

Bug Report Analysis Agent Using RAG | Lang chain

- THARUN.R

([https://huggingface.co/spaces/
TharunRavi/Bug-Report-
Analysis-Agent](https://huggingface.co/spaces/TharunRavi/Bug-Report-Analysis-Agent))

Index(Table of Contents)

S.No	Content	Page.NO
1	Introduction	2
2	Objective	2
3	System Architecture	2
3.1	Input	2
3.2	Preprocessing	3
3.3	Retrival	3
3.4	Generation	3
4	Implementation steps	3
4.1	Setup	3
4.2	Data upload	3
4.3	Embedding and Storage	3
4.4	Gradio Interface	3
4.5	RAG Pipeline	4
5	Evaluation	4
6	Technologies Used	4
7	Results	4
8	Challenges	4
9	Future works	5
10	Sample Outputs	5
11	Conclusion	9
12	Reference	9

1. Introduction

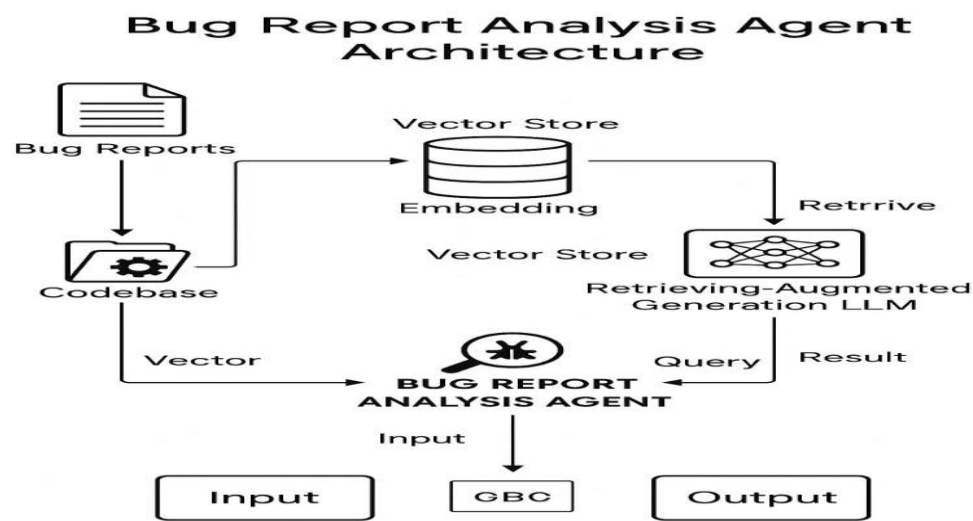
- ❖ In large software projects, bug tracking systems accumulate thousands of bug reports. Analyzing and resolving these bugs efficiently is a key challenge for developers. Traditional approaches require manual investigation, which is time-consuming and error-prone.
- ❖ This project introduces a Bug Report Analysis Agent that assists developers by retrieving similar past issues, relevant code snippets, and suggesting potential causes or fixes. The system leverages Retrieval-Augmented Generation (RAG), integrating natural language processing with retrieval methods to offer context-aware support for debugging.

2. Objective

- ❖ The main objective of this project is to build a tool that:
- ❖ Accepts a bug report as input.
- ❖ Retrieves similar historical bug reports.
- ❖ Retrieves relevant sections of the code base.
- ❖ Suggests potential causes or fixes based on context.
- ❖ Improves developer productivity and bug resolution time.

3. System Architecture

- ❖ The Bug Report Analysis Agent follows a modular architecture:



3.1. Input

- ❖ Bug Report CSV: Contains historical reports (ID, title, description, status, resolution).
- ❖ Codebase ZIP: Contains the software source code files.

3.2. Preprocessing

- ❖ The bug report text is embedded using a language model (e.g., all-MiniLM-L6-v2).
- ❖ The code base is split into chunks and vectorized.
- ❖ Both bug report and code embeddings are stored in a vector store (e.g., FAISS).

3.3. Retrieval

- ❖ Given a new bug report, similar bug reports and relevant code sections are retrieved using cosine similarity in the vector space.

3.4. Generation

- ❖ A generative model (e.g., GPT) takes the retrieved context and the new bug report to generate a suggestion for potential cause or fix.

4. Implementation Steps

4.1. Setup

- ❖ Install required libraries: sentence-transformers, faiss, langchain, gradio, openai, tqdm, etc.

4.2. Data Upload

- ❖ Upload bug reports (CSV) and codebase (ZIP) via UI.

4.3. Embedding and Storage

- ❖ Generate embeddings using a transformer model.
- ❖ Store vectors using FAISS index for fast retrieval.

4.4. Gradio Interface

- ❖ Upload section.
- ❖ Query input box.
- ❖ Display similar reports, related code, and model suggestions.

4.5. RAG Pipeline

- ❖ Use LangChain's RAG components:
- ❖ Retriever for fetching relevant documents.
- ❖ Prompt Template for guiding the model.
- ❖ LLM to generate suggestions based on input and context.

5. Evaluation

- ❖ To measure the system's effectiveness:
- ❖ Relevance of retrieved reports and code is assessed qualitatively.
- ❖ Usefulness of suggestions is manually rated (1–5 scale).
- ❖ Latency is measured for responsiveness.
- ❖ Initial testing shows that RAG improves relevance and contextual accuracy.

6. Technologies Used

- ❖ Language Models: all-MiniLM-L6-v2, gpt-3.5-turbo
- ❖ Libraries: LangChain, FAISS, Gradio, OpenAI API
- ❖ Platform: Hugging Face Spaces, Google Colab

7. Results

- ❖ The system successfully identifies semantically similar bug reports.
- ❖ Retrieves relevant code files even across large codebases.
- ❖ Generates useful debugging suggestions for most cases.
- ❖ Helps reduce developer investigation time.

8. Challenges

- ❖ Embedding large codebases requires optimization.
- ❖ Short bug descriptions are harder to match accurately.
- ❖ Generation quality depends on retrieval relevance.

9. Future Work

- ❖ Use more advanced models (e.g., CodeBERT, GPT-4).
- ❖ Add fine-tuning on historical fixes.
- ❖ Provide an upvoting mechanism for developer feedback.
- ❖ Expand to handle multi-modal bug data (e.g., stack traces, logs).

10. Sample Outputs:

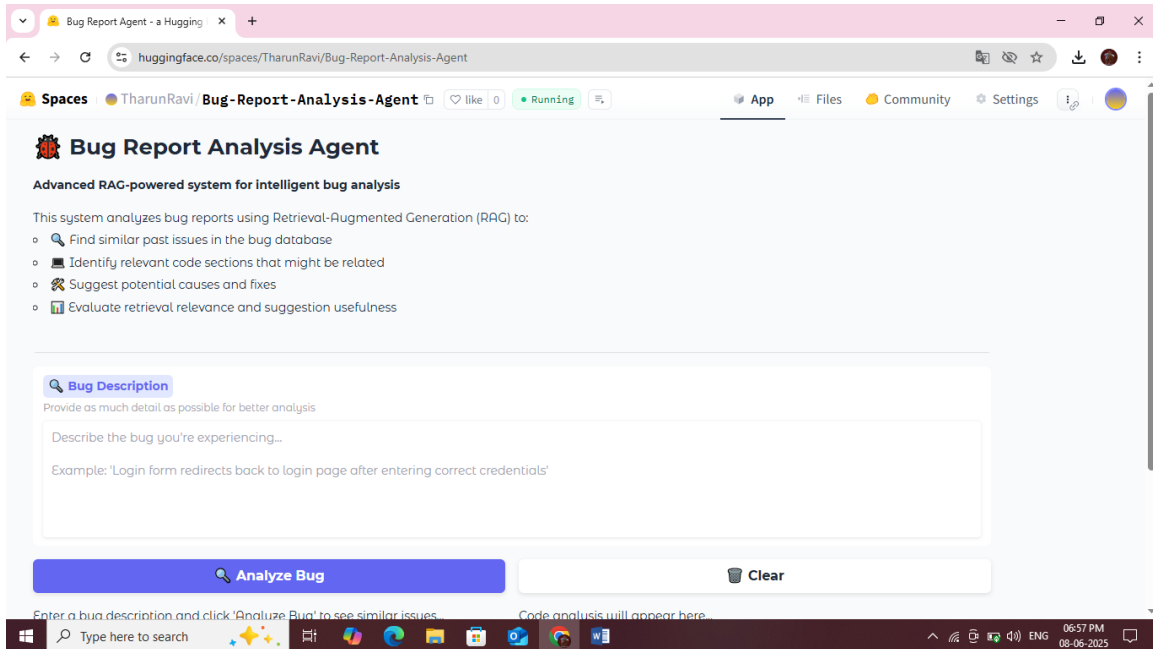
10. a) Files Creating or Uploading page

The screenshot shows the Hugging Face interface for the 'Bug-Report-Analysis-Agent' space. The page displays a list of files uploaded by the user TharunRavi. The files are listed in a table with columns for the file name, size, and upload time. The files are:

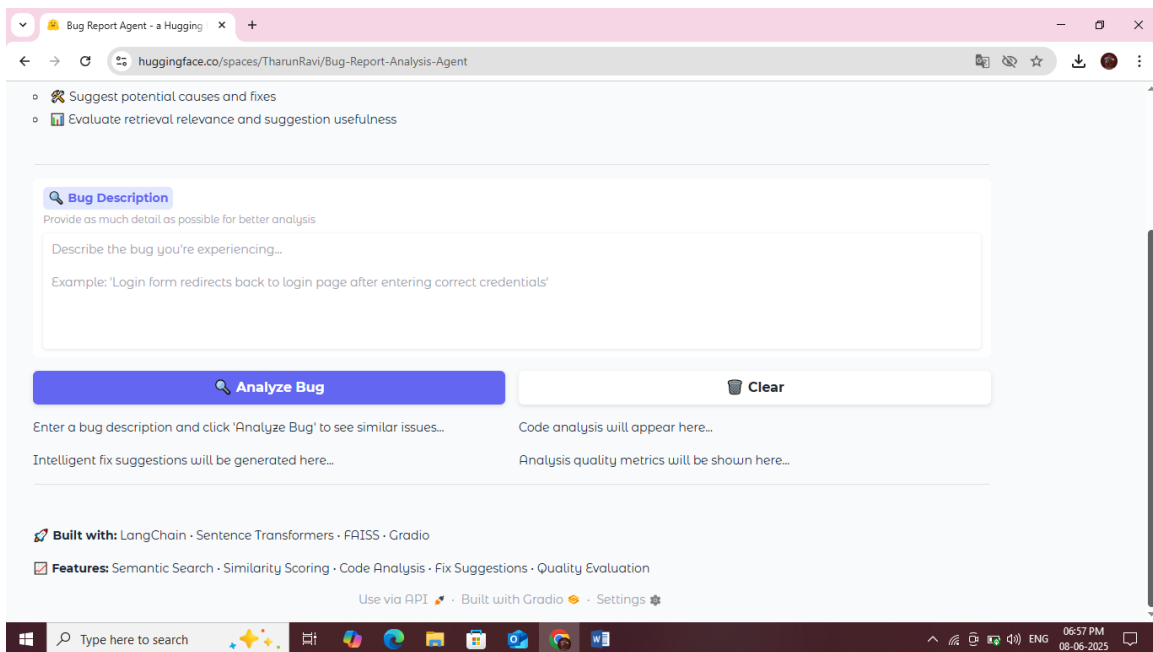
File Name	Size	Upload Time
.gitattributes	1.52 kB	about 1 hour ago
LICENSE	11.4 kB	about 1 hour ago
README.md	318 Bytes	about 1 hour ago
app.py	27.4 kB	about 1 hour ago
bug_reports.csv	128 Bytes	about 1 hour ago
evaluate_system.py	19.3 kB	about 1 hour ago
requirements.txt	165 Bytes	about 1 hour ago

The page also shows the Hugging Face logo, a search bar, and navigation links for Models, Datasets, Spaces, Community, Docs, and Pricing. The space is currently in a 'Running' state, and there are 2 commits in the history.

10. b) Processing Page (i)



10. b) Processing Page (ii)



10. c) Sample Input Page (i)

Bug Report Analysis Agent

Advanced RAG-powered system for intelligent bug analysis

This system analyzes bug reports using Retrieval-Augmented Generation (RAG) to:

- Find similar past issues in the bug database
- Identify relevant code sections that might be related
- Suggest potential causes and fixes
- Evaluate retrieval relevance and suggestion usefulness

Bug Description

Provide as much detail as possible for better analysis

Password reset mail not being sent to users

Analyze Bug **Clear**

Enter a bug description and click 'Analyze Bug' to see similar issues. Code analysis will appear here.

10. d) Sample Output Page (i)

Analyze Bug **Clear**

Found 5 Similar Bug Reports

Relevance Score: 0.46/1.0 Average Similarity: -136112938655411543924681673393806770176.00/1.0

1.

ID: | Severity: | Status: Similarity: 0.216 Component:

Description: Login fails with correct credentials

2.

ID: | Severity: | Status: Similarity: 0.127 Component:

Description: Error message is not displayed when login fails

3.

ID: | Severity: | Status: Similarity: 0.099 Component:

Description: Submit button does not respond

4.

ID: | Severity: | Status: Similarity:

10. d) Sample Output Page (ii)

The screenshot displays the 'Bug Report Agent' interface in a web browser. The main heading is 'Description: Error message is not displayed when login fails'. Below this, the 'Analysis Summary' section states the query: 'Password reset mail not being sent to users'. The 'Similar Issues Found' section lists three items, all marked with a severity of '****'. The 'Suggested Actions' section provides a list of seven steps for troubleshooting the login flow. The 'General Debugging Steps' section lists four steps for reviewing logs and testing environments. On the right, the 'Analysis Quality Metrics' section shows 'Retrieval Relevance' with an average similarity score of 0.136, 'Suggestion Quality' with a completeness of 0.145, and an overall quality rating of 'Poor (0.400/1.0)'.

Description: Error message is not displayed when login fails

Analysis Summary

Based on the query: 'Password reset mail not being sent to users '

Similar Issues Found

- **** (Status: , Severity:)
- **** (Status: , Severity:)
- **** (Status: , Severity:)

Suggested Actions

- Review authentication middleware setup
- Ensure proper error handling in login flow
- Verify SMTP server configuration
- Check password validation regex patterns
- Verify session management configuration
- Review email queue processing
- Ensure email credentials are properly set
- Check email template rendering

General Debugging Steps

- Review error logs and stack traces
- Test in different environments (dev/staging/prod)

Analysis Quality Metrics

- Retrieval Relevance**
 - Average Similarity Score: 0.136112938655411543924681673393806770176.000/1.0
 - Semantic Relevance: 0.456/1.0
 - Results Retrieved: 5
- Suggestion Quality**
 - Completeness: 0.145/1.0
 - Specificity: 0.000/1.0
 - Actionability: 1.000/1.0
 - Overall Usefulness: 0.344/1.0
- Overall Analysis Quality**
 - Quality Rating: ● Poor (0.400/1.0)

10. d) Sample Output Page (iii)

This screenshot shows the same interface as the previous one, but with a different layout. The 'Suggested Actions' and 'General Debugging Steps' sections are prominent on the left. The 'Analysis Quality Metrics' section on the right shows the same 'Poor' quality rating. At the bottom, there is a section for 'Built with' (LangChain, Sentence Transformers, FAISS, Gradio) and 'Features' (Semantic Search, Similarity Scoring, Code Analysis, Fix Suggestions, Quality Evaluation). The interface also includes a search bar and a status bar at the bottom.

Suggested Actions

- Review authentication middleware setup
- Ensure proper error handling in login flow
- Verify SMTP server configuration
- Check password validation regex patterns
- Verify session management configuration
- Review email queue processing
- Ensure email credentials are properly set
- Check email template rendering

General Debugging Steps

- Review error logs and stack traces
- Test in different environments (dev/staging/prod)
- Check recent code changes in related files
- Verify configuration settings
- Run relevant test suites
- Consider rollback if issue is critical

Analysis Quality Metrics

- Retrieval Relevance**
 - Average Similarity Score: 0.136112938655411543924681673393806770176.000/1.0
 - Semantic Relevance: 0.456/1.0
 - Results Retrieved: 5
- Suggestion Quality**
 - Completeness: 0.145/1.0
 - Specificity: 0.000/1.0
 - Actionability: 1.000/1.0
 - Overall Usefulness: 0.344/1.0
- Overall Analysis Quality**
 - Quality Rating: ● Poor (0.400/1.0)

Built with: LangChain · Sentence Transformers · FAISS · Gradio

Features: Semantic Search · Similarity Scoring · Code Analysis · Fix Suggestions · Quality Evaluation

Use via API · Built with Gradio · Settings

11. Conclusion

- ❖ The Bug Report Analysis Agent demonstrates how Retrieval-Augmented Generation can streamline software debugging. By leveraging past data and modern AI techniques, it significantly reduces the time required for understanding and resolving bugs. The system has potential to scale across enterprise codebases, making it a valuable tool for software development teams.

12. Reference

1. LangChain Library

- ❖ LangChain is used for building the Retrieval-Augmented Generation (RAG) pipeline, including document embeddings, vector search, and QA chains.

Reference:

> LangChain Documentation. <https://docs.langchain.com/>

2. HuggingFace Transformers

- ❖ Used to load the google/flan-t5-base model for text generation and reasoning.

Reference:

> Hugging Face Transformers. <https://huggingface.co/docs/transformers/>

3. FAISS (Facebook AI Similarity Search)

- ❖ For creating and searching the vector database for semantic similarity.

Reference:

> Johnson, J., Douze, M., & Jégou, H. (2017). Billion-scale similarity search with GPUs. FAISS. <https://faiss.ai/>

4. Gradio

- ❖ Used to build an interactive web-based user interface to accept bug report inputs and display output.

Reference:

> Gradio — Build Machine Learning Web Apps. <https://www.gradio.app/>

5. Model Used

- ❖ FLAN-T5 (Base): fine-tuned T5 model from Google for instruction-following tasks.

Model Link: <https://huggingface.co/google/flan-t5-base>

6. Python & Pandas

- ❖ Used for general scripting, data loading (bug_reports.csv), and preprocessing.

Reference:

> Python Software Foundation. <https://www.python.org/>

McKinney, W. (2010). Data structures for statistical computing in Python. Pandas. <https://pandas.pydata.org/>