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.2026.81771

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

A RAG-Based Approach for Automated Fake News Verification

Venna Chandra Sekhar Reddy[1], Khadri Syed Faizz Ahmad[2], Vodarevu Shaik Mohammad Kaif[3], Jannu Bhanu Prakash[4], J Sujan Kumar[5]
Department of Computer Science & Engineering, Acharya Nagarjuna University, India

Abstract: ***This paper presents a real-time fake news detection system based on a Retrieval-Augmented Generation (RAG) framework integrated with Large Language Models (LLMs). Traditional fake news detection approaches rely on static datasets and require frequent retraining, making them inefficient in dynamic information environments. To address this limitation, the proposed system leverages live data retrieval from Google News, enabling up-to-date and evidencebased verification.The system employs the all-MiniLM-L6-v2 SentenceTransformer model to generate semantic embeddings of retrieved news headlines and utilizes FAISS for efficient similarity search. By comparing user-provided claims with relevant news evidence, the system determines whether the claim is real or fake. Additionally, a caching mechanism is introduced to store previously retrieved results and model responses, significantly reducing redundant computations and improving response time.To further enhance performance, Particle Swarm Optimization (PSO) is applied to optimize the retrieval parameter (top-k), enabling the system to adaptively select the most relevant evidence. Experimental results demonstrate that the system achieves high accuracy, particularly when operating with a smaller number of retrieved documents. The proposed approach is computationally efficient, scalable, and capable of delivering near real-time predictions, making it suitable for practical fake news detection applications.***

Keywords: Fake News Detection; Retrieval-Augmented Generation (RAG); Large Language Models (LLMs); Semantic Similarity; FAISS; Sentence Transformers; Real-Time Information Retrieval; Particle Swarm Optimization (PSO); Google News RSS; Cache Optimization; Fact-Checking Systems; Natural Language Processing (NLP)

I. INTRODUCTION

The ever-increasing volume of misinformation presents a considerable obstacle for today's digital world. The effect of fake news on public opinion, the stability of societies, and the processes by which individuals make decisions are all significant across all areas of society; for example, the effects of fake news are evident in politics, healthcare, and finance. Therefore, with the amount of information published on the internet continuing to grow at an exponential rate, it is becoming increasingly important to verify the credibility of news that appears to be trustworthy.

Historically, traditional approaches to detect fake news have been limited to using supervised machine learning algorithms trained on fixed data sets, which may be useful in a controlled environment; however, these techniques have been found ineffective when working with developing falsehoods and changes to information patterns. Consequently, these metrics will need to be retrained regularly, making them prohibitively costly from a computational perspective on an ongoing basis and impractical for use in a real-time environment. To solve the issue described by a growing number of researchers, there has been a shift towards retrieval-augmented approaches that integrate external retrieval of evidence and better language model-based reasoning (e.g., large language models); therefore, this combined use of evidence from executed searches and language model reasoning produces improvement in the level of factual verification since all predictions can rely on the most up-to-date evidence (e.g., VeraCT Scan has created a retrieval-augmented framework for verifying the accuracy of factual statements, as this framework extracts claims from text, retrieves web-based evidence, and uses joint reasoning based on both the retrieved evidence and the predictive accuracy of the language model being used). Methods that use RAG, followed by LLM reasoning techniques, have shown a higher level of success in fact-checking than traditional techniques.

II. RELATED WORK

Recent work has increasingly shifted fake news detection toward retrieval-augmented and evidence-based methods. Sharma and Meena proposed a retrieval augmented generation classification algorithm for fake news detection, showing that retrieval can improve classification over static approaches.[5] Bing et al. extended this direction with RAG-augmented reasoning for political fact-checking using LLMs, introducing structured reasoning to support more complex verification tasks.[6] The IEEE Computer Society survey further shows that large language models and RAG-based methods are becoming central to modern fake news detection research.[7]

The foundations of these systems come from earlier retrieval-augmented language modelling work. Lewis et al. introduced Retrieval-Augmented Generation for knowledge-intensive NLP tasks, establishing the core retrieval-plus-generation paradigm.[8] Guu et al. proposed REALM, which integrates retrieval during pre-training to improve factual grounding and knowledge access.[9] For evidence-based verification, Thorne et al. introduced FEVER, a large-scale dataset that became a standard benchmark for fact extraction and verification.[10]

Building on these foundations, more recent studies have proposed advanced fact-checking frameworks. Lopez-Joya et al. presented a blueprint for a new fact-checking system, while H. Li et al. explored the use of retrieval-augmented large language models for fact-checking.[11,12] Díaz García et al. introduced VERIFAID, a FAISS-based framework for fake news detection, and Y. Bai et al. proposed FCRV, which emphasizes full-context retrieval and verification for misinformation detection.[13,14] Dash et al. showed that evidence-backed fact checking with RAG and few-shot learning can improve claim verification by combining retrieved evidence with prompting.[15]

Overall, these works show a clear move from static classification toward dynamic retrieval, grounded reasoning, and evidence-based verification. This trend motivates our proposed RAGbased fake news detection framework, which aims to improve freshness, reliability, and explainability by leveraging external evidence.

III. RESEARCH GAP

The system I am proposing is a time fake news detection pipeline. It checks news claims provided by users by searching news comparing meanings using a large language model to reason storing data in memory for quick access and optimizing with a particle swarm method Of using a fixed computer program trained on old data my system searches Google News for each claim. It uses news headlines as proof to classify the claim. When a user submits a statement or headline it goes through a query simplification step. This step removes punctuation. Keeps only words that are longer than three letters. It keeps most five important words. The simplified query helps to get rid of words and reduces getting irrelevant results. To prevent doing the work twice the system stores results in memory. If the same query is used again it reuses the stored results of searching again. The simplified query is then sent to Google News to search for headlines. The response is in XML format. Is parsed to get the titles of the returned items. Currently only headline titles are. At most ten headlines are kept as evidence. These headlines are stored in a cache for news. This allows the system to quickly return headlines if the same query is used again. This reduces the time it takes and the load, on systems.

IV. PROPOSE METHODOLOGY

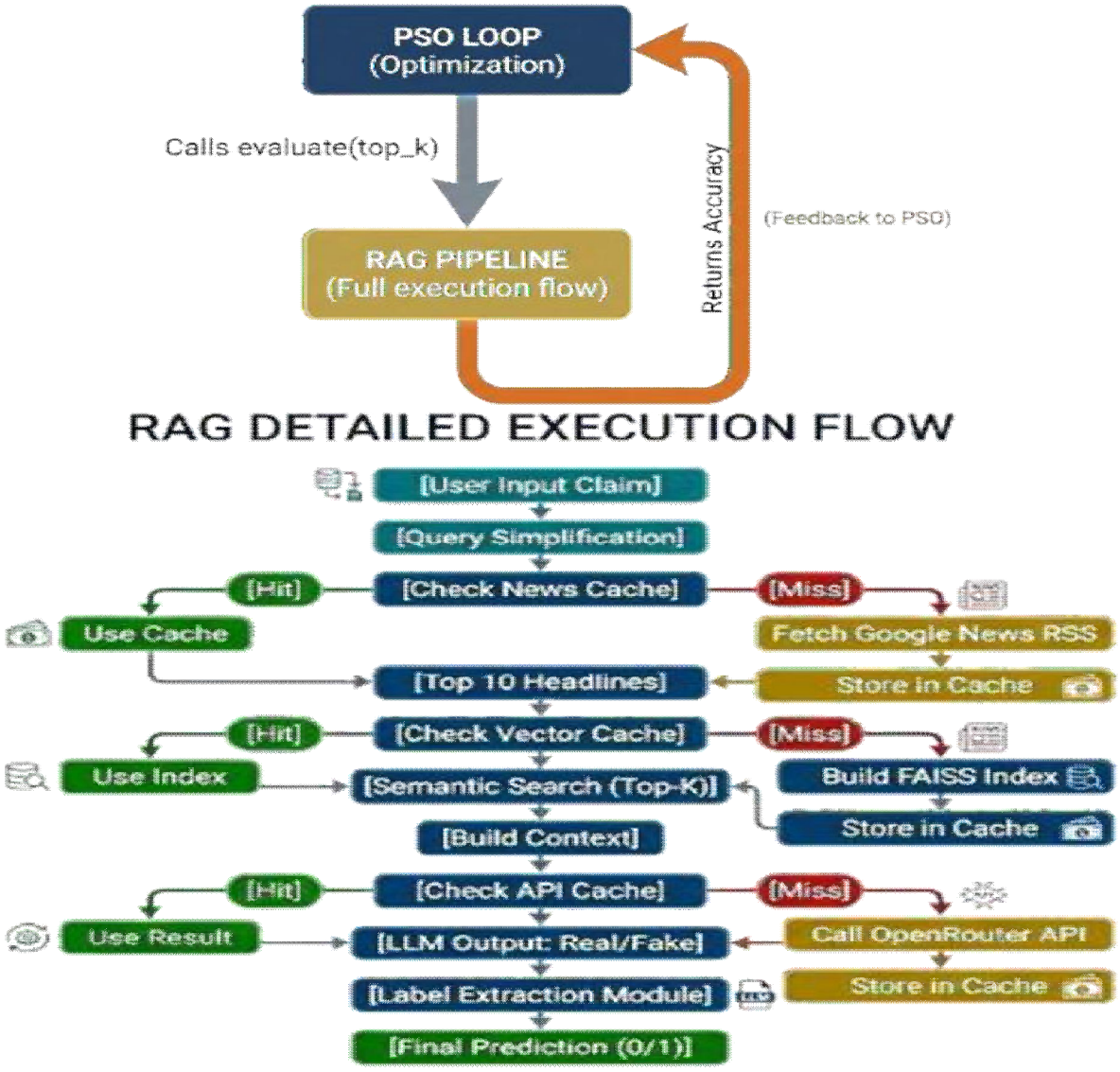
The proposed system is a time fake news detection pipeline that checks user-provided news claims. It does this by using news retrieval searching for similar meanings large language model reasoning, in-memory caching and particle swarm optimization.

The system does not rely on a classifier that was trained on a fixed set of data. Instead it looks up the news claim on Google News. Uses recent headlines as proof to decide if the claim is true or not.

When a user submits a statement or headline the system first simplifies the query. It removes punctuation. Only keeps words that are more than three characters long and it only keeps at most five important words. This simplified query is used to search Google News, which helps to get rid of words and reduce irrelevant results. The system also stores the results of queries in memory so if the same query is submitted again it can use the previous results instead of searching Google News again. The simplified query is sent to Google News and the system gets a list of headlines. It only uses the titles of these headlines. It only keeps at most ten headlines as evidence. These headlines are stored in a cache so if the same query is submitted again the system can use the cached headlines instead of searching Google News again which makes the system faster and less dependent, on external services.Next, the retrieved headlines are transformed into dense semantic embeddings using the pre-trained all-MiniLM-L6-v2 SentenceTransformer model. All embeddings are L2-normalized and inserted into a FAISS IndexFlatIP index, where inner product on normalized vectors effectively corresponds to cosine similarity. To further optimize repeated evaluations, the system uses a vector-store cache keyed by the simplified query: if an index for that query already exists, it is reused instead of recomputed, saving both embedding time and indexing overhead. For each new claim, the retriever embeds the original text using the same SentenceTransformer and queries the FAISS index to retrieve the top-k most similar headlines. These top-k headlines are concatenated into a context block that represents the external evidence supporting or refuting the claim. The context, together with the original claim, is then passed to an LLM via the OpenRouter API, which is instructed to behave as a strict fact-checking assistant and output a binary verdict (“Real” or “Fake”). To avoid making requests to the application programming interface the system stores the application programming interface responses in a cache. This cache is organized by the prompt so when the system gets the same context and claim it can use the previous large language model outputs instead of getting new ones from a remote server.

The large language model response is turned into a number by a module that uses rules to classify it. The response text is converted to lowercase. Checked for certain words. If the word "real" is found the claim is given the label 1. If the words "fake" or "misleading" are found it is given the label 0. This simple system makes the pipeline work well when the models wording is a little different and it makes sure that the decision is always either 0 or 1 which is what is needed for evaluation.

The system also has a module that adjusts parameters to get the results. This module uses a method called particle swarm optimization to find the value for the top k parameter. The top k parameter is the variable that is being optimized. It is kept within a certain range. A fitness function is used to evaluate each value by running the entire pipeline on a small part of the dataset and calculating the accuracy. The particle swarm optimization method then searches for the value of top k that gives the accuracy without using gradients, which is well-suited for this type of optimization. Once the optimization is complete the best top k value is used for the evaluation and some fixed values like 3, 5 7 and 10 are also tried to see how the results change when the retrieval depth is different. Finally the system is tested using a dataset that has labels. For each claim the pipeline records the predicted label and how long it took to process, which makes it possible to calculate the accuracy, precision, recall and F1 score using metrics for binary classification. It also creates a confusion matrix to summarize the positives true negatives, false positives and false negatives and it reports the average time it takes to process each claim, which shows how well the system performs from start, to finish including the effect of caching on how fast the responses are. All these parts work together to create a system that's modular, efficient and can adapt to detect fake news in real time based on live news evidence and the fake news detection system is focused on detecting fake news.



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

V. SOLUTION ARCHITECTURE

The proposed solution is a system that takes in what the user says finds outside information to back it up uses a kind of computer program to think about it and makes sure it gets the right information. This system does all of this while storing some information in memory so it can work faster and not have to look things up over again. The system is made up of parts that work together which makes it easier to add or remove things as needed. This solution is called a pipeline. It helps the system work better by using the computer program to think about the information and by storing things in memory to reduce delays. The solution uses the pipeline to get the information and to make sure it is working correctly which is why it is a good way to solve the problem. The pipeline is a solution because it uses the computer program and stores information, in memory which makes it work faster and better.

A. Input Layer

The input part of the system has a way to work with the user. It takes a news statement or headline that the user gives it. This can be something the user says is true or a claim they want to check. The system then sends this statement to the steps, which are the preprocessing and retrieval pipeline. This means that every time someone wants to check if something's true the system does it in the same way following the same steps using the news statement or headline or claim that the user provided.

B. Query Simplification Module

The query simplification module makes the input claim simpler so that it is easier to find what we are looking for and use what we already have in the cache. This module gets rid of punctuation breaks the text into parts and only keeps the words that have more than three letters taking no more than five important words. The simplified query that comes out of this process is used in two ways: it is used as the search string, for Google News. It is also used as a key to keep things steady in the news and vector-store cache. The query simplification module uses this query to help with the search.

C. Caching Layer

To avoid doing the work over and over and making too many outside calls the system has a special part called a caching layer. This caching layer uses memory dictionaries in Python.

1) The news cache, which is also called news_cache keeps track of the news headlines from Google News for each query. If we need to do the query again the system just uses the cached headlines instead of going out to get new ones.
2) The vector-store cache or vector_cache keeps track of the FAISS index for each query. This index is made from the headline embeddings. If we need to do the query again the system can just use this index instead of making a new one.
3) The API cache or api_cache keeps track of what the model says for each prompt. This prompt includes the context and the claim. If we need to ask the question again the system just uses the cached answer instead of asking the model again. This helps because it means we do not have to make as calls, to OpenRouter. The news cache and the vector-store cache and the API cache all help the system work better by avoiding computation and external calls. The news cache and the API cache and the vector-store cache are all parts of the system.

This caching layer sits logically between the retrieval, embedding, and LLM modules, and significantly reduces average latency and external API load, especially when evaluating multiple claims or re-running the pipeline with different parameters.

D. Google News Retrieval Module

The Google News retrieval module uses the simplified query to construct a Google News RSS URL with an optional time-window constraint (via the when parameter). Before performing the HTTP request, it checks news_cache; if a cached entry exists for that query, it immediately returns the stored headlines. Otherwise, it sends the request, parses the XML response, extracts the title of each <item>, truncates the list to at most ten headlines, and inserts this list into news_cache for future reuse.

E. Embedding Module

The embedding module converts all retrieved headlines into dense vector representations using the pre-trained all-MiniLM-L6-v2 SentenceTransformer model, which produces 384dimensional sentence embeddings suitable for semantic similarity. When building a vector store for a given simplified query, the system first checks vector_cache: if a FAISS index for that query is already available, it is reused directly; otherwise, the module computes embeddings, normalizes them, builds the index, and stores it in the cache.

F. FAISS Vector Index

The normalized headline embeddings are stored in a FAISS IndexFlatIP index configured with inner-product similarity. Because both stored vectors and query vectors are L2-normalized, inner product corresponds to cosine similarity, enabling efficient nearest-neighbour search in the semantic space. Each index is keyed in vector_cache by the simplified query, making it possible to reuse the same index across multiple claims that map to similar search terms.

G. Retriever

The retriever embeds the original user claim using the same SentenceTransformer model and searches the FAISS index for the top-k most similar headlines. The news headlines that we find are then put together in a context. This context is made from lists of information that we already have. That we just made. It has the important news, from recently that is related to what we are looking at and we use this context to help us think about what it means

H. LLM Reasoning Module

The LLM reasoning module makes a prompt with the evidence and the original claim. It sends this to a language model using the OpenRouter API. The prompt tells the model to be very careful and check the facts. It should then give a verdict that says "Verdict: /Fake".

Before the module sends the request it checks if it already has an answer in the api_cache. It uses the prompt to look for this answer. If it finds one it just gives that answer. If not it sends the request. Waits, for the result. When it gets the result it stores it in the cache so it can use it again time. The LLM reasoning module does this to make sure it does not have to ask the question twice. It uses the LLM reasoning module and the large language model to get the verdict. The verdict is what the LLM reasoning module is looking for.

I. Classification Output Module

The classification output module works on the LLM response. It uses rules to pull out labels. The text is lowercased and scanned for keywords: if “real” appears, the system outputs label 1; if “fake” or “misleading” appears, it outputs label 0. This produces the final binary prediction that is returned to the user or passed to the evaluation module.

J. Optimization Module

The optimization module applies Particle Swarm Optimization to tune the top_k retrieval parameter. Treating top_k as a search variable, PSO repeatedly runs the full cached pipeline on a small validation subset and computes accuracy for each candidate value. Because caching is enabled at the news, index, and API levels, repeated evaluations with similar queries reuse existing data and responses, making the PSO search significantly faster and more cost-efficient than un cached evaluation.

K. Evaluation Module

The evaluation module runs the pipeline on a labeled test dataset and records predictions and processing time per claim. It computes accuracy, precision, recall, and F1-score using scikitlearn’s binary classification metrics, and constructs a confusion matrix to summarize classification behaviour. The system also tells us about the time it takes to get a response, which is the total time it takes for the network to make calls do computations, search, for vectors make decisions with the LLM and get faster because of the cache. This gives us a picture of how well the cache-aware architecture makes predictions and how fast it responds in real time so we can see how good the predictive quality is and how well the system responds right away.

VI. EXPERIMENTATION AND RESULTS

A. Experimental Setup

The experiments used a Retrieval-Augmented Generation framework and Particle Swarm Optimization for tuning the system in a task to detect news. This system was made using Python. It is designed to check claims by getting evidence searching for similar meanings using a large language model to reason and helping the system work faster by using a cache.

This is distinct from Retrieval-Augmented Generation systems that utilize an external source of information retrieval and knowledge generation. The system we have developed leverages Google News to receive current news articles and headlines that correspond to each claim. The system aggregates these news articles and headlines into a searchable format, providing users with access to meaningful, searchable content related to their claims; therefore, the system is most useful for verifying events where evidence is available in real-time.

To reduce the time of execution associated with the system and decrease the amount of work required by users, we have implemented three separate cache files: One cache file saves news articles and headlines received from the Google News source; another saves responses from the OpenRouter; and the third stores pre-built indexes for repeated queries. This cache design improves the efficiency of the system's execution, especially when multiple checks are performed on the same claims or when Particle Swarm Optimization is used to identify optimal configurations of the system.

The system uses a model called all-L6-v2 SentenceTransformer to put sentences into a form that can be searched for similar meanings. It puts the news headlines into this form. Stores them in a way that makes it easy to find similar headlines, for each claim. The system uses the OpenRouter API and the OpenAI/gpt-3.5-turbo model to reason. Check claims. For each claim it puts the headlines together and asks the model to say if the claim is real or fake. It then turns the answer into a number.

Particle Swarm Optimization is used to find the number of headlines to get for each claim. The system checks different settings and finds the one that works best. This makes the system able to adapt to situations because it can find the best number of headlines to use without needing to be told what that number is.

B. Experiment Design

The retrieval parameter top_k was tested to see how it influences the performance and runtime of the RAG-based news detection system. For each configuration, the entire pipeline of this news detection system was executed, including: simplifying the query, collecting news from Google News, generating embeddings, performing searches using FAISS, classifying results using LLMs, and processing the output. Twenty news claims were used; each claim was marked as either real or fake. For each claim the system made a guess at the label of the claim, where 0 is fake and 1 is real, then compared the guess to the true label.

Using top_k values of 3, 5, 7 and 10, the system also recorded a variety of metrics for the performance evaluation of each of the configurations, including accuracy, precision, recall, F1 score, confusion matrix, and average execution time. The Particle Swarm Optimization technique is applied to find the top_k value generating the best accuracy for the system. Results were constructed using these metrics for each top_k value, where each result documents the assigned metrics. These results were used to generate Matplotlib plots to visualize the accuracy, precision, and so on.

C. Evaluation Protocol

The evaluation follows a consistent protocol for each candidate top_k value.

1) Run full pipeline per claim

For each news claim in the test dataset the system runs the RAG pipeline with a set top_k. The system notes the time at the start and finish of each claim. It uses the Python time module to do this. The difference, between these two times is the time it takes to process the news claim. This is called the latency for that news claim.

2) Collect predictions and compute metrics

After all claims are processed we collect the labels and predicted labels into two separate arrays. The overall accuracy is calculated as the number of predictions divided by the total number of predictions. We also compute precision, recall and F1-score using the classification metrics from scikit-learn. These metrics give us an understanding of how our model is performing. A confusion matrix is created to show the counts of positives true negatives, false positives and false negatives. We also calculate the latency by taking the mean of the latency recorded for each claim. The accuracy, precision, recall and F1-score are all metrics, for evaluating the models performance on the claims.The confusion matrix helps us to see where the model is making mistakes. The average latency tells us how long it takes for the model to process each claim.

3) Parameter sweep

The steps are repeated for top_k values, which are 3, 5, 7 and 10. For each of these top_k values we get a report. The report has top_k value, accuracy, precision, recall, F1score time taken and a confusion matrix. These reports are stored in a list called results. Then we use Matplotlib to create some plots. The plots are based on the results for top_k values. We look at the results, for top_k values of 3, 5 7 and 10.These results help us understand how top_k affects the performance.

- A performance plot where accuracy, precision, recall, and F1-score are plotted against top_k on the same figure;
- A latency plot where average latency (in seconds) is plotted against top_k. These plots let us take a look at the trade-off between retrieval depth and predictive performance and also retrieval depth and runtime. We can see how retrieval depth affects performance and how it affects runtime. The plots are useful, for checking the trade-off between retrieval depth. These two things, which are predictive performance and runtime.

4) Best model analysis

The system runs a search for the value of top_k at the same time as it tries other fixed parameters. It uses the Particle Swarm Optimization method or PSO for short to find the top_k value that makes the accuracy as high as possible. The Particle Swarm Optimization method uses a fitness function that looks at a smaller set of data, which is called the small_dataset. When the system finds the top_k value it tests the system again with all the data using this best top_k value.

Then it makes a picture of the confusion matrix, which's a table that shows how well the system is working. The system uses Seaborn to make this picture.

It labels the sides of the picture with the words "Fake" and "Real". People often use these pictures of confusion matrices to understand how the system is making decisions and to find any mistakes it is making such, as saying something is fake when it is really real or saying something is real when it is really fake.

5) Final qualitative test (optional)

The system is ready to use when we have the settings in place. We can use it to look at claims and see what the system thinks in real time. This step is not about measuring how well the system works with numbers.

It is about seeing how the RAG–LLM pipeline works with information and how fast it can give us an answer, with the special architecture that remembers things.

VII. EVALUATION METRICS

The system is evaluated using these metrics. We use a system where things are either "Real" (that is label 1) or "Fake" (that is label 0).

1) Accuracy is how often the system gets it right. It is the number of answers divided by the total number of questions. This tells us how good the system is overall. It does not tell us what kind of mistakes it makes.

2) Precision for the "class is when the system says something is real and it actually is. If the system has precision for the "Real" class that means when it says something is real it is usually right. This is important when it is bad to say something is real when it is not.

3) Recall for the "class is when the system finds all the real things. If the system has recall for the "Real" class, that means it does not miss many real things. This is important when it is bad to miss things.

4) F1-Score is a way to combine precision and recall for the "class. It gives us one number that balances precision and recall, for the "class. This is useful when there are Real" things or more "Fake" things or when making mistakes is bad.

5) Confusion Matrix is a table that shows what the system says versus what's really true. It looks like this:

6) True Negativesre when the system says something is "Fake" and it really is "Fake".

7) False Positives are when the system says something is "Real". It is really "Fake".

8) False Negatives are when the system says something is "Fake". It is really "Real".

9) True Positives are when the system says something is "Real". It really is "Real". We can use a heatmap to look at this table and see if the system is making the mistakes over and over like if it always says things are "Real" or always says things are "Fake".

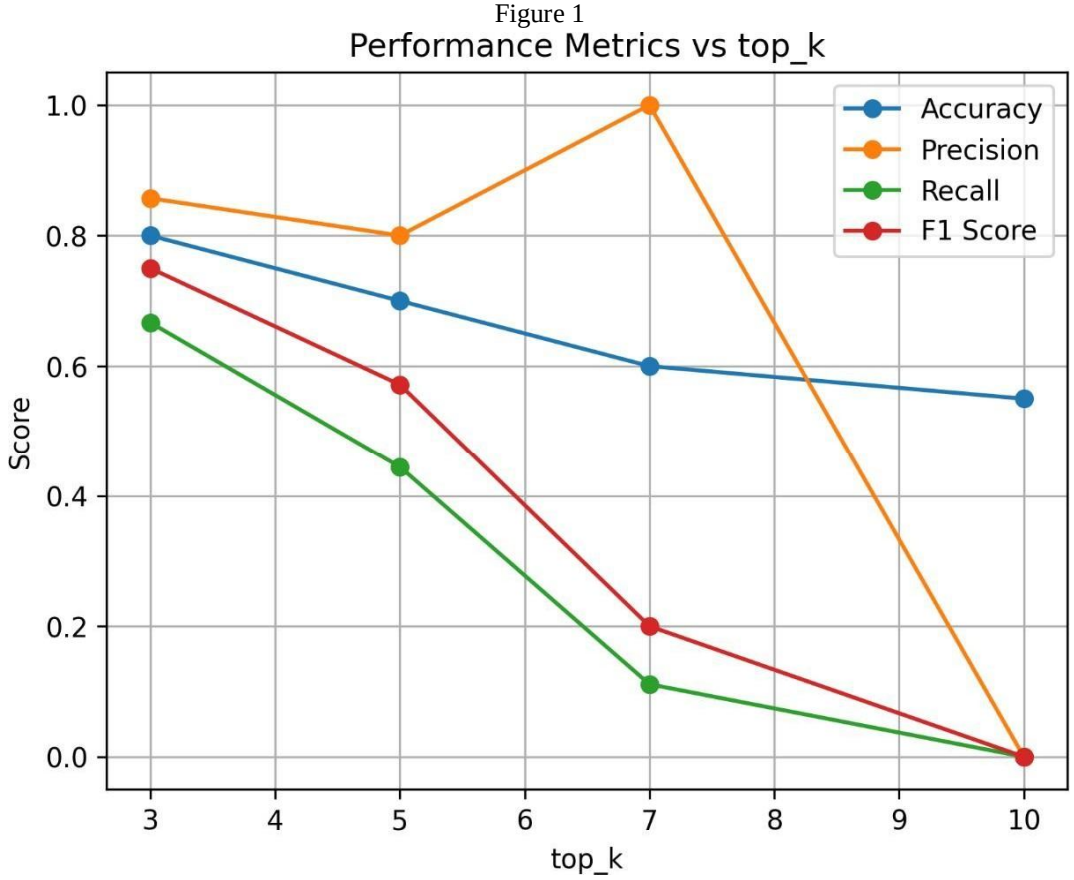
A. Latency

Latency refers to the time taken for the processing of the claim, including the initial receipt of the claim, the retrieval of the claim from FAISS, and the use of the LLM for decision-making.

The reason why this is important is that it demonstrates the functionality of the application within real-life scenarios. We assess the time taken to perform the aforementioned tasks and compare it to an alternate measure (i.e., top_k) to observe how a greater depth of introspection of our data correlates with the timeliness of our response to events. We will examine the cumulative latency of processing the claim using retrieval, embedding, FAISS search, and LLM inference to determine how these components interact with one another to provide a full picture of the overall system performance.

This allows for a straightforward method of evaluating your RAG-based news detector, as it provides both an experimental protocol for evaluating performance as well as relevant measurement techniques to evaluate performance in real-time. The end result will be a straightforward assessment of the accuracy and efficacy of your RAG-based fake news detection algorithm and its performance.

Figure 1



B. Confusion Matrix Analysis

A confusion matrix was generated to visualize the distribution of classification outcomes. It provides a detailed breakdown of:

- Correct classifications (TP and TN)
- Misclassifications (FP and FN)

This analysis helps assess whether the model exhibits bias toward a specific class and identifies error patterns in semantic claim evaluation.

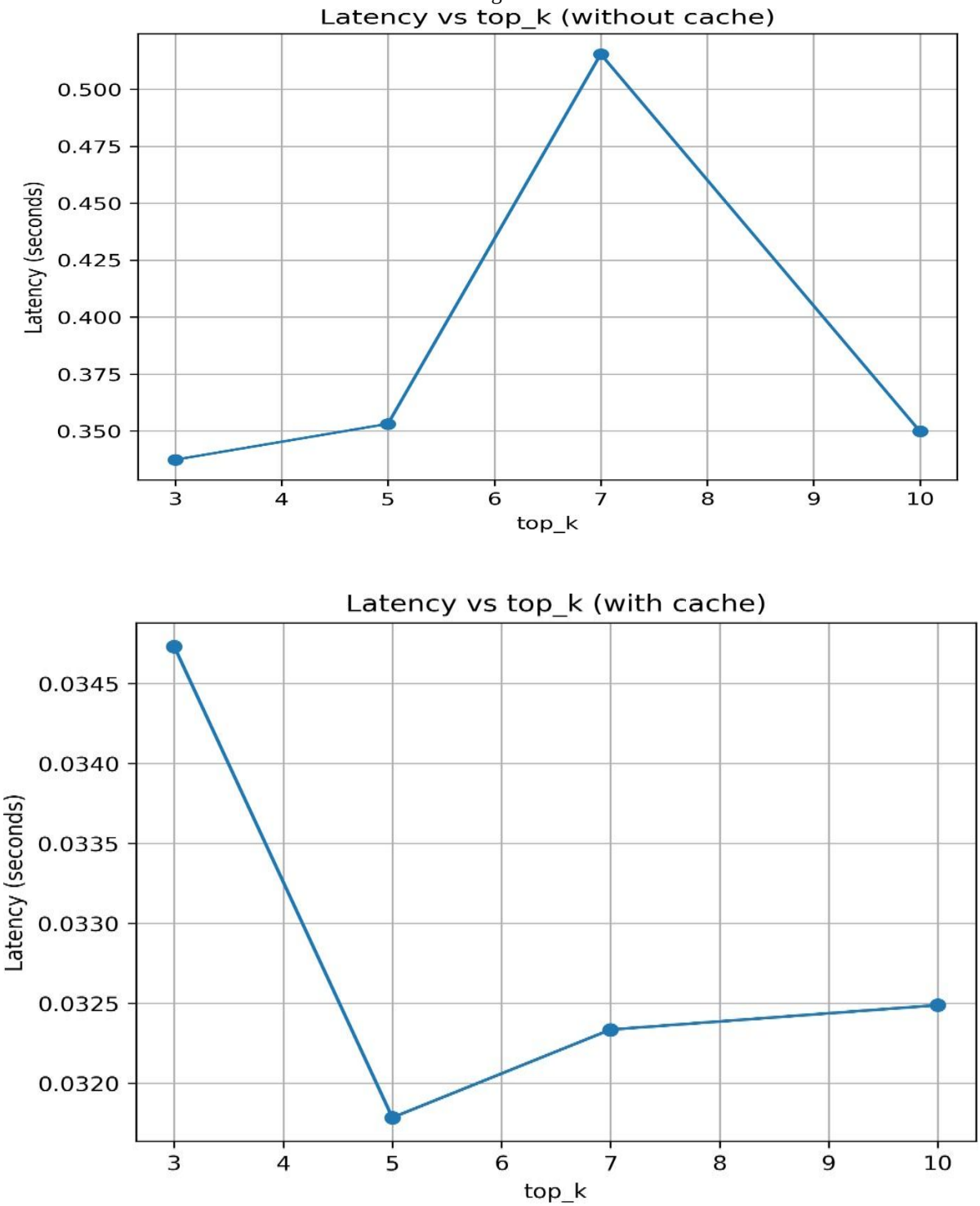
Figure 2

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

C. Retrieval Latency

System performance was evaluated based on the speed (measured in second) it took to produce results, with quick and consistent times suggesting that our system can perform effectively when checking claims in a timely manner. System performance also measured retrieval time to validate that it would work for checking claims in a real-time environment.

Figure 3



VIII. RESULTS

A. Binary Classification Performance

The RAG + PSO framework was tested to see how well it works when it retrieves numbers of documents. We looked at how accurate it was how precise it was, how well it recalled information and its F1-score. The results are shown in Figure 1 which compares these metrics to how many documents were retrieved. It is clear that the RAG + PSO framework works best when it only retrieves a documents. The documents the RAG + PSO framework retrieves the worse it performs.

From Figure 1 (k = 3, 5, 7, 10), the observed trends are:

1) When we look at k equals 3 the system gets the balance just right. The system is about 80 percent of the time. The precision of the system is 86 percent. The recall of the system is 67 percent. The F1-score of the system is 75 percent. The system gets the balance right at k equals 3.
2) When k equals 5 all the metrics are not as good as before. The accuracy of the system is around 70 percent. The precision of the system is 80 percent. The recall of the system is 44 percent. The F1-score of the system is 57 percent.
3) When k equals 7 the scores of the system are even worse. The accuracy of the system is 60 percent. The precision of the system is 20 percent. The recall of the system is 29 percent. The F1-score of the system is 20 percent.
4) When k equals 10 the precision of the system the recall of the system and the F1-score of the system are all very close, to zero. The accuracy of the system is around 55 percent.

The results indicate that:

- The system is really good at getting it when there are not many options like when we are looking at just three things. This means it does not often say something is real when it is not.
- When we look at more and more things the system gets worse at finding the real ones. This is because looking at many things can be confusing and makes it harder for the system to make good decisions.
- We can see that there is a trade-off, between how the system works and how many things it looks at. When it only looks at a things it makes better decisions but when it looks at a lot of things it can get too much information and that hurts the quality of the decisions made by the Large Language Model or LLM for short I mean the Large Language Model classification.

B. Confusion Matrix Analysis

The confusion matrix for the configuration that was chosen as the best in your script, where you used top_k = 4 is shown in Figure 2. This confusion matrix, for the configuration shows us:

- True Fake → Predicted Fake: 11
- True Fake → Predicted Real: 0
- True Real → Predicted Fake: 5
- True Real → Predicted Real: 4

This pattern is really good at finding claims. It does not get fooled by news. However it sometimes says real claims are fake when they are not. This happens five times. The system is being careful. It would rather say something is fake than miss something that's really fake. This is like being, on the side. Depending on what we're using this for this way of being safe might be okay or we might need to make some changes. For example we might need to adjust the questions we ask or how we figure out what is real and what is not.

C. Impact of Retrieval Parameter (k)

The impact of the retrieval parameter is shown in Figure 1. This figure shows how performance metrics change when we use top_k values.

1) When k is 3: We get the results with this dataset. We get the F1-score and the best balance between precision and recall.
2) When k is 5: Using a larger context does not help. It actually leads to recall and F1score. This means that adding headlines does not always give us useful information.
3) When k is 7 and k is 10: The results get much worse. Precision and recall get very close to zero when k is 10. This shows that using a context introduces a lot of noise. It becomes harder for the LLM to focus on the important evidence.

So when we look at this small test set of 20 samples we see that k values like 3 or 4 work best. These values give us evidence without adding unnecessary or misleading information.

The PSO-based optimization layer helps us find the option but with very small datasets the best result can change depending on the specific examples. We should look at this as a guide than a final answer.

The retrieval parameter k is important. The retrieval parameter k affects the results. The best retrieval parameter k values are the ones that give us the balance, between precision and recall.4.4 Latency and Real-Time Performance

D. *Latency and Real-Time Performance*

When we look at how it takes to get results, which is called latency and we consider the top_k we can see what happens when we do not use a cache and when we do use a cache like you can see in Figures 3 and 4.

1) When we do not use a cache the average time it takes to get results for each claim is around 0.34 to 0.53 seconds. The longest time it takes is around k = 7 and it is a little faster when k is 3 k is 5 or k's 10. This is because we have to pay the cost of getting results in real time making embeddings searching with FAISS and calling the LLM for each claim.
2) When we use a cache the time it takes to get results gets faster around 0.031 to 0.035 seconds and it stays about the same no matter what k is. This is because when we do the evaluations again we can use the cached Google News results, FAISS indices and API responses so the main cost is just the time it takes to process things in Python not the time it takes to get data from the network or run the model.

Overall even when we do not use a cache the system can still get results in under one second per claim which is pretty good, for this dataset and it makes it possible to use the system to check fake news in real time. In life when we have to check a lot of claims or do the same queries over and over using a cache can make the system run much faster which is a big help.**4.5**

E. *Key Observations*

From the experimental analysis:

The RAG + PSO pipeline does not always get better when we use a top_k. In fact the RAG + PSO pipeline works best when we use a number of retrieval depths. If we give the RAG + PSO pipeline much context it actually gets worse.

1) The system is really good at finding content. For example when we use top_k = 4 the system never misses any content. However the RAG + PSO pipeline sometimes thinks real claims are fake which is not good. This is a problem with the RAG + PSO pipeline. The RAG + PSO pipeline needs to be better at telling the difference, between fake claims.
2) Caching dramatically improves latency while preserving the same decision behaviour, confirming that in-memory news, vector, and API caches are effective for repeated evaluations and real-time scenarios.

IX. SUMMARY

This work presents a real-time fake news detection framework that combines live Google News retrieval, sentence-level embeddings, FAISS-based semantic search, and LLM-based binary classification within a retrieval-augmented generation (RAG) setup. A user-provided claim is simplified into a keyword query, used to retrieve recent headlines via Google News RSS, and these headlines are encoded with the all-MiniLM-L6-v2 SentenceTransformer, L2-normalized, and indexed in FAISS so that the most semantically similar headlines can be efficiently selected as evidence. The retrieved evidence and original claim are then passed to an OpenRouterhosted LLM, whose Real/Fake verdict is converted into a binary label for evaluation.

To see how well the pipeline works a small group of fake claims was tested from start to finish with different settings for the retrieval parameter top_k. The script then calculated how accurate it was, how precise it was, how well it remembered things, its F1-score, a chart to show what it got right and wrong. How long it took on average to check each claim.

The pipeline used something called Particle Swarm Optimization to find the best top_k setting that makes it the most accurate. This helps the pipeline look as deep as it needs to.

When you look at the charts that show how well it does with top_k settings and how long it takes with and without storing things in a cache and the chart that shows what it got right and wrong you can see that the best settings let the pipeline be very accurate and still check claims very quickly which means it could be used to check fake news in almost real time. The pipeline is good at finding a balance between being accurate and not taking long which is important, for checking fake news quickly.

X. FUTURE WORK

1) Larger and more diverse datasets: Current experiments rely on a small, hand-crafted test set; future work should evaluate the model on larger benchmark datasets spanning multiple domains and languages to better assess robustness and generalization.
2) Richer evidence and source modeling: At present, the system uses only Google News headlines as evidence. Future extensions could incorporate article snippets or full articles, fact-checking databases, and explicit source credibility features, helping the model reason over supporting and refuting evidence more effectively.

3) Advanced RAG strategies and reasoning: More sophisticated RAG setups—multi-hop retrieval, iterative retrieval–refine loops, claim decomposition, and contrastive evidence presentation—could further improve fact-checking accuracy and explanation quality, especially for complex or multi-claim statements.

4) Calibration, thresholds, and abstention: The current version always outputs a binary decision. Future work could add confidence estimation, tunable thresholds, and an "insufficient evidence / abstain" class to reduce overconfident predictions on ambiguous or low-evidence claims.

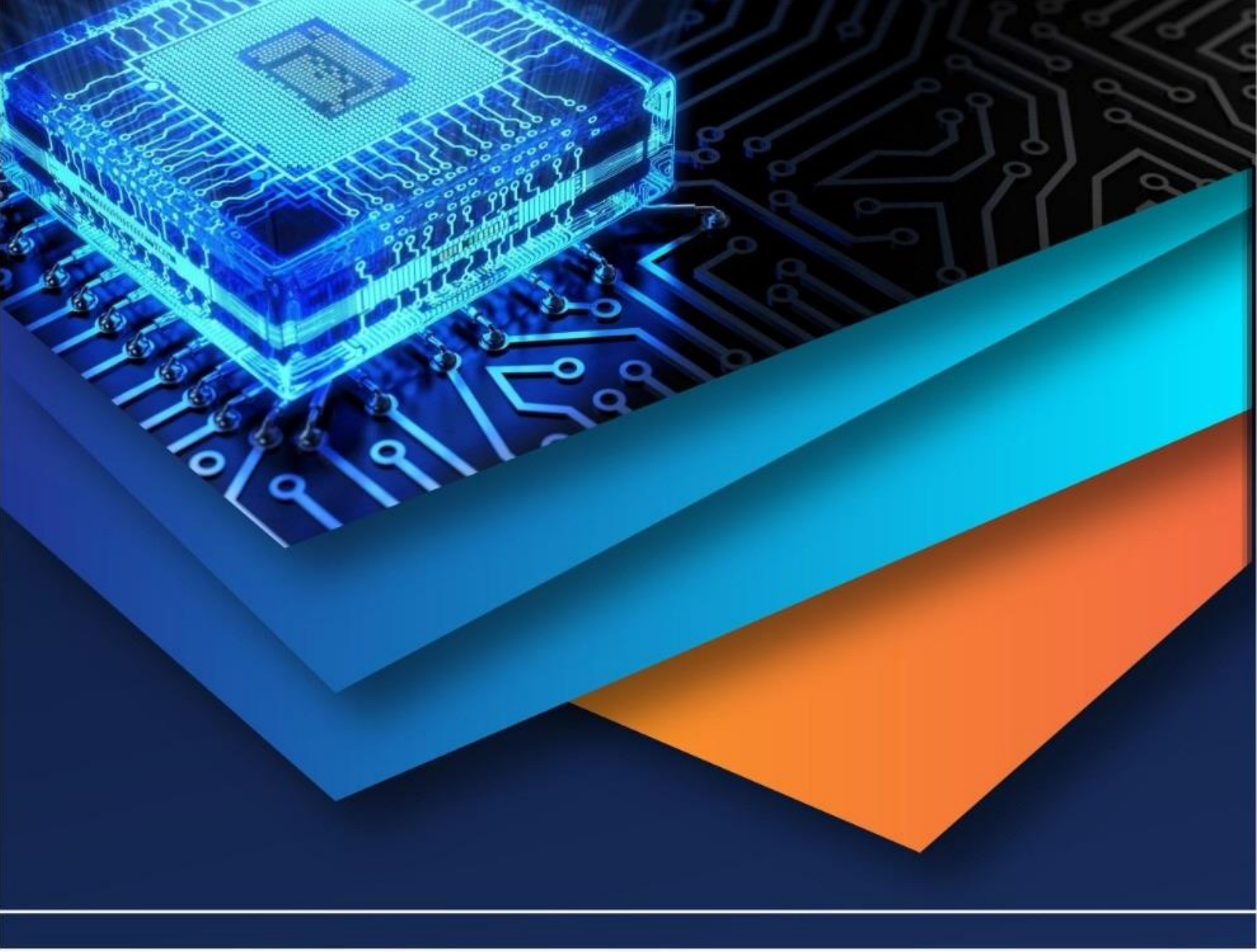
5) Infrastructure and latency improvements: Since the current system depends entirely on OpenRouter's free tier, it is subject to rate limits and network-induced latency. Future work should explore more scalable deployment options—such as hosting smaller LLMs locally, using paid or dedicated OpenRouter plans, or caching and batching strategies—to reduce latency and support higher throughput.

6) User interface and human-in-the-loop verification: Integrating the backend into an interactive interface for journalists or end-users, with transparent display of retrieved evidence and model rationales, would support humanin-the-loop verification and allow collecting user feedback for further improvement.

7) Comparative benchmarking: Finally, the system should be compared against baseline fake news detectors (traditional classifiers, non-RAG LLM prompts, and other RAG-based fact-checking approaches) to quantify the benefit of live retrieval plus optimized top_k under realistic constraints.

REFERENCES

[1] L. Li et al., "VeraCT Scan: Retrieval-Augmented Fake News Detection System," arXiv preprint, 2024.
[2] Y. Bai et al., "A Large Language Model-based Fake News Detection Framework with Retrieval and Verification," NSF/Preprint, 2024.
[3] R. Singal et al., "Evidence-backed Fact Checking using RAG and Few-Shot Learning," ACL FEVER Workshop, 2024.
[4] N. Rubtsova et al., "Fake News Detection with Retrieval Augmented Generation Framework," OpenReview, 2024.
[5] S. Sharma and S. Meena, "Retrieval Augmented Generation Classification Algorithm for Fake News Detection," 2024.
[6] J. Bing et al., "RAG-Augmented Reasoning for Political Fact-Checking using LLMs," arXiv preprint, 2024.
[7] IEEE Computer Society, "A Survey of Large Language Models in Fake News Detection," IEEE Computer Magazine, 2025.
[8] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," NeurIPS, 2020.
[9] K. Guu et al., "REALM: Retrieval-Augmented Language Model Pre-Training," ICML, 2020.
[10] O. Thorne et al., "FEVER: A Large-Scale Dataset for Fact Extraction and Verification," NAACL-HLT, 2018.
[11] S. Lopez-Joya et al., "The blueprint of a new fact-checking system," 2025.
[12] H. Li et al., "Use of Retrieval-Augmented Large Language Model for Fact-Checking," 2025.
[13] J. Díaz García et al., "VERIFAID: Verification FAISS-based framework for fake news detection," 2025.
[14] Y. Bai et al., "FCRV: Full-Context Retrieval and Verification for misinformation detection," 2024.
[15] A. Dash et al., "Evidence-backed Fact Checking using RAG and Few-Shot Learning," 2024.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY