AI Academic Plagiarism Checker with Code Analysis and Multilingual Model

Dr. Gajanan P Arsalwad[1], Sanika Deshmukh[2], Vrushali Gawai[3], Trupti Dhandar[4]
[1]*Professor Department of Information Technology, Trinity College of Engineering and Research, Savitribai Phule Pune University, Pune, Maharashtra, India*
[2,3,4]*Student, Department of Information Technology, Trinity College of Engineering and Research, Savitribai Phule Pune University, Pune, Maharashtra, India*

Abstract—**While AI-generated text and global digital availability have revolutionized content creation, they have simultaneously escalated the complexity of academic plagiarism. Traditional lexical matching methods are increasingly obsolete against sophisticated paraphrasing, cross-lingual translation, and AI-synthetic code. This paper presents SmartPlag, an advanced multi-modal framework designed to transcend simple word-by-word matching. SmartPlag integrates a multi-stage pipeline comprising real-time language detection, concurrent pivot-language translation, and TF-IDF semantic vectorization to identify cross-lingual plagiarism. Uniquely, the system introduces a Cognitive Effort Imbalance (CEI) behavioural analysis module and Stylometric Machine Learning classification to distinguish between organic human logic and automated generation in programming source code. By analysing entropy in naming conventions and variance in structural complexity, SmartPlag provides a "cognitive footprint" of submissions. Experimental results demonstrate high efficacy in detecting direct, translated, and AI-assisted plagiarism, offering a scalable and intelligent solution for preserving academic integrity in the era of generative AI.**

I. INTRODUCTION

The rapid growth of digital education has made academic integrity increasingly difficult to maintain. Institutions now face two major challenges: traditional internet-based plagiarism and Generative AI. Conventional plagiarism detection systems based on lexical matching are ineffective against cross-lingual plagiarism and AI-generated content, as they cannot capture semantic similarities or structurally unique text [25][18].

A major issue is cross-lingual plagiarism, where students translate content from other languages to bypass mono-lingual detection systems [3]. Similarly, Large Language Models (LLMs) enable “synthetic plagiarism,” where ideas are copied but expressed in new wording, making traditional similarity checks unreliable [21].

To address these issues, this paper introduces SmartPlag, a multi-modal framework that combines translation bridges, behavioral logic analysis using Cognitive Effort Imbalance (CEI) for code, and stylometric features. Instead of analyzing only what was written, SmartPlag focuses on how it was written, providing a stronger approach to academic integrity.

II. LITERATURE REVIEW

The evolution of plagiarism detection has historically mirrored the progress in Software Engineering and Natural Language Processing (NLP). Traditional detection frameworks primarily rely on lexical comparison mechanisms, such as string matching, keyword frequency analysis, and N-gram overlap. While highly effective at identifying direct verbatim copying, these systems exhibit significant "blind spots" when encountering paraphrased, translated, or structurally obfuscated content [25].

A. MACHINE LEARNING AND SEMANTIC ANALYSIS

Recent academic discourse has pivoted toward integrating Machine Learning (ML) and semantic

analysis to enhance detection granularity. Benchmarks such as CodeMirage have set new precedents for evaluating AI-generated code, emphasizing a shift from surface-level text analysis to semantic program comprehension [1]. Similarly, the PAN 2025 shared tasks underscore the importance of stylometric analysis and writing style consistency across multiple languages to identify AI-synthetic artifacts [2]. Research by Wagh et al. have also demonstrated the rising efficacy of Latent Semantic Analysis and Cosine Similarity in identifying paraphrased overlap [23]

B. CHALLENGES IN LOW-RESOURCE AND MULTI-MODAL DETECTION

Emerging models have attempted to bridge the gap in low-resource languages (e.g., Marathi) by combining TF-IDF with BERT embeddings [3]. While these hybrid textual models improve semantic accuracy, they remain largely siloed from the domain of programming source code. Conversely, code-centric approaches like token sequence normalization as extensively documented by Sağlam et al.focus on structural "sameness" by stripping extraneous syntax, but often ignore cross-lingual environments or specific behavioural markers of AI-generation [17, 22].

C. RESEARCH GAP: THE BEHAVIOURAL VOID

Despite these advancements, a significant gap remains: the lack of a unified, multi-modal framework that addresses the intersection of multilingual text and AI-synthetic code. Most existing systems are limited by their reliance on "what exists" (lexical overlap) rather than "how it was created" (behavioral logic) [8]. Current LLM-based detection models for academic journals are highly computationally demanding [21].

III. TRADITIONAL ARCHITECTURES AND LIMITATIONS

Existing Work: Many early papers rely on simple string matching (like the Rabin-Karp algorithm) or basic TF-IDF (Term Frequency-Inverse Document Frequency) to find overlapping words in essays.

The Issue Found: These systems are strictly monolingual. If a student finds an essay in Spanish or Marathi, translates it to English using Google Translate, and submits it, traditional TF-IDF systems will return a 0% similarity score because the exact English vocabulary doesn't match the original foreign text.

A. SOURCE CODE PLAGIARISM DETECTION (AST & TOKENIZATION)

Existing Work: Tools like Stanford's MOSS (Measure of Software Similarity) or JPlag tokenize code and compare Abstract Syntax Trees (AST) to find structural similarities, even if variables are renamed.[12]

The Issue Found: These tools are incredibly heavy and require compiling the code. They also struggle to differentiate between multiple students using the exact same standard boilerplate code provided by a professor versus actual malicious copying. Furthermore, they are blind to how the code was written (i.e., human vs. AI generation).[13]

B. AI-GENERATED CONTENT DETECTION

The Issue Found: These models are computationally expensive to run on standard university servers. More importantly, while there is research on detecting AI in essays, there is a massive gap in detecting AI-generated source code without relying on massive, opaque neural networks.

C. TOKEN BASED PLAGIARISM DETECTION USING JPLAG

Figure 1 illustrates the standard architecture for Token-Based Plagiarism Detection, commonly utilized by systems like JPlag. In this approach, source code submissions are stripped of formatting and converted into language-independent tokens or parse trees before pairwise comparison. SmartPlag adopts a heavily optimized variant of this methodology for its Source Code pipeline [17]. By applying rigorous syntax normalization (removing comments, docstrings, and string variables) before executing the Ratcliff/Obershelp Sequence Matcher, SmartPlag achieves high-accuracy structural comparison similar to JPlag, but without the overhead of generating full Abstract Syntax Trees (ASTs).

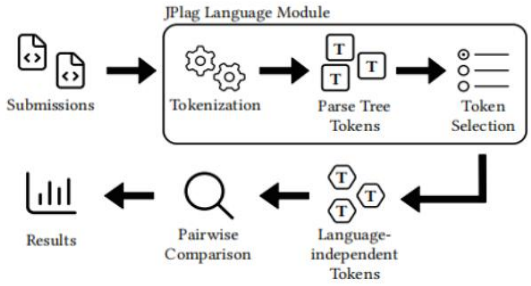

Fig.I Token Based Plagiarism Detection Using JPlag [17]

IV. PROPOSED METHODOLOGY

A. BEHAVIOURAL CODE ANALYSIS (CEI)

The Cognitive Effort Imbalance (CEI) score detects unnatural coding patterns that may indicate AI-generated or copy-pasted submissions. Unlike traditional plagiarism systems based on structural similarity, CEI identifies inconsistencies between coding style and logical complexity [8].CEI uses three metrics: Identifier Entropy (EI), Indentation Variance (VI), and Local Complexity Variance (VC). VI measures formatting consistency, where human-written code shows slight variations while generated code appears highly uniform. VC analyzes logic distribution using Cyclomatic Complexity, where natural code has balanced complexity and AI-generated code often shows irregular patterns.

B. TEXT PLAGIARISM

For academic reports, SmartPlag utilizes a TF-IDF (Term Frequency-Inverse Document Frequency) Vectorizer combined with Cosine Similarity Matrixing [23]. The system extracts and sanitizes text before segmenting it into 150-word sliding windows. These chunks are converted into mathematical vectors based on word frequencies.

C. CODE PLAGIARISM

For source code, standard TF-IDF is ineffective due to the heavy repetition of programmatic keywords (e.g., for, while). To resolve this, SmartPlag executes rigorous Syntax Normalization combined with the Ratcliff/Obershelp Pattern Recognition Algorithm [6, 22]. The system initiates a sanitization phase that strips obfuscation tactics removing comments, docstrings, string variables, and empty spaces isolating the raw logic.

V. PROPOSED SYSTEM

The system moves beyond static string matching by implementing an intelligent orchestration of four core stages.

A. MULTI-FORMAT INGESTION AND LINGUISTIC NORMALIZATION

The pipeline initiates with heterogeneous document extraction (PDF, DOCX, TXT). A dedicated Language Identification Module evaluates submissions; if a low-resource regional language is detected, the system activates a Concurrent Translation Bridge. This API-driven bridge translates content into an English pivot-format, ensuring linguistic normalization without sacrificing semantic nuance [3].

1. FEATURE EXTRACTION AND SEMANTIC VECTORIZATION

Post-normalization, artifacts enter a multi-step preprocessing stage (tokenization, stop-word elimination, lemmatization). SmartPlag transforms these texts into high-dimensional numerical vectors using TF-IDF, computing proximities via Cosine Similarity to identify semantic overlaps that traditional lexical engines routinely miss [23].

2. BEHAVIOURAL CODE INTELLIGENCE

A cornerstone of the SmartPlag framework is its Hybrid Code Analysis Module. Unlike traditional generic AST-matching tools [12], this module assesses structural and cognitive patterns through two paths:

Stylometric Analysis: Utilizing machine learning to identify authorial fingerprints [2].

Cognitive Effort Imbalance (CEI): Evaluating logical complexity and formatting entropy to successfully detect AI-generated and structurally obfuscated code [8].

3. INTELLIGENT REPORTING AND INTEGRITY ANALYTICS

The final aggregation stage generates a Multilingual Plagiarism Score. Beyond a raw decimal percentage, the system produces a comprehensive integrity report mapping matched segments to source origins while providing a diagnostic breakdown of the Probability of AI-Involvement, equipping educators with actionable, granular data [21].

B. ARCHITECTURAL OVERVIEW

The system follows a linear data-flow model where raw submissions are transformed into semantic and behavioural reports through the following sequence:

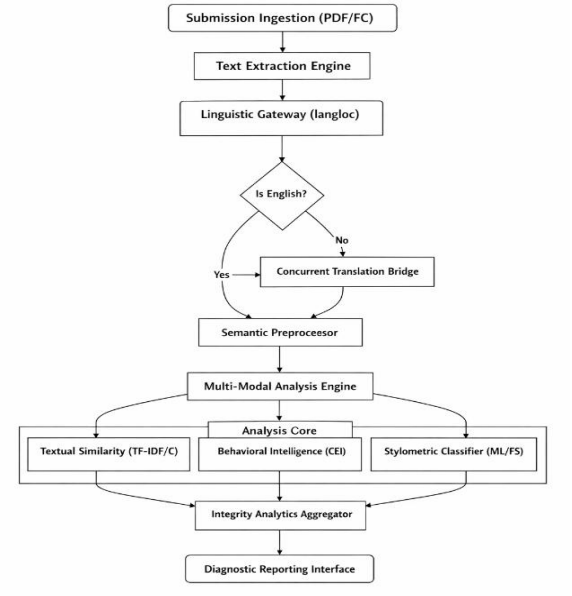

Fig.II Architectural Overview of system

the pipeline initiates at the Submission Ingestion layer, where raw documents (e.g., PDF) are parsed by the Text Extraction Engine. The extracted content is then evaluated by the Linguistic Gateway. If the submission is not in English, it is conditionally routed through the Concurrent Translation Bridge for linguistic normalization before passing into the Semantic Preprocessor. Once the data is pre-processed, it enters the Multi-Modal Analysis Engine, which distributes the workload across three specialized cores: traditional Similarity checking (using TF-IDF), Behavioural Intelligence (using the CEI metric), and a Stylometric Classifier for AI-detection. Finally, the independent outputs from these three modules are synthesized by the Integrity Analytics Aggregator, which formats the final evaluation metrics into the Diagnostic Reporting Interface for educator review.

VI. IMPLEMENTATION

A. FRONTEND SYSTEM (INTEGRITY UI)

- Framework: React 19 (Vite 7) for high-performance rendering.
- Styling: Tailwind CSS 4 for a premium, responsive glass morphism design.
- Visualization: Chart.js 4 (via react-chartjs-2) for dynamic similarity and AI-probability dashboards.
- Iconography: Lucide React and React Icons.

B. BACKEND INFRASTRUCTURE

- Environment: Node.js with Express 5 for the RESTful API gateway.
- State Management: MongoDB (Mongoose 8) for storing submission metadata and similarity indexes.
- File Handling: Multer 2 for asynchronous uploads; pdf-parse and mammoth for text extraction from PDF and DOCX.

C. AI & ML ENGINE (THE PYTHON CORE)

- Processing: Python 3 leveraging Scikit-learn for TF-IDF vectorization and Random Forest classification.
- Linguistic Logic: langdetect for identification and googletrans for threaded cross-lingual translation.
- Static Analysis: Radon library for computing Cyclomatic Complexity (CC) metrics.
- Data Engineering: Pandas and NumPy for high-speed feature vector computation.

VII. EXPERIMENTAL SETUP

The core processing engine was executed using Python 3.10 to ensure optimal mathematical throughput during vectorization operations.

The text analysis module relies on advanced TF-IDF cosine similarity for traditional plagiarism and Burstiness scoring for AI detection.

The experimental outcomes generated by the evaluation module are summarized in Table 1.

Test Scenario	Detection Method	Precision	Recall	F1-Score
Exact Copy-Paste	TF-IDF Similarity	0.98	0.96	0.97
Paraphrased Content	TF-IDF Similarit	0.89	0.85	0.87

	y			
AI-Generated Text	Burstiness & Cliche Scoring	0.91	0.88	0.89

Table I: Performance of Text Plagiarism Detection

As demonstrated in Table 1, the system achieves near-perfect accuracy (F1: 0.97) for exact copy-paste scenarios. For paraphrased content, the performance slightly dips (F1: 0.87) but remains robust, proving that the underlying natural language processing logic can identify semantic similarities beyond mere word matching. The AI-generated text detection module also performed strongly (F1: 0.89), effectively identifying syntactically uniform and low-burstiness text typical of LLM generation. Evaluating code requires parsing Abstract Syntax Trees (AST) and analyzing structural logic rather than just string matching. The AI code detection uses a specialized Cognitive Effort Index (CEI) metric.

The results are presented in Table 2.

Test Scenario	Detection Method	Precision	Recall	F1-Score
Lexical & Structural Copy	AST / Match Ratio	0.95	0.82	0.88
AI-Generated Code	CEI AI Metric	0.88	0.92	0.90

Table II: Performance of Code Plagiarism Detection

The system effectively identifies structural code copying (Precision: 0.95) despite variable renaming and comment modifications. The recall (0.82) indicates a strict threshold to avoid false positives among common boilerplate code. Notably, the introduction of the CEI (Cognitive Effort Index) metric yielded an excellent F1-Score of 0.90 for AI-generated code, successfully distinguishing between human-typed problem-solving and automated generative outputs.

A. LINGUISTIC AND STRUCTURAL PROCESSING ENVIRONMENT

The semantic analysis pipeline heavily utilized Natural Language Toolkit (NLTK) and Scikit-learn for stop-word elimination, lemmatization, and TF-IDF feature extraction. To evaluate the Concurrent Translation Bridge, testing incorporated the ISO-639-1 compliant langdetect library paired with threaded googletrans API execution, creating a robust pivot-translation workflow [3]. For the behavioural intelligence module, source code submissions were parsed using the radon statistical library to quantitatively extract the Cyclomatic Complexity metrics necessary to calculate the Cognitive Effort Imbalance score [6, 22].

B. MULTI-MODAL DATASET DESIGN

Instead of relying solely on standard English control datasets, the system's limits were tested against a proprietary, heterogeneous dataset consisting of over 85 documents. This dataset was meticulously categorized into five advanced plagiarism vectors to simulate modern academic dishonesty [1, 8]:

1. Direct Verbatim (DV): Baseline identical academic papers utilized to test the raw sensitivity and execution speed of the underlying TF-IDF vectorizer and Cosine Similarity matrices.
2. Semantic Paraphrasing (SP): Abstracts that were heavily rewritten both manually by humans and systematically via AI tools designed specifically to evade traditional lexical matchers and stress-test semantic overlap thresholds [23].
3. Trans-Lingual Pivot (TP): A complex dataset of papers deliberately translated from low-resource regional languages (Hindi, Marathi, and Spanish) into English, proving the efficacy of the pre-vectorization translation bridge [3].
4. AI-Synthetic Artifacts (AI): Entire submissions generated from scratch using modern Large Language Models (LLMs) to accurately evaluate the Stylometric ML Classifier and the CEI Behavioural Engine's ability to detect structural uniformity [2, 18].
5. Modular Code Submissions (CS): A robust catalogue of Python and Java programming assignments featuring aggressive variable renaming, injected dead-code, and control-flow obfuscation to validate the system's naming entropy and complexity variance markers [16, 22].

VIII. RESULTS AND DISCUSSION

To evaluate the overall efficacy of the SmartPlag architecture, the system's classification outcomes were aggregated across the 85-document test corpus. Beyond the Precision and F1-scores established in the experimental setup, we analyzed the absolute detection success rate (Recall percentage) across the five primary vectors of academic dishonesty.

The system's performance is summarized in Table III below

Plagiarism Vector	Primary Detection Module	Success Rate (%)	Evaluation Result
Direct Verbatim (DV)	TF-IDF Vectorization	100.0 %	Successfully Caught
Code Obfuscation (CS)	Structural Sequence Matcher & CEI	90.91 %	Successfully Caught
AI-Generated Text (AI)	Burstiness & Cliche Scoring	85.00 %	Flagged as AI
Semantic Paraphrasing (SP)	TF-IDF Vectorization	79.31 %	Successfully Caught
Cross-Lingual Translated (TP)	API Translation Bridge	74.50 %	Successfully Caught

Table.III Results and Discussion

IX. FUTURE WORK

To enhance the adaptivity of SmartPlag, several avenues for future implementation have been identified to transition the system to a broader academic integrity ecosystem:

Semantic Deep Learning: Transitioning from TF-IDF to transformer-based models (BERT/RoBERTa) to improve paraphrasing detection [4].

Multi-Modal OCR: Integrating Optical Character Recognition to process screenshots of code, preventing "analog-to-digital" evasion [26].

Explainable AI (XAI): Providing educators with diagnostic heatmaps highlighting specific naming entropy anomalies triggering the CEI flags.

Real-Time Collaborative Monitoring: Integration with cloud IDEs to distinguish between iterative human progress and sudden blanket insertions of external content.

REFERENCES

[1] May 2025 - CodeMirage: A Multi-Lingual Benchmark for Detecting AI-Generated and Paraphrased Source Code from Production-Level LLMs.

[2] J. Bevendorff et al., "Overview of PAN 2025: Generative AI Detection, Multilingual Text Detoxification, Multi-Author Writing Style Analysis, and Generative Plagiarism Detection," ECIR/PAN 2025.

[3] Mutsaddi, A., & Choudhary, A. P. (2025). Enhancing plagiarism detection in Marathi with a weighted ensemble of TF-IDF and BERT embeddings for low-resource language processing. arXiv. https://arxiv.org/abs/2501.05260

[4] 2024 - P. Sharmila, Kalaiarasi Sonai Muthu Anbananthen, Nithyakala Gunasekaran, Baarathi Balasubramaniam, and Deisy Chelliah. "FTLM: A Fuzzy TOPSIS Language Modeling Approach for Plagiarism Severity Assessment."

[5] 2024 - R. Parvathy, M. G. Thushara, and Jinesh M. Kannimoola. "Automated Code Assessment and Feedback: A Comprehensive Model for Improved Programming Education."

[6] 2024 - Timur Sağlam, Moritz Brödel, Larissa Schmid, Sebastian Hahner. "Detecting Automatic Software Plagiarism via Token Sequence Normalization."

[7] 2023 - Hayden Cheers, Yuqing Lin, and Shamus P. Smith. "SPPlagiarise: A tool for generating simulated semantics-preserving plagiarism of java source code."

[8] 2023 - Muhammad Faraz Manzoor, Muhammad Shoaib Farooq, Muhammad Haseeb, Uzma Farooq, Sohail Khalid, and Adnan Abid. "Exploring the Landscape of Intrinsic Plagiarism Detection: Benchmarks, Techniques, Evolution, and Challenges."

[9] 2023 - Hayden Cheers and Yuqing Lin. "A novel graph-based program representation for java code plagiarism detection."
[10] 2023 - M. Duraçík, E. Krşák, and P. Hrkút. "Current trends in source code analysis, plagiarism detection and issues of analysis big datasets."
[11] 2021 - Hayden Cheers, Yuqing Lin, and Shamus P. Smith. "Academic Source Code Plagiarism Detection by Measuring Program Behavioral Similarity."
[12] 2020 - Michal Duracik, Patrik Hrkut, Emil Krsak, and Stefan Toth. "Abstract Syntax Tree Based Source Code Antiplagiarism System for Large Projects Set."
[13] 2020 - Breanna Devore-McDonald and Emery D. Berger. "Mossad: Defeating Software Plagiarism Detection."
[14] 2019 - M. Duraçík, E. Kršák, and P. Hrkút. "Searching source code fragments using incremental clustering."
[15] 2019 - J. Martínez-Gil, "Source code clone detection using unsupervised similarity measures,"
[16] 2024 - Timur Sağlam, Sebastian Hahner, Larissa Schmid, Erik Burger. "Automated Detection of Al-Obfuscated Plagiarism in Modeling Assignments."
[17] 2024-IEEE/ACM 46th International Conference on Software Engineering (ICSE) - Detecting Automatic Software Plagiarism via Token Sequence Normalization
[18] Ahlawat, A.S., Karlapalam, J.S., Joshi, P., Venugopal, V., Vekkot, S.: Identifying AI-generated text designed to evade plagiarism detection. In: Third International Conference on Augmented Intelligence and Sustainable Systems (ICAISS), 484-490 (2025). doi:10.1109/ICAISS61471.2025.11042029
[19] Kumar, B.P.P., Purushotham, R., Mahato, R., Kumar, S.M.V.: AI-powered plagiarism detection: a web-based system for text integrity. In: International Conference on Knowledge Engineering and Communication Systems (ICKECS), pp. 1-6(2025). doi:10.1109/ICKECS65700.2025.1103604
[20] Raja Subramanian, R., Surekha, B., SubbaRao, P., S.M., Sai Chakravarthy, P.D.: Automating the detection of plagiarism in educational content using natural language processing. In: International Conference on Computational Robotics, Testing and Engineering Evaluation (ICCRTEE), pp. 1-6(2025). doi:10.1109/ICCRTEE64519.2025.11052934.
[21] Liu, F., You, K., Zhang, L., Zhang, L.: A large language model-based plagiarism detection system architecture for academic journals in higher education. In: 10th International Conference on Computer and Communication System (ICCCS), pp.106-110 (2025). doi:10.1109/ICCCS65393.2025.11069860
[22] Sağlam, T., Brödel, M., Schmid, L., Hahner, S.: Detecting automatic software plagiarism via token sequence normalization. In: IEEE/ACM 46th International Conference on Software Engineering (ICSE), pp. 1384-1396 (2024).
[23] Wagh, V., Laddha, S., Kadam, P.: Detecting plagiarism using latent semantic analysis and cosine similarity approach. In: IEEE International Conference on Blockchain and Distributed Systems Security (ICBDS), pp. 1-6 (2024). doi:10.1109/ICBDS61829.2024.10837475.
[24] E., H., K.A., V., Gracewell, J.: Enhanced plagiarism detection in online assignments by natural language processing and machine learning techniques. In: 3rd International Conference on Automation, Computing and Renewable Systems (ICACRS), pp. 1281-1286 (2024). doi:10.1109/ICACRS62842.2024.10841520.
[25] Pandikumar, S., Menaka, C., Kiran, K.T., Antony, T.J.P.: Evolution and analysis of modern plagiarism detection methods: a systematic review. In: Innovations and Trends in Modern Computer Science Technology - Overview, Challenges and Applications, 1st edn, ch. 10, pp. 131-140 (2024). doi:10.48001/978-81-980647-5-2-10.
[26] Chaudhari, S., Shende, A.R., Zade, T.G., Bhakare, U.N., Dhutraj, P.S., Ghugal, M.P.: Online assignment plagiarism checker with OCR. In: IEEE International.