

# An Intelligent Plagiarism Detection System Using TF-IDF Vectorization and Cosine Similarity with Sentence-Level Granularity

Vedant Sonawane, Manisha Devgunde, Venkatesh Shinde, Megha Bansode

<sup>1</sup>Department of Artificial intelligence and machine learning, Zeal college of engineering and research, pune, India

<sup>1</sup>Department of Artificial intelligence and machine learning, Zeal college of engineering and research, pune, India

<sup>1</sup>Department of Artificial intelligence and machine learning, Zeal college of engineering and research, pune, India

<sup>1</sup>Department of Artificial intelligence and machine learning, Zeal college of engineering and research, pune, India

\*\*\*

**Abstract**— Plagiarism detection has become increasingly important across academic, research, and digital publishing domains due to the rapid growth of online content. Many existing systems function as opaque tools, offering limited insight into how similarity is determined or which sections are flagged. This study introduces an interpretable plagiarism detection framework that integrates TF-IDF vectorization with cosine similarity to evaluate textual overlap at both document and sentence levels. In addition, a logistic regression classifier is incorporated as a fallback mechanism when reference data is unavailable. The system is implemented as a RESTful API using the Flask framework, supporting multiple file formats including PDF, DOCX, and TXT. Each sentence is individually analysed and visually highlighted based on similarity scores, enabling precise identification of potentially copied content. Experimental evaluation on diverse real-world datasets demonstrates consistent performance with low latency. The proposed system emphasizes modularity, transparency, and extensibility.

**Keywords**—plagiarism detection, TF-IDF, cosine similarity, logistic regression, sentence-level analysis, Flask, natural language processing, similarity scoring

## I. INTRODUCTION

The widespread availability of digital resources has made copying and reusing textual content easier than ever before. As a result, plagiarism has emerged as a serious concern in academic environments, research communities, and content publishing platforms. It not only compromises intellectual integrity but also reduces the credibility of original work.

Early plagiarism detection systems primarily relied on exact text matching techniques. While these approaches are effective for identifying direct copying, they struggle to detect more subtle forms such as paraphrasing or content restructuring. Modern tools provide advanced detection but are often expensive and inaccessible.

This paper proposes a transparent and efficient plagiarism detection system using TF-IDF vectorization and cosine similarity, supported by a logistic regression classifier. The system is deployed using Flask and provides detailed sentence-level analysis.

## II. LITERATURE REVIEW

Plagiarism detection has been an active area of research for more than two decades. Early systems focused on syntactic matching. Schleimer et al. [1] proposed Winnowing, a document fingerprinting algorithm used as the basis for the MOSS (Measure of Software Similarity) system, which detects copied source code by hashing fixed-length character n-grams. While highly accurate for exact matches, such approaches are inadequate for semantically paraphrased content.

The introduction of vector space models significantly advanced the field. Salton and McGill [2] formalized TF-IDF as a means of representing documents in a high-dimensional term space, enabling similarity comparison through cosine distance. Subsequent researchers applied this framework to plagiarism detection, demonstrating that even moderately paraphrased text

preserves enough term overlap to be detectable [3]. Potthast et al. [4] organized the PAN shared tasks on plagiarism detection, providing standardized evaluation benchmarks and revealing that hybrid approaches combining IR-based retrieval with detailed linguistic analysis outperform purely statistical methods.

Machine learning approaches have further strengthened detection capabilities. Alzahrani et al. [5] surveyed semantic approaches and found that combining syntactic and semantic features improves recall against paraphrased plagiarism. Support vector machines (SVMs) and logistic regression have been widely adopted as classifiers for binary plagiarism labeling due to their interpretability and effectiveness on text features derived from TF-IDF [6]. More recently, transformer-based models such as BERT have been applied to cross-lingual and obfuscated plagiarism detection [7], achieving state-of-the-art results at the cost of increased computational overhead.

On the deployment side, several open-source tools have emerged. CopyCatch and PlagScan offer web-based interfaces but require subscription fees. Research prototypes built on Python and Flask [8] have demonstrated that lightweight deployments can achieve competitive detection performance while remaining accessible to institutions without large IT budgets. The system described in this paper builds on these foundations, combining TF-IDF with cosine similarity in a sentence-level pipeline that prioritizes interpretability and ease of integration.

### III. METHODOLOGY

#### A. System Architecture

The proposed system follows a layered architecture comprising input processing, feature extraction, similarity computation, classification, and output generation. Table I summarizes the principal components at each layer.

**TABLE I** System Architecture Layers and Components

Layer	Component / Description
Input Layer	Text (typed), File upload (PDF/DOCX/TXT), REST JSON API
Preprocessing	Sentence splitting, tokenization, lowercase normalization, punctuation removal
Feature Extraction	TF-IDF Vectorizer (sklearn), transforms text to sparse numerical vectors
ML Model	Logistic Regression classifier (binary: plagiarized / original)
Similarity Engine	Cosine similarity against reference corpus (sentence-level and document-level)
Output Layer	JSON response: percentage, label, highlighted sentences, matched source fragments
Persistence	JSON-based scan history; Dashboard analytics (average, high/medium/low counts)

The input layer abstracts over three distinct submission modes: typed text via a web form or JSON API call, an uploaded binary document (PDF, DOCX, or TXT), and programmatic requests from downstream applications. All inputs converge at the preprocessing stage before entering the feature pipeline.

### **B. Data Preprocessing**

Raw text extracted from any of the supported input modalities is subjected to a lightweight preprocessing pipeline prior to vectorization. First, the text is split into individual sentences using a rule-based splitter that identifies sentence boundaries at period, exclamation mark, and question mark characters followed by whitespace. Sentences shorter than ten characters are discarded to avoid noise introduced by headers, page numbers, or formatting artifacts.

Each retained sentence is then cleaned: all characters are converted to lowercase, punctuation is stripped, and common English stopwords are removed. For document-level analysis, the full cleaned text is passed to the similarity engine as a single unit; for sentence-level analysis, each sentence is processed individually to enable granular highlighting.

### **C. TF-IDF Vectorization**

Text representation is achieved through Term Frequency-Inverse Document Frequency (TF-IDF) vectorization. For a term  $t$  in document  $d$  belonging to corpus  $D$ , the TF-IDF weight is defined as:

$$tfidf(t, d, D) = tf(t, d) \times \log(|D| / (1 + df(t, D)))$$

where  $tf(t, d)$  is the normalized term frequency in document  $d$ ,  $|D|$  is the total number of documents in the corpus, and  $df(t, D)$  is the number of documents containing term  $t$ . The resulting sparse high-dimensional vectors efficiently capture the relative importance of each term and suppress the influence of ubiquitous stopwords that survive preprocessing.

The vectorizer is fit on the training corpus during model building and serialized to disk as a pickle file, ensuring that inferencetime transformations are performed in the same feature space as training. This is critical for consistency: any discrepancy between the fit vocabulary and the inference vocabulary would silently degrade detection accuracy.

### **D. Model Training**

A logistic regression model is trained on TF-IDF vectors derived from a dataset of paired text samples, where each sample is labeled 0 (original) or 1 (plagiarized). Logistic regression was chosen for three reasons. First, it is computationally efficient, enabling rapid inference on server-side requests. Second, its probabilistic output (softmax over the binary class distribution) provides a confidence estimate that can be used as a fallback plagiarism percentage when no reference corpus is available. Third, its coefficients are interpretable, facilitating future auditing and error analysis.

Both the trained model and the fitted vectorizer are serialized using Python's pickle module and loaded at application startup. This design decouples the training phase from the serving phase, allowing the model to be retrained or updated independently of the deployment infrastructure.

### **E. Similarity Computation**

The core detection mechanism computes cosine similarity between the input text vector and every vector in the reference corpus matrix. For two vectors  $A$  and  $B$ , cosine similarity is:

$$\cos(\theta) = (A \cdot B) / (||A|| \times ||B||)$$

At the document level, the maximum cosine similarity across all reference sentences is reported as the plagiarism percentage. This conservative choice avoids penalizing documents that contain only partial overlap. At the sentence level, each split sentence is independently compared against the full reference matrix; sentences whose best-match similarity meets or exceeds a configurable threshold (default 0.55) are flagged as plagiarized and annotated with the matching source fragment. When the reference corpus is unavailable, the model's predicted probability for the plagiarized class is used in its place.

## IV. IMPLEMENTATION

### A. Flask Backend

The application backend is implemented in Python 3 using the Flask microframework. Flask was selected for its minimal overhead, its compatibility with the scientific Python ecosystem (NumPy, scikit-learn), and the ease with which it exposes Python functions as HTTP endpoints. The application is configured with a 10 MB upload size limit and a random 24-byte secret key for session management. All heavy ML artefacts (model and vectorizer) are loaded once at startup and held in memory, eliminating per-request deserialization overhead.

### B. API Design

The system exposes four RESTful endpoints. The primary /detect endpoint accepts HTTP POST requests and handles three distinct content types: (i) application/json bodies containing a text field, (ii) multipart/form-data requests carrying an uploaded file, and (iii) URL-encoded form bodies with a plain text field. Guard clauses enforce minimum text length requirements and reject unsupported file types with informative 400-series error responses.

A successful detection request returns a JSON payload containing the computed plagiarism percentage, a categorical label (Plagiarism Detected or No Plagiarism Detected), the timestamp of the analysis, aggregate counts (total sentences, flagged sentences, word count), and the full sentence-level annotation array. Supplementary endpoints (/api/history and /api/stats) expose scan history records and aggregate statistics, enabling frontend dashboards and third-party integrations to consume realtime analytics.

### C. File Processing

Three file formats are supported. Plain text files (.txt) are decoded from UTF-8 bytes directly. PDF documents are processed via the pdfplumber library, which extracts text from each page and concatenates the results. DOCX files are parsed using the python-docx library, which iterates over document paragraphs and joins their text content. Both PDF and DOCX support modules are loaded conditionally at startup; if a dependency is absent, the endpoint returns an informative error message rather than crashing, preserving backward compatibility in constrained deployment environments.

### D. Sentence Highlighting

Sentence-level analysis produces a structured annotation array in which each entry records the original sentence text, a boolean plagiarized flag, a numerical similarity score (0–100), and the best-matching reference fragment. This array is consumed by the frontend to render color-coded highlighting: sentences exceeding the similarity threshold are rendered with a distinct background color, and hovering over a flagged sentence reveals the matched source snippet in a tooltip. This granular visual feedback transforms the system from a simple binary classifier into a diagnostic tool that helps users understand which specific claims or passages require citation review.

## V. RESULTS AND DISCUSSION

### A. Experimental Setup

The system was evaluated against a set of real-world documents spanning multiple domains, including academic research papers, encyclopedic articles, and cybersecurity technical notes. These documents were submitted to the /detect endpoint in both plain-text and file-upload modes. Evaluation focused on three observable metrics: the computed plagiarism percentage, the number of sentences flagged relative to the total sentence count, and the binary classification label assigned by the threshold rule (30% decision boundary).

### B. Sample Results

Table II summarizes the results obtained for four representative test documents drawn from the system's scan history. These results reflect the behavior of the cosine similarity engine against the training corpus.

**TABLE II** Detection Results for Representative Test Documents

Input Text Type	Sentences	Flagged	Similarity %	Label
Wikipedia excerpt	8	0	30.8%	Plagiarism Detected
Research journal text	142	2	27.2%	No Plagiarism
Cybersecurity notes	13	0	24.3%	No Plagiarism
Short greeting text	1	0	50.0%	Plagiarism Detected

The Wikipedia excerpt, despite containing only eight sentences, triggered the plagiarism label because its maximum sentence-level similarity to a reference document reached 30.8% — marginally above the 30% threshold. The academic journal text, although considerably longer (142 sentences), produced only two flagged sentences and an overall similarity of 27.2%, placing it just below the threshold. This behavior correctly reflects that a document may contain a small number of highly similar passages while being predominantly original.

The short greeting text achieved a relatively high similarity score of 50% because very short inputs produce sparse TF-IDF vectors whose cosine distances are less discriminative, a known limitation of bag-of-words representations for very short strings. This motivates the 20-character minimum input guard clause implemented in the detection endpoint.

### C. Performance Characteristics

Response latency for text inputs under 1,000 words was observed to be under 500 milliseconds on a standard development machine (Intel Core i5, 8 GB RAM), including sentence splitting, TF-IDF transformation, cosine similarity computation against a corpus of several hundred reference sentences, and JSON serialization. File uploads incur additional latency proportional to extraction complexity; PDF files with embedded images or complex layouts can increase processing time to 1–2 seconds. For production deployments handling concurrent users, a WSGI server such as Gunicorn with multiple worker processes is recommended.

### D. Threshold Sensitivity

The 30% document-level decision threshold and 55% sentence-level threshold were chosen empirically based on the distribution of cosine similarity scores in the training corpus. Lowering the document-level threshold increases recall at the cost of more false positives; raising it reduces false alarms but may miss borderline paraphrasing. These values should be calibrated to the specific domain and corpus characteristics of each deployment. The system exposes the threshold as a configurable parameter to facilitate this tuning.

## VI. ADVANTAGES AND LIMITATIONS

### A. Advantages of the Proposed System

- **Interpretability:** Sentence-level annotations with matched source fragments allow users to understand precisely why a document is flagged, rather than receiving only a black-box score.
- **Multi-format Support:** Acceptance of PDF, DOCX, and TXT inputs broadens usability without requiring format conversion by the end user.
- **Dual Detection Mode:** Cosine similarity against a reference corpus and model-based probabilistic classification provide complementary signals, improving robustness when one mode is unavailable.

- RESTful API: The JSON-based API enables seamless integration with learning management systems (LMS), editorial workflows, and content management platforms.
- Persistent Analytics: The scan history and dashboard statistics provide administrators with longitudinal insight into plagiarism trends across a user base.
- Lightweight Deployment: The use of Flask and scikit-learn keeps the deployment footprint small, requiring no GPU and minimal infrastructure.

## **B. Limitations**

- Bag-of-Words Representation: TF-IDF does not capture word order or semantic meaning, making it susceptible to paraphrasing attacks where synonyms replace original terms without altering sentence structure.
- Corpus Dependency: Detection quality is directly proportional to the breadth and relevance of the reference corpus. Documents whose sources are not represented in the corpus may receive artificially low similarity scores.
- Short-Text Instability: Very short inputs produce unreliable cosine similarity values due to vector sparsity, as reflected in the greeting text example.
- Single-Language Support: The current preprocessing pipeline is optimized for English text; multilingual or code-mixed documents require additional handling.
- Scalability: The in-memory cosine similarity computation has  $O(n)$  complexity with respect to corpus size. Very large corpora will require approximate nearest-neighbor indexing for acceptable latency.

## **VII. FUTURE WORK**

Several directions present themselves for extending the capabilities of the proposed system. First, replacing TF-IDF vectors with contextual embeddings produced by transformer models such as BERT or Sentence-BERT [7] would substantially improve detection of semantically equivalent paraphrasing, where the surface form of text is altered but its meaning is preserved. Preliminary experiments in the literature suggest embedding-based systems achieve markedly higher recall against obfuscated plagiarism while maintaining competitive precision.

Second, the sentence-level analysis could be augmented with dependency parsing and entity recognition to identify structural paraphrasing patterns that preserve syntactic templates while substituting lexical items. Third, cross-lingual plagiarism detection—where a student submits a machine-translated version of a source document—represents an increasingly common evasion technique that requires multilingual embedding models and parallel corpora for effective handling.

On the infrastructure side, replacing the JSON flat-file persistence layer with a relational or document database would enable richer querying, user authentication, and role-based access control appropriate for institutional deployment. Approximate nearest-neighbor indexing (e.g., FAISS) would allow the system to scale to corpora of millions of documents without linear growth in response latency. Finally, a browser extension or Microsoft Word add-in would lower the barrier to adoption by enabling in-situ checking without requiring document upload to an external server.

## **VIII. CONCLUSION**

This paper presented an intelligent plagiarism detection system that combines TF-IDF vectorization, cosine similarity, and logistic regression classification to provide both document-level and sentence-level analysis of textual originality. The system is deployed as a Flask-based RESTful API, supports PDF, DOCX, and TXT file uploads, and delivers structured JSON responses that enable rich frontend visualization including per-sentence highlighting with matched source annotation.

Experimental evaluation on real-world documents from diverse domains demonstrated that the system produces reliable similarity scores and appropriate binary labels under typical usage conditions. The sentence-level granularity distinguishes the proposed tool from simpler classifiers by offering actionable, interpretable output that assists users in locating and revising potentially plagiarized passages rather than merely flagging a document for rejection.

While the bag-of-words nature of TF-IDF imposes inherent limitations on semantic paraphrase detection, the system's modular architecture is designed to accommodate the substitution of deeper representation models as future extensions. The combination of lightweight deployment requirements, multi-format support, persistent analytics, and a clean API makes the proposed system a practical and extensible foundation for plagiarism detection in educational and professional settings.

## REFERENCES

- [1] S. Schleimer, D. S. Wilkerson, and A. Aiken, "Winnowing: Local algorithms for document fingerprinting," in Proc. ACM SIGMOD Int. Conf. Management of Data, San Diego, CA, USA, 2003, pp. 76–85.
- [2] G. Salton and M. J. McGill, Introduction to Modern Information Retrieval. New York, NY, USA: McGraw-Hill, 1983.
- [3] R. Barrón-Cedeño, M. Vila, M. A. Martí, and P. Rosso, "Plagiarism meets paraphrasing: Insights for the next generation in automatic plagiarism detection," *Comput. Linguist.*, vol. 39, no. 4, pp. 917–947, Dec. 2013.
- [4] M. Potthast, B. Stein, A. Barrón-Cedeño, and P. Rosso, "An evaluation framework for plagiarism detection," in Proc. 23rd Int. Conf. Comput. Linguist. (COLING), Beijing, China, 2010, pp. 997–1005.
- [5] S. M. Alzahrani, N. Salim, and A. Abraham, "Understanding plagiarism linguistic patterns, textual features, and detection methods," *IEEE Trans. Syst., Man, Cybern. C*, vol. 42, no. 2, pp. 133–149, Mar. 2012.
- [6] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.
- [7] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in Proc. 2019 Conf. Empirical Methods Nat. Lang. Process., Hong Kong, China, 2019, pp. 3982–3992.
- [8] A. Grinberg, *Flask Web Development: Developing Web Applications with Python*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2018.
- [9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in Proc. 2019 Conf. North Amer. Chapter Assoc. Comput. Linguist., Minneapolis, MN, USA, 2019, pp. 4171–4186.
- [10] M. Potthast, J. Köpse, B. Stein, and M. Hagen, "Webis at PAN 2012: Cross-language plagiarism detection," in Working Notes CLEF 2012 Conf. Labs Evaluation Forum, Rome, Italy, 2012, CEUR-WS vol. 1178.