iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** V **Month of publication:** May 2026

DOI: https://doi.org/10.22214/ijraset.2026.83241

www.ijraset.com

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

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue V May 2026- Available at www.ijraset.com

AI-Based Test Scenario Generation

Prof. Renu Kachhoria[1], Saish Pisolkar[2], Gouri Bhapkar[3], Riya Chavan[4], Manas Patil[5], Darshan Pawar[6]
Artificial Intelligence and Data Science Vishwakarma Institute of Technology Pune, India

Abstract: ***Fast moving and accurate quality assurance procedures are required in the software development world today it helps to make sure the correct functionality of the software as the systems needs keep on changing .Manually writing the test cases based on technical documents can be expensive , error prone and also it can consume lot of time. This work showcases Docling as a test case auto generation and document processing system. It transforms software specifications written as PDFs into structured items which will be ready for testing. Docling has the potential for extracting the text , recognizing metadata and document features , performing embedding-based retrieval and also supporting reasoning with the help of Large Language Models (LLMs). It makes use of LlamaIndex for accurate content lookup with the capacity to connect with other models via the Gemini API . The system makes use of easy web interaction as well as APIs for uploading files, observing the process of document analysis and testing case export. The experimental work indicates substantial savings of human efforts with good applicability and tracing of test cases generated. Docling expands testing efficiency with systematic testing of requirements. It is an effective technique towards automated testing of software quality assurance.***

Keyword: Document Processing, Test Case Generation, Quality Assurance, LlamaIndex, Large Language Models, PDF Extraction, Embedding Retrieval, Automation, Software Testing, Artificial Intelligence

I. INTRODCTION

Software testing is a crucial part of software development life cycle (SDLC) . Software testing assesses to recognize whether the developed system is error free and functioning accurately as per the customers requirement. With advancements in software systems and growing complexities in their operation and functionalities , test planning methods also need cop-up with these changes and also adapt to various functions and constraints of software systems. Earlier , test engineers used to examine the requirements specification documentation manually. The test engineers recognize the functionalities of the software system and then develop the test cases based on those functionalities. This process becomes more challenging when it relates to the software requirements specification (SRS) , Project Documentation and Legal Compliance Documentation. The utilization of Artificial Intelligence (AI) is a favorable solution to fasten the process of quality assurance, mainly with the expanded adoption of automation frameworks, and also the use of Large Language Models (LLMs). The use of LLM-based systems is capable of understanding the semantics of natural languages, as well as the ability to automatically create test artifacts. Nevertheless, the current approach also experiences some difficulties. This includes unstructured information in documents, a lack of contextual understanding, the inability to trace the gathered requirements, as well as the inability to combine domain knowledge. Docling is expected to ease the ease of use with web-based interfaces as well as REST APIs. Users will be able to upload files, view stages of document processing, and download results. These results comprise feature summaries, markdown text reports, as well as test case datasets. All of these can be scalable.
The contributions of this project are summarized below:

1) Provide a system for automatically creating test cases from documents. From a travel document , for instance.
2) A hybrid pipeline that includes embeddings-driven retrieval, LLM-based semantic analysis and pdf extraction.
3) Including a role-optimized test generation prompt method.
4) The incorporation of an API-enabled lightweight interactive user interface for software development teams.

Docling meets this challenge with its flexible pipeline for new domains and document formats requiring little re-engineering. Multi-document reasoning, compliance-based validation, and specialized domain-model inclusions are part of ongoing updates, which its modular design welcomes. In this regard, Docling is a state-of-the-art approach that integrates the requirement engineering and automated testing phases to realize high software quality in shorter delivery cycles

II. LITERATURE REVIEW

This has become the main focus in software engineering: automating test case generation from software requirement documents due to much manual work required in traditional quality assurance workflows. It is that LLMs can generate structured test cases that are representative of functional behavior and understand textual requirements.

Schäfer et al's. Study [1] on automated unit test generation with LLMs is one example . They demonstrated how using domain-specific prompts increased test coverage. However, because they primarily concentrated on code-based testing rather than testing based on requirements, there is a gap in document-driven approaches .

In the work of Celik et al. [2], a detailed analysis of LLM-based automated test case generation was done, revealing some challenges that must be overcome, including problems about context window limits and hallucination issues. Their findings underline the need for hybrid systems which combine retrieval and reason-ing-a design principle in the proposed system.

Masuda et al. [3] discuss how high-level functional test cases can be generated directly from industry use case natural language requirements. Here, too, the conclusion is that models need to be endowed with contextual document understanding to enhance relevance. This supports the approach of Docling to employ embedding-powered document retrieval.

Roy et al. in [4] presented a retrieval-informed chain process in the context of conversational extraction in PDFs on the basis of embedding and vector indexing. The presented technique demonstrated that query-based retrieval on chunked docs improves the context-awareness of query responses, thus reducing hallucinations within the output generated by the LLM. It is particularly significant in scenario-oriented task examples like requirement analysis and quality assurance, wherein accuracy on semantic extraction is a critical task. It thus validates the significance of knowledge retrieval performed using embedding in domain-rich software docs.

According to the official technical documentation available in [5], the_advantage_of_LlamaIndex is the excellent unified interface it offers when linking the LLM with various data sources such as PDF files, HTML pages, fields, or databases. The indexing and node reference functions of the software guarantee traceability of the extracted information to the initial requirement source.

Moreover, the documentation on the generative capabilities of Gemini, a generative AI, mentions its strong long-context understanding and formatting capabilities for structured output [6]. This is particularly useful for automatically generating test cases, as it also supports boundary and positive and negative flows. The application of prompt engineering techniques on Gemini-like LLMs has been demonstrated to boost domain-specific information transformation tasks, like test case matrices generated from textual requirements.

Studies on the application of NLP and ML for converting requirements to test cases also provide more information on the design of test cases through automation. Scholars such as Lee et al. [7] and Khan et al. [8] illustrated how rules and the use of ML can reduce redundancy and increase coverage efficiency for test generation. But the problem of ambiguity or complex requirements, which was normally encountered by traditional NLP approaches, has been handled effectively by newer LLM-based reasoning systems.

There are discussions in the literature [9, 10] on vector store-supported search systems such as FAISS and Chroma, which demonstrate efficient and scalable search capabilities in embedding-space search. Together with hybrid ranking techniques, Docling uses related concepts in LlamaIndex to select appropriate contexts before the formulation of test cases.

Some recent work on software testing also illustrates the value addition of automation regarding the reduction of dependence on human resources, achieving better consistency, as well as shortening the cycle time for software release [11][12]. The current trend in organizations is towards the application of Generative AI-based Quality Assurance (GAI-QA), where the use of automated semantic reasoners helps human testers, owing to the increasing complexity of the system being developed, as well as the shortening cycle time. Docling finds a place here, guaranteeing a proper understanding of the documents, using smart document processing.

In conclusion, from the research works above, it can be noted that there is a paradigm shift from traditional requirement-based test design to completely automated and retrieve-and-augment LLM solutions. Leveraging innovative solutions for PDF extraction, embedding-based retrievals, and LLM reasoning strategies, Docling uses these developments to offer a scalable and reliable tool for human engagement in smart test case generation.

III. METHODOLOGY

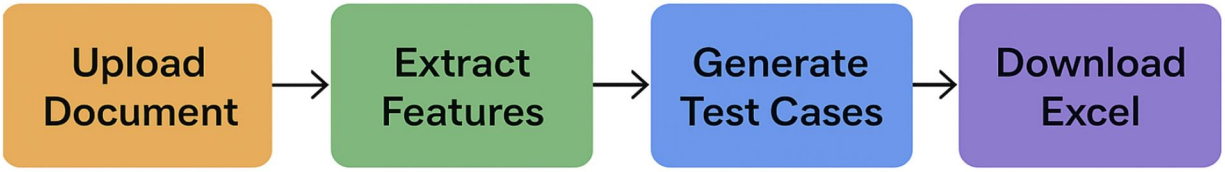


Fig.1 Project Workflow diagram

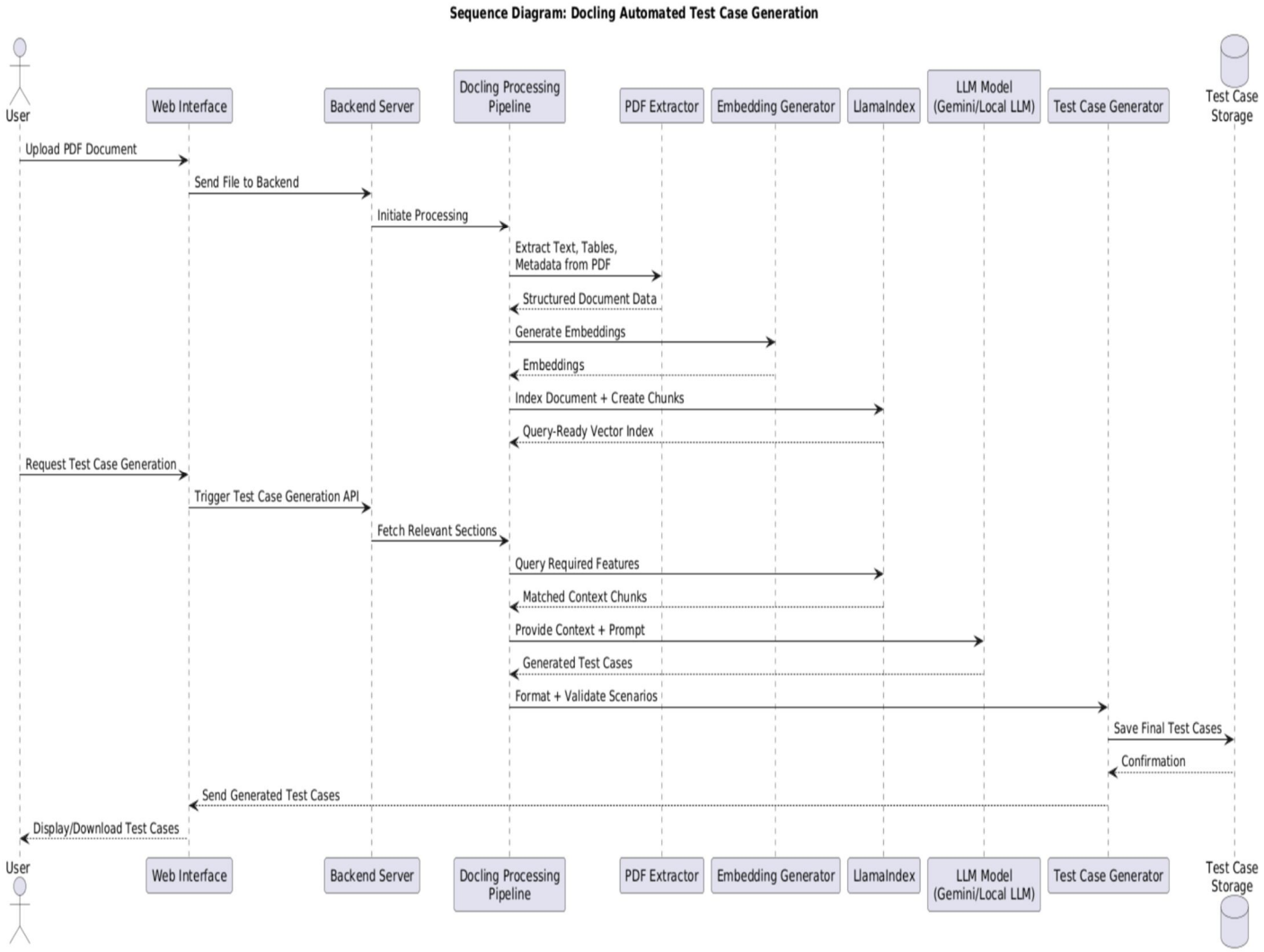


Fig.2 Project Sequence Diagram

A. System Architecture

"Docling Document Processing and Test Case Generation System": This is a modularly designed system. Its modules are the front-end, back-end services, "AI modules," and "vector search engine."

B. Document Processing Pipeline

1) User Input: The users upload a PDF/DOCX through the frontend or API.

2) Document Conversion: Docling will be converting the document to an intermediate structured format, retaining text, tables, images, and layout.

3) Text & Image Extraction:

- Text: Extracted, and optionally enhanced to Markdown.
- Images: high-resolution extraction, these are analyzed with Gemini AI to generate descriptive metadata necessary from image.

4) Feature Extraction: LLM analyzes the preprocessed document to extract software features, options, and requirements and will be stored in structured JSON.

5) Vector Indexing & Semantic Search: It will generate the Embeddings for all text sections, scenarios, and help documents. Cosine similarity retrieves the most common context for each feature.

6) Test Case Generation: For every feature, contextual prompts are built, based on retrieved content and few-shot examples. The structured test cases produced by Gemini AI contain the fields: ID, title, description, preconditions, steps, expected results, priority, and tags. These fields are dynamic and can be changed in prompt as per the user needs.

7) Excel Generation & Frontend Delivery: The test cases are combined into a well-designed Excel file, complete with formatting, headers, and frozen panes. Users can download the test cases and process summaries through the frontend or API.

C. AI & Machine Learning Components

LLM Integration:

- Gemini 1.5 Flash for text generation.
- Fallback Options: OpenAI models via gemini_service.py.

Optimization Strategies:

- The vector search cuts down the number of tokens per feature from 20,500 to 1,750 tokens.
- Rate limiting enforces API compliance.
- The use of the embedding cache improves performance in repetitive or batch document processing.

D. Mathematical Modeling and System Formulation

1) Feature Extraction Using LLMs

$$F = fLLM^{(D)} \quad \text{..... (1)}$$

Where:

- $fLLM$ represents the large language model (Gemini 1.5 Flash).

A LLM (Large Language Model) is used to model the feature extraction process as a transformation function. To determine software features and requirements , the LLM examines the structured document content , which includes text, tables and metadata. In order to facilitate traceability and serve as the basis for automated test case generation and downstream semantic retrieval , the extracted features are stored in structured Json format.

2) Embedding Generation for Semantic Indexing

$$e_i = \phi(t_i) \quad \text{..... (2)}$$

Where:

- ϕ(·) is the embedding model.
- d is the embedding dimension.

An Embedding model is used to convert each textual unit that was taken out of the document into a dense vector representation. In order to facilitate effective semantic retrieval during context construction, these embeddings , which are stored in a vector database using LlamaIndex, capture semantic relationships between document sections.

3) Semantic Similarity Computation

$$sim(e_q, e_i) = \frac{e_q . e_i}{||e_q|| \, ||e_i||} \quad \text{..... (3)}$$

Where:

- e_q is the embedding of a feature query.
- e_i is the embedding of a document section.
- $sim(.)$ measures semantic closeness.

Semantic relevance between extracted features and document sections is measured using cosine similarity .

By ensuring that only the most contextually relevant document fragments are retrieved, this similarity metrics minimizes hallucination during test case generation and passes less irrelevant information to the LLMs.

4) Context-Aware Test Case Generation

$$TC_f = gLLM(f, C_f) \quad \ldots\ldots (4)$$

Where:

- $gLLM$ is the test case generation prompt.
- TC_f denotes the set of test cases generated for feature f.

Semantically relevant document sections are combined with the feature description for each extracted feature to create context-aware prompt .
Based on user-defined schema, the LLM creates structured test cases with feeds like pre-condition, execution step , expected result , priority and tags that can be dynamically changed.

5) The Project's RAG Workflow

- Source of Knowledge :
  The external knowledge base is made up of the processed document content (text, tables, descriptions of images, and scenarios).
- Indexing :
  LlamaIndex is used to embed and store every document section in a vector index.
- Acquisition :
  The system uses semantic similarity to retrieve the most pertinent document fragments for each extracted feature.
- Generation :
  To create structured test cases, the retrieved context is injected into the LLM prompt.

6) AI & Vector Search Engine Algorithm :
Preprocessing Documents : Extract text, tables, images, and metadata from the uploaded document to create a structured representation.

- Extraction of Features :
  Utilize the large language model as described in Equation (1) to extract functional features and requirements from the structured document.
  The extracted features should be stored in a structured JSON format.
- Generation of Embedding :
  As stated in Equation (2), use the embedding model to create a semantic embedding for each extracted textual unit t in the document.

All embeddings should be kept in the vector database.

- Embedding Feature Queries :
  Create a matching query embedding for every extracted feature.
- Calculating Semantic Similarity :
  Using Equation (3), find the cosine similarity between each document embedding and the feature query embedding.
- Retrieving Context :
  To create the contextual set, extract the top-ranked document sections with the highest similarity scores.
- Context-Aware Creation of Test Cases :
  Using the LLM prompt formulation specified in Equation (4) and the retrieved semantic context, create structured test cases for every feature.
- Generation of Output :
  Combine all of the test cases that were created and export them to users in an organized Excel file.

IV. RESULTS

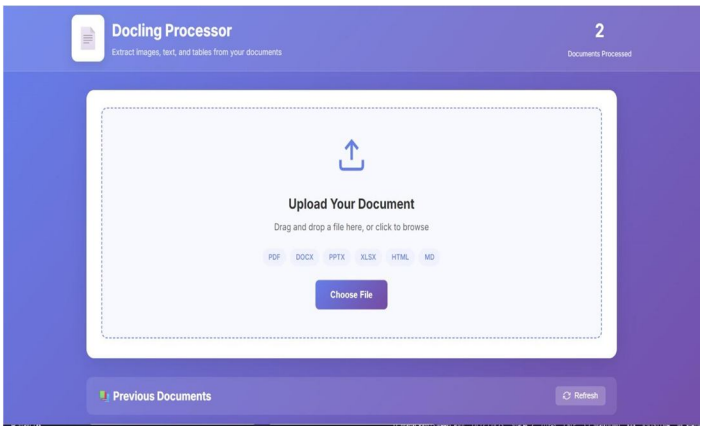


Fig.3 Document Upload Interface

The system automatically starts the processing pipeline as soon as a document is uploaded. The document preprocessing stage that gets the input ready for LLM-based feature identification modeled in Equation (1) is reflected in the execution stages that are shown to the user, which include structure extraction, text extraction, and image extraction.

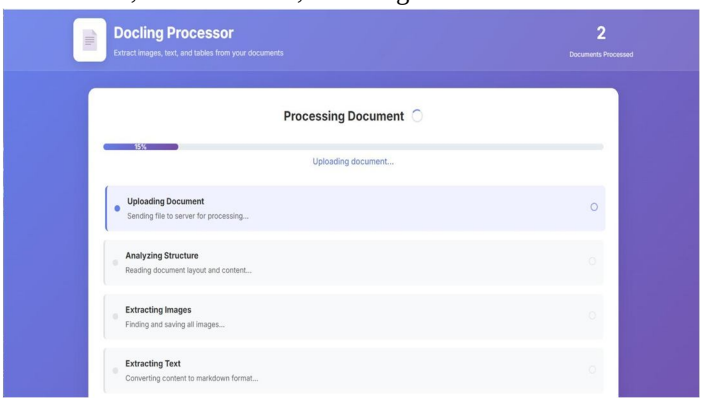


Fig.4 Document Processing

After preprocessing, the system uses the LLM transformation specified in Equation (1) to examine the extracted content and determine functional features and requirements. Equation (2) describes how the extracted textual units are transformed into semantic embeddings to facilitate effective semantic indexing. Together with the generated folder structure and output files, the interface shows a condensed view of the processed results, including the total number of extracted images, tables, and features. Green status markers are used to visually indicate successful extraction and processing stages, giving the user clear feedback.

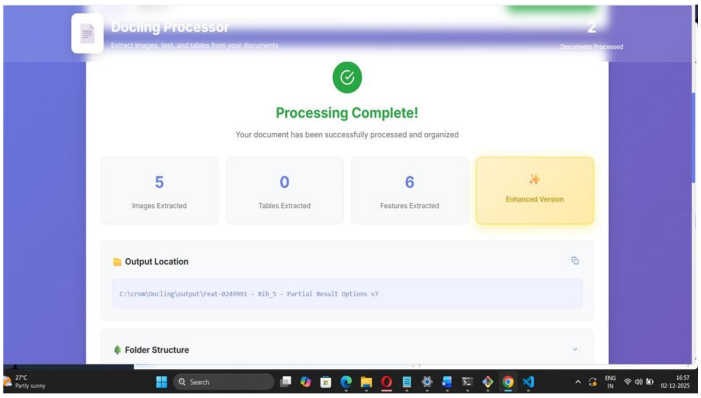
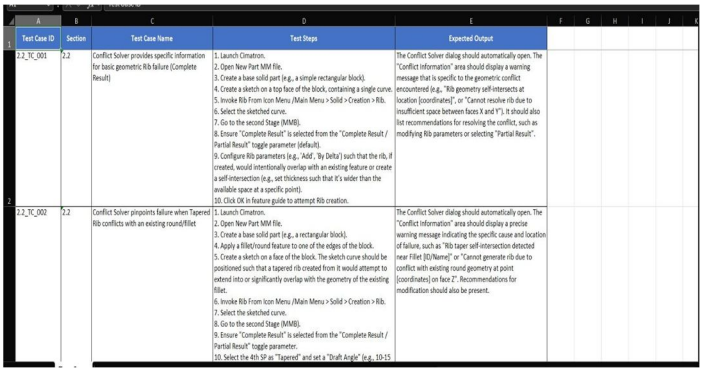


Fig.5 Process Completed

Equation (3) uses the cosine similarity formulation to calculate the semantic relevance between extracted features and document sections. This guarantees that only data that is pertinent to the context is chosen for further processing. The context-aware function defined in Equation (4) automatically generates structured test cases based on this filtered semantic context.



Test Case ID	Section	Test Case Name	Test Steps	Expected Output
2.2_TC_001	2.2	Conflict Solver provides specific information for basic geometric Rib failure (Complete Result)	1. Launch Cimatron. 2. Open New Part MM file. 3. Create a base solid part (e.g., a simple rectangular block). 4. Create a sketch on a top face of the block, containing a single curve. 5. Invoke Rib From Icon Menu /Main Menu > Solid > Creation > Rib. 6. Select the sketched curve. 7. Go to the second Stage (MMB). 8. Ensure "Complete Result" is selected from the "Complete Result / Partial Result" toggle parameter (default). 9. Configure Rib parameters (e.g., 'Add', 'By Delta') such that the rib, if created, would intentionally overlap with an existing feature or create a self-intersection (e.g., set thickness such that it's wider than the available space at a specific point). 10. Click OK in feature guide to attempt Rib creation.	The Conflict Solver dialog should automatically open. The "Conflict Information" area should display a warning message that is specific to the geometric conflict encountered (e.g., "Rib geometry self-intersects at location [coordinates]", or "Cannot resolve rib due to insufficient space between faces X and Y"). It should also list recommendations for resolving the conflict, such as modifying Rib parameters or selecting "Partial Result".
2.2_TC_002	2.2	Conflict Solver pinpoints failure when Tapered Rib conflicts with an existing round/fillet	1. Launch Cimatron. 2. Open New Part MM file. 3. Create a base solid part (e.g., a rectangular block). 4. Apply a fillet/round feature to one of the edges of the block. 5. Create a sketch on a face of the block. The sketch curve should be positioned such that a tapered rib created from it would attempt to extend into or significantly overlap with the geometry of the existing fillet. 6. Invoke Rib From Icon Menu /Main Menu > Solid > Creation > Rib. 7. Select the sketched curve. 8. Go to the second Stage (MMB). 9. Ensure "Complete Result" is selected from the "Complete Result / Partial Result" toggle parameter. 10. Select the 4th SP as "Tapered" and set a "Draft Angle" (e.g., 10-15	The Conflict Solver dialog should automatically open. The "Conflict Information" area should display a precise warning message indicating the specific cause and location of failure, such as "Rib taper self-intersection detected near Fillet [ID/Name]" or "Cannot generate rib due to conflict with existing round geometry at point [coordinates] on face 2". Recommendations for modification should also be present.

Fig.6 Generated Test Case Output

The excel file that has been exported contains structured test cases that have been created automatically. Various test case IDs, sections, test case descriptions, special execution instructions, and expected results are shown in each row because it indicates the benefit that can be gained from this tool, which converts document requirements to quality test scenarios.

V. CONCLUSION

In our paper, we propose the system called Docling for intelligent document processing as well as test case generation through the power of artificial intelligence. It effectively translates PDFs and word documents into structured form, identifies functional features in the document, and also develops test cases based on the semantic understanding of the document using the power of Large Language Models and semantic vector search.

Google's Gemini AI engine for feature extraction as well as image analysis and the powerful LlamaIndex Embeddings for optimal contextual search make Docling highly efficient for test case generation.

Modularity in structure and architecture, along with the responsive interface, efficient backend facilities, and intelligent AI components, is designed within the system to scope out scalability and adaptability capabilities within various document formats.

Future enhancements could also include batch processing, multi-language support, analytics, and distributed processing-continuing performance improvements and enhancing its applicability in a business environment.

A. Funding

This research did not receive grant from funding agencies in the public, commercial or non-for-profit sectors.

B. Data Availability

The data used in this study contain confidential information and cannot be publicly shared due to non-disclosure agreements.

C. Declaration of Competing Interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

REFERENCES

[1] Generating High-Level Test Cases from Requirements using LLM: An Industry Study — Masuda, S. et al., 2025. This work demonstrates how LLMs can generate high-level test cases automatically from requirement documents with no need for RAG.

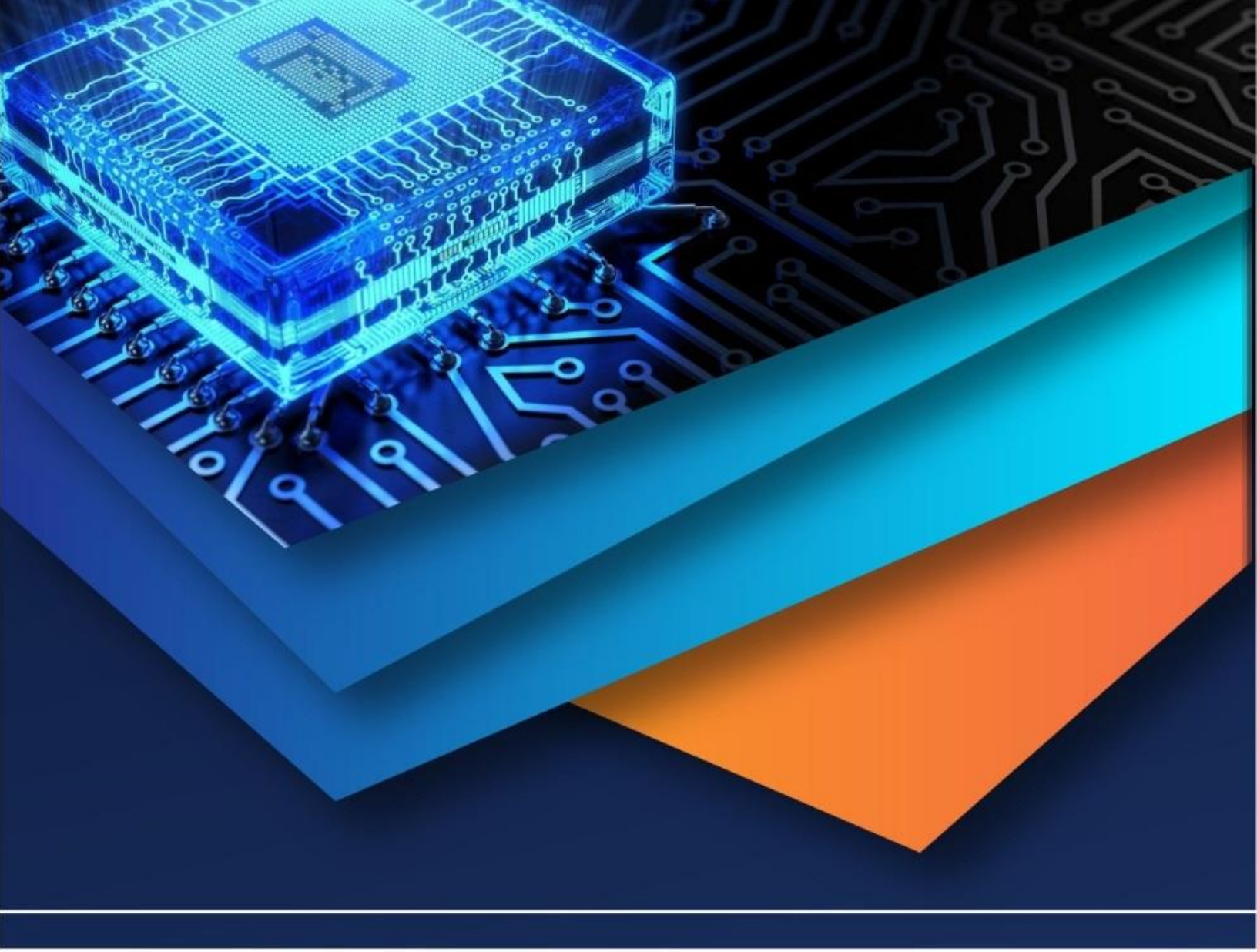
[2] A Review of Large Language Models for Automated Test Case Generation — Celik, A. et al., 2025. A survey of approaches using LLMs for test-case generation, summarizing benefits and limitations.

[3] KTester: Leveraging Domain and Testing Knowledge for More Effective LLM-based Test Generation — Li, A. et al., 2025. A framework that combines project-specific structure with testing-domain knowledge to guide LLM-based test generation.

[4] An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation — Schäfer, M., Nadi, S., Eghbali, A., Tip, F., 2023. Empirical results on LLM-based unit test generation across many real-world APIs.

[5] TESTEVAL: Benchmarking Large Language Models for Test Case Generation — Wang, W. et al., 2025. Introduces a benchmark suite to evaluate the test-case generation capability of LLMs by measuring coverage and correctness.
[6] BRMiner: Enhancing Automatic Test Case Generation by Extracting Relevant Inputs from Bug Reports by Ouédraogo, W.C. et al., 2025. This demonstrates how to combine LLMs with classical techniques to extract inputs and improve automated test generation.
[7] A method of test case generation for IoT devices using framework TCG-IoT by Kumar, S. et al. (2025) gives a good example of automated test-case generation within domain-specific context, which is useful for extending the applicability of the approach.
[8] Conversational Text Extraction with Large Language Models Using Retrieval-Augmented Systems by Roy, S. et al., 2025: This presents a pipeline using embeddings and RAG to extract text from PDFs via conversational interface, similar to document-processing tasks in Docling.
[9] Automation of data extraction from scientific literature and reports — Moreira-Filho, J.T. et al., 2025. This demonstrates the automation of document parsing and extraction workflows related to PDF/document ingestion tasks in Docling.
[10] LlamaIndex: Build powerful RAG pipelines (blog) — an official tutorial on how to integrate indexing of vector stores with embeddings and LLMs to enable document retrieval and generation.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue V May 2026- Available at www.ijraset.com



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY