ChatGPT

Building a Local AI-Generated Content and Plagiarism Detection System

Detecting AI-written text and plagiarism in student papers requires a multi-faceted approach. Our proposed system splits each document into sentences, then flags each sentence as **AI-generated**, **plagiarized**, or **original**, explaining each flag with model outputs and recommended actions. We prioritize robustness (handling paraphrases or adversarial edits) and full offline operation. Key components include modern NLP models (BERT/RoBERTa, SentenceTransformers), semantic search, and ensemble techniques (e.g. contrastive learning, watermarking, zero-shot methods). Below we outline the design, citing relevant research and open-source tools.

AI-Generated Text Detection

Model-Based Detectors. Contemporary detectors use supervised classifiers or statistical cues to differentiate human vs. AI text. For example, Ghostbuster (Verma *et al.*, 2023) passes text through several weaker LMs (from unigram to GPT-3) to compute features, then trains a simple linear classifier. This yields very high accuracy (99% F1 on held-out essays/creative/news texts). Ghostbuster's code and datasets are publicly released, providing a strong baseline. In practice one can fine-tune a RoBERTa/DistilBERT model on labeled human vs. AI text (using resources like OpenAI's GPT-2 output dataset or community models). For instance, open HuggingFace models such as `openai/roberta-base-openai-detector` (for GPT-2) or community "ChatGPT-content-detector" checkpoints can be used or further fine-tuned.

Zero-Shot and Statistical Methods. Other techniques require no training data. *DetectGPT* (Mitchell *et al.*, 2023) is a zero-shot method: it generates small random perturbations of a sentence and checks if the original has unusually low log-probability under a candidate LLM. AI-generated text tends to lie in regions of high "negative curvature" of probability space. If an open-source LLM (e.g. GPT-Neo/GPT-J) is available locally, one can implement a DetectGPT-like check by sampling perturbations and comparing log-likelihoods. Similarly, classic tools like GLTR (Gehrmann *et al.*, 2019) use token-probability statistics to hint at AI generation (though GLTR is more for human inspection).

Advanced Learning Techniques. Recent work shows that contrastive pretraining can improve detection robustness. For example, the SENTRA model (Tan *et al.*, 2022) leverages contrastive learning on large unlabeled corpora to distinguish AI vs. human text. Bhattacharjee *et al.* (2023) further apply *contrastive domain adaptation* for unsupervised detection. In practice, one could augment the classifier's training with examples of paraphrased or adversarial AI text (using ChatGPT or GPT-3.5 to generate difficult negatives). *Watermarking* is another route: modern LLMs (e.g. Google DeepMind's SynthID-Text, 2024) can embed hidden signals during generation. If a known watermark scheme was used, the detector can check for that signal efficiently. (However, watermark detection only works if the LLM was configured with that scheme.)

Limitations and Ensemble. No single method is foolproof. As Sadasivan *et al.* (2023) note, detectors degrade under paraphrasing attacks. Thus our system will combine multiple signals. For each sentence we can run both a fine-tuned classifier and a zero-shot check; if either strongly indicates AI, we flag it. We should also watch for biases (e.g. non-native writing can trigger false positives). In summary, use an ensemble of models (RoBERTa/DistilBERT classifiers, logistic regression on features, DetectGPT-like checks) to maximize accuracy, reporting which model fired and why. For example:

- *Sentence flagged as AI:* "RoBERTa classifier score 0.96 indicates AI generation, and DetectGPT log-prob gap is high. Consider rewriting or citing."
- *Sentence flagged original:* "No detector signals above threshold; treat as original writing."

This sentence-level diagnostic lets instructors review suspicious lines individually.

Plagiarism Detection (Semantic Search)

We detect plagiarism by semantic similarity rather than relying on large proprietary databases. We encode each sentence (or small chunk) into a vector using a sentence embedding model (e.g. **SentenceTransformers** such as `all-mpnet-base-v2` or fine-tuned SBERT). These embeddings are stored in a local vector index (e.g. FAISS, Chroma or a database like PostgreSQL+pgvector [1]). The corpus we embed should include open academic content: for instance, Kaggle and arXiv host the full arXiv paper archive (≈1.7M articles). Wikipedia, CORE research papers, and open textbooks can supplement. We deliberately avoid closed tools like Turnitin; instead, we build our own "knowledge base" from open corpora (the arXiv Kaggle dataset and arXiv bulk S3 access are freely available).

At runtime, each input sentence is embedded and a cosine-similarity K-NN search is performed against the index. If a nearest neighbor's score exceeds a high threshold (e.g. cosine > 0.8), we mark the sentence as plagiarized and report the matching source snippet. We can also fine-tune a second classifier that takes the embedding of the input and its nearest neighbor to verify plagiarism (as in recent work by Wahle *et al.*, 2022). In fact, Wahle *et al.* provide pretrained HuggingFace models for paraphrase detection (e.g. `jpelhaw/roberta-base-pd` for paraphrase detection) which can classify whether two sentences convey the same content [2].

Sentence embeddings are robust to paraphrase: SBERT captures deep semantic relationships, so we can catch not only exact copy-paste but also heavy rewrites. For example, a university homework copy might be paraphrased by an LLM, but it will still appear near the original in embedding space. (By contrast, traditional string-matching detectors often fail on paraphrase.) This approach handles adversarial cases better: if a student runs a sentence through ChatGPT for rewriting, the embedding will still likely correlate with the source content, unlike simple n-gram overlap.

Figure: Example embedding-based plagiarism pipeline. Text is split into segments, encoded via a BERT/SBERT model, and indexed in a vector database. Queries compute embeddings and retrieve nearest neighbors for similarity. (Based on an AWS reference architecture.)

The figure above (inspired by AWS's plagiarism detection diagram) illustrates this pipeline. Text is extracted and chunked, then a BERT-based model produces embeddings. Those vectors are stored (e.g. in OpenSearch, FAISS, or pgvector). Each incoming sentence is likewise embedded and a KNN search retrieves similar known sentences. If the similarity is high, we flag potential plagiarism and cite the source. AWS's reference design highlights this: it uses Step Functions and Lambda to convert text to embeddings and then a cosine-based KNN search over a pre-indexed corpus.

Sentence-Level Reporting and Actions

After classifying each sentence as (AI, plagiarized, or original), we generate an actionable report. This report lists sentences in the submitted document and annotates each. For an AI-flagged sentence, the report notes **"Detected by [ModelName]"** and gives the confidence or probability. It might quote a snippet of evidence (e.g. key phrases typical of LLM output) or explain that the sentence had extremely low perplexity under human text models. For a plagiarized sentence, we include the title/ID of the

matched source and a similarity score, suggesting to **“add citation or rewrite”**. Original sentences are marked safe.

Next-step suggestions are crucial. We can template suggestions (“Cite this sentence” or “Rewrite for originality”) or even generate them via an LLM. For example, using LangChain we could prompt an offline Llama-2 or Vicuna model with the flagged sentence and ask for rewrite suggestions or citation formatting. In all cases, the report should be clear: **why** a sentence was flagged (e.g. “high semantic overlap with source X”) and **which model** triggered it.

Accuracy and Robustness

We emphasize accuracy over speed. To improve robustness, we use ensembling: multiple detectors vote. This guards against model-specific biases; for instance, Verma *et al.* note that Ghostbuster (trained on ChatGPT outputs) may not catch LLaMA-generated text, so we include several models. We also adversarially augment training data: generate paraphrased negatives using GPT-3.5 or open LLMs, and include known plagiarism obfuscations (as in Wahle *et al.*, 2022).

Empirical results show high performance for these methods: Ghostbuster’s ensemble approach is ~99% accurate, and SBERT-based plagiarism retrieval can achieve near-perfect recall on paraphrase datasets when tuned. However, we remain cautious: Sadasivan *et al.* (2023) warn that detectors have a fundamental limit and can be fooled by adaptive rewriting. Therefore the system flags uncertainties conservatively (high threshold) and always includes human review steps.

Implementation and Open-Source Tools

Our stack is fully open-source and offline. We use Python with HuggingFace’s Transformers and Sentence-Transformers libraries. For example, LangChain’s integration allows us to easily call any HF model for embeddings and LLM tasks. A pipeline might look like:

1. **Loader/Splitter**: Read the uploaded file (PDF or text), extract sentences (e.g. via SpaCy or regex), chunking if needed.
2. **AI-Detector**: Run each sentence through a pretrained classifier (e.g. a HuggingFace RoBERTa fine-tuned on AI-text detection) or apply DetectGPT logic if an open LLM is available.
3. **Plagiarism-Search**: Embed each sentence with SBERT and query the FAISS/Chroma vector index. If a nearest neighbor’s similarity > threshold, flag it (optionally double-check with a classifier like `jpelhaw/roberta-base-pd` [2]).
4. **Report Generator**: Collect flags and run a summarization or explanation LLM chain (using LangChain) to produce a human-readable report, mentioning which model flagged what and recommended actions.

We rely only on local models. For embeddings, LangChain’s `HuggingFaceEmbeddings` class can load e.g. `"all-MiniLM-L6-v2"` or custom SentenceTransformer models. We store vectors in an offline store: either FAISS indexes on disk or an open-source vector DB (Chroma, Milvus, or `pgvector` in Postgres [1]). As one example, an open-source student project uses Postgres+pgvector for semantic search [1], which is easily replicated. For the AI detector, any HuggingFace model (e.g. DistilBERT or RoBERTa) with sequence-classification head can run fully offline on GPU/CPU.

We have identified relevant resources for implementation: - **Ghostbuster** code by Verma *et al.* (GitHub: *vivek3141/ghostbuster*). - **HuggingFace models** for AI detection and paraphrase/plagiarism (e.g. PirateXX’s detectors, Wahle’s paraphrase models [2], OpenAI’s RoBERTa detector). - **Sentence-**

Transformers library for embeddings (supported in LangChain). - **Vector stores**: FAISS or `pgvector` (as in [28]). - **LangChain** pipelines for orchestration (text splitting, embedding, retrieval, and report generation).

Finally, we organize the system with a hybrid architecture. For scalability, we can separate “real-time” and batch processing: the pipeline above can run on a queue (one document per job) with workers. As one reference architecture notes, a document database (for storing submissions) can be paired with a graph DB for analysis logs, using separate clusters for on-demand vs. offline jobs.

In summary, by combining **modern NLP models** (BERT/RoBERTa classifiers, sentence embeddings) with **semantic search** and **ensemble logic**, we can build a robust, offline tool that flags AI-written or plagiarized sentences and explains each flag. The detailed report will cite the offending sentence, the detection model, and actionable advice (e.g. “add citation” or “rewrite in your own words”), leaving no sentence unexamined. All components are open-source: for example, Ghostbuster’s pipeline and AWS’s embedding-based plagiarism design provide tested blueprints that we adapt into a local, self-contained system.

Sources: Recent detection research and tools, plagiarism-detection studies [2], AWS reference architecture, LangChain documentation, and open-source projects [1].

---

[1] GitHub - Kyle6012/plagiarism-detection: Plagiarism and AI-generated content detection system with text and image comparison,.
https://github.com/Kyle6012/plagiarism-detection

[2] GitHub - jpwahle/iconf22-paraphrase: The official implementation of the iConference 2022 paper "Identifying Machine-Paraphrased Plagiarism".
https://github.com/jpwahle/iconf22-paraphrase