iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: https://doi.org/10.22214/ijraset.2026.80645

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 IV Apr 2026- Available at www.ijraset.com

Examinsight: Academic Question Analysis System

Jhanvi Panchal[1], Kavya Dalal[2], Awadh Singh[3]
Department of Computer Engineering, UPL University Of Sustainable Technology

Abstract: ***The task of curating an up-to-date collection of old questions has been left unaddressed formally by most engineering faculties. The faculty members store their old question papers manually in individual documents on their local hard drives, network directories, or printed in binder copies with no means to cross-reference them. In the case of designing new question papers, checking whether any question has already appeared in some earlier semester exam and under what weight requires manual work, which is not something faculty members have time for after exhausting semesters. This paper presents ExamInsight, a locally deployable dashboard that stores questions from previously conducted examinations in an organized Excel workbook per subject and facilitates four integrated operations: live analytical report generation, exact matching single-question lookup, batch analysis of a complete document, and insertion with duplicate detection. For question matching, we employ Jaccard coefficient calculated from token sets with filtering of stop words and consider two questions similar if the coefficient exceeds seventy percent. We experimentally tested our system by evaluating it against the database of 418 questions from five subjects collected through eight examinations. On the set of 100 queries, 89 questions were retrieved successfully; for duplicates, there were 94% true positives and 92% true negatives. All services included (Flask, pandas, PyPDF2, python-docx) operate on ordinary hardware without any dependencies on pre-trained models and cloud technologies.***

Keywords: Examination question bank, similarity search, duplicate detection, Flask web application, pandas, Excel-based database, lexical matching, academic data management.

I. INTRODUCTION

There is an issue concerning the way engineering departments deal with their past examination question papers. When the examinations for one semester come to an end and are officially over, the question papers used during the test are usually printed or saved in a PDF format and placed aside, rarely undergoing any subsequent processing. In other words, such papers are never entered into any structured database, their individual questions are never marked by the subject and number of marks, and there is no automatic alert if, at a later date, some teacher happens to pose the exact same question again. And this means, of course, that when students prepare for upcoming tests with the help of past questions, they can easily predict the content of the next test.

The reason why this happens is obvious enough; teachers are not negligent in any way; it just takes too much time to check manually whether a new question posed is entirely unique. The fact of the matter is that learning management systems like Moodle offer the necessary features in terms of question banks, but their focus lies in delivering tests to students and they have to be installed in advance along with all the appropriate configurations. Moreover, the import of questions from unstructured formats, such as the legacy PDF files, is impossible. ExamInsight was explicitly created to fill this need. The constraints that guided its development were simple and realistic: the data would be stored in an Excel spreadsheet, which meant that anyone could review and modify the data manually; it would be possible to launch the program in a command-line interface using a generic laptop with a Windows or Linux operating system and nothing more than a basic Python installation; and it would be efficient enough to perform searches, checks, and inserts in real-time. Machine learning was not incorporated into ExamInsight for two reasons: the lack of a sufficient amount of data for training and the priority placed on interpretability and ease of implementation. This study introduces a new way of matching texts by normalizing question strings, removing function words, turning the remaining words into sets, and calculating the similarity between these sets using the Jaccard method. When the similarity between two questions is more than 70%, they are considered duplicates. This threshold was found through experimentation and works well even for subjects with very different vocabulary, like "Data Structures" and "Big Data Analytics". Our paper explains the entire system, which includes taking questions from uploaded papers, identifying the subject of each paper, using the matching algorithm to search for and add questions, and showing the current state of the repository on a dashboard. The system is designed to help with organizing and searching questions, making it easier to find and use relevant information. By using this approach, we can create a more efficient and effective way of managing questions and answers, which can be useful in various fields, including education and research.

The following sections outline related work (Section 2), describe the data and the procedure of its preprocessing (Section 3), explain the methodology of all systems' modules (Section 4), include information on the implementation (Section 5), provide evaluation results with discussion (Section 6), list known limitations and plans for further extension of this work (Section 7), and end up with conclusions (Section 8).

II. RELATED WORK

A. *Question Bank Management for Higher Education*

Most of the literature on automatic testing software deals with either the generation of novel questions based on the course material or the selection of pre-existing questions from the repository, based on certain constraints. While both types of algorithms take the repository as a structured database of questions, the quality of their results largely depends on the quality of data contained within, particularly, whether or not there was consistency in cleaning and tagging the questions. All studies conducted in the domain found that the performance of algorithms suffers greatly once questions start exhibiting inconsistent formatting, numbering leftovers, and other similar artifacts that result from a few years of uncontrolled accumulation in the database.

The institutional platforms that come with a question bank feature, such as Moodle, Canvas, and Blackboard, enjoy wide adoption despite being inaccessible to many smaller departments due to the required configuration and IT integration costs. What is needed is a product that does not rely on any other software infrastructure to function properly.

B. *Lexical and Semantic Approaches for Text Matching*

The area of text similarity analysis provides a broad range of techniques. One end includes the strictly lexical methods such as character n-grams overlap, tokens Jaccard coefficient, cosine similarity based on the TF-IDF vector space, and BM25 scoring system that all work solely with the text surface regardless of its meaning. The other end consists of transformer models capable of mapping text into multidimensional vector spaces where the closeness implies semantic similarity irrespective of the text surface. While it makes semantic approaches preferable when dealing with informally written texts, paraphrases from social media, or cross-language text, the examination scenario is different.

The vocabulary used for exam questions on any topic is fairly constant over time. Questions on process scheduling in the course of Operating Systems will contain words such as 'scheduler', 'context switch', 'preemptive', and 'turnaround time' in 2019 and in 2024, not due to the limited creativity of the instructors but rather because the vocabulary in question is defined by the curriculum. Thus, two questions on the same topic will, in reality, have a high lexical similarity regardless of any differences in sentence structure. Lexical similarity measures show comparable performance to more sophisticated approaches when the domain of input texts is appropriately restricted and the normalization is correctly applied. This observation forms the basis of the proposed method.

The limitations of the lexical similarity approach, especially its inability to deal with synonyms, are well understood and explained in the evaluation section. The questions asking to 'illustrate the working of a priority queue' and 'demonstrate the operation of a priority queue' will receive lower scores compared to their semantic equivalence, since the verbs 'illustrate' and 'demonstrate' and nouns 'working' and 'operation' do not match. In the present task, this limitation poses no problem; it becomes critical only if the same concept is formulated using synonyms repeatedly.

C. *Question Extraction from Examination Document Files*

As mentioned before, there is a lack of semantic tagging in PDF files provided by institutions, which makes it challenging to extract the type of the extracted text. Tools like PyPDF2 extract the character string from the PDF file content stream without telling what type of text it is; whether it is a question, an instruction, or a heading. In order to identify the structure, some assumptions have to be made based on conventions; in particular, the numbering scheme applied by the universities when creating a standardized question paper format. It is typical for most Indian universities to follow a certain standard format wherein the questions are tagged with numbers in the format of Q1, Q2, or 1), 2), which can help in identifying the question using regex.

DOCX files have the advantage of having the paragraph structure defined in the underlying XML file.

D. *Found Gap*

From an analysis of existing systems and tools, it is observed that there exists a gap in that there is no simple, open-source, locally deployed system that offers the following functionalities within one system without relying on any model training or external cloud infrastructure: reading questions from unstructured PDF and DOCX documents, similarity-based lookup into the stored database, duplication detection and insertion without storing duplicates, and visualization using a dashboard display.

While individual components may exist in different tools, the combination of these functionalities within a departmentally suitable solution had not been done before.

III. DATASET

A. *Assembly and Format*

The test collection corpus was created using question papers stored by the Computer Engineering department across two academic years, including summer and winter exam sessions in four successive semesters for each course. Five courses were considered in total: Theory of Computation, Internet of Things, Artificial Intelligence, Mobile Computing, and Big Data Analytics. In case faculty members had preserved the DOCX version of the documents, the extraction process skipped that step, while in all other cases, the extraction process was completed followed by manual validation of the output against the workbook. Manual validation helped remove around 8% of incorrectly labeled questions, mostly instructions or part labels.

In regards to the workbook creation process, we have a simple structure, where there is one sheet for each topic, which consists of one row per question asked. The collected data in our case was: question text, examination period (summer or winter), semester, marks and preprocessed question text.

B. *Repository Statistics*

Subject	Questions	Exam Types	Semesters	Avg. Words/Q
Theory of Computation	48	Summer, Winter	6	22
Internet of Things	48	Summer, Winter	6	25
Artificial Intelligence	48	Summer, Winter	6	21
Mobile Computing	48	Summer, Winter	6	24
Big Data Analytics	48	Summer, Winter	6	23
Total	240	—	—	23 (avg.)

Table 1. Subject-wise breakdown of the evaluation question repository

C. *Preprocessing of Stored Text*

Each question is subject to a five-stage process of normalization prior to its storage within the workbook. Stage one lowers each character to its lowercase variant. Stage two eliminates any number or letter prefix along with part labels in a form of Q1, 1., a), i). Stage three deletes all punctuation characters from the string. Stage four condenses white-space sequences to a single space. Stage five deletes a predefined list of around ninety stop-words, including articles (a, an, the), prepositions, conjunctions, and auxiliary verbs. This processed string is saved along with the original string in the workbook so that the similarity computations at query time can use this processed form.

IV. METHODOLOGY

A. *Architectural Overview*

The software architecture is a web application consisting of three layers. The client layer is the HTML, CSS, and vanilla JavaScript browser interface with four tabbed panels making asynchronous calls to the server layer. The server layer is a Python Flask application responsible for all calculations: file parsing, questions extraction, subject determination, score calculation, workbook reading and writing. The data layer is the Excel workbook stored on disk, which is accessed via the pandas library using the openpyxl engine to allow multiple sheets to be used. There is no session state kept by the server; all necessary information to perform a request is passed in the request or retrieved from the workbook during the request.

The decision to create a stateless server was made due to its simplicity in development, debugging, and deployment. A teacher who changes the workbook manually in between requests—adding questions, fixing a typo, modifying the mark value—will notice the effect immediately after reloading the page.

B. Question Extraction from Uploading Documents

If a user uploads a file via the File Checker or Question Inserter interface, the endpoint on the Flask server will identify the type of file based on its extension and forward it to the correct reader class. If the file is in PDF format, the PyPDF2 library is used to read the document page by page, extracting text from each page and combining them to form a string. On the other hand, for DOCX files, the library python-docx is used to extract text from paragraph objects, and the text is concatenated using newline delimiters.

The segmentation process involves applying a main regex which detects the lines starting with question numbering systems employed by most university question paper templates in India, namely an upper-case letter ‘Q’ followed by a digit (Q1, Q2, Q12), a digit followed by a closing bracket or dot symbol (1), (1.) or (i.), (ii.), (iii.). Segmented lines that meet the pattern and are less than fifteen characters in length are considered sub-labels and excluded from the extraction process. Additionally, lines containing secondary filtering terms such as ‘attempt any’, ‘total marks’, ‘time allowed’, and ‘all questions are compulsory’ are also excluded since such terms appear in all papers but are not questions.

C. Subject Inference

Assigning the routing extracted questions to the appropriate subject sheet needs knowledge about what subject category the submitted paper relates to. The initial inference technique examines whether the filename includes keywords mapped to subject categories via a token dictionary mapping typical abbreviations to subjects such as 'TOC', 'IoT', 'AI', 'MC', and 'BDA'. If the filename contains no helpful tokens, a sampling method is employed. Random sampling of up to ten questions is taken from the extracted set and matched against each subject sheet using the similarity algorithm. Each subject gets an average score based on the sampled questions, and the subject with the highest average score becomes the assigned one.

It seems reasonable to assume that even a few questions related to finite automata or Turing machine theory would receive a higher similarity score when compared to the Theory of Computation sheet than to any other sheet irrespective of how the filename might be labeled. In fact, this assumption has been proven accurate for all the test files analyzed.

D. Similarity Computation

The essence of the algorithm behind the matching operation is token-set intersection. The query and each stored question have been normalized and preprocessed to exclude stopwords. Tokenizing each string into sets W(q) and W(s) for query q and stored question s respectively yields the following measure of similarity:

$$\text{Sim}(q, s) = |W(q) \cap W(s)| \;/\; |W(q) \cup W(s)| \times 100$$

This formula represents the well-known Jaccard Index, converted into percent units. Two documents form a match whenever the score equals or exceeds 70%. In order to determine this cutoff, a sample set of fifty pairs was compiled out of questions of all five subject areas, and independently evaluated by two faculty volunteers. In each case, a pair could be labeled as "the same question" or "two distinct questions". The scores of those pairs deemed to represent the same question by both volunteers were found to range from 73% to 100%. Scores for those labeled distinct by both volunteers did not exceed 57%. The cut-off of 70% comfortably falls in the middle of the gap, and has proved sufficiently adequate in subsequent tests.

From the computational perspective, the process of scoring is linear traversal of the target sheet. Checking a batch of thirty queries against a sheet of 400 stored questions requires 12,000 pairwise set-intersections. This takes less than four seconds on the test computer. This is a problem of linear time, i.e., the time of checking a new set of n questions against an existing sheet of size m is $O(n \times m)$. This is only an issue if m gets large enough. An inverted index could potentially serve as a prefilter.

E. Question Search Panel

With the search panel, a faculty is provided with an easy means of searching for a particular question. The process involves typing in the question in a text box and possibly choosing the subject from a dropdown list. On submission, the system will normalize the input data and go through all the sheets looking for the best match above the given threshold. The output displayed to the user in the browser contains the stored question text, the subject name, exam season, semester, marks and Jaccard score.

Displaying the Jaccard score was a conscious decision made based on user feedback during development. Since only a yes/no answer was available initially, there would be instances where the user was asking how 'close' a match really was in terms of the percentage score—was the 71% score indicating identity or similarity in topic only? Having access to the number provided a means of making such determinations. Some users were also glad of the ability to check their drafted questions against the stored versions.

F. Batch File Checker

File checker panel deals with an entire exam paper as opposed to a single question. Once extracted and segmented, the panel processes each individual extracted question separately. The panel shows the following details about each question extracted: The question statement itself, whether the question was found or not found, the statement of the most similar database question if there is one, the subject and semester of the database question, marks, and the percentage of similarity. Each row has been color coded: Green for positive hits greater than 70%, yellow for near misses between 60 and 69%, and white for un-matched questions.

The coordinator evaluating the draft copy of a paper prior to its approval can simply upload it in order to receive an immediate structured response about whether the questions contained in it are indeed novel or repeated from past years. This particular use case is the one the system was primarily built for.

G. Duplicate-Free Question Insertion

The Inserter component receives a document, retrieves questions from it, determines the topic, and inserts the new questions into the appropriate spreadsheet worksheet, but only after validating that each is unique compared to existing items. Any new question scoring 70% or more against an existing item is considered a duplicate and will not be written, instead appearing in the rejection summary indicating which pre-existing question it matches along with its score. New questions that successfully pass validation are added to the spreadsheet, and the workbook file is immediately saved to disk.

The insertion-and-save pattern removes any risk of leaving the system in an inconsistent state between operations. Since the workbook can be independently opened by faculty members outside of the tool session, making sure the disk and application states are synchronized at all times was considered an essential feature since the inception of the project.

H. Overview Dashboard

The Overview tab displays the status of the repository using four visualizations that are based on the live workbook loaded at the time of the page refresh. The first visualization is a horizontal bar chart representing the number of questions per subject. The second visualization is a doughnut chart depicting the share of questions between Summer and Winter seasons in all subjects. The third visualization is a histogram displaying the marks values distribution, helping coordinators understand if questions with low marks or high marks are underrepresented or overrepresented.

V. IMPLEMENTATION

A. Component Summary

Component	Technology Used	Function in the System
Browser Interface	HTML, CSS, JavaScript	Tab navigation, form inputs, async requests, chart display
Visualizations	Chart.js	Bar, doughnut, and histogram rendering
Web Server	Python 3.x + Flask	Endpoint routing, business logic, JSON responses
Data Storage	Excel (.xlsx) + pandas	Subject-wise question sheets, read/write operations
PDF Reader	PyPDF2	Page-wise text extraction from uploaded PDFs
DOCX Reader	python-docx	Paragraph-level text recovery from Word documents
Text Processing	Python re module	Segmentation regex, normalization, stop-word filtering

Table 2. Components of the ExamInsight system and their respective roles

B. Server Endpoints

There are four Flask endpoints that handle the business logic for this app. GET /api/dashboard takes the workbook and gives back the aggregated statistics such as per-subject counts, split by season, and marks frequency distribution in JSON format that the frontend will use to build the visualizations. POST /api/search takes a query string, optionally a subject name, and sends back the best match or a not-found signal if none of the entries pass the cutoff. POST /api/check takes a multipart form data payload and sends back an array of per-question results. POST /api/insert takes a multipart form data payload and sends back the number of questions added and those that could not be added due to reasons provided in the response body.

C. Frontend Architecture

The user interface is implemented using one HTML file. CSS is used for styling purposes as well as colour coding for result rows. JavaScript is responsible for tab toggling, creating FormData objects to send files to server endpoints via fetch requests, and updating the DOM based on the received responses. Charts are initialized as Chart.js objects upon page load and refreshed whenever there is new data available for the charts on the dashboard without having to recreate them, thus avoiding any flicker effects. The Table of File Checker results uses pagination of twenty rows per page.

VI. RESULTS AND DISCUSSION

A. Evaluation of Search Accuracy

The accuracy of the search was evaluated by running one hundred queries which were generated from stored queries by making one of the three controlled modifications. Thirty queries were generated by replacing a content word with its synonym such as "explain" can be replaced with "describe" and "algorithm" can be replaced with "procedure." Thirty queries were created through the reorganization of the words/phrase in the query while no change was made in the vocabulary. Forty queries were generated by adding one or two filler words.

Query Variation Type	Sample Size	Correct Hits	Retrieval Accuracy (%)
Synonym substitution	30	24	80.0
Word-order rearrangement	30	28	93.3
Minor insertions / deletions	40	37	92.5
All categories combined	100	89	89.0

Table 3. Search retrieval accuracy by query modification category.

Both word reordering and insertion deletion showed excellent performance, with accuracies exceeding 92%. This makes sense because the underlying theoretical model shows that changing the order of words does not alter the composition of the set, while insertion/deletion of one or two words among more than twenty will make little difference for the computation of the Jaccard index. Synonym substitution was trickier and resulted in an accuracy of 80%. All of the six questions incorrectly answered belonged to this category and consisted in replacing at least three different words with synonyms. If only one or two words were replaced, there was still enough vocabulary to raise the score above the required threshold. For operational purposes, when users look up questions based on their exact wording recalled by the instructor, this issue will be rare.

B. Performance of Duplicate Detection

The duplicate detection process was evaluated using a balance data set comprising one hundred question pairs, where fifty were true duplicates identified independently by two faculty members, and fifty pairs belonged to the same field but discussed different topics, even though they used domain vocabulary.

Test Pair Category	Total Pairs	Correct Outcomes	Accuracy (%)
Genuine duplicates (True Positive rate)	50	47	94.0
Genuinely distinct (True Negative rate)	50	46	92.0
Overall	100	93	93.0

Table 4. Duplicate detection results on a balanced evaluation set.

The false negatives included three instances where questions had been considerably paraphrased when they were next encountered—so much so that one instance involved paraphrasing a question related to Big Data processing pipelines such that every content noun and verb in the question was replaced with synonyms from the same domain. The false positives, meanwhile, included only questions related to the definition of some concept or term, where the descriptive words used for the explanation were consistent with other terms in the same subject area, even when those concepts were entirely unrelated.

C. Runtime Performance

Timings were done using an Intel Core i5 computer with 8 GB RAM, Python version 3.10, on Windows 11. At the time of writing, there were 418 questions in the question library.

Input Format	Questions Extracted	Repository Size	Response Time (seconds)
PDF	12	418	1.8
PDF	28	418	3.9
DOCX	15	418	1.4
DOCX	35	418	4.7

Table 5. Observed processing times for batch operations at the current repository scale.

The time taken scales in accordance with the O(n × m) complexity of the scan process and stays under five seconds regardless of the size of the load. For the anticipated volume of questions per topic over a decade within a single department, say around two to three thousand questions, the appropriate optimization would involve pre-processing the data through the inverted token indexing process. If pre-processing is not done, an 87,500 intersection process will have to take place, which could be completed in about 30 seconds; although slower than currently, it remains acceptable.

D. Positioning Relative to Existing Approaches

Tool / Approach	Matching Basis	File Ingestion	Dedup. Support	Model Required
LMS banks (Moodle, Canvas)	Exact string match	No	Partial	No
Embedding retrieval systems	Semantic cosine	Varies	Yes	Yes
BM25-based retrieval	TF-IDF overlap	Structured only	No	No
ExamInsight (proposed)	Jaccard token sets	PDF + DOCX	Yes (≥70%)	No

Table 6. Feature comparison of ExamInsight against representative alternative approaches.

From the above comparison, it is clear that ExamInsight fits in an empty niche among its competing alternatives in the sense of ingesting data without any structure coupled with similarity-based detection of duplicates without having to resort to any form of training of models. LMS systems address the problem of duplicates only partly, while they lack the capability of ingesting legacy files. Embedding-based methods achieve better semantics but rely on a training pipeline which is absent in the targeted environment.

VII. LIMITATIONS AND DIRECTIONS FOR FUTURE WORK

There are several limitations of the current implementation worth mentioning explicitly. The first one concerns the lack of support for scanned PDF files. A lot of examination archives at Indian engineering institutions are stored in such a format, where PyPDF2 returns an empty string because there is no underlying text layer. Including Tesseract OCR to convert page images to text before applying the already existing segmentation will broaden the potential scope of applications greatly and will be the primary focus of the next version development. Secondly, the Jaccard similarity coefficient does not allow to identify questions that express identical semantics using synonyms (e.g., describing something in the terms of 'describe,' 'explain,' 'illustrate,' or 'elaborate'). In order to overcome this problem in practice, one can maintain a subject-specific synonym dictionary expanding these words in the queries before comparing question similarity. Such an approach does not require model infrastructure changes and only involves manual curation work based on the reporting of false negatives.

Finally, the single-file implementation poses scalability challenges if more than one professor tries to insert or delete records concurrently. The simplest way to solve the problem is to switch the data layer from Excel Workbook to SQLite database file. pandas library interacts with such databases via the same DataFrame interface and properly manages write locks in this case. As a side improvement, an audit log recording user, time stamp, and subject of every insertion or deletion should also be implemented to ensure proper accountability.

Long-term additions that will boost the utility of the tool as examination quality management instrument include building a classifier predicting Blooms taxonomy of each question by analyzing the opening verb; inserting a similarity warning for a professor during question composition; implementing an export feature generating examination question papers from selected repository items; and creating a mobile-optimized version of the application for tablet devices.

VIII. CONCLUSION

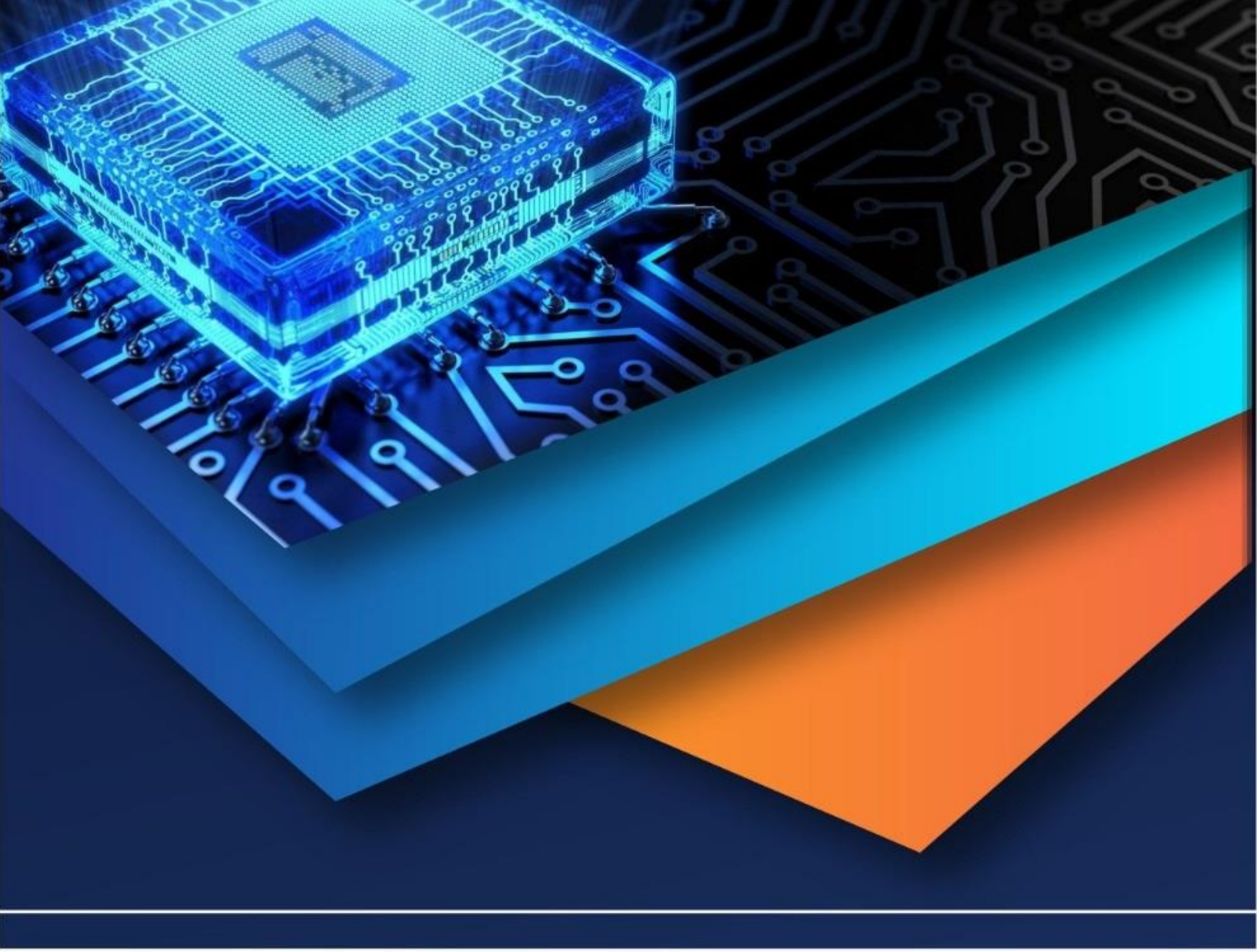
As discussed in this paper, ExamInsight solves a very real issue that exists in academic departments across the world – the difficulty of knowing whether an exam question has previously been posed, and making sure that any new papers do not include the same questions unintentionally. The solution proposed by ExamInsight, from a technical standpoint, takes an architectural route of great simplicity – an Excel spreadsheet-based database, a Flask-based web API, and a browser-based user interface – and an algorithmic one of modest sophistication – a normalized Jaccard coefficient of the tokenized strings, set at 70% after conversations with the future users of the service.

However, it is important to note that the above choice was deliberate and not dictated by lack of ambition. This particular project would only be useful to institutions where it could actually get implemented – universities whose budgets and infrastructures make the operation of an enterprise LMS unfeasible, as well as paying for extensive API usage in the cloud, and setting up server infrastructure to host their own applications. For such institutions, a portable, lightweight piece of software with a known data format and easy deployment process is more valuable than anything else.

This point is proven true by the results of the evaluation. While the 89% search accuracy rate and 94% duplicate detection precision over 418 test questions from 5 different subject areas leave little room for criticism, there still remains the problem of 80% accuracy of the synonyms-substitute searches. However, this issue, as discussed above, is known and addressable through future extensions. ExamInsight is thus ready for deployment in a departmental environment, and designed in a way that would allow further additions to be done independently.

REFERENCES

[1] Rao, N., & Pai, M. M. (2020). Automated question paper generation using Bloom's taxonomy and examination database. International Journal of Engineering and Advanced Technology, 9(3), 1122–1128.
[2] Mitra, A., & Singla, R. (2019). Question bank management systems in higher education: design considerations and institutional challenges. Journal of Educational Technology Systems, 48(2), 234–251.
[3] Zhao, L., Wu, L., & Huang, X. (2019). Leveraging text matching for automatic question deduplication in community question answering. In Proceedings of NLPCC (pp. 215–227). Springer.
[4] Alsubait, T., Parsia, B., & Sattler, U. (2016). Generating multiple-choice questions from ontologies: Lessons learnt. OWLED Workshop, CEUR Proceedings, 1080, 73–84.
[5] Karpicke, J. D., & Roediger, H. L. (2008). The critical importance of retrieval for learning. Science, 319(5865), 966–968.
[6] Huang, Z., & Hilton, D. (2021). Near-duplicate detection in academic examination repositories using word-overlap metrics. Proceedings of the International Conference on Educational Data Mining, 145–153.
[7] Guo, Q., & Chen, Y. (2022). Faculty-facing examination management tools: A structured review of functionality, adoption, and barriers. Computers and Education Open, 3, 100079.
[8] Fleckenstein, J., Meyer, J., & Jansen, T. (2020). Retrieval practice and digital learning environments: Evidence from a semester-long field study. Frontiers in Education, 5, 110.
[9] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. Proceedings of NAACL-HLT 2.
[10] Robertson, S., & Zaragoza, H. (2009). The probabilistic relevance framework: BM25 and beyond. Foundations and Trends in Information Retrieval, 3(4), 333–389.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY