**Current End-to-End Solution Perspective**

This Project develops an AI-enabled Requirements Engineering Specification (RES) using RAG that automates the elicitation, analysis, and formalization of software requirements from unstructured natural-language data sources. The entire pipeline operates as an end-to-end solution, beginning with data acquisition and ending with validated, structured requirements ready for inclusion in a Software Requirements Specification (SRS).

**Step-by-Step End-to-End Process**

**Input Acquisition -** The system collects textual inputs from multiple sources-stakeholder interviews, chatbot conversations, domain documents, or open datasets such as PROMISE or Banking Chatbot datasets. These represent the raw unstructured requirements statements ().

**Pre-processing and Normalization** - The text data is cleaned, tokenized, and normalized using NLP techniques (stop-word removal, lemmatization, sentence segmentation). This ensures linguistic consistency and prepares the text for automated analysis.

**Feature Extraction and Representation** - Techniques such as TF-IDF and Sentence-BERT embeddings are used to convert the text into numerical vectors that capture semantic meaning. These serve as the model’s features for classification or retrieval.

**Requirement Classification** – A supervised model (e.g., fine-tuned BERT) is to classify requirement statements into Functional Requirements (FR) and Non-Functional Requirements (NFR).

Functional Requirements (FR): What the system should do (e.g., “The system shall authenticate users using a PIN”).

**Knowledge-Base Construction and Retrieval (RAG Module) -** The classified and validated requirements are indexed using FAISS to form a domain-specific knowledge base. When a new query or user prompt arrives, the RAG architecture retrieves relevant documents from this base and provides them as grounded context to the generation model.

**Requirement Generation and Refinement (LLM Component)** - Using a generative model such as FLAN-T5, the system produces formalized requirement statements following an organizational template (“The system shall …”) based on retrieved knowledge and prompt instructions. Feedback loops continuously improve the model’s accuracy and domain alignment.

**Validation and Evaluation -** The generated requirements are assessed using quantitative metrics such as Precision, Recall, F1-score, and Hallucination Rate, along with semantic similarity and stakeholder review to ensure correctness and completeness.

**Output -** The final outputs include:

A structured Software Requirements Specification (SRS) document containing validated FR and NFR sets.

Traceability links between raw stakeholder inputs and formalized requirements.

Metrics reports showing performance of each pipeline component.

**Examples of Pairs**

**x= Input Text (statement from stakeholder, document, or dataset)**

**= Output Label (target category or extracted requirement type)**

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **X** | **y** | **Type of Requirement** |
| 1 | “Users should be able to transfer money between accounts.” | Functional | FR |
| 2 | “The system must encrypt all transaction data.” | Non- Functional | NFR - security |
| 3 | “Customers can check their account balance using voice commands.” | Functional | FR |

A diagram of a chatbot

AI-generated content may be incorrect.

**AI Improvement End-to-End Solution Perspective**

**Input Layer (requirements documents, domain datasets, stakeholder text data) →  
AI Processing Layer (RAG, NLP, ML classification, anomaly detection, clustering, recommendations) →  
Output Layer (validated requirements, quality scores, alerts, summaries, requirement clusters).**

**1. Predictive Requirement Quality Modelling (ML)**

Train ML models on historical requirement datasets (e.g., PROMISE, PURE, Banking Chatbot data) to predict the **quality and correctness** of new requirements.

* **Input:** Requirement attributes (length, ambiguity indicators, similarity to existing requirements, classification features).
* **AI Processing:** Train supervised ML models (Random Forest, Gradient Boosting, Neural Networks).
* **Output:** Probability score that a generated requirement is *clear, testable, and correct*.

**2. Anomaly Detection in Requirement Specifications**

Detect unusual or risky patterns in requirements to flag early signals of **low-quality or conflicting requirements**.

* **Input:** Requirement repository (time-series of revisions, acceptance/rejection logs, classifier scores).
* **AI Processing:** Methods – Autoencoders, Isolation Forest, LSTM anomaly detection.
* **Output:** Alerts such as:
  + “Unusual spike in ambiguous requirements.”
  + “Requirement conflicts detected in NFR set.”

**3. Natural Language Processing (NLP) on Requirement Texts**

Analyse textual content of stakeholder inputs, SRS documents, and generated requirements.

* **Input:** Requirement text, user stories, stakeholder comments.
* **AI Processing:**
  + **Sentiment analysis:** Detect acceptance/resistance trends from stakeholder feedback.
  + **Topic modelling:** Identify recurring requirement themes (usability, performance, security, reliability).
  + **Ambiguity & clarity detection:** Flag vague requirements using linguistic features.
  + Models – BERT, GPT embeddings, LDA/BERTopic.
* **Output:** Requirement health indicators, e.g.,
  + “30% of NFRs are performance related.”
  + “Detected ambiguity in 15% of FRs.”

**4. AI-Assisted Requirement Process Mining**

Beyond discovering requirement engineering workflows, AI predicts **requirement acceptance/rejection** and recommends **optimal elicitation flows**.

* **Input:** Event logs from requirement lifecycle (elicitation, validation, acceptance/rejection, revisions).
* **AI Processing:**
  + Classify requirement traces as “efficient” vs “inefficient.”
  + Predict likelihood of rejection.
  + Recommend stakeholder interaction patterns that improve clarity.
* **Output:** Workflow optimizations, e.g.,
  + “High probability this requirement will be rejected due to ambiguity; suggest rephrasing before review.”

**5. Clustering Requirements by Specification Profiles**

Group requirements into clusters for better **traceability, prioritization, and risk management**.

* **Input:** Requirement features (functional vs non-functional, similarity embeddings, acceptance history).
* **AI Processing:** Unsupervised ML (k-means, DBSCAN, hierarchical clustering).
* **Output:**
  + Clusters of requirements (e.g., security-focused NFRs, usability-related FRs, ambiguous requirements).
  + Dashboards for comparison:
    - “Your project’s NFR set falls into Cluster 2: high ambiguity, low stakeholder acceptance.”