EvalGuard AI: Continuous, Reference-Free Evaluation and Statistical Drift Monitoring

Samiksha Bandu Jadhav[1*], **Nafisa Jainuddin Patel**[1] and **Gauri.V.Sonawane**[2]

[1] B.Tech, Computer Science and Design
MET Institute of Technology, Nashik, India

[2] Assistant Professor
MET Institute of Technology, Nashik, India

* Corresponding author

Abstract

The EvalGuard AI system focuses on addressing the challenge of checking the quality of Retrieval-Augmented Generation (RAG) systems over time. A RAG system answers ques tions by first retrieving documents and then generating an answer with a Large Language Model (LLM). Its answer quality can fall silently when documents, models or settings change, and checking the answers by hand is slow and inconsistent. This project pro poses an automated system that evaluates the answers of a RAG pipeline, stores every evaluation run, and tells the user when the quality has drifted. The system is built in Python 3.11 with a FastAPI backend and a Streamlit dashboard, and it evaluates a fixed benchmark for each domain, across eight domains: Education, Healthcare, Banking and Finance, Legal, E-commerce, IT and Technical Support, Human Resources, and Travel and Hospitality. Document embeddings are created with Sentence Transformers and stored in the ChromaDB vector database, and answers are generated with the Gemini API or a local Ollama model. Every answer is scored on faithfulness, answer relevance, retrieval hit rate and latency, and every run is saved in an SQLite database. To find drift, the scores of two runs are compared with the Kolmogorov Smirnov (KS) test using NumPy and SciPy, and the result is reported as Stable or Drift. The output is presented on an interactive Streamlit dashboard with seven pages: Overview and Trends, Cross-Domain Benchmark, Run Deep-Dive, Drift Detection Lab, Live Play ground, Trigger Evaluation and Ingest Custom Docs. The dashboard shows scores, trends over runs and drift status in a simple visual form, so that developers and teams can find quality problems early without writing any code. This system reduces manual effort and provides data-driven insights for keeping a RAG application trustworthy.

Key Words: *Retrieval-Augmented Generation (RAG), RAG Evaluation, Drift Detection, Large Language Models (LLM), Faithfulness, Answer Relevance, Kolmogorov–Smirnov Test, Continuous Monitoring*

I. INTRODUCTION

Large language models (LLMs) can write fluent answers, but they sometimes state things that the source material does not support. Retrieval-Augmented Generation (RAG) reduces this problem by first retrieving passages from a document collection and then asking the model to answer from them[1]. RAG assistants are now built for study material, medical information, banking help desks and similar uses, where an unsupported answer can mislead a user.

Building such an assistant is only the first step. Its quality depends on the documents, the embedding model, the retriever, the language model and the prompt, and any of these can change after release. A document update or a model change can lower the quality of answers, and nobody notices until users complain. Judging answers by hand does not scale, and a one-time check does not detect a slow decline.

This paper presents EvalGuard AI, a framework that evaluates a RAG application *continuously*. A fixed benchmark of questions is run again whenever a change is made; every answer is scored automatically; the scores are stored with their history; and a statistical test decides whether the new scores differ from an earlier baseline. A dashboard presents the results so that a developer can use the tool without writing code.

The contributions of this work are deliberately scoped:

1) A reference-free scoring pipeline that rates each answer for faithfulness and relevance and also records retrieval hit rate and latency (Section VI).

2) A run history in SQLite, so that any two runs can be compared at any time.

3) A drift check that applies the Kolmogorov–Smirnov two-sample test to the score distributions of two runs and reports Stable or Drift for each metric.

4) A multi-domain benchmark setup and a seven-page Streamlit dashboard, including a cross-domain comparison and a live playground for single questions.

5) An honest report of the observed results, including weak ones, and of the limits of the evidence (Sections VIII and IX).

EvalGuard AI is a student reference implementation. It does not claim to solve RAG evaluation in general and does not claim to match human judgment. The rest of the

paper reviews related work, states the problem and gap, describes the architecture, methodology and implementation, reports the observations, and discusses limitations and future work.

II. BACKGROUND AND RELATED WORK

A. Retrieval-Augmented Generation

Lewis et al. combined a neural retriever with a sequence-to-sequence generator and showed strong results on open-domain question answering[2]. Dense retrieval, which represents questions and passages as vectors and finds the closest ones, is widely used for the retrieval step[3]. Sentence embeddings make this practical for short passages[4], and compact models such as MiniLM keep the cost low[5]. These works explain how a RAG system is built; they do not describe how to watch its quality after deployment.

B. Automatic Evaluation of RAG

Ragas proposes metrics that need no human-written reference answers, such as faithfulness, answer relevance and context relevance, which are computed with an LLM[6]. ARES trains lightweight judge models on synthetic data and reports accurate scoring with few human labels[7]. RAGChecker evaluates the retriever and the generator separately with fine-grained metrics[8]. RAGBench offers a large benchmark with explainable metrics across several industry domains[9], and a survey by Yu et al. organises what can be measured in a RAG system[10]. Using an LLM to grade other models is common, but it has known biases, for example towards longer answers[11]. These works supply the metrics and the benchmarks. They are mostly one-time evaluations and do not themselves keep a run history or raise an alert when quality changes.

C. Detecting Change in Data and Scores

Rabanser et al. study how to detect dataset shift and find that two-sample statistical tests applied to the data or to model outputs are simple and effective[12]. The Kolmogorov–Smirnov (KS) test is one of these: it compares the cumulative distributions of two samples without assuming their shape. EvalGuard AI applies this idea to the scores of RAG answers instead of to the input data.

D. Hallucination and Grounding

A language model can produce a fluent sentence that the source text does not support. This is usually called hallucination, and a survey of RAG evaluation lists it among the main quality risks that an evaluation has to expose[13]. RAG reduces the risk by giving the model passages to answer from[14], but it does not remove it: the retrieved passages may be missing the needed fact, or the model may add statements of its own. For this reason, checking whether each statement of an answer is supported by the retrieved text, which is the idea behind faithfulness metrics[15], is treated as a separate question from checking whether the answer is on topic. EvalGuard AI keeps the two apart and also records whether retrieval itself succeeded, so that a low score can be traced to the retriever or to the generator.

E. Metrics Used in the Literature

Table I lists the metric ideas that the reviewed work uses and the question each one answers. EvalGuard AI adopts the first two for every answer and adds the hit rate and the latency, which are cheap to record and useful for monitoring.

Table I: Metric Ideas and the Question They Answer

Metric	Question answered	Source
Faithfulness	Is the answer supported by the retrieved text?	16
Answer relevance	Does the answer address the question?	17
Context relevance	Is the retrieved text useful?	18
Retrieval hit rate	Was the needed information retrieved?	19
Latency	How long did the answer take?	Recorded here

F. Monitoring Learned Systems After Release

A model or pipeline that works well when it is built can behave differently later, because the data it sees or the components it uses change. The dataset shift study of Rabanser et al. examines methods that raise an alarm when such a change happens and reports that comparing distributions with a statistical test is a practical approach[20]. Their setting is general machine learning. EvalGuard AI carries the same

reasoning to a RAG application by treating the scores of the answers, not the raw inputs, as the quantity to compare between a baseline and a later run.

G. Related Work from the Supervisor's Group

Earlier work from the project supervisor's group applies learning-based methods to traffic signal control and traffic management[21], malware detection[22] and private data retrieval from partitioned data[23]. These studies are outside RAG evaluation, and we do not build on their methods. We cite them only as context: learned components used in decisions that matter need to be checked over time, which is the concern that motivates EvalGuard AI.

H. Positioning

Table II: Comparison of Related Approaches

Approach	Main capability	Limitation here
Ragas[24]	Reference-free metrics	No run history or drift check
ARES[25]	Trained judge models	Needs about 150 human labels
RAGChecker[26]	Retriever and generator metrics	Needs ground-truth answers
RAGBench[27]	Large multi-domain benchmark	Benchmark, not a monitor
Shift tests[28]	Detect distribution change	Not applied to RAG scores
EvalGuard AI	Scores, history and KS drift check	Small benchmarks; automatic scores

Table II summarises the approaches. Each solves one part of the problem. EvalGuard AI does not propose a new metric or a new statistical test. Its contribution is to combine reference-free scoring, a stored history, a standard drift test and a dashboard in one tool that can be run again after every change.

III. RESEARCH PROBLEM AND OBJECTIVES

A. Problem Statement

RAG applications answer from documents, yet their quality can decline silently when the documents, the language model or the settings change. Manual checking is slow,

costly and not repeatable, and most checks are done only once. The problem addressed here is to build a tool that (i) scores RAG answers automatically without human-written references, (ii) records speed and retrieval success together with answer quality, (iii) keeps the results of every run, and (iv) tells the developer, with a statistical test, when the quality of a new run differs from an earlier one.

B. Objectives

1) Score each answer for faithfulness and answer relevance without reference answers.
2) Measure retrieval hit rate and response latency for every question.
3) Run fixed benchmarks on several domains.
4) Detect a change in quality between two runs with the Kolmogorov–Smirnov test.
5) Store every run so that trends can be viewed and runs compared.
6) Provide one dashboard, including a live playground for single questions.

C. Requirements

The requirements that follow from the objectives are given in Tables III and IV.

Table III: Functional Requirements

ID	Requirement
FR1	The user can select a domain and run its benchmark
FR2	Each answer is scored for faithfulness and relevance
FR3	Retrieval hit and latency are recorded for each question
FR4	Every run and its question-wise scores are stored
FR5	The user can view the history and trends of runs
FR6	Two runs can be compared with the KS test
FR7	A single question can be tested in a live playground
FR8	The user can add own documents to a domain

Table IV: Non-Functional Requirements

Quality	Requirement
Usability	Use without writing code
Repeatability	Same benchmark and method in every run
Modularity	Dashboard, backend, storage and model replaceable
Security	API key kept out of code and dashboard
Portability	Runs on an ordinary laptop

IV. RESEARCH GAP AND MOTIVATION

The reviewed literature offers good metrics, benchmarks and statistical tests, but each addresses only one part. Metric frameworks score a run once; shift-detection studies deal with general data and not with RAG answer quality. What is less visible is a small, usable tool that joins automatic scoring, a stored history, a standard drift test and a dashboard so that a team can monitor a RAG application over time. Table V lists the practical gaps. They are drawn from the limitations of the surveyed approaches and not from a measured comparison.

Table V: Identified Gaps and the EvalGuard AI Approach

Gap	EvalGuard AI approach
Manual checking is slow and not repeatable	Automatic scoring of a fixed benchmark
Unsupported answers go unnoticed	Faithfulness score for every answer
Quality is checked only once	Every run stored with its history
Quality drops are hard to prove	KS test gives Stable or Drift
Speed and retrieval are tracked separately	Hit rate and latency recorded with quality

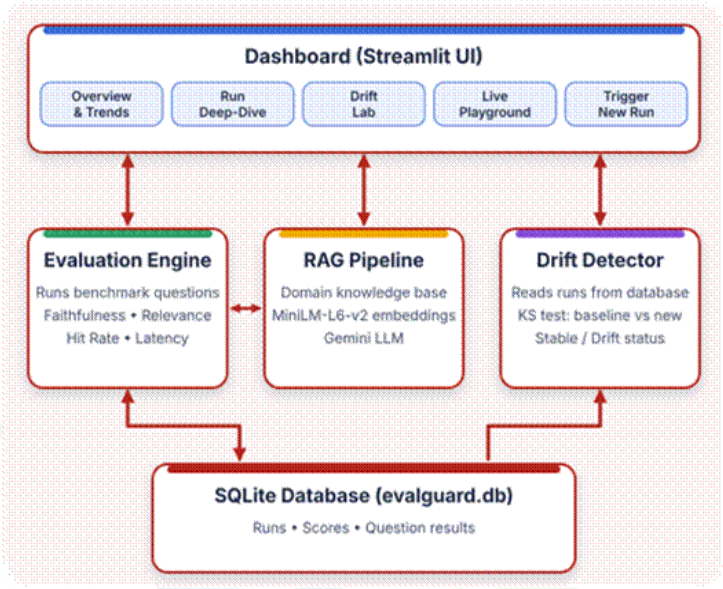
V. PROPOSED SYSTEM ARCHITECTURE

A. Overview

EvalGuard AI has a dashboard layer, a backend layer, a RAG pipeline and a storage layer (Fig. 1). The user works in a Streamlit dashboard. The dashboard calls a Python

backend built with FastAPI, which runs the evaluation, the metric calculation and the drift test. The RAG pipeline retrieves context from a ChromaDB vector store and sends the question and the context to a language model, which is either the Gemini API or a local Ollama model. SQLite stores every run, its scores and the question-wise results.

Figure 1: Overall architecture of EvalGuard AI.



B. Components and Responsibilities

The system has six modules. (1) The *dashboard* has seven pages: Overview and Trends, Cross-Domain Benchmark, Run Deep-Dive, Drift Detection Lab, Live Playground, Trigger Evaluation and Ingest Custom Docs. (2) The *evaluation engine* runs the benchmark of a domain and collects the scores. (3) The *RAG pipeline* embeds the question, retrieves context and generates the answer. (4) The *metric module* scores faithfulness, relevance, hit and latency. (5) The *drift detector* reads two stored runs and applies the KS test. (6) The *storage layer* keeps the runs in SQLite and the document vectors in ChromaDB. The layers are separate so that each can be replaced; for example, Ollama can replace Gemini without changing the dashboard.

C. Information Stored for Each Run

Table VI lists the information kept for every run. Keeping the question-wise rows is a design decision: a comparison of two means can hide a change in the spread of the scores, whereas the KS test uses all rows.

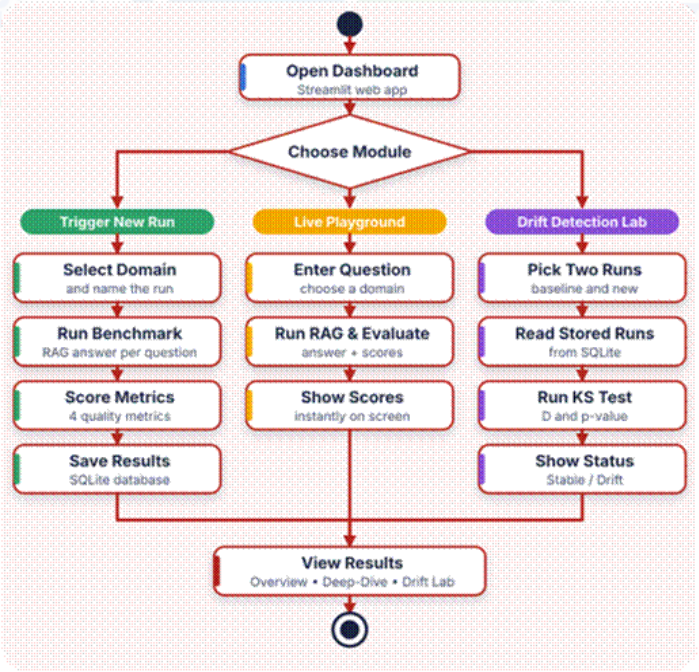
Table VI: Information Stored for Each Run

Level	Content
Run	Run name, domain, time of the run, mean faithfulness, mean relevance, hit rate, mean latency
Question	Question, retrieved context, generated answer
Scores	Faithfulness, relevance, hit and latency of that question

D. Operational Workflow

Fig. 2 shows the operational workflow as an activity view. A run starts when the user chooses a domain or types a single question. The benchmark questions are processed one after another by the RAG pipeline, every answer is scored, and the results are written to the database. When the user later opens the Drift Detection Lab and selects two runs, the drift detector reads their stored scores and reports Stable or Drift for each metric, which the dashboard displays. The evaluation and the drift check are separate actions, so a check can be repeated for any pair of runs without running the benchmark again.

Figure 2: Operational workflow of EvalGuard AI (activity view).



E. Design Decisions

Table VII records the main design decisions and the reason for each.

Table VII: Main Design Decisions

Decision	Reason
Reference-free scores	No human-written answers needed for each question
Store question-wise rows	Allows a distribution test, not only a mean
KS two-sample test	Simple and makes no assumption about the score shape
SQLite and ChromaDB	Need no separate server
Gemini or Ollama	Cloud or local model without changing the pipeline
Streamlit dashboard	Interface without separate front-end code

F. Deployment

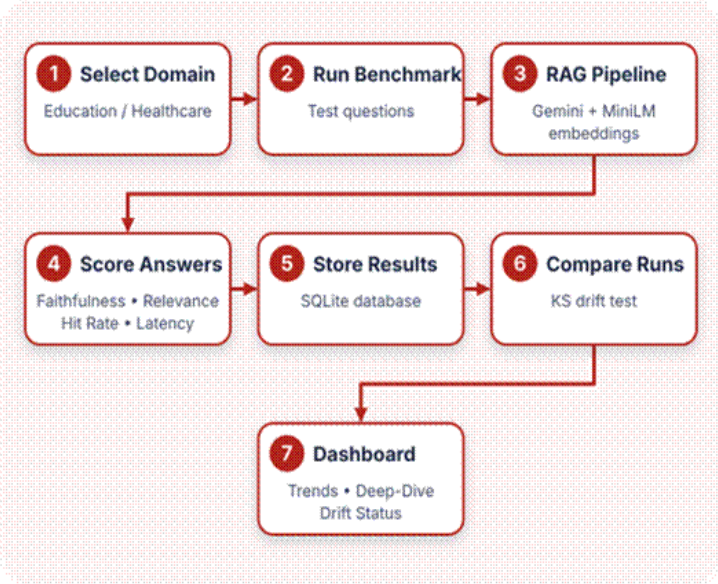
EvalGuard AI runs on an ordinary laptop. The Streamlit application is started from the command line and opened in a browser, and the backend, ChromaDB and SQLite run on the same machine. Only the Gemini API needs an internet connection; a local Ollama model removes this need. Answer generation therefore depends on the availability of the chosen model service.

VI. METHODOLOGY AND EVALUATION WORKFLOW

A. Evaluation Run

Fig. 3 shows the workflow of one run. The user selects a domain. The benchmark questions $Q=\{q_1,\ldots,q_n\}$ of that domain are taken one by one. For each question q_i the pipeline retrieves a context c_i from the knowledge base K of the domain, the language model writes an answer a_i from q_i and c_i, and the metric module computes the scores. The system is written as $S=\{I,P,O\}$, with input $I=\{Q,K,B\}$ (benchmark questions, knowledge base and baseline scores), process P (the pipeline above) and output $O=\{M,T\}$ (metric scores and drift status).

Figure 3: Workflow of one evaluation run.



B. Metrics

Four metrics are recorded for each question.

- **Faithfulness** f_i: the share of the statements in the answer that are supported by the retrieved context. With S the set of statements in the answer and $V \subseteq S$ the supported ones, $f_i = |V|/|S|$. A low value indicates unsupported content, that is, a possible hallucination.
- **Answer relevance** r_i: how well the answer addresses the question asked, on a scale from 0 to 1.
- **Retrieval hit** $h_i \in \{0,1\}$: whether the retrieved context contains the information needed for the question; the hit rate of a run is the mean of h_i.
- **Latency** t_i: the time taken for the question, in milliseconds.

The score of a run is the mean of each metric, for example $F = (1/n)\ \Sigma(i=1..n)\ f_i$. These scores follow the reference-free idea of Ragas[29]. Because an LLM may be used as a judge, its known biases apply[30], so the same judge and prompt are used in every run to keep runs comparable.

C. Evaluation Algorithm

The steps of one run are as follows.

1) Take the selected domain d and its benchmark questions $Q = \{q_1, \ldots, q_n\}$.

2) For each q_i, embed the question and retrieve the context c_i from the vector store of the domain.

3) Generate the answer a_i from q_i and c_i with the Gemini API or Ollama, and measure the time t_i.

4) Compute the faithfulness f_i and the relevance r_i of a_i, and record whether the retrieval was a hit h_i.

5) Save $(q_i,a_i,f_i,r_i,h_i,t_i)$ in the database.

6) After the last question, compute the mean of each metric and store the run with its name, domain and time.

Two stored runs can then be compared at any time, since the question-wise scores are kept and not only the means. This is what allows a distribution test and not only a comparison of averages.

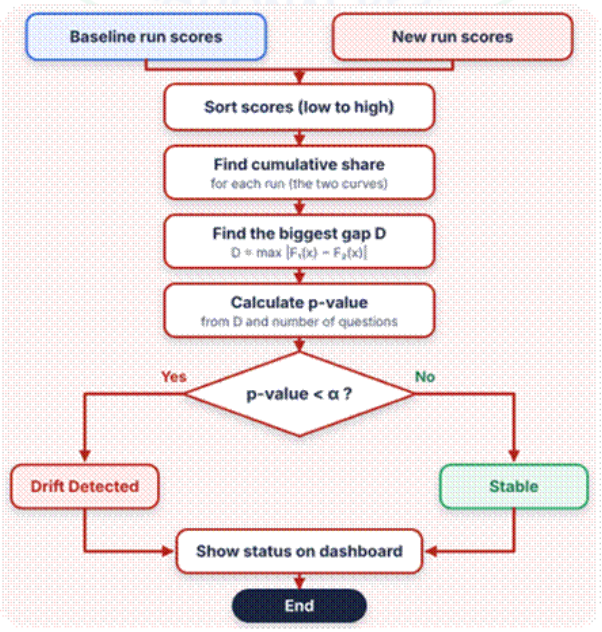
D. Drift Detection with the KS Test

To decide whether a candidate run differs from a baseline run, the scores of the two runs for one metric are treated as two samples. If $F_1(x)$ and $F_2(x)$ are their empirical cumulative distribution functions, the KS statistic is

$$D = max_x \mid F_1(x) - F_2(x) \qquad (1)$$

A p-value is obtained from D and the sample sizes, using SciPy. With significance level α, the rule is: if the p-value is below α, the metric is reported as *Drift*; otherwise it is *Stable*. Fig. 4 summarises these steps. The test is applied to every metric separately.

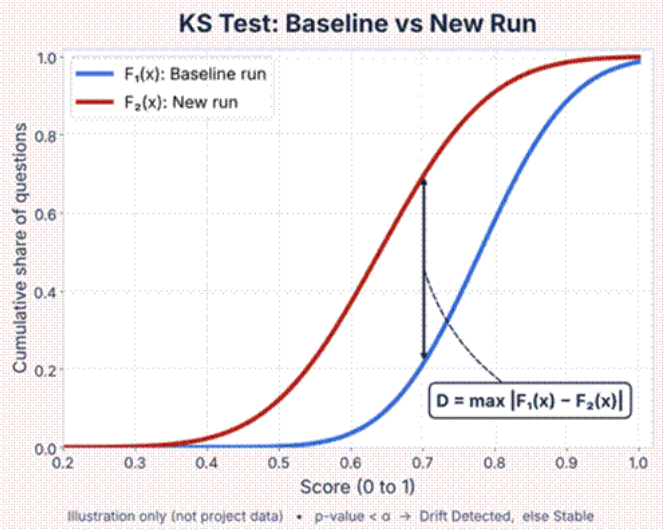
Figure 4: Steps of the KS drift check.



E. Illustration of the KS Statistic

Fig. 5 illustrates the idea with an invented example that is not project data. Suppose the faithfulness scores of a baseline run are mostly high and those of a later run are spread towards low values. When the scores of each run are sorted and the share of scores below each value is plotted, two cumulative curves are obtained. If both runs come from the same source, the curves lie close to each other, and the largest vertical gap *D* is small. If the quality has changed, the curves separate and *D* becomes large, which gives a small p-value. The test therefore looks at the whole shape of the scores and not only at their average. This matters because two runs can have the same average and still differ, for example when one run has all answers of medium quality and the other has some perfect and some very poor answers.

Figure 5: Illustration of the KS statistic (invented example, not project data).



A limit of the test should be stated. With only a few questions per run, the p-value is calculated from few samples, so a real change may not be detected, and a small run-to-run variation may occasionally look like a change. The p-value is therefore a guide for the developer and not a final verdict.

VII. IMPLEMENTATION AND TECHNOLOGIES

A. Technology Stack

Table VIII gives the stack. The backend is written in Python 3.11 with FastAPI, which exposes the evaluation and drift logic to the Streamlit dashboard. Embeddings are produced with Sentence Transformers; the model shown in the dashboard status box is all-MiniLM-L6-v2[31]. ChromaDB stores the document embeddings and returns the

closest passages for a question. Answers are generated by the Gemini API or a local Ollama model. SQLite stores runs, scores and question-wise results, and NumPy and SciPy provide the array handling and the KS test. Pytest is used to test the components.

Table VIII: Technology Stack of EvalGuard AI

Layer	Technology	Role
Frontend	Streamlit	Dashboard
Backend	Python 3.11, FastAPI	Evaluation logic, API
Main database	SQLite	Runs, scores, history
Vector database	ChromaDB	Document embeddings
LLM	Gemini API / Ollama	Answer generation
Embeddings	Sentence Transformers	Text to vectors
Drift test	NumPy, SciPy	KS test
Testing	Pytest	Component tests

B. Domains and Benchmarks

Eight domains are configured: Education, Healthcare, Banking and Finance, Legal, E-commerce, IT and Technical Support, Human Resources, and Travel and Hospitality. Each has a knowledge base and a benchmark of questions. The dashboard notes that the Healthcare, Banking and Legal evaluations use synthetic reference standards created for benchmarking, so high scores indicate retrieval and grounding quality and not any statutory approval. The Ingest Custom Docs page lets a user add own documents to a domain.

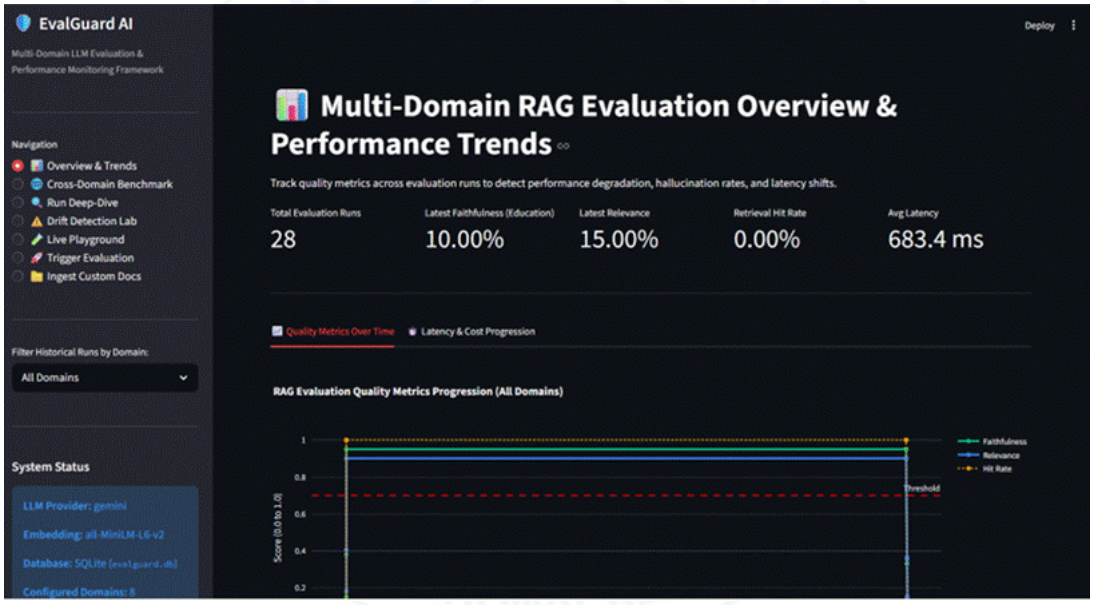
C. Dashboard

The Overview and Trends page (Fig. 6) shows the number of runs, the latest faithfulness, relevance, hit rate and average latency, and charts of the metrics across runs with a threshold line. The Cross-Domain Benchmark page compares all domains in bar charts with a target line. The Run Deep-Dive page lists the questions of one run with their individual scores, which helps to find the question behind a low score. The Drift Detection Lab (Fig. 8) selects a baseline and a candidate run and shows the status, the mean of each metric, the KS statistic and p-value, and the score distributions. The Live Playground scores a single typed question, and Trigger Evaluation starts a whole benchmark and saves the run.

Table IX: Dashboard Pages of EvalGuard AI

Page	Function
Overview and Trends	Run count, latest scores, trend charts with a threshold
Cross-Domain Benchmark	Bar charts of all domains with a target line
Run Deep-Dive	Question-wise scores of one run
Drift Detection Lab	KS statistic, p-value and status of two runs
Live Playground	Answer and scores for one question
Trigger Evaluation	Runs a whole benchmark and saves it
Ingest Custom Docs	Adds own documents to a domain

Figure 6: Overview and Trends page of the dashboard.



D. Hardware and Software Requirements

Table X gives the requirements for running the system. The hardware values are recommended values and not measured minimums.

Table X: Hardware and Software Requirements

Item	Requirement
Device	Laptop or desktop
Memory	8 GB RAM recommended
Storage	10 GB or more
Network	Internet access for the Gemini API
Software	Python 3.11 and the libraries of Table VIII

Item	Requirement
Browser	Any modern web browser

E. Testing Approach

The components are tested with Pytest, for example the metric calculation and the drift decision on small sets of values with known results. The functional behaviour of the dashboard was checked by using it, as listed in Table XII. The results of automated tests are not reported in this paper, because they have not been recorded for it.

F. Security and Data Handling

The API key of the Gemini service is read from the environment of the machine and is not written in the source code or shown on the dashboard. Stored runs stay in a local SQLite file. Because the answers depend on retrieved text, the benchmark and the documents of a domain are fixed during a run so that results are comparable.

VIII. EVALUATION, RESULTS AND DISCUSSION

A. Evaluation Approach

Evaluation was carried out on the running application through its dashboard. It was functional and descriptive and not a controlled study. We report values exactly as the dashboard displayed them. We did not measure agreement between the automatic scores and human judgment, we did not repeat runs to estimate variation, and we make no claim about accuracy against human raters.

B. Observed Results

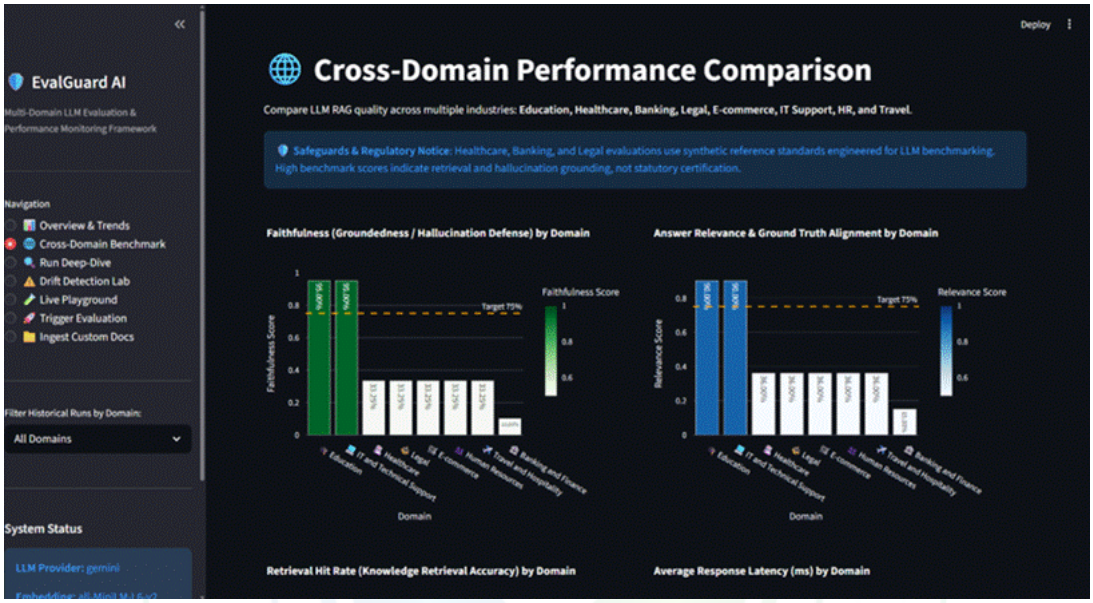
The Overview page showed 28 stored evaluation runs (Fig. 6). Its latest run showed a faithfulness of 10.00%, a relevance of 15.00%, a retrieval hit rate of 0.00% and an average latency of 683.4 ms. The Cross-Domain page (Fig. 7) showed that Education and IT and Technical Support scored 95% faithfulness and 90% relevance, five domains scored 33.25% and 36.00%, and Banking and Finance scored 10.00% and 15.00%. These values are summarised in Table XI.

Table XI: Scores Shown on the Cross-Domain Page

Domain group	Faithfulness	Relevance
Education; IT and Technical Support	95.00%	90.00%

Domain group	Faithfulness	Relevance
Five other domains	33.25%	36.00%
Banking and Finance	10.00%	15.00%

Figure 7: Cross-Domain Benchmark page: faithfulness and relevance by domain.



C. Functional Checks

Table XII lists the functional checks. Pass means that the behaviour is visible in the dashboard screenshots collected for this paper; To Test means that the check is defined but has not been recorded. We do not claim the To Test items as passed.

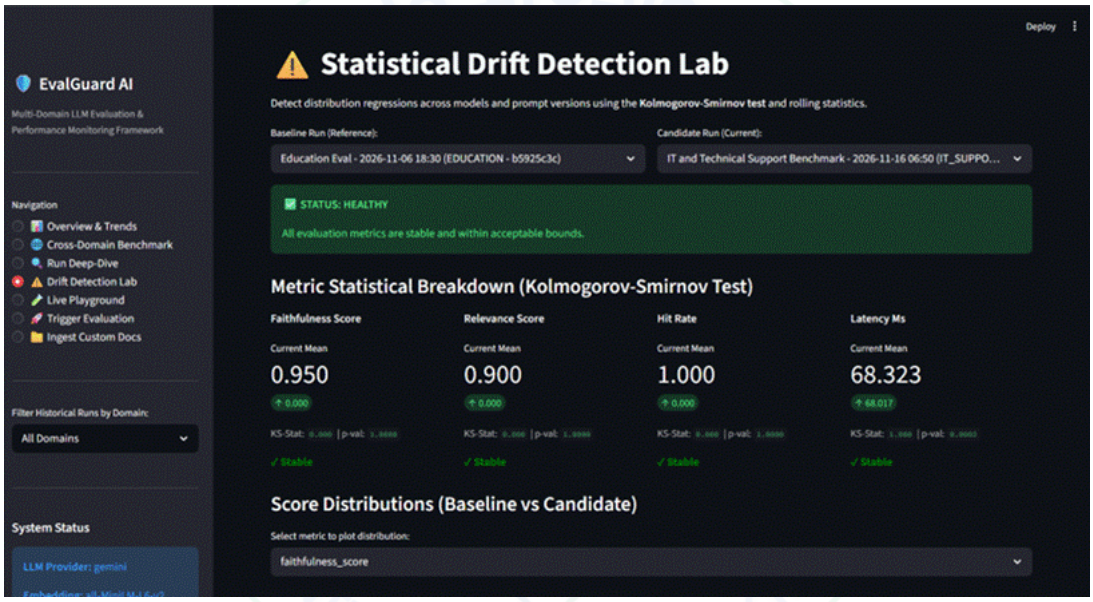
Table XII: Functional Checks and Their Status

ID	Check	Status
F1	Overview shows run count and latest scores	Pass
F2	Cross-domain charts for eight domains	Pass
F3	Drift Lab shows status, means, KS and p-value	Pass
F4	Live Playground accepts a question	Pass
F5	Playground returns answer with scores	To Test
F6	Trigger Evaluation saves a new run	To Test
F7	A clearly changed run is marked Drift	To Test
F8	Ingest Custom Docs adds a document	To Test

D. Drift Check Example

In the Drift Detection Lab (Fig. 8) an Education run was the baseline and an IT and Technical Support run was the candidate. The status was healthy. The mean faithfulness was 0.950, the mean relevance 0.900, the mean hit rate 1.000 and the mean latency 68.323 ms. For faithfulness, relevance and hit rate the KS statistic was 0.000 with a p-value of 1.0000. For latency the KS statistic was 1.000 with a p-value of 0.0002, and the metric was still displayed as Stable. Because the two runs come from different domains, this comparison shows how the test works; it is not a measure of quality change over time.

Figure 8: Drift Detection Lab comparing a baseline and a candidate run.



E. Discussion

Three observations follow. First, the benchmark separates domains clearly: two domains reach the 75% target, while others fall far below it, which suggests that their knowledge bases or retrieval need work. This is an observation from the dashboard and not a diagnosed cause. Second, the weak latest run on the Overview page shows why continuous monitoring is useful, because the drop is visible at once and not only after user complaints. Third, the drift example shows the mechanics of the test and the display, but a fair drift study needs two runs of the same domain before and after a real change.

The evidence supports the claim that the pipeline scores answers, stores runs and applies the KS test as designed. It does not support claims about the accuracy of the scores, about the causes of the low domains, or about the improvement that

monitoring brings in practice.

F. Interpreting the Scores

A faithfulness value near 1 means that almost all statements in the answers were supported by the retrieved text, while a low value means that many were not. A low relevance with a high faithfulness means that the answers were true to the text but did not address the question. A low hit rate means that the retriever did not bring the needed passage, so the generator had little to work from, and a low faithfulness is then expected. Reading the four metrics together is what lets a developer decide where to look first: the knowledge base, the retriever, the prompt or the model. The latency value tells whether a change made the system slower even when the quality stayed the same.

G. Domain-wise Observations

The Cross-Domain page shows that the results are not uniform across domains. Education and IT and Technical Support reach the 75% target line with 95% faithfulness and 90% relevance. Five other domains stay at 33.25% and 36.00%, and Banking and Finance is lowest with 10.00% and 15.00%. The equal values of the five domains suggest that they share the same condition, for example a knowledge base that is not loaded or a retriever that brings no useful context, but this is only a hypothesis. We have not checked it, and it should be examined before any conclusion is drawn about the domains. The value of the framework is that the difference is visible at all, because a team that looked at one domain only would not see it.

H. Applications

The framework can be used by a student team that builds a question-answering assistant on study material, by a team that answers medical or banking queries from documents and needs to check grounding, by a company that maintains a support assistant on product manuals, and by a developer who changes the language model, the embedding model or the prompt and wants to compare the new run with the old one before release. In each case the same steps apply: run the fixed benchmark, store the run, and compare it with the baseline.

I. Advantages

The main advantages are the following. The scoring is automatic and needs no human-written reference answers. The same benchmark and method are used every time, so

runs can be compared fairly. The history is stored, so quality can be followed over time. A statistical test gives a clear Stable or Drift result instead of an opinion. The question-wise view shows which question caused a low score. The tool is used through a dashboard and needs no code, and its parts can be replaced independently.

J. Threats to Validity

Several factors limit what can be concluded. The metrics are computed automatically, so their absolute values may differ from what a person would give. The benchmarks are written for this project, so the scores describe these questions and documents and not the domain in general. The KS test assumes independent samples, and the answers of one run share the same model and documents, which may not be fully independent. Finally, the observed values come from one installation and a limited number of runs, and they were not repeated.

IX. LIMITATIONS AND FUTURE SCOPE

A. Limitations

The scores are automatic estimates, and if an LLM is used as the judge, its bias applies[32]. The benchmarks are small, so the KS test has few samples per run, and its result is unreliable with few questions. Several domains scored low and the reasons have not been analysed. Healthcare, Banking and Legal use synthetic reference standards. The language model is an external service when Gemini is used, so speed and availability vary. Runs can be compared fairly only on the same benchmark. No agreement study with human raters, repeated runs or user study has been carried out.

B. Ethical and Safety Considerations

The Healthcare, Banking and Legal domains use synthetic reference standards created for benchmarking, and the dashboard says so. A high score in these domains shows how well the answers are grounded in the supplied documents; it is not a certification of medical, financial or legal correctness, and the system must not be used as a substitute for professional advice. When a user adds own documents, the user is responsible for the right to use them and for removing personal data before they are added.

C. Future Scope

The most valuable next step is a validation study that compares the automatic scores

with human ratings and repeats runs to measure their variation. Further directions are: larger benchmarks per domain; a context-relevance metric; correct handling of latency in the drift decision; automatic alerts when drift is detected; scheduled runs; comparison of different language models and retrievers; and corrective work on the domains that scored low.

X. CONCLUSION

EvalGuard AI shows how a RAG application can be watched continuously. It scores each answer for faithfulness and relevance without reference answers, records retrieval hit rate and latency, stores every run, and uses the Kolmogorov–Smirnov test to report whether the scores of two runs differ. The implemented prototype has clear limits: small benchmarks, automatic scores that are not validated against human judgment, and a descriptive evaluation. Within that scope it gives developers a simple, inspectable way to notice quality problems early and a base for systematic validation.

Endnotes

1. P. Lewis et al., “Retrieval-augmented generation for knowledge-intensive NLP tasks,” in Adv. Neural Inf. Process. Syst. (NeurIPS), 2020.

2. P. Lewis et al., “Retrieval-augmented generation for knowledge-intensive NLP tasks,” in Adv. Neural Inf. Process. Syst. (NeurIPS), 2020.

3. V. Karpukhin et al., “Dense passage retrieval for open-domain question answering,” in Proc. EMNLP, 2020.

4. N. Reimers and I. Gurevych, “Sentence-BERT: Sentence embeddings using Siamese BERT-networks,” in Proc. EMNLP-IJCNLP, 2019.

5. W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, “MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers,” in Adv. Neural Inf. Process. Syst. (NeurIPS), 2020.

6. S. Es, J. James, L. Espinosa-Anke, and S. Schockaert, “RAGAS: Automated evaluation of retrieval augmented generation,” arXiv:2309.15217, 2023.

7. J. Saad-Falcon, O. Khattab, C. Potts, and M. Zaharia, “ARES: An automated evaluation framework for retrieval-augmented generation systems,” arXiv:2311.09476, 2023.

8. D. Ru et al., “RAGChecker: A fine-grained framework for diagnosing retrieval-augmented generation,” arXiv:2408.08067, 2024.

9. R. Friel, M. Belyi, and A. Sanyal, “RAGBench: Explainable benchmark for retrieval-augmented generation systems,” arXiv:2407.11005, 2024.

10. H. Yu et al., “Evaluation of retrieval-augmented generation: A survey,” arXiv:2405.07437, 2024.

11. L. Zheng et al., “Judging LLM-as-a-judge with MT-Bench and Chatbot Arena,” in Adv. Neural Inf. Process. Syst. (NeurIPS), 2023.

12. S. Rabanser, S. Günnemann, and Z. Lipton, “Failing loudly: An empirical study of methods for detecting dataset shift,” arXiv:1810.11953, 2018.

13. H. Yu et al., “Evaluation of retrieval-augmented generation: A survey,” arXiv:2405.07437, 2024.

14. P. Lewis et al., “Retrieval-augmented generation for knowledge-intensive NLP tasks,” in Adv. Neural Inf. Process. Syst. (NeurIPS), 2020.

15. S. Es, J. James, L. Espinosa-Anke, and S. Schockaert, “RAGAS: Automated evaluation of retrieval augmented generation,” arXiv:2309.15217, 2023.

16. S. Es, J. James, L. Espinosa-Anke, and S. Schockaert, “RAGAS: Automated evaluation of retrieval augmented generation,” arXiv:2309.15217, 2023.

17. S. Es, J. James, L. Espinosa-Anke, and S. Schockaert, “RAGAS: Automated evaluation of retrieval augmented generation,” arXiv:2309.15217, 2023.

18. S. Es, J. James, L. Espinosa-Anke, and S. Schockaert, “RAGAS: Automated evaluation of retrieval augmented generation,” arXiv:2309.15217, 2023; D. Ru et al., “RAGChecker: A fine-grained framework for diagnosing retrieval-augmented generation,” arXiv:2408.08067, 2024.

19. D. Ru et al., “RAGChecker: A fine-grained framework for diagnosing retrieval-augmented generation,” arXiv:2408.08067, 2024.

20. S. Rabanser, S. Günnemann, and Z. Lipton, “Failing loudly: An empirical study of methods for detecting dataset shift,” arXiv:1810.11953, 2018.

21. G. Sonawane and S. Khan, “Designing a model on smart traffic control system,” Int. J. Innov. Technol. Exploring Eng., vol. 9, no. 3, 2020; G. Sonawane, P. Kote, and D. S. Khan, “Smart traffic control system based on neural network,” in Proc. 2nd Int. Conf. Commun. Inf. Process. (ICCIP), 2020; G. V. Sonawane, A. Y. Panpatil, R. R. Raskar, M. B. Sonje, E. Muniyandy, et al., “Topology-aware federated graph reinforcement learning for real-time edge-based adaptive traffic signal control in smart cities,” in Proc. 2026 Int. Conf. Innovations in Computing and Emerging [complete venue title], 2026; G. V. Sonawane, P. R. Patil, and P. R. Bhaladhare, “AI for real-time traffic management in communication networks,” Metallurgical and Materials Engineering, 2025; G. V. Sonawane, P. R. Patil, and P. R. Bhaladhare, “Neural network-based traffic control system,” J. Electr. Syst., vol. 20, no. 3, 2024, doi: 10.52783/jes.7946.

22. G. V. Sonawane, “Effective technique used for malware detection using machine learning,” Int. Res. J. Eng. Technol., vol. 6, no. 12, pp. 1102–1103, 2019.

23. G. V. Sonawane and N. Thoutam, “Design and analysis of three party protocol to retrieve private data by using vertical partitioned data,” Int. J. Electron., Commun. Soft Comput., 2015; G. V. Sonawane and N. Thoutam, “Design and analysis of two party protocol to retrieve private data by using vertical partitioned data—A review,” Int. J. Electron., Commun. Soft Comput., 2014; P. Kote, G. Sonawane, J. Khairnar, and K. Lata, “Analysis on efficient traffic detection system using machine learning techniques,” [publication details not available].

24. S. Es, J. James, L. Espinosa-Anke, and S. Schockaert, “RAGAS: Automated evaluation of retrieval augmented generation,” arXiv:2309.15217, 2023.

25. J. Saad-Falcon, O. Khattab, C. Potts, and M. Zaharia, “ARES: An automated evaluation framework for retrieval-augmented generation systems,” arXiv:2311.09476, 2023.

26. D. Ru et al., “RAGChecker: A fine-grained framework for diagnosing retrieval-augmented generation,” arXiv:2408.08067, 2024.

27. R. Friel, M. Belyi, and A. Sanyal, “RAGBench: Explainable benchmark for retrieval-augmented generation systems,” arXiv:2407.11005, 2024.

28. S. Rabanser, S. Günnemann, and Z. Lipton, “Failing loudly: An empirical study of methods for detecting dataset shift,” arXiv:1810.11953, 2018.

29. S. Es, J. James, L. Espinosa-Anke, and S. Schockaert, “RAGAS: Automated evaluation of retrieval augmented generation,” arXiv:2309.15217, 2023.

30. L. Zheng et al., “Judging LLM-as-a-judge with MT-Bench and Chatbot Arena,” in Adv. Neural Inf. Process. Syst. (NeurIPS), 2023.

31. W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, “MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers,” in Adv. Neural Inf. Process. Syst. (NeurIPS), 2020; N. Reimers and I. Gurevych, “Sentence-BERT: Sentence embeddings using Siamese BERT-networks,” in Proc. EMNLP-IJCNLP, 2019.

32. L. Zheng et al., “Judging LLM-as-a-judge with MT-Bench and Chatbot Arena,” in Adv. Neural Inf. Process. Syst. (NeurIPS), 2023.

© 2026 Assa'ada International Research Journal
This work is licensed under a **Creative Commons Attribution 4.0 International License (CC BY 4.0)**.
A Publication of Jamia Sa-adiya Arabiya, Sa-adabad, Deli, Kalanad (PO), Kasaragod, Kerala 671317, India