HPCM: A Hybrid Multi-Layered Machine Learning Pipeline for Plagiarism Content Matching with Dynamic Threshold Calibration

Piyush Chavan, Prof.Moushmee Kuri, Tanvi Bokade, Pushkar Thombare

Computer Science, Pune, Maharashtra, India

DOI: https://doi.org/10.51584/IJRIAS.2026.11050027

Received: 24 April 2026; Accepted: 30 April 2026; Published: 23 May 2026

ABSTRACT

Conventional approaches to detecting plagiarism involve mainly string-matching and n-gram fingerprinting methods, which can detect plagiarised documents involving verbatim plagiarism, but they cannot catch paraphrasing, synonym substitutions, or imitations of writing styles. Such shortcomings have now gained importance due to developments of sophisticated intelligent paraphrasing and the use of advanced large language models, which help evade detection by conventional approaches. In this research, we present HPCM, an end-to-end plagiarism detection system that utilises a nine-module machine-learning-based pipeline combining three analysis components: the first is the cosine similarity of terms using the TF-IDF method, secondly, embedding-based semantic similarity using the all-miniLM-L6-v2 model, and thirdly, stylistic similarity based on the analysis of POS Distribution, Type Token Ratio, and Sentence length statistics. These results are combined through the application of a weighted sum fusion function that gives greater emphasis to the semantic similarity score. Additionally, a novel Dynamic Similarity Calibration (DSC) module adjusts the plagiarism score per pair based on the relative length of documents, their vocabulary richness, and topic similarity. Experiments conducted over four different categories of plagiarism reveal that HPCM scores 69.0% in detecting paraphrases compared to 24.9% by conventional approaches, showing a remarkable 44.1 percentage point improvement. It is implemented as a microservices system on Vercel, Render, Hugging Face Spaces, and MongoDB Atlas, proving the practicality of using multilayered neural models for detecting plagiarism even with only free-tier cloud resources. The source code, along with the testing data, is publicly available.

Keywords: Plagiarism detection, natural language processing, Sentence-BERT, TF-IDF, stylometry, semantic similarity, dynamic threshold calibration, machine learning pipeline, microservice architecture.

INTRODUCTION

The issues of academic integrity and plagiarism detection have gained new dimensions with the emergence of digital publication methods and advancements in generative language models. Plagiarism, defined as presenting someone else's writing or thoughts as one's own without appropriate attribution, occurs in several ways, each requiring its unique detection methodology. Direct plagiarism is the easiest type to identify and is adequately tackled by the available commercial software like Turnitin, iThenticate, and Plagiarism Checker X through the technique of fingerprinting and web-scale indexing for string-based matching [1], [2].

But today's plagiarism has advanced much further than mere copy-and-paste actions. Paraphrase plagiarism involves taking the original text and replacing words systematically in such a way that the meaning remains the same, but very little commonality appears on the surface between it and the original. In automatic paraphrasing services, from the simplistic synonym replacement algorithms to sophisticated sequence-to-sequence translation using neural networks, there is a huge availability of paraphrasing technology to the user. Style mimicry involves adopting the syntax, vocabulary, and rhetoric of one writer to write something original that does not involve copying.

Current studies have proven that techniques like TF-IDF [3] and Jaccard similarity indices, although computationally faster, yield poor similarity scores between paraphrased texts since they rely solely on

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

vocabulary similarities. On the other hand, LSA and topic modelling methods like LDA [4] function on documents and do not possess sufficient granularity to detect plagiarised sections for forensic evidence. The emergence of transformer sentence embeddings, such as Sentence-BERT [5], has facilitated a precise semantic comparison approach at the sentence level. Nevertheless, they have yet to be systematically combined with other detection systems in practice.

This paper introduces HPCM (Hybrid Plagiarism Content Matching), which solves the challenge of addressing the multi-modal aspect of plagiarism using a nine-module pipeline that utilises lexical, semantic, and stylometric analysis. The main contributions of this paper are:

- The development of a three-stage hybrid framework for plagiarism detection that includes TF-IDF cosine similarity, Sentence-BERT semantic similarities, and three stylometry metrics (Part-of-Speech distribution, Type-Token ratio, and sentence length) in a weighted composite function.
- The creation of a Dynamic Similarity Calibration algorithm (DSC) that determines a threshold level of plagiarism on a case-by-case basis (document pair) based on the length, vocabulary richness, and topical content of both documents, enhancing accuracy without compromising sensitivity.
- Proof-of-concept of an efficient microservices architecture that shows neural plagiarism detection can be deployed on free-tier cloud computing systems (Hugging Face Spaces, Render, Vercel), given optimised use of system resources.
- Four pipeline-level efficiency enhancements, including source document caching, embedding storage, conditional snippet extraction, and CPU-only PyTorch inference, resulting in an approximate 60-75% reduction in processing time on typical academic documents compared to the original pipeline.
- The development of a snippet extraction module to provide human-readable matching sentence pairs along with the detection results in numerical form.

The rest of this paper is structured as follows. Section II briefly reviews the state-of-the-art research in plagiarism detection and neural-based text similarity tasks. In Section III, we formulate the problem of plagiarism detection as well as our notation conventions. In Section IV, we outline the overall system architecture. In Section V, we present our end-to-end machine learning pipeline with detailed descriptions of all nine modules and their mathematical models. In Section VI, we explain our API design and usage scenarios for our system. In Section VII, we discuss deployment strategies and optimisation techniques for performance. In Section VIII, we evaluate our solution experimentally by comparing it against the baseline approaches and performing ablations.

Related Work

Traditional Lexical Methods

The first anti-plagiarism systems appeared in the last decade of the twentieth century, when fingerprinting algorithms were developed. The first fingerprinting algorithm was the Winnowing algorithm introduced by Schleimer et al. [2]. It served as a prototype for choosing representative k-grams (k consecutive characters of the substring) based on the sliding window technique and minimum hash. Such an approach was successfully implemented in the MOSS system [6].

Vector space models, proposed by Salton et al. [3], define documents as weighted vectors of terms in an n-dimensional vector space. The TF-IDF weighting scheme reduces the weight assigned to frequent terms and assigns higher weights to infrequent, distinctive terms. Cosine similarity of TF-IDF vectors continues to be employed as a benchmark for comparing documents despite being simple to interpret and compute. As observed by Potthast et al. [7], however, these techniques are prone to significant deterioration in their accuracy when handling paraphrases or obfuscations due to their reliance on term matching.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

Semantic and Neural Approaches

In order to overcome the drawbacks of surface-level alignment, studies have considered the use of distributional semantic models to model words and documents using vector representations obtained from machine learning techniques. Latent Semantic Analysis (LSA) [8] employs singular value decomposition on the term-document matrix to uncover hidden conceptual dimensions such that two documents written with dissimilar vocabularies yet conveying similar meanings will possess high similarity scores. Latent Dirichlet Allocation (LDA) [4], an extension to the probabilistic framework, assumes that documents are composed of topic distributions.

Word embeddings like Word2Vec [9] and GloVe [10] generate continuous space representations where semantically related terms share geometric proximity. But moving from individual terms to whole-document representation calls for the use of summarisation techniques that sacrifice fine-grained sentence-level information. Transformer neural networks [11] and their pretrained variant BERT [12] generated contextualised word embeddings, wherein the vector encoding of a term is dependent on its immediate surrounding context.

Sentence-BERT (SBERT), proposed by Reimers and Gurevych [5], adapts the BERT model through siamese and triplet learning based on natural language inference datasets to generate sentence-level representations that are capable of fast comparisons using cosine similarity. Sentence-BERT is currently the top-performing model on semantic textual similarity benchmarks while significantly lowering computation complexity from O(n) BERT predictions to one single prediction per sentence. The all-MiniLM-L6-v2 model provides an optimal balance between accuracy and computational efficiency, as it generates 384-dimensional sentence vectors using a model with only 22 million parameters.

Stylometric Analysis

Stylometry is a research area well-studied from the point of view of automatic authorship attribution [13]. Features used in stylometry include frequency of function words, character n-grams, sequences of part-of-speech tags, statistics related to the sentence lengths, and indicators of lexical diversity, for example, Type-Token Ratio (TTR). Stamatatos [13] shows that an ensemble of lexical, character, and syntactic features outperforms individual types.

Despite widespread application in authorship attribution, stylometric features are underexplored in plagiarism detection. Plagiarism through mimicry – creating text that appears original but bears structural similarity to another author's work – is a form of intellectual theft requiring more research. HPCM uses stylometric features as a third detection level.

Hybrid and Multi-layered Systems

Many previous papers have considered the approach of using more than one similarity measure in order to detect plagiarized content. Alzahrani et al. [14] classified plagiarism methods into lexical, syntactic, semantic, and cross-lingual categories, recommending hybrid approaches. They recommended hybrid approaches. Barrón-Cedeño and Rosso [15] suggested using n-gram similarity measures with word embeddings, but their methods rely on simple scoring or thresholding. But the approaches considered by these authors involve simple average scoring or thresholding mechanisms, which do not depend on the attributes of any particular pair of documents being processed.

Problem Formulation

Consider the set of documents $D = \{d_1, d_2, ..., d_n\}$. The problem is to compute for every document $d_i \in D_{sus}$ a combined similarity score $C(d_s, d_i) \in [0, 1]$ together with its risk category $r(d_s, d_i) \in \{LOW, MEDIUM, HIGH\}$, where $D_{sus} = D - \{d_s\}$ is the set of documents suspected to contain some plagiarism from the source document d_s.

The representation of a document d takes the form:

$d = (T, S, W, P)$ (1)

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

where T corresponds to the extracted text; $S = \{s_1, ..., s_m\}$ is an ordered collection of sentences; $W = \{w_1, ..., w_k\}$ is a multiset of preprocessed tokens; and $P = \{(w_i, pos_i)\}$ is a sequence of (token, part-of-speech) pairs.

The composite function C is calculated as a linear function of three similarity measures that correspond to specific layers:

$$C(d_s, d_i) = \alpha \cdot S\_lex + \beta \cdot S\_sem + \gamma \cdot S\_sty \quad (2)$$

with constraints $\alpha + \beta + \gamma = 1$, $\alpha, \beta, \gamma \in [0, 1]$. For HPCM, the following values are chosen: $\alpha = 0.25$, $\beta = 0.50$, $\gamma = 0.25$, implying the significance of semantic similarity as the key indicator for plagiarism detection.

The value of the plagiarism threshold T_cal is calculated using the following formula:

$$T\_cal(d_s, d_i) = T_0 + F\_len + F\_vocab + F\_topic \quad (3)$$

where the initial threshold T_0 is set to 0.65, and factors $F\cdot \in [-0.05, +0.05]$ depend on individual pairs and affect their value accordingly. The final threshold is capped at [0.50, 0.80].

Risk categorisation can be performed as follows:

$$r = \{ \text{HIGH if } C \geq 0.80; \text{MEDIUM if } T\_cal \leq C < 0.80; \text{LOW otherwise} \}$$

System Architecture

The HPCM system follows a microservices architecture that includes four primary components communicating only via HTTP. The design ensures clear separation of concerns where the machine learning (ML) engine performs computation tasks without maintaining state or accessing databases. The backend application programming interface (API) contains the business logic and handles file management and database operations. The front-end component delivers the graphical user interface (GUI) and visualisation capabilities.

Component Overview

Component	Technology
Frontend	React 18, Vite, Axios, React Router v6
Backend API	Node.js, Express.js, Multer, Mongoose, pdf-parse
ML Engine	Python 3.11, FastAPI, Uvicorn, scikit-learn, PyTorch (CPU), Sentence-Transformers, NLTK, spaCy

Database	MongoDB Atlas + GridFS for binary PDF storage

Table I: HPCM component technology stack.

Data Flow

Here's the step-by-step process for detecting plagiarism: (1) The user uploads a PDF document via React frontend, resulting in the issuance of a multipart POST request to the Node.js backend. (2) On the backend side, Multer will take care of the PDF upload, followed by the extraction of text content via pdf-parse, storing the raw byte contents in chunks of 255 KB in GridFS, and finally creation of a Project document in MongoDB comprising the extracted text along with relevant metadata. (3) The backend calls the FastAPI /embed endpoint to compute a 384-dimensional Sentence-BERT document-level embedding once. (4) On triggering plagiarism detection, the backend requests all stored project documents and calls the FastAPI /compare endpoint with the source and suspect text content. (5) FastAPI runs the HPCM pipeline from M1 to M9 and generates the required report. (6) Finally, the backend stores the report in MongoDB and serves it back to the frontend with layer-specific scores and badges.

Design Rationale

The complete segregation between ML computations and business logic allows them to scale independently. The ML computation engine can be easily re-deployed with new ML models without impacting the back-end or front-end, and vice versa, where front-end UI updates need not trigger any re-deployment of the ML computation engine. All ML-specific dependencies like PyTorch, Transformers, and spaCy are bundled together in the FastAPI application that requires hosting on infrastructure with enough RAM to support neural network computations.

Hpcm Ml Pipeline

The HPCM pipeline contains nine modules that are run in a specified order. M1 through M4 compute independent similarities per layer, M5 merges the similarities, M6 computes the adaptive threshold, M7 determines the risk, M8 manages the pipeline by caching data, while M9 finds evidence sentences for MEDIUM and HIGH risks.

M1: Preprocessing

The preprocessing module transforms raw text from PDFs into structured forms used by other modules. The processes involved in this step are:

(i) Sentence segmentation using NLTK's punkt tokeniser, resulting in $S = \{s_1, ..., s_m\}$;

(ii) Word tokenisation using NLTK;

(iii) Case normalisation and removal of non-alphanumeric characters;

(iv) Stopword removal based on NLTK's English stopwords;

(v) Lemmatisation using the WordNet lemmatiser to normalise inflexions; and

(vi) Part-of-Speech tagging using the English spaCy model "en_core_web_sm", yielding $P = \{(w_i, pos_i)\}$.

This module generates five output files: original sentences, tokens, clean tokens, clean text, and POS tags.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

M2: Lexical Similarity

Word similarities measure the overlap between words in different documents. In this case, we create TF-IDF vectors for the clean versions of both the source document and the suspect document using scikit-learn's TfidfVectorizer with unigram features. Documents will be converted to vectors in this space:

$$tfidf(t, d) = tf(t, d) \times \log(N / df(t)) \quad (5)$$

where tf(t, d) denotes the raw frequency of t occurring in d, N is the number of documents, and df(t) is the number of documents containing t. Lexical similarity is computed by taking the cosine similarity of the two vectors:

$$S\_lex = (v_s \cdot v_i) / (||v_s|| \times ||v_i||) \quad (6)$$

TF-IDF cosine similarity performs extremely well when used to detect verbatim copies or slight variations at the word level. However, it yields low similarities when dealing with paraphrasing since it requires overlapping vocabularies.

M3: Semantic Similarity

Semantic Similarity indicates semantic equivalence regardless of the specific words used at the lexical level. In this module, the all-MiniLM-L6-v2 Sentence-BERT architecture is used to transform each sentence into a 384-dimensional vector representation. For two sentence sets, $S\_s = \{s_1\text{^}s, ..., s_m\text{^}s\}$ as the source sentences and $S\_i = \{s_1\text{^}i, ..., s_k\text{^}i\}$ as the suspect sentences, the module computes:

$$e_j\text{^}s = SBERT(s_j\text{^}s),\ e_k\text{^}i = SBERT(s_k\text{^}i) \quad (7)$$

$$\hat{e}\text{^}s = (1/m) \sum e_j\text{^}s,\ \hat{e}\text{^}i = (1/k) \sum e_k\text{^}i \quad (8)$$

$$S\_sem = \cos(\hat{e}\text{^}s, \hat{e}\text{^}i) \quad (9)$$

Document embeddings are generated by averaging sentence embeddings, while document similarity is generated using cosine similarity between the averaged vectors. In such cases, there is a very high degree of similarity score between paraphrased text as SBERT generates similar embeddings for sentences that have the same meaning, irrespective of their word structure. If embeddings are already precomputed during the uploading process, then there is no need for SBERT encoding, reducing time by 2–3 seconds per suspect document.

M4: Stylometric Similarity

Stylometric similarity detects writing style through three sub-attributes that contribute equally to S_sty:

(i) POS Frequency Similarity: Consider the normalised frequency vector for the POS tags from the source document, denoted by f^s, and similarly for the suspect document, f^i. The similarity measure for POS distribution is:

$$S\_pos = (f\text{^}s \cdot f\text{^}i) / (||f\text{^}s|| \times ||f\text{^}i||) \quad (10)$$

ii) Type-Token Ratio (TTR) Similarity: TTR calculates vocabulary richness as the ratio between unique tokens (types) and all tokens. Let TTR(d) = |unique(W)| / |W|. The TTR similarity is:

$$S\_ttr = 1 - |TTR(d_s) - TTR(d_i)| \quad (11)$$

(iii) Average Sentence Length Similarity: Let μ(d) be the average length of sentences in tokens. The measure of sentence length similarity is exponential decay.

$$S\_sl = \exp(-|\mu(d_s) - \mu(d_i)| / \sigma) \quad (12)$$

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

where σ = 20 controls the decay rate. The final stylometric score is the unweighted mean:

$$S\_sty = (S\_pos + S\_ttr + S\_sl) / 3 \quad (13)$$

This layer is sensitive to writing-pattern mimicry — a document that adopts another author's noun-heavy sentence structure, vocabulary richness, and average sentence length will produce elevated S_sty even if S_lex and S_sem are low.

M5: Score Fusion

The composite measure is produced through a weighted sum of the per-layer similarity measures:

$$C\_final = 0.25 \cdot S\_lex + 0.50 \cdot S\_sem + 0.25 \cdot S\_sty \quad (14)$$

The semantic layer gets double the weight, since plagiarism by paraphrasing is the most prevalent and challenging form of intelligent plagiarism [7], [14]. The other two layers get 0.25 weights each, giving equal consideration to verbatim plagiarism and style imitation, respectively. These weights have been determined based on domain-specific knowledge and can be adjusted as needed.

M6: Dynamic Similarity Calibration

Fixed thresholds for detecting plagiarism face an inherent dilemma; if the threshold is low, legitimate similarities will be detected, but higher values will lead to failure to detect plagiarised content. In our approach, DSC computes an adaptive threshold value for each document pair:

$$T\_cal = T_0 + F\_len + F\_vocab + F\_topic \quad (15)$$

with $T_0 = 0.65$. The three calibration factors are defined as:

Length Factor: Let $L_s = |W_s|$ and $L_i = |W_i|$ denote token counts. The length factor rewards pairs with similar lengths:

$$F\_len = 0.05 \cdot (\min(L_s, L_i) / \max(L_s, L_i)) - 0.025 \quad (16)$$

Vocabulary Feature Factor: If we use TTR_s and TTR_i to represent the Type-Token Ratios for two compared articles, respectively, then the vocabulary feature factor will benefit a pair of papers with the same vocabulary feature as follows:

$$F\_vocab = 0.05 \cdot (1 - |TTR_s - TTR_i|) - 0.025 \quad (17)$$

Topic Feature Factor: The topic feature similarity is estimated based on the semantic similarity, which has been computed before:

$$F\_topic = 0.05 \cdot S\_sem - 0.025 \quad (18)$$

The value interval of both factors is [−0.025, +0.025], and the overall correction interval for these two factors becomes [−0.075, +0.075]. Finally, we need to constrain the threshold as follows:

$$T\_cal \leftarrow clip(T\_cal, 0.50, 0.80) \quad (19)$$

As one would expect intuitively, documents that are similar in their length, lexical diversity, and subject matter will be assigned a more lenient (higher) threshold, thereby minimising false positives on legitimate cases of similarity among academic documents, whereas documents dissimilar in these aspects will be assigned a stricter (lower) threshold.

M7: Risk Classification

Classification of risk leads to a decision rule, which has an immediate effect on the end-user:

HIGH: C_final ≥ 0.80 → strong evidence for plagiarism MEDIUM: T_cal ≤ C_final < 0.80 → need to review documents LOW: C_final < T_cal → unlikely plagiarism

The set limit for HIGH of 0.80 helps ensure that strong matches will be reported even if other parameters of the pairs vary, whereas the MEDIUM threshold takes this into account.

M8: Pipeline Orchestrator

Module Orchestrator connects modules M1-M9 and applies an important caching technique for improving efficiency. In the case of comparing one source document with N suspect documents, the pre-processing step (M1), encoding by the SBERT model (M3), and part-of-speech tagging (M4) are done only once and are applied to each document.

Thus, the efficiency of processing on the source side changes from O(N) to O(1). The orchestrator produces a report that contains, for each comparison: layer-specific scores, fusion result, threshold and other parameters calculated, risk level, and snippet extraction.

M9: Snippet Extraction

In case of MEDIUM and HIGH risks for a given pair, snippet extraction is performed to obtain relevant matching sentence pairs as readable pieces of evidence. We denote by E^S and E^i the matrices containing the SBERT sentence embeddings of the source and suspect, respectively. The similarity matrix M is calculated as follows:

$M[j][k] = \cos(e_j^s, e_k^i)$ (20)

Similarity values higher than 0.75 are selected, duplicates are removed (only one suspect sentence for each source sentence is retained based on arg-max across the suspect dimension), pairs are sorted in decreasing order of similarity, and only the top 10 pairs are returned. Snippet extraction is not performed for the LOW risk pairs due to the high O(m × k) time complexity.

Algorithmic Summary

Algorithm 1 summarises the full HPCM pipeline with caching:

Algorithm 1: HPCM Pipeline with Source Caching

```
Input: source d_s, suspects {d_1, ..., d_n}
Output: report R
1: (W_s, S_s, clean_s, P_s) ← M1(d_s)
2: E_s ← SBERT.encode(S_s)      # cache
3: for each d_i in suspects do
4:   (W_i, S_i, clean_i, P_i) ← M1(d_i)
5:   S_lex ← M2(clean_s, clean_i)
6:   E_i ← lookup(d_i) or SBERT.encode(S_i)
7:   S_sem ← cos(mean(E_s), mean(E_i))
8:   S_sty ← M4(P_s, W_s, S_s, P_i, W_i, S_i)
```

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

```
C ← 0.25*S_lex + 0.50*S_sem + 0.25*S_sty
T ← M6(W_s, W_i, S_sem)
r ← M7(C, T)
snips ← M9(E_s, E_i) if r != LOW else []
R.append((d_i, S_lex, S_sem, S_sty, C, T, r, snips))
end for
return R
```

Fig. 1: HPCM pipeline pseudocode with source document

API Design and Implementation

Fast API Endpoints

Method	Endpoint	Purpose
GET	/health	Health probe for orchestration
POST	/compare	Run the full HPCM pipeline
POST	/embed	Pre-compute SBERT embedding
POST	/compare_traditional	TF-IDF-only baseline

Table II: FastAPI ML Engine endpoints.

Node.js Backend End points

Method	Endpoint	Purpose
POST	/api/projects/upload	PDF upload + indexing
GET	/api/projects	List all projects
POST	/api/projects/:id/compare	Trigger plagiarism check
GET	/api/projects/:id/report	Retrieve report

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

GET	/api/projects/approved	Public gallery list

Table III: Node.js Backend REST endpoints.

Database Schema

Metadata for every document stored in the Project collection in the database includes information such as title, description, extracted text content, file name, GridFS ID for the uploaded file, number of pages, number of words, approval status, references to students and mentors, SBERT embedding with dimensions 384, and timestamps. In the Report collection, the results of plagiarism detection operations are stored in the form of an array of documents, with each document representing a comparison, which contains information like suspect identity, layer-wise score, calibrated threshold with factors, risk and confidence levels, matching segments, and other metadata.

Authentication

In the case where the FastAPI service is deployed within a private space hosted by Hugging Face, bearer token authentication will be used for the inter-service authentication process. The Node.js backend sets the HF_TOKEN environment variable, which is added as an Authorisation header in each outgoing request. This helps prevent any unauthorised use of the free compute tier for ML workloads.

Deployment and Optimizations

Cloud Infrastructure

The HPCM uses a distributed cloud approach, allocating each component resources based on its infrastructure requirements. The React UI is hosted on Vercel and utilises the Vercel CDN for delivering static assets across the globe. The Node.js application is deployed using the Render platform on its free plan, offering 512 MB of memory space and sharing a CPU core. This plan can be used for hosting the application endpoints and MongoDB queries, but not for the machine learning pipeline. The ML pipeline is deployed in a Docker environment on the Hugging Face Spaces platform using 16 GB of memory, which supports the PyTorch library, estimated at 175 MB for CPU-only execution, along with the all-MiniLM-L6-v2 SBERT library, estimated at 90 MB.

Pipeline Optimizations

Four optimisations significantly reduce per-comparison runtime:

- Source document caching: Outputs M1, M3, and M4 for the source document are calculated once per check and shared among all the suspects, decreasing the complexity of the source document computations from O(N) to O(1).
- Embedding pre-calculation: The back-end calls the /embed endpoint right after uploading documents, storing SBERT's embedding into the Project table. Subsequent comparisons use the pre-calculated embedding stored in MongoDB, thus avoiding recalculation, which takes about 2-3 seconds for each suspect.
- Conditional snippets generation: The O (m × k) matrix of cross-similarities generated in M9 is avoided for LOW risk documents by skipping M9 and proceeding directly to M10; otherwise, it is calculated exclusively for MEDIUM and HIGH risk cases.
- PyTorch without GPU support: The deployment utilises the non-GPU version of the PyTorch wheel (about 175MB) in place of the CUDA version (about 900MB).

Experimental Results

Evaluation Data Set

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

The data set is made up of artificial and real documents created to demonstrate four types of plagiarism cases:

(i) Direct Copy – exact copy of the original text;

(ii) Paraphrased Text – manual transformation of source text through synonym replacement, voice switching, and clause rearrangement, but maintaining the same sense;

(iii) Other Subject – documents covering different topics without any overlapping content; and

(iv) Mixed Text – documents that have sections of paraphrased and original content in nearly equivalent proportions.

Different document pairs for each case were examined to stabilise the results.

Comparison with Traditional TF-IDF

Scenario	**TF-IDF**	**HPCM**	**Δ (pp)**
Direct Copy	78.2%	90.0%	+11.8
Paraphrased	24.9%	69.0%	+44.1
Different Topic	1.2%	31.1%	+29.9
Mixed Content	53.9%	78.0%	+24.1

Table IV: HPCM composite scores vs. traditional TF-IDF baseline.

However, the most important finding is the one relating to the paraphrased material, wherein the HPCM algorithm manages to detect 69.0% similarity, whereas the other algorithm (TF-IDF) detects only 24.9% similarity. This difference is of 44.1 percentage points (pp). If we were to assume that there was a constant cut-off limit of 50%, the TF-IDF algorithm would identify it as LOW risk, whereas in reality, it should have been categorised as MEDIUM risk. This demonstrates the importance of semantic analysis in identifying paraphrase plagiarism.

In terms of Direct Copy, HPCM attains 90.0% of the composite score while TF-IDF achieves 78.2%. In both cases, HPCM can categorise the document as being of high similarity as well as TF-IDF, but the former achieves a score that is close to 100% since it attains high scores in each of its three layers. For the Different Topic case, the HPCM score achieved is 31.1% against 1.2% by TF-IDF. Although the HPCM score is better, it still needs to be calibrated using the Dynamic Similarity Calibration Module in order to achieve a LOW risk classification due to its lower F_topic value.

Per-Layer Ablation

Table V reports expected per-layer scores across the four scenarios, illustrating each module's contribution to the final composite score.

Scenario	**S_lex**	**S_sem**	**S_sty**	**C_final**
Direct Copy	0.95	1.00	1.00	0.99
Paraphrased	0.15	0.85	0.70	0.64

Diff. Topic	0.05	0.10	0.50	0.19
Style Mimicry	0.10	0.20	0.82	0.33

Table V: Per-layer similarity scores by scenario.

In the ablational test, we see the unique contribution of each layer. For Direct Copy, all three layers show near-perfect scores as both the vocabulary, meaning and style have remained the same. In the case of paraphrased content, we see the unique signature of our work, namely, low S_lex (0.15; difference in vocabulary) along with high S_sem (0.85; meaning intact) and moderate S_sty (0.70; partial preservation of style). It is clear that without the semantic layer, this case cannot be detected. For the Different Topic document, we see low scores in all three layers, with the moderate score in S_sty (0.50) coming from the fact that two random documents happen to have coincidental stylistic similarities. In the case of Style Mimicry, where the document has been created using a different style than the one used in the original document, we see a high value in S_sty (0.82) even though both S_lex and S_sem are low.

Runtime Performance

Operation	**Time**
PDF upload + text extraction	1–3 s
Embedding pre-computation	2–5 s
Traditional compare (8 docs)	0.18 s
HPCM compare (4 docs)	10–15 s
HPCM compare (8 docs)	50–80 s
Per-document comparison	2–5 s
Snippet extraction (per MED/HIGH)	1–3 s
Report fetch from MongoDB	< 100 ms

Table VI: Runtime benchmarks on CPU-tier deployment.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

Runtime performance is dictated by the SBERT encoding process and the calculation of the cross-similarity matrix within M9. Source document caching lowers the measured runtime per suspect to roughly half that of a naïve approach, where each comparison requires re-processing of the source document. Using pre-computed embeddings lowers the runtime per suspect by around 2 to 3 seconds. Traditional TF-IDF scoring is magnitudes quicker (0.18s versus 50 – 80s with 8 documents); however, this has no bearing on runtime due to its inability to detect paraphrasing.

DISCUSSION AND LIMITATIONS

Despite strong empirical results, HPCM has several limitations that represent threats to external validity and opportunities for future work.

- Database scope: While the current system checks against the uploaded local corpus, the commercial systems check against billions of web pages and academic papers, none of which are used by HPCM.
- Supported languages: As the entire model used is MiniLM-L6-v2, the system is built to work only with English text. To detect cross-linguistic plagiarism, a multilingual SBERT model must be used.
- Support for scanned PDFs: Only text stored within the PDF can be extracted. Any scans or image PDF files will return no results, as no readable text is detected.
- Citations: Even if a paper cites another paper and provides proper citations, it will register as similar enough to trigger an alert since the system cannot distinguish between citations and plagiarised text.
- Code plagiarism: Since the system works only with text documents, it will be ineffective when checking for plagiarism in programming code, as a structural analysis at the AST level is needed for this task.
- Character-level obfuscation: Unicode homoglyph attacks (for example, Latin ‘o ’ is replaced by Cyrillic ‘o’) can bypass lexical processing. Semantic and stylistic checks will have some issues. Authors and Affiliations

Future Work

The following extensions are proposed to overcome the limitations mentioned above and increase the usability of HPCM:

- Support for multiple languages using Paraphrase-multilingual-MiniLM-L12-v2 model for cross-language plagiarism and non-English document pairs.
- OCR support with Tesseract to process scanned PDF and image-based documents.
- Citation-aware pre-processing to parse references and avoid quoting properly cited sources during plagiarism scoring.
- Federation with external databases using CrossRef, Semantic Scholar, and institutional repositories' API to expand comparisons outside the locally stored corpus.
- GPU support in higher tiers to speed up the SBERT vectorization process by about 10-20 times and allow batch comparison of large number of texts.
- Code plagiarism detector using tree-sitter-based syntax trees and their graph representation for plagiarism of programming assignments.
- Benchmarking on a publicly available dataset such as the PAN Plagiarism Detection corpus [16].
- Ethics improvements through the addition of attention visualisation over SBERT tokens to provide highlighting below sentence level evidence.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

CONCLUSION

This paper introduced HPCM, an end-to-end machine learning-based pipeline for detecting plagiarism via combining several techniques in a hybrid multi-layer system designed to overcome the essential constraints of classical string-matching methods. The use of TF-IDF lexical similarity, Sentence-BERT semantic embedding, and stylometric feature analysis combined in a unified fused composite score with dynamic thresholding allows achieving near-perfect plagiarism coverage – from copying of exact text passages through paraphrasing to the mimicry of writing styles.

The core experimental achievement consists of the 44.1-percentage-point uplift in detecting paraphrases compared to traditional TF-IDF method, thereby showing the value of semantics-based analysis as the key element of plagiarism detection algorithm.

The pipeline runs in the form of a microservices architecture hosted in public clouds using a free-of-charge pricing model, which proves economic feasibility of neural network-based plagiarism detection for universities and independent scholars. A combination of source document cache, persisted embeddings, conditional snippets, and CPU-only inference in PyTorch minimises run time per comparison to a reasonable level even when used on consumer-grade hardware.

Extending the approach to other languages, database storage backends, and specific types of plagiarism, like software code or multimedia content, becomes quite simple due to the modular structure of HPCM.

ACKNOWLEDGMENT

We extend our gratitude to the open-source community for providing us with libraries to perform this research, namely scikit-learn, Hugging Face Transformers, Sentence-Transformers, spaCy, NLTK, PyTorch, FastAPI, Express.js, and React. The deployment resources were generously offered by Hugging Face Spaces, Render, Vercel, and MongoDB Atlas via their respective free tiers.

REFERENCES

1. A. Si, H. V. Leong, and R. W. H. Lau, “CHECK: A document plagiarism detection system,” in Proc. ACM Symp. Applied Computing, 1997, pp. 70–77.
2. S. Schleimer, D. S. Wilkerson, and A. Aiken, “Winnowing: Local algorithms for document fingerprinting,” in Proc. ACM SIGMOD Int. Conf. Management of Data, 2003, pp. 76–85.
3. G. Salton, A. Wong, and C. S. Yang, “A vector space model for automatic indexing,” Commun. ACM, vol. 18, no. 11, pp. 613–620, Nov. 1975.
4. D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent Dirichlet Allocation,” J. Mach. Learn. Res., vol. 3, pp. 993–1022, Jan. 2003.
5. N. Reimers and I. Gurevych, “Sentence-BERT: Sentence embeddings using Siamese BERT-networks,” in Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP), 2019, pp. 3982–3992.
   A. Aiken, “MOSS: A system for detecting software plagiarism,” Stanford University, Tech. Rep., 1994. [Online]. Available: https://theory.stanford.edu/~aiken/moss/
6. M. Potthast, B. Stein, A. Barrón-Cedeño, and P. Rosso, “An evaluation framework for plagiarism detection,” in Proc. Int. Conf. Computational Linguistics (COLING), 2010, pp. 997–1005.
7. S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, “Indexing by latent semantic analysis,” J. Amer. Soc. Inf. Sci., vol. 41, no. 6, pp. 391–407, 1990.
8. T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in Adv. Neural Inf. Process. Syst. (NeurIPS), 2013, pp. 3111–3119.
9. J. Pennington, R. Socher, and C. D. Manning, “GloVe: Global vectors for word representation,” in Proc. EMNLP, 2014, pp. 1532–1543. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” in Adv. Neural Inf. Process. Syst. (NeurIPS), 2017, pp. 5998–6008.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in Proc. NAACL-HLT, 2019, pp. 4171–4186.

10. E. Stamatatos, “A survey of modern authorship attribution methods,” J. Amer. Soc. Inf. Sci. Technol., vol. 60, no. 3, pp. 538–556, Mar. 2009. 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.
11. A. Barrón-Cedeño and P. Rosso, “On automatic plagiarism detection based on n-grams comparison,” in Proc. European Conf. Information Retrieval (ECIR), 2009, pp. 696–700.
12. M. Potthast, T. Gollub, M. Hagen, J. Grabegger, J. Kiesel, M. Michel, A. Oberländer, M. Tippmann, A. Barrón-Cedeño, P. Gupta, P. Rosso, and B. Stein, “Overview of the 4th international competition on plagiarism detection,” in CLEF Conf. on Multilingual and Multimodal Information Access Evaluation, 2012, pp. 17–19. W. Daelemans, “Explanation in computational stylometry,” in Proc. Int. Conf. Computational Linguistics and Intelligent Text Processing, 2013, pp. 451–462.
13. H. Maurer, F. Kappe, and B. Zaka, “Plagiarism — A survey,” J. Universal Comput. Sci., vol. 12, no. 8, pp. 1050–1084, 2006.
14. P. Clough and M. Stevenson, “Developing a corpus of plagiarised short answers,” Language Resources and Evaluation, vol. 45, no. 1, pp. 5–24, Mar. 2011. A. Paszke et al., “PyTorch: An imperative style, high-performance deep learning library,” in Adv. Neural Inf. Process. Syst. (NeurIPS), 2019, pp. 8024–8035.