



International Journal of Innovative Research in Science Engineering and Technology (IJIRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 9.935

Volume 15, Issue 10, October 2026

AI Requirements Intelligence Platform: An Integrated Approach to SRS Extraction, Ambiguity Detection, and Requirement Conflict Analysis

Nivethitha R¹, Inbasekaran C², Harishkumar J³

Assistant Professor, Department of Computer Science and Engineering, K.L.N. College of Engineering,
Tamil Nadu, India¹

Student, Department of Computer Science and Engineering, K.L.N. College of Engineering, Tamil Nadu, India^{2,3}

ABSTRACT: Software Requirements Specification (SRS) documents commonly contain natural-language statements that may be ambiguous, duplicated, inconsistent, or contradictory. Manual review of lengthy specifications is time-consuming and can produce inconsistent results. This paper presents the AI Requirements Intelligence Platform (ARIP), a modular system designed to support automated SRS quality analysis. ARIP accepts requirements as direct text or PDF documents. Its processing pipeline extracts and cleans text, identifies requirement-bearing sentences, filters non-requirement content using a TF-IDF and Logistic Regression classifier, and applies ambiguity and conflict analysis to the retained requirements. The platform uses spaCy for sentence and linguistic processing, RapidFuzz for duplicate checking, and a hybrid similarity-and-rule-based conflict engine to identify potentially incompatible requirements. Detected issues are presented with classifications, confidence values, explanations, and severity levels. A quality assessment is consolidated into a score on a 0–100 scale, and results can be exported through a structured PDF report generated using ReportLab. The project report describes a balanced requirement-classification dataset containing 4,000 samples, with 2,000 requirement and 2,000 non-requirement examples. Demonstrations show SRS upload, requirement extraction, ambiguity findings, and PDF report generation. The implementation provides an integrated workflow for early-stage requirements review; broader evaluation on independently labelled SRS documents remains necessary to establish generalizability.

KEYWORDS: Requirements Engineering, Software Requirements Specification, Natural Language Processing, Machine Learning, Ambiguity Detection, Conflict Detection, TF-IDF, Logistic Regression, Quality Assessment.

I. INTRODUCTION

A Software Requirements Specification (SRS) communicates the expected behaviour, constraints, and quality attributes of a software system. Requirements are often written in natural language so that stakeholders, analysts, developers, and testers can understand them. However, natural-language requirements can contain vague terms, missing measurable limits, repeated statements, and constraints that conflict with other requirements. Such issues can cause different interpretations and may be discovered late in development.

Traditional requirement-quality review relies on manual inspections, checklists, and static keyword rules. Manual review requires sustained reviewer effort and may vary between reviewers. Keyword-based tools can identify expressions such as “fast,” “user-friendly,” and “appropriate,” but may also flag irrelevant document text and cannot independently establish whether two requirements contradict each other.

Natural Language Processing (NLP) and machine learning offer ways to automate parts of requirements engineering. Sentence classification can separate requirement statements from document noise, while text similarity and linguistic rules can identify candidate conflicts. The AI Requirements Intelligence Platform (ARIP) combines these capabilities in

one workflow: SRS extraction, requirement classification, ambiguity detection, conflict analysis, quality scoring, and structured PDF reporting.

The main contribution is an integrated prototype combining (i) layout-aware SRS extraction, (ii) requirement/non-requirement classification, (iii) ambiguity classification and explanation, (iv) candidate-pair conflict analysis, (v) consolidated quality scoring, and (vi) downloadable reporting. ARIP is intended to support, not replace, expert review.

II. LITERATURE SURVEY / MATERIALS AND METHODS / SYSTEM MODEL

This section reviews existing research in requirements engineering and describes the methods and tools used in the ARIP system. It also presents the system model, explaining how SRS extraction, requirement classification, ambiguity detection, and conflict analysis are integrated.

2.1 Literature Survey

S. No.	Author(s) and Year	Research Focus / Method	Key Contribution / Relevance to ARIP	Limitation / Research Gap
1	Ferrari, Spagnolo, and Gnesi (2017)	PURE: A Dataset of Public Requirements Documents.	Provides a public requirements-document dataset that supports empirical research on natural-language software requirements.	A dataset resource by itself does not provide an integrated workflow for ambiguity detection, conflict analysis, quality scoring, and report generation.
2	Canedo and Mendes (2020)	Software requirements classification using machine-learning algorithms.	Supports the use of machine-learning methods for classifying software requirements from textual features.	Classification alone does not cover the full ARIP workflow, including ambiguity explanations, conflict identification, and PDF reporting.
3	Bashir et al. (2025)	Requirements ambiguity detection and explanation with large language models; industrial study.	Examines LLM-based ambiguity identification and explanation, relevant to making automated ambiguity findings more understandable.	An ambiguity-focused approach does not necessarily provide requirement extraction, duplicate checking, conflict analysis, quality scoring, and audit reporting as one pipeline.
4	Lobus, Felfernig, Garber, and Tran (2024)	Leveraging LLMs for software-requirement quality assurance.	Motivates using language models to examine natural-language requirements and provide quality-related feedback.	The reference details and the precise evaluated scope should be verified from the publisher record before final journal submission.
5	Helmaczi, Cevik, and Yildirim (2022)	Prompt-based few-shot learning approach to software conflict detection.	Explores the use of a small number of examples in prompts to help identify potential conflicts between requirements.	Prompt-based conflict detection alone does not constitute a complete SRS-processing and reporting workflow;

S. No.	Author(s) and Year	Research Focus / Method	Key Contribution / Relevance to ARIP	Limitation / Research Gap
				performance can depend on prompts and examples.
6	Necula, Dumitriu, and Greavu-Şerban (2024)	Systematic literature review on NLP in software requirements engineering.	Surveys NLP use in requirements engineering and provides background for positioning ARIP among automated analysis approaches.	A review synthesizes existing work rather than implementing the integrated ARIP pipeline.
7	Zhao et al. (2021)	Natural Language Processing for Requirements Engineering: A Systematic Mapping Study.	Maps research themes and methods in NLP-based requirements engineering.	A mapping study does not itself deliver an operational tool that combines requirement classification, ambiguity detection, conflict analysis, scoring, and report generation.
8	Veizaga, Shin, and Briand (2024)	Automated smell detection and recommendation in natural-language requirements.	Addresses automated detection of requirement smells and recommendations, relevant to identifying and improving problematic statements.	Requirement-smell detection is one quality-analysis task and may not cover SRS extraction, cross-requirement contradiction analysis, or integrated audit reports.
9	Gartner and Gohlich (2024)	Automated requirement contradiction detection through formal logic and LLMs.	Directly relates to identifying contradictions between requirements using formal reasoning and language-model techniques.	ARIP instead emphasizes a lightweight similarity, linguistic-feature, and rule-based pipeline; comparative evaluation is needed to establish relative performance.
10	ISO/IEC/IEEE 29148:2018(2018)	Requirements-engineering life-cycle standard.	Provides a standards context for requirements engineering and the quality of requirements specifications.	A standard defines guidance and processes rather than supplying an automated classifier or conflict-detection implementation.

2.2 Materials and Dataset

The project report describes a requirement-sentence classification dataset containing 4,000 rows: 2,000 labelled YES (requirement) and 2,000 labelled NO (non-requirement), reportedly covering more than 12 domains. Textual features are generated using TF-IDF and classified with Logistic Regression. The uploaded report does not provide complete dataset records, the full split protocol, or an independent annotation procedure; these details should be included in a final submission if available.

Table 2.1: ARIP Components, Methods, Tools, and Their Purposes

Component	Method / tool	Purpose
PDF extraction	PyMuPDF (fitz); pdfplumber also appears in the report	Extract text and document structure.
Sentence processing	spaCy	Segment sentences and extract linguistic features.
Requirement classification	TF-IDF + Logistic Regression	Classify sentences as requirement or non-requirement.
Duplicate checking	RapidFuzz	Identify repeated or highly similar statements.
Ambiguity analysis	Supervised classifier and vague-term triggers	Label clarity and provide possible reasons for review.
Conflict analysis	TF-IDF cosine similarity, spaCy, rule-based checks	Identify potentially incompatible requirement pairs.
Reporting	ReportLab	Generate a structured PDF analysis report.

2.3 System Model

ARIP follows a client-server design. The web interface accepts direct text or an SRS PDF and communicates with a Python backend through REST endpoints. The report identifies FastAPI and Uvicorn as the API stack and lists /health, /analyze, /extract-pdf, /detect-conflicts, and /generate-report as key endpoints.

The workflow contains four logical layers: document ingestion, requirement understanding, quality analysis, and reporting. The ingestion layer extracts and cleans content. The understanding layer segments sentences, classifies requirements, removes non-requirement content, checks duplicates, and normalizes retained statements. The analysis layer detects ambiguity and candidate conflicts and consolidates findings into a quality score. The reporting layer displays results and generates a PDF.

III. PROPOSED METHODOLOGY AND DISCUSSION

The proposed methodology integrates SRS extraction, requirement classification, ambiguity detection, conflict analysis, and quality assessment into a unified workflow. The discussion highlights the prototype's capabilities and the need for further evaluation to establish its accuracy, reliability, and effectiveness.

3.1 End-to-End Processing Pipeline

This table describes the eight stages of ARIP, from SRS document input to requirement analysis and report generation. It highlights the operations performed at each stage and the corresponding outputs.

Table 3.1: Stages of the Proposed ARIP Workflow

Stage	Operation	Output
1	Direct text input or SRS PDF upload	Raw text or document
2	PDF text/layout extraction and cleaning	Cleaned document content
3	Sentence segmentation using spaCy	Individual sentence candidates
4	TF-IDF + Logistic Regression classification	YES/NO labels
5	Filtering, duplicate checking, normalization	Refined requirements with identifiers
6	Ambiguity analysis	Clear/Ambiguous label, confidence, feedback
7	Candidate-pair conflict analysis	Potential conflict type, severity, evidence
8	Quality scoring and report generation	Score, dashboard findings, PDF report

3.2 Requirement Sentence Classification

The classifier reduces the content passed to later stages by separating actionable requirement statements from document noise. Each sentence is represented using TF-IDF features and classified by Logistic Regression as a requirement or

non-requirement. The project report describes a balanced 4,000-row dataset. The classification stage is intended to filter headings, figure captions, background descriptions, and other text that does not express an actionable requirement. A complete journal evaluation should specify the data split, model settings, class-wise precision and recall, and confusion matrix.

3.3 Ambiguity Detection and Explainability

Retained requirements are classified as Clear or Ambiguous. The report identifies vague expressions such as “fast,” “user-friendly,” “reliable,” “robust,” “as soon as possible,” and “as needed.” For example, “The system should provide fast response to users” is difficult to verify unless a response-time threshold and operating conditions are specified. ARIP reports confidence and feedback to help the analyst revise the requirement.

Trigger terms are useful indicators but do not prove that a requirement is defective in every context. A flagged statement should therefore be reviewed by a requirements engineer, and the interface should distinguish automated warnings from confirmed defects.

3.4 Requirement Conflict Detection

The conflict engine selects candidate requirement pairs using TF-IDF cosine similarity and then examines linguistic features such as actions, polarity, roles, conditions, numerical constraints, and temporal expressions. The project report describes direct contradiction, permission/authorization, behavioural, temporal, numeric/constraint, and condition-based conflict categories.

For example, one requirement may state that users can download PDF reports at any time, while another prevents downloads during maintenance hours. These statements may conflict unless the first explicitly excludes maintenance periods. ARIP is designed to return requirement IDs, conflict type, severity, confidence, explanation, and evidence. Similarity alone is not proof of contradiction; conflict rules should be evaluated against manually labelled positive and negative pairs.

3.5 Quality Score and Reporting

ARIP calculates a Software Requirement Quality Index from 0 to 100. The report lists factors such as the clear-to-ambiguous ratio, conflict severity, requirement precision, numerical information, and domain-risk indicators. The uploaded report does not specify the complete formula or factor weights. A journal submission should include the formula, rationale for weights, and worked examples to make the score reproducible.

ReportLab is used to generate a structured PDF containing analysis summaries, requirement-level results, ambiguity findings, conflict details, quality assessment, and recommendations. The web interface supports document upload, extracted-requirement review, ambiguity/conflict visualization, report preview, and PDF download.

3.6 Discussion

The integration of a lightweight classifier, NLP processing, and explicit conflict rules makes ARIP modular and offers some traceable explanations. Nevertheless, lexical similarity and trigger-word rules can miss implicit contradictions or generate false positives. Domain differences, PDF formatting, sentence boundaries, and label quality may also affect performance. Human validation remains necessary, particularly for high-impact requirements.

IV. EXPERIMENTAL RESULTS

The experimental results demonstrate ARIP’s prototype workflow, including requirement classification, ambiguity detection, conflict analysis, and PDF report generation. The evaluation confirms the demonstrated functionality, while further testing is required to validate model performance and generalization.

4.1 Dataset and Demonstration Summary

The uploaded project report describes a balanced 4,000-row classification dataset and demonstrates an SRS-analysis workflow. In the displayed sample, 10 requirements were analysed: 7 were labelled clear and 3 ambiguous, with no conflicts detected in that sample. These are demonstration observations, not comprehensive model-performance measurements.

Table 4.1: Summary of Experimental Results and Evaluation Limitations

Evaluation item	Result shown or stated	Interpretation
Classifier dataset	4,000 rows: 2,000 YES and 2,000 NO	Dataset size and balance are documented; split details are not included.
SRS demonstration	10 requirements analysed	Small workflow demonstration.
Ambiguity findings	7 clear and 3 ambiguous	Shows clarity findings for the sample.
Conflict findings	0 conflicts detected	Applies only to this sample; does not establish recall or precision.
PDF report	Generation and browser download shown	Demonstrates report export.
Classifier performance	Project conclusion claims 100% test accuracy and 100% macro F1	The uploaded report does not include sufficient detailed test-set evidence to independently verify these figures.

4.2 Example Ambiguity Findings

This table presents example requirements containing vague or subjective terms and explains their potential issues. It also suggests refinement directions to make requirements more precise, measurable, and verifiable.

Table 4.2: Examples of Ambiguous Requirements and Refinement Directions

Example requirement	Potential issue	Refinement direction
The system should provide fast response to users.	“Fast” is not measurable.	Specify a response-time threshold, workload, and measurement conditions.
The application must have a user-friendly interface.	“User-friendly” is subjective.	Define usability criteria or measurable task-completion targets.
The system should use appropriate security mechanisms.	“Appropriate” does not identify controls.	Specify required controls, standards, and verification criteria.

4.3 Example Conflict

Requirement A: “The system shall allow users to download PDF reports at any time.” Requirement B: “The system shall prevent users from downloading PDF reports during maintenance hours.” The pair contains a potential temporal conflict because one statement is unrestricted and the other introduces a maintenance-period restriction. An illustrative revision is: “The system shall allow users to download

PDF reports at any time except during scheduled maintenance periods.” The intended business rule must be confirmed by the system owner.

4.4 Evaluation Limitations

The ARIP prototype demonstrates the SRS analysis workflow, but its evaluation requires further validation. The train, validation, and test split procedures are not fully documented. The annotation process for ambiguity and conflict detection needs clearer guidelines. Confusion matrices, precision, recall, F1-score, and false-positive rates should be reported. Runtime measurements are needed to evaluate performance on large SRS documents. Baseline comparisons are required to establish the effectiveness of the proposed approach. Any 100% accuracy claim must be verified using held-out data and duplicate-leakage checks. Future work should evaluate robustness across diverse domains and previously unseen SRS documents.

4.5 Result and Output

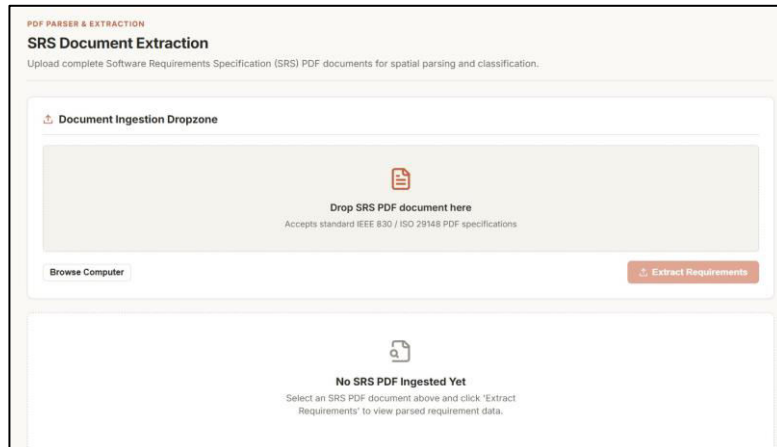


Figure 4.1: ARIP SRS Document Extraction Interface

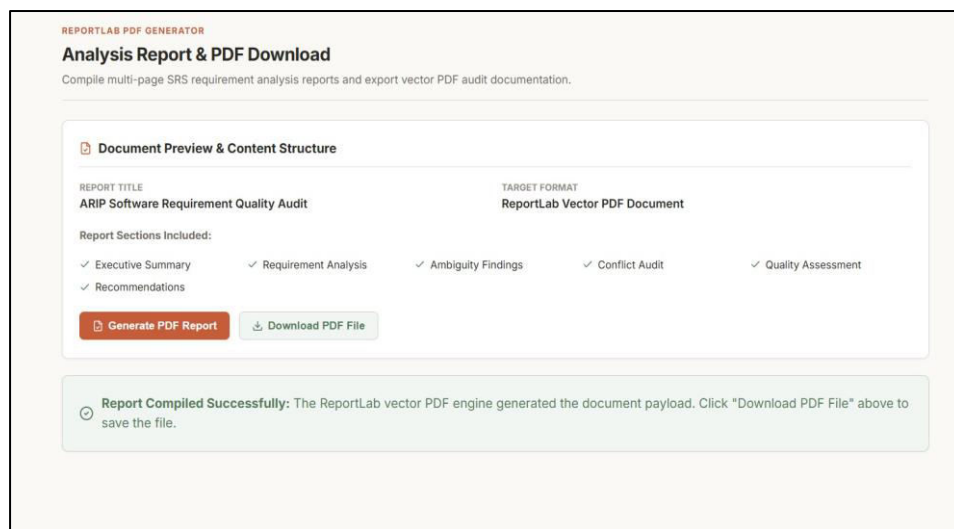


Figure 4.2: ARIP Analysis Report and PDF Generation Interface

V. CONCLUSION

This paper presented ARIP, an integrated prototype for analysing natural-language software requirements. The platform combines SRS PDF extraction, sentence segmentation, requirement classification, duplicate checking, ambiguity detection, candidate-pair conflict analysis, quality scoring, and PDF reporting. Its modular workflow is intended to reduce repetitive manual review and help analysts locate potentially unclear or inconsistent statements earlier in development.

The project report demonstrates requirement extraction, ambiguity findings, conflict presentation, and downloadable PDF output. The sample identifies three ambiguous statements among ten requirements and reports no conflict in that sample. These observations demonstrate functionality but do not establish general performance. Future work should evaluate independently annotated SRS documents from multiple domains, report module-specific metrics, validate the quality-score formula, and compare ARIP with rule-based and language-model baselines. ARIP should be treated as decision support, with final decisions reviewed by domain experts.

REFERENCES

- [1] A. Ferrari, G. O. Spagnolo, and S. Gnesi, "PURE: A Dataset of Public Requirements Documents," in Proceedings of the 25th IEEE International Requirements Engineering Conference (RE), 2017, pp. 502–505, doi: 10.1109/RE.2017.29.
- [2] E. D. Canedo and B. C. Mendes, "Software Requirements Classification Using Machine Learning Algorithms," Entropy, vol. 22, no. 9, Art. no. 1057, 2020, doi: 10.3390/e22091057.
- [3] S. Bashir, A. Ferrari, M. Abbas, P. E. Strandberg, Z. Haider, M. Saadatmand, and M. Bohlin, "Requirements Ambiguity Detection and Explanation with LLMs: An Industrial Study," in Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME), 2025, pp. 620–631, doi: 10.1109/ICSME64153.2025.00063.
- [4] S. Lobus, A. Felfernig, D. Garber, and T. N. T. Tran, "Leveraging LLMs for the Quality Assurance of Software Requirements," 2024. Complete publisher details should be verified before submission.
- [5] R. K. Helmaczi, M. Cevik, and S. Yildirim, "A Prompt-Based Few-Shot Learning Approach to Software Conflict Detection," in Proceedings of CASCON, 2022.
- [6] S.-C. Necula, F. Dumitriu, and V. Greavu-Şerban, "A Systematic Literature Review on Using Natural Language Processing in Software Requirements Engineering," Electronics, vol. 13, no. 11, Art. no. 2055, 2024, doi: 10.3390/electronics13112055.
- [7] L. Zhao, W. Alhoshan, A. Ferrari, K. J. Letsholo, M. A. Ajagbe, E.-V. Chioasca, and R. T. Batista-Navarro, "Natural Language Processing for Requirements Engineering: A Systematic Mapping Study," ACM Computing Surveys, vol. 54, no. 3, Art. no. 55, 2021, doi: 10.1145/3444689.
- [8] A. Veizaga, S. Y. Shin, and L. C. Briand, "Automated Smell Detection and Recommendation in Natural Language Requirements," IEEE Transactions on Software Engineering, 2024, doi: 10.1109/TSE.2024.3361033.
- [9] A. E. Gärtner and D. Göhlich, "Automated Requirement Contradiction Detection Through Formal Logic and LLMs," Automated Software Engineering, vol. 31, Art. no. 49, 2024, doi: 10.1007/s10515-024-00452-x.
- [10] ISO/IEC/IEEE, ISO/IEC/IEEE 29148:2018, Systems and Software Engineering—Life Cycle Processes—Requirements Engineering. International Organization for Standardization, 2018.



INTERNATIONAL
STANDARD
SERIAL
NUMBER
INDIA



INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

 9940 572 462  6381 907 438  ijirset@gmail.com



www.ijirset.com

Scan to save the contact details