Team jr at Generative Plagiarism Detection 2025: A Two-Stage Approach: From TF-IDF/Jaccard Filtering to Transformer Classification

Notebook for the PAN lab at CLEF 2025

Jieren Luo[1], Mancheng Huang[2], Biao Liu[1] and Zhongyuan Han*[1]

[1]Foshan University, Foshan, China
[2]China United Network Communications Group Co., Ltd. Foshan Branch

Abstract

Detecting plagiarism in text generated by large language models is challenging due to the high fluency and variability of AI-authored passages. In the first stage, we apply two lightweight filters in sequence. We compute the TF-IDF cosine similarity and then use character 3-gram Jaccard similarity. This quickly removes sentence pairs that are unlikely to contain plagiarism. In the second stage, remaining candidates are batched and fed through a fine-tuned BERT base classifier. Evaluated on the PAN 2025 validation set, our system achieves a micro_F1 of 0.2158 (micro_recall 0.1767, micro_precision 0.5819) and a macro_F1 of 0.1890 (macro_recall 0.1503, macro_precision 0.5642).

Keywords

PAN 2025, Generative Plagiarism Detection, TF-IDF Filtering, Jaccard Similarity, BERT Classification

1. Introduction

With the rise of openly accessible content generation utilities built on large language models, the issue of inherent text appropriation has become a critical concern for various sectors[1]. The PAN 2025 "Generative Plagiarism Detection" task requires participants to process pairs of documents—one suspicious and one source file—and identify all contiguous maximal-length passages of reused text[2]. At inference time, systems output an XML file for each pair, which specifies the offsets and lengths in the suspicious document and the source document.

Classic filters such as TF-IDF or Jaccard similarity are very fast but often miss paraphrased or obfuscated passages. Deep models like BERT deliver higher accuracy but incur high computational cost, making them impractical for large-scale validation. To balance efficiency and accuracy, we propose a two-stage hybrid pipeline. First, we apply TF-IDF cosine similarity followed by character 3-gram Jaccard filtering to quickly discard unlikely sentence pairs. Second, we batch the remaining candidates and apply a fine-tuned BERT classifier for the final decision. Recent studies have shown that NLP-based deep models like BERT improve semantic-level detection of complex plagiarism types[3]. Experiments on the PAN 2025 validation set show that our approach improves overall F1 performance without sacrificing speed[4].

2. Method

To achieve both high throughput and strong detection quality, we split our system into two stages. Stage One applies lightweight IR filters (TF–IDF cosine similarity followed by character 3-gram Jaccard)

CLEF 2025 Working Notes, 9 – 12 September 2025, Madrid, Spain
*Corresponding author.
wolike666@gmail.com (J. Luo); huangmc3@chinaunicom.cn (M. Huang); hyticen@gmail.com (B. Liu); hanzhongyuan@gmail.com (Z. Han*)
0009-0007-4867-4214 (J. Luo); 0009-0007-0130-4937 (M. Huang); 0009-0000-3031-9758 (B. Liu); 0000-0001-8960-9872 (Z. Han*)

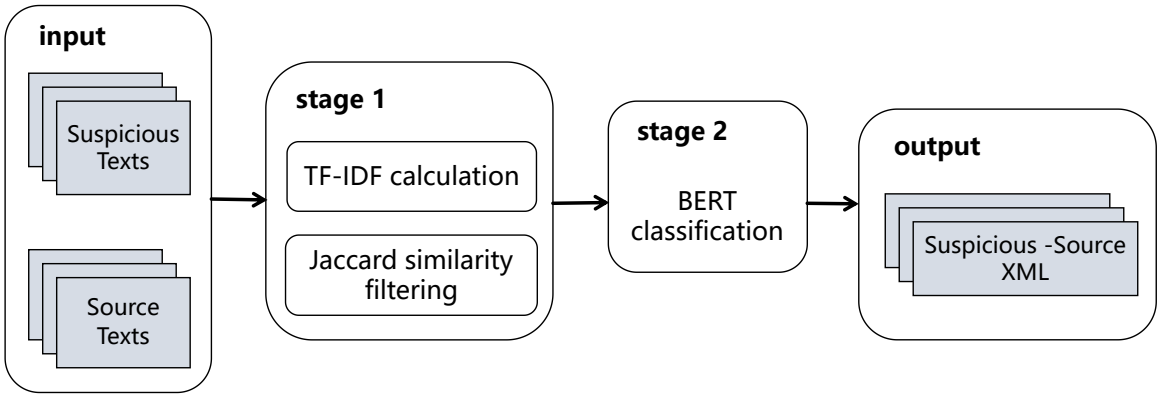

Figure 1: Two-Stage Generative Plagiarism Detection Pipeline

to prune most sentence pairs, and Stage Two runs a fine-tuned BERT classifier only on the remaining high-confidence candidates.

2.1. Stage One: IR-Based Filtering

In the first stage, we leverage two lightweight information-retrieval techniques to quickly eliminate the vast majority of sentence pairs that are unlikely to contain reused text. We begin by fitting a single TF-IDF vectorizer over all sentences from both suspicious and source documents; this global TF-IDF model captures the overall distribution of terms in the entire corpus. At inference time, each candidate sentence pair is transformed into sparse TF-IDF vectors and compared via cosine similarity [5], a technique also demonstrated to be effective in detecting both direct copying and online plagiarism when paired with semantic models. Pairs whose similarity falls below our threshold are discarded immediately, since they share too little contextual content to plausibly be paraphrased or reused. The surviving sentence pairs then undergo a second round of filtering based on character 3-gram Jaccard similarity: we convert each sentence into a set of character 3-gram shingles and compute the ratio of their intersection to their union. If this Jaccard score is below the appropriate obfuscation-specific threshold, we again discard the pair. By cascading TF-IDF (to remove grossly dissimilar pairs) and character 3-gram Jaccard (to weed out noisy overlaps), we reduce the candidate pool by several orders of magnitude. This ensures that only a small, high-likelihood subset progresses to the more expensive Transformer-based classification in the next stage.

2.2. Stage Two: Transformer-Based Classification

In the second stage, all sentence pairs that survived IR-based filtering are prepared for Transformer-based classification. First, we organize the remaining pairs into chunks of a fixed size (e.g., 100 pairs per chunk) to balance GPU memory usage and throughput. Each chunk is then split into batches (e.g., 64 pairs per batch), and every sentence pair within a batch is tokenized and padded to a uniform length before being moved onto the GPU. We utilize a Hugging Face text-classification pipeline backed by a BERT-base model fine-tuned on our prepared training data [6]. During inference, the pipeline receives each batch of tokenized sentence pairs and outputs a softmax confidence score for “plagiarism” versus “non-plagiarism”. We compare each pair’s plagiarism confidence to its obfuscation-specific threshold (higher for medium/hard, slightly lower for simple). Only those pairs whose BERT score exceeds the threshold are marked as positive detections. This approach ensures that BERT is applied only to the small subset of pairs most likely to contain reused text—maximizing classification accuracy while keeping GPU utilization efficient.

2.3. Output Generation

In the final stage, we generate structured output files in the PAN XML format to report all detected cases of plagiarism. For each file pair processed in a chunk, the system collects all positively classified sentence pairs, merges overlapping or adjacent segments using a configurable merge_gap, and writes the resulting intervals to an XML document. Each plagiarism instance is encoded as a <feature name="detected-plagiarism" ... /> element, specifying both the suspicious and source text offsets and lengths. This step ensures that the output aligns with PAN's evaluation format and can be directly used for scoring. Our implementation automates this process within each chunk, leveraging batch BERT results and filtering logic to generate accurate and concise detection records, efficiently saving them to disk.

3. Experiment

3.1. Experimental Setup

1. **Preprocessing:** We first read the pairs file to identify every suspicious–source document pair. For each listed pair, both the suspicious and the source documents are loaded from their respective folders, and their raw text is normalized (e.g., ensuring consistent UTF-8 encoding). We then apply NLTK's sentence tokenizer to split each document into a sequence of sentences [7]. Once all sentences are extracted, we form the initial candidate pool by pairing each sentence in the suspicious document with each sentence in the source document. This complete cross-product of sentences becomes the input to our first filtering stage, ensuring that potential plagiarism fragment (regardless of its position) is considered.
2. **Data:** The PAN 2025 organizers provide three separate datasets—spot_check, train, and validation—each following the same subfolder layout: a `src/` folder, a `susp/` folder, a pairs file listing suspicious-source pairs, and a corresponding `_truth` folder containing XML annotations[1]. The train and validation datasets only differ in scale: train contains roughly 2.4 million pairs (used for model fine-tuning), validation contains approximately 238 k pairs (held out for final scoring), and spot_check is a small sample meant for rapid local testing. During evaluation, our system must produce an XML file named `<suspicious>-<source>.xml` for each pair in the pairs list; each output XML includes `<feature name="detected-plagiarism" . . . >` elements that are compared against the ground-truth `<feature name="plagiarism" . . . >` entries in the corresponding `_truth` folder.
3. **Parameters:** We extract positive and negative sentence pairs using a helper script that reads each `_truth` XML, splits the specified spans into sentences (positive examples), and then randomly samples non-overlapping sentences as negatives at a fixed ratio (approximately 0.43 negatives per positive). To limit computational overhead and simulate a low-resource setting, we randomly selected 10,000 sentence pairs from the full training set for BERT fine-tuning, drawing positives and negatives in a 7:3 ratio (i.e., 7,000 positive pairs and 3,000 negative pairs). We fine-tuned BERT with a learning rate of 2×10^{-5}, weight decay of 0.01, for 3 epochs, using a per-device train batch size of 4 (effective batch size 16) and mixed-precision. For filtering, we compute TF-IDF cosine similarity (keeping pairs above 0.7) and then apply character 3-gram Jaccard (removing pairs below 0.4 for "simple" versus 0.7 for other obfuscations). Surviving pairs are grouped into chunks of 100 and processed in batches of 64 by a fine-tuned BERT model, using confidence cutoffs of 0.75 for "simple" and 0.8 for "medium/hard."
4. **Baseline:** The PAN reference implementation uses a simple character-level n-gram approach to detect near-copy plagiarism[2]. First, it preprocesses each suspicious document by removing all punctuation and whitespace and then sliding a fixed-length (50-character) window over the text to build a hash table of every 50-character n-gram and its positions. Next, for each source document, it similarly slides the same 50-character window and looks up whether that n-gram

[1] https://zenodo.org/records/14969012
[2] https://github.com/pan-webis-de/pan-code/tree/master/clef25/generated-plagiarism-detection

exists in the suspicious document's index. Whenever a match is found, the baseline attempts to extend the match forward (skipping over any punctuation/whitespace) to produce the longest possible contiguous aligned span. All detected spans are collected and written out as <feature name="detected-plagiarism" ... > entries in an XML file named <suspicious>-<source>.xml[8]. This "IR-only" method is fast and easy to implement but tends to miss paraphrased or heavily obfuscated passages.

5. **Evaluation Metrics:** We use the standard PAN toolkit to measure micro_recall and macro_recall and precision on character-level overlaps, along with granularity [9] (the average number of detected segments per true case). During evaluation, each ground-truth <feature name="plagiarism" ... > is compared against our <feature name="detected-plagiarism" ... > entries. Micro-averaged metrics count total overlapping characters across all documents, while macro-averaged metrics compute recall and precision separately for each case or detection before averaging. Granularity quantifies how many detected spans overlap each true plagiarism case—ideally one-to-one. These measures are computed by the official plagiarism-detection-evaluation.py script, which outputs micro_recall, micro_precision, macro_recall, macro_precision, and granularity for final scoring.

3.2. Results and Analysis

Table 1
Overall Performance

Method	micro_recall	micro_precision	micro_F1	macro_recall	macro_precision	macro_F1	granularity
Baseline	0.1062	**0.5697**	0.1081	0.0734	0.5096	0.0775	2.1511
Ours	**0.1767**	0.5819	**0.2158**	**0.1503**	**0.5642**	**0.1890**	**1.3885**

Table 1 reports results on the validation subset. Our approach raises the micro_F1 score from 0.1081 to 0.2158 and the macro_F1 score from 0.0775 to 0.1890. Granularity (average number of detected segments) decreases from 2.15 to 1.39, indicating fewer false positives.

Table 2
Ablation on IR filters

Configuration	micro_recall	micro_precision	micro_F1	macro_recall	macro_precision	macro_F1	granularity
TF-IDF only	0.1994	0.5076	0.2239	0.1701	0.4997	0.1984	**1.4272**
Jaccard only	0.1587	0.2359	0.1534	0.1390	0.2156	0.1367	1.3564
TF-IDF and Jaccard	**0.2011**	**0.5423**	**0.2315**	**0.1717**	**0.5115**	**0.2029**	1.4067

To demonstrate the complementary effect of our two-stage IR filtering, we compare three configurations on the spot_check subset:

- **TF-IDF only**: apply only the TF–IDF cosine-similarity threshold (threshold_tfidf=0.7), no Jaccard.
- **Jaccard only**: skip TF–IDF (threshold_tfidf=0), apply only 3-gram Jaccard.
- **TF-IDF and Jaccard**: the default pipeline combining both filters.

As Table 2 demonstrates that adding the 3-gram Jaccard filter to TF–IDF improves overall F1 performance compared to using TF–IDF alone. Although both filters rely on token overlap, Jaccard's emphasis on exact contiguous 3-gram matches helps eliminate noisy or boilerplate pairs that TF–IDF's cosine similarity still admits. This complementary effect validates the benefit of cascading the two filters before invoking the Transformer classifier.

Table 3 reports per-obfuscation performance on the validation subset. As obfuscation strength increases from "simple" to "hard", we observe a clear and steady decline in both recall and F1, with precision also dropping across the board[3]. For "simple" cases, our pipeline achieves its best performance

[3]https://github.com/wolike666/evaluate_by_obfuscation-for-PAN25

Table 3
Our Method for Per-Obfuscation Analysis

Obfuscation	micro_recall	micro_precision	micro_F1	macro_recall	macro_precision	macro_F1	granularity
simple	**0.3947**	**0.5728**	**0.4674**	**0.3063**	**0.4912**	**0.3773**	1.4097
medium	0.2622	0.2420	0.2517	0.1780	0.1931	0.1853	**1.4496**
hard	0.2144	0.0549	0.0874	0.1274	0.0436	0.0650	1.4333

(micro_F1=0.4674), whereas medium and hard obfuscations exhibit lower micro_F1 scores of 0.2517 and 0.0874, respectively. The decreasing precision—from 0.5728 in the "simple" category to just 0.0549 for "hard"—indicates that while the model is generally precise when it does predict plagiarism, it increasingly fails to detect heavily obfuscated passages. Granularity remains close to 1.43 in all categories, showing that detections tend to form single, contiguous spans rather than fragmented segments. These findings confirm that our two-stage IR filtering plus BERT classification excels at catching lightly obfuscated reuse but that further enhancements—such as more targeted filtering thresholds or ensemble modeling—are needed to boost recall on the most challenging cases.

4. Conclusions

We presented a two-stage pipeline for generative plagiarism detection that combines fast IR-based filtering (TF-IDF and Jaccard) with a fine-tuned BERT classifier. By first pruning the vast majority of sentence pairs with simple filters and then applying BERT only to the remaining candidates, our method achieves a strong balance between throughput and accuracy. On the PAN 2025 validation set, this hybrid approach more than doubles the micro_F1 compared to the IR-only baseline while reducing false positives and keeping fragmentation low. Although detection of heavily obfuscated passages remains challenging, our framework provides a solid foundation for further improvements—such as enhanced filtering or ensemble models—to boost recall on the hardest cases. To advance plagiarism detection systems, future studies should prioritize the refinement of cross-lingual detection via knowledge graphs and multilingual embeddings [10]. Meanwhile, hybrid architectures fusing conventional rule-based methods with AI-driven algorithms could present a scalable pathway to enhance detection efficiency and adaptability.

Acknowledgments

This work is supported by the National Social Science Foundation of China (24BYY080).

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT (OpenAI) and DeepSeek for grammar and spelling checks, paraphrasing and rewording, and for translation assistance. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] M. Sajid, M. Sanaullah, M. Fuzail, T. S. Malik, S. M. Shuhidan, Comparative analysis of text-based plagiarism detection techniques, PLOS ONE 20 (2025) e0319551. doi:10.1371/journal.pone.0319551.

[2] J. Bevendorff, D. Dementieva, M. Fröbe, B. Gipp, A. Greiner-Petter, J. Karlgren, M. Mayerl, P. Nakov, A. Panchenko, M. Potthast, A. Shelmanov, E. Stamatatos, B. Stein, Y. Wang, M. Wiegmann,

E. Zangerle, Overview of PAN 2025: Voight-Kampff Generative AI Detection, Multilingual Text Detoxification, Multi-Author Writing Style Analysis, and Generative Plagiarism Detection, in: J. C. de Albornoz, J. Gonzalo, L. Plaza, A. G. S. de Herrera, J. Mothe, F. Piroi, P. Rosso, D. Spina, G. Faggioli, N. Ferro (Eds.), Experimental IR Meets Multilinguality, Multimodality, and Interaction. Proceedings of the Sixteenth International Conference of the CLEF Association (CLEF 2025), Lecture Notes in Computer Science, Springer, Berlin Heidelberg New York, 2025.

[3] A. Amirzhanov, C. Turan, A. Makhmutova, Plagiarism types and detection methods: a systematic survey of algorithms in text analysis, Frontiers in Computer Science 7 (2025). URL: https://www.frontiersin.org/articles/10.3389/fcomp.2025.1504725/full. doi:10.3389/fcomp.2025.1504725.

[4] M. Fröbe, M. Wiegmann, N. Kolyada, B. Grahm, T. Elstner, F. Loebe, M. Hagen, B. Stein, M. Potthast, Continuous Integration for Reproducible Shared Tasks with TIRA.io, in: Advances in Information Retrieval. 45th European Conference on IR Research (ECIR 2023), Lecture Notes in Computer Science, Springer, Berlin Heidelberg New York, 2023, pp. 236–241.

[5] D. M. Setu, T. Islam, M. Erfan, S. K. Dey, M. R. Al Asif, M. Samsuddoha, A comprehensive strategy for identifying plagiarism in academic submissions, Journal of Umm Al-Qura University for Engineering and Architecture 2 (2025) 310–325. doi:10.1007/s43995-025-00108-1.

[6] M. Khadhraoui, H. Bellaaj, M. B. Ammar, H. Hamam, M. Jmaiel, Survey of bert-base models for scientific text classification: Covid-19 case study, Applied Sciences 12 (2022) 2891. URL: https://www.mdpi.com/2076-3417/12/6/2891. doi:10.3390/app12062891.

[7] M. Wang, F. Hu, The application of nltk library for python natural language processing in corpus research, Theory and Practice in Language Studies 11 (2021) 1041–1049. doi:10.17507/tpls.1109.09.

[8] A. Greiner-Petter, M. Fröbe, J. P. Wahle, T. Ruas, B. Gipp, A. Aizawa, M. Potthast, Overview of the Generative Plagiarism Detection Task at PAN 2025, in: G. Faggioli, N. Ferro, P. Rosso, D. Spina (Eds.), Working Notes of CLEF 2025 – Conference and Labs of the Evaluation Forum, CEUR Workshop Proceedings, CEUR-WS.org, 2025.

[9] M. Potthast, M. Hagen, A. Beyer, M. Busse, M. Tippmann, P. Rosso, B. Stein, Overview of the 6th international competition on plagiarism detection, in: Working Notes for CLEF 2014, 2014, pp. 845–872.

[10] A. Amirzhanov, C. Turan, A. Makhmutova, Plagiarism types and detection methods: a systematic survey of algorithms in text analysis, Frontiers in Computer Science 7 (2025). URL: https://www.frontiersin.org/articles/10.3389/fcomp.2025.1504725. doi:10.3389/fcomp.2025.1504725.