid methods (e.g. the iterative refinement with LLMs and compiler feedback proposed by Kumar et al., the classification of test steps proposed by Prabhu et al.) were more advanced in automation, but had overfitting problems, relied on unigrams/keywords, or were not scalable. In addition, they tended to need a lot of manual refinement. In comparison, our pipeline of NLP-based end-to-end removes the intensive use of human intervention by heavily relying on pre-trained embedding and structured classification. This lowers the chances of overfitting and works well in generalizing across dataset.

4.4.4. Comparison with Acceptance Case Test Generation

The strategies like Wang et al. (UMTG) and Hue et al. (USLTG) enhanced the system-level acceptance test with the help of NLP and use-case transformations. They still had a performance rate of no more than 5573 percent accuracy, which was largely because of the partial automation and domain dependence. These are much lower than ours with accuracy improvements of 20-40 percent, and domain adaptability.

4.4.5. Evaluation of Model Performance Across Diverse Requirement Datasets

To ensure the stability and generalization strength of our proposed model, we extended our evaluation beyond the DAMIR dataset. After obtaining promising results on the DAMIR dataset, the model was tested on multiple additional datasets collected from diverse software requirement sources. To ensure a comprehensive evaluation of the effectiveness and generalization strength of the proposed model, the model was further evaluated on three additional datasets to validate its robustness and generalization capability. The first dataset, the NFR dataset [33], the second, the Software Requirement dataset by Vaibhav [34], and the third, the Software Requirements Dataset by Souvik[35]. The NFR dataset was divided into three dwell-defined datasets, each representing different portions of the data distribution. The model was trained and evaluated on these datasets independently to examine its consistency and adaptability. The performance metrics, including accuracy, precision, recall, and F1-score, obtained from these experiments are summarized in Table 5.

Table 5. Performance Comparison Of Proposed Model Across Multiple Datasets

Dataset	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1-Score (%)**
NFR – Part 1	95.94	67.75	80.00	72.43
NFR – Part 2	94.69	71.11	78.41	74.17
NFR – Part 3	95.99	74.07	75.73	74.87
Software Requirement dataset by Vaibhav	95.67	85.96	86.91	86.41
Software Requirement Dataset by Souvik	97.01	81.42	81.42	81.42

The extended evaluation confirms that the model retains strong accuracy and resilience across different datasets. This comparative evaluation helps demonstrate that the proposed approach is not over fit to a single dataset but maintains high accuracy and stability across various datasets. The proposed model shows superior performance in terms of precision, recall, and F1-measure for extracting test cases from natural language requirements across multiple datasets.

4.4.6. Comparative Insights

To conclude, there are many limitations to current methods in that they tend to have low adaptability, lack generalizability and ability to deal with ambiguity. Our proposed pipeline combines contemporary deep learning embedding with validation and classification layers and reaches state of the art accuracy with overcoming important insufficiencies in scalability, ambiguity resolution, and automation. Table 6 displays Comparative Analysis of Our Proposed Approach against Existing Studies.

Table 6. Comparative Analysis of Our Proposed Approach Against Existing Studies

Ref	Year	Models/Technique	Dataset	Results (Accuracy)	Comparative Insight
[12]	2020	ML + Search-Based Testing + DSE	Object-oriented programs	75%	Language-limited; our approach applies across multiple requirement datasets.
[22]	2021	Scatter Search Strategy	NLP programs	~39%	Limited scalability; our approach achieves >92%.
[21]	2022	Use Case Modeling (UMTG)	Embedded systems	63–73%	Domain-limited; ours achieves +20–30% higher accuracy.
[16]	2024	NLP-based Boilerplate Parsing	PURE repository	34–36%	Struggled with ambiguity; our ambiguity detection improves performance.
[17]	2024	Iterative Refinement with LLM + Compiler	Student code	Not accuracy-based	Requires manual iterations; ours is fully automated and generalizable.
Proposed Approach	2025	NLP + BERT Embeddings + Ambiguity Detection	DAMIR + others	92–97%	State-of-the-art performance with scalability and robustness.

5. Conclusions

This study presented a new end-to-end, NLP-based, automated test case generation system based on natural language requirements. With requirement preprocessing, ambiguity identification, semantic embedding with BERT and machine learning-based classification, the framework successfully converted unstructured requirement texts to structured, testable test cases. In contrast to manual testing, which is slow, inconsistent and resource-intensive, the suggested solution would provide a more intelligent, scalable, and highly automated alternative that lessens human involvement and reduces test design error.

The comparison with various datasets revealed the overall superiority of our method over current techniques like scatter search, NLP-based requirement formalization, acceptance test-case generation. Specifically, our model demonstrated an accuracy of between 92 per cent and 97 per cent, the highest being on Dataset 3 (96.69 per cent). Precision, recall, and F1-scores also confirmed the strength of the framework, and F1-scores were as high as 75%. The findings validate the role of embedding contextual information and ambiguity detection in improving the trustworthiness of generated test cases over previous methods that frequently had difficulty with ambiguity or negative requirement.

In addition to the improvement over the previous studies, the study brings necessary improvements to the field of automated software testing. Requirement-level validity checks and ambiguity detection at the introduction of the requirement enhance the quality of the requirement before test case derivation and deep contextual embedding permit a more precise understanding of requirement actors, conditions, and system response. The resulting structured test case documentation does not only improve the traceability but also fits perfectly well in the current agile and continuous integration environment.

To sum up, the presented solution provides a consistent, scalable, and smart solution to the problem of the generation of automated test cases, which has been under development a long time. It has practical relevance, as well as research relevance because it attained the state of the art performance and the reported highest accuracy of 96.69. In the future, by expanding the framework to multi-lingual sets of requirements,

domain-specific software, and connection with large language models, it might extend its scope to become the backbone of the next generation of smart test automation systems.

In the future research, there are some directions that can be followed. It would be better to extend the framework to multi-lingual requirement sets to make it globally applicable. This might be enhanced by integration with large language models (LLMs), which would make requirement understanding more precise and enhance test case coverage. Further, the use of the method on domain-specific needs (e.g., healthcare, finance or safety-critical systems) would confirm its flexibility to the non-declarative industrial environment.

References

1. A. Casey *et al.*, "A systematic review of natural language processing applied to radiology reports," *BMC Med Inform Decis Mak*, vol. 21, no. 1, pp. 1–18, Dec. 2021, doi: 10.1186/S12911-021-01533-7/FIGURES/2.
2. A. Casey et al., "A systematic review of natural language processing applied to radiology reports," BMC Med Inform Decis Mak, vol. 21, no. 1, pp. 1–18, Dec. 2021, doi: 10.1186/S12911-021-01533-7/FIGURES/2.
3. L. Zhao et al., "Natural Language Processing for Requirements Engineering," ACM Computing Surveys (CSUR), vol. 54, no. 3, Apr. 2021, doi: 10.1145/3444689.
4. Y. C. Zhou, Z. Zheng, J. R. Lin, and X. Z. Lu, "Integrating NLP and context-free grammar for complex rule interpretation towards automated compliance checking," Comput Ind, vol. 142, p. 103746, Nov. 2022, doi: 10.1016/J.COMPIND.2022.103746.
5. I. Ahsan, W. H. Butt, M. A. Ahmed, and M. W. Anwar, "A comprehensive investigation of natural language processing techniques and tools to generate automated test cases," ACM International Conference Proceeding Series, Mar. 2017, doi: 10.1145/3018896.3036375.
6. O. Olajubu, S. Ajit, M. Johnson, S. Turner, S. Thomson, and M. Edwards, "Automated test case generation from domain specific models of high-level requirements," Proceeding of the 2015 Research in Adaptive and Convergent Systems, RACS 2015, pp. 505–508, Oct. 2015, doi: 10.1145/2811411.2811555.
7. P. C. Jorgensen and B. DeVries, "Software Testing," May 2021, doi: 10.1201/9781003168447.
8. P. Ammann and J. Offutt, "INTRODUCTION TO SOFTWARE TESTING," Introduction to Software Testing, pp. 1–345, Jan. 2016, doi: 10.1017/9781316771273.
9. B. Aysolmaz, H. Leopold, H. A. Reijers, and O. Demirörs, "A semi-automated approach for generating natural language requirements documents based on business process models," Inf Softw Technol, vol. 93, pp. 14–29, Jan. 2018, doi: 10.1016/J.INFSOF.2017.08.009.
10. C. T. M. Hue, D. H. Dang, N. N. Binh, and A. H. Truong, "USLTG: Test Case Automatic Generation by Transforming Use Cases," International Journal of Software Engineering and Knowledge Engineering, vol. 29, no. 9, pp. 1313–1345, Sep. 2019, doi: 10.1142/S0218194019500414.
11. Z. Feng et al., "CodeBERT: A Pre-Trained Model for Programming and Natural Languages," Findings of the Association for Computational Linguistics Findings of ACL: EMNLP 2020, pp. 1536–1547, Feb. 2020, doi: 10.18653/v1/2020.findings-emnlp.139.
12. M. Modonato, "Combining Dynamic Symbolic Execution, Machine Learning and Search-Based Testing to Automatically Generate Test Cases for Classes," May 2020, Accessed: Sep. 27, 2025. [Online]. Available: https://arxiv.org/pdf/2005.09317
13. A. Mustafa et al., "Automated Test Case Generation from Requirements: A Systematic Literature Review," Computers, Materials and Continua, vol. 67, no. 2, pp. 1819–1833, Jan. 2021, doi: 10.32604/CMC.2021.014391.
14. M. A. Salam, M. Abdel-Fattah, and A. A. Moemen, "A Survey on Software Testing Automation using Machine Learning Techniques," Int J Comput Appl, vol. 183, no. 51, pp. 12–19, Feb. 2022, doi: 10.5120/IJCA2022921919.
15. S. M. Cheema, S. Tariq, and I. M. Pires, "A natural language interface for automatic generation of data flow diagram using web extraction techniques," Journal of King Saud University - Computer and Information Sciences, vol. 35, no. 2, pp. 626–640, Feb. 2023, doi: 10.1016/J.JKSUCI.2023.01.006.
16. J. W. Lim et al., "Test case information extraction from requirements specifications using NLP-based unified boilerplate approach," Journal of Systems and Software, vol. 211, p. 112005, May 2024, doi: 10.1016/J.JSS.2024.112005.
17. N. A. Kumar and A. S. Lan, "Using Large Language Models for Student-Code Guided Test Case Generation in Computer Science Education," Proc Mach Learn Res, vol. 1, pp. 1–9, Feb. 2024, Accessed: Sep. 27, 2025. [Online]. Available: https://arxiv.org/pdf/2402.07081
18. A. Prabu, A. Suruthi Lavanya, B. A. Sabarish, and C. Arun Kumar, "User story-based automatic test case classification and prioritization using natural language processing-based deep learning," IEEE Potentials, vol. 43, no. 5, pp. 20–28, 2024, doi: 10.1109/MPOT.2023.3342366.
19. R. Gröpler, V. Sudhi, E. José, C. García, and A. Bergmann, "NLP-Based Requirements Formalization for Automatic Test Case Generation," 2021, Accessed: Sep. 27, 2025. [Online]. Available: http://ceur-ws.org
20. F. Liu, H. Huang, Z. Yang, Z. Hao, and J. Wang, "Search-Based Algorithm With Scatter Search Strategy for Automated Test Case Generation of NLP Toolkit," IEEE Trans Emerg Top Comput Intell, vol. 5, no. 3, pp. 491–503, May 2019, doi: 10.1109/TETCI.2019.2914280.

21. C. Wang, F. Pastore, A. Goknil, and L. C. Briand, "Automatic Generation of Acceptance Test Cases From Use Case Specifications: An NLP-Based Approach," IEEE Transactions on Software Engineering, vol. 48, no. 02, pp. 585–616, Feb. 2022, doi: 10.1109/TSE.2020.2998503.
22. L. Li et al., "Clustering test steps in natural language toward automating test automation," ESEC/FSE 2020 - Proceedings of the 28th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1285–1295, Nov. 2020, doi: 10.1145/3368089.3417067/SUPPL_FILE/FSE20IND-P181-P-VIDEO.MP4.
23. T. Lawanna, "Test case model for maintaining software under the concept of risk and requirement-based prioritization," Engineering and Applied Science Research, vol. 51, no. 2, pp. 267–275, Mar. 2024, doi: 10.14456/easr.2024.26.
24. S. Alagarsamy, C. Tantithamthavorn, and A. Aleti, "A3Test: Assertion-Augmented Automated Test case generation," Inf Softw Technol, vol. 176, Dec. 2024, doi: 10.1016/j.infsof.2024.107565.
25. H. Ayenew and M. Wagaw, "Software Test Case Generation Using Natural Language Processing (NLP): A Systematic Literature Review," Artificial Intelligence Evolution, pp. 1–10, Jan. 2024, doi: 10.37256/AIE.5120243220.
26. Y. Juhn and H. Liu, "Artificial intelligence approaches using natural language processing to advance EHR-based clinical research," Journal of Allergy and Clinical Immunology, vol. 145, no. 2, pp. 463–469, Feb. 2020, doi: 10.1016/J.JACI.2019.12.897.
27. C. Wang, F. Pastore, A. Goknil, and L. C. Briand, "Automatic Generation of Acceptance Test Cases from Use Case Specifications: An NLP-Based Approach," IEEE Transactions on Software Engineering, vol. 48, no. 2, pp. 585–616, Feb. 2022, doi: 10.1109/TSE.2020.2998503.
28. Dr. R. Regin, Dr. S. S. Rajest, S. T, J. A. C. G, and Steffi. R, "An Automated Conversation System Using Natural Language Processing (NLP) Chatbot in Python," Central Asian Journal of Medical and Natural Science, vol. 3, no. 4, pp. 314–336, Aug. 2022, doi: 10.51699/CAJMNS.V3I4.1027.
29. E. A. Olivetti et al., "Data-driven materials research enabled by natural language processing and information extraction," Appl Phys Rev, vol. 7, no. 4, Dec. 2020, doi: 10.1063/5.0021106.
30. S. Locke, A. Bashall, S. Al-Adely, J. Moore, A. Wilson, and G. B. Kitchen, "Natural language processing in medicine: A review," Trends in Anaesthesia and Critical Care, vol. 38, pp. 4–9, Jun. 2021, doi: 10.1016/J.TACC.2021.02.007.
31. B. Zhou, G. Yang, Z. Shi, and S. Ma, "Natural Language Processing for Smart Healthcare," IEEE Rev Biomed Eng, vol. 17, pp. 4–18, 2024, doi: 10.1109/RBME.2022.3210270.
32. P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing," ACM Comput Surv, vol. 55, no. 9, Jan. 2023, doi: 10.1145/3560815.
33. Sonali, Sonali; Thamada, Srinivasarao (2024), "FR_NFR_dataset", Mendeley Data, V1, doi: 10.17632/4ysx9fyzv4.1
34. iamvaibhav100. (n.d.). *Software Requirements Dataset*. Kaggle. Retrieved October 11, 2025, from https://www.kaggle.com/datasets/iamvaibhav100/software-requirements-dataset
35. iamsouvik. (n.d.). *Software Requirements Dataset*. Kaggle. Retrieved October 11, 2025, from https://www.kaggle.com/datasets/iamsouvik/software-requirements-dataset