On Improving the Efficiency of AI-Generated Text Detection

Bogdan Ichim[1,2] and Andrei-Cristian Năstase[1]
[1]*Faculty of Mathematics and Computer Science, University of Bucharest, Str. Academiei 14, Bucharest, Romania*
[2]*Simion Stoilow Institute of Mathematics of the Romanian Academy, Str. Calea Grivitei 21, Bucharest, Romania*

Keywords: AI-Generated Text Detection, AI-Generated Content, Large-Scale AI Detection, AI-Generated Text Classification, Dimensionality Reduction, PCA (Principal Component Analysis), TSVD (Truncated Singular Value Decomposition), FastICA (Fast Independent Component Analysis), BERT Embeddings, SimCSE Embeddings, TF-IDF (Term Frequency-Inverse Document Frequency).

Abstract: This paper proposes methods of making AI-Generated Text Detectors more computationally efficient without paying a high price in prediction accuracy. Most AI-Detectors use transformer-based architectures with high-dimensional text embedding vectors involved in the pipelines. Applying dimension reduction algorithms to these vectors is a simple idea for making the whole process more efficient. Our experimental results reveal that this may lead from 5 up to 500 times improvements in the training and inference times, with only marginal performance degradation. These findings suggest that integrating such methods in large-scale systems could be an excellent way to enhance the processing speed (and also reduce the electric energy consumption). In particular, real-time applications might benefit from such enhancements.

1 INTRODUCTION

In parallel with the recent advances in artificial intelligence, texts generated by computers have become increasingly indistinguishable from human writing (see Casal & Kessler, 2023; Jakesch et al., 2023). Even AI-generated reviews are challenging to distinguish (Ignat et al., 2024). Large Language Models (LLM's) can write extremely coherent (and sometimes relevant) texts, and that, to some, might signal serious future problems (Bender et al., 2021).

While AI-generated texts have many uses in fields such as marketing (generating content, thumbnails, video edits, and so on), or customer service, as with anything, LLM's are a double-edged sword – potential misuses of such tools include the possibility of impersonation, public opinion manipulation, and disinformation in general.

Therefore, it's becoming more and more important to be able to recognize AI-generated text in order to preserve confidence in digital information and communication. Robust and efficient detection systems are essential for ensuring this.

Methods for detecting AI-generated texts have been proposed by several authors as can be seen in Section **2** of this paper. All these techniques are computationally very expensive since the models take a long time to train and the sizes of the datasets can grow out of control. This may result in significant costs for the hardware, for the electric energy used and might even impact the environment. There is interest for making such computations more efficient (Patel et al., 2024).

By adding dimension reduction algorithms to the AI-detection pipelines we may improve training and inference times and at the same time significantly lower the computational costs. While high-dimensional text embeddings capture nuanced semantic features, the dimensionality reduction algorithms maintain the essential features with minimal information loss, while at the same time making models smaller, faster, and more efficient.

Therefore, the contributions in this paper are the following:

- We have prepared a new dataset, which may be useful also for other experiments;
- We are studying how dimensionality reduction techniques like Principal Component Analysis (PCA), Truncated Singular Value Decomposition (TSVD) and FastICA can be used to minimize the size of a host of different text embedding vectors generated by SBERT, SimCSE and the more traditional TF-IDF with a minimal reduction in prediction accuracy.

Ichim, B. and Năstase, A.-C.
On Improving the Efficiency of AI-Generated Text Detection.
DOI: 10.5220/0013433600003928
In *Proceedings of the 20th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2025)*, pages 731-738
ISBN: 978-989-758-742-9; ISSN: 2184-4895

The newly prepared dataset, together with the implementation used for experiments, is available on demand from the authors.

2 RELATED WORK

Researchers have recognized the valuable role of dimensionality reduction in the detection of AI-generated text. For example, as shown in (Rojas-Simón et al., 2024), feature reduction on ASCII-based lexical representations improves classification accuracy while reducing overheads. Our experiments complement this type of work. While embedding-based methods are capable of capturing more semantic information, character-based vectorization may serve as a viable and less computationally expensive alternative. Hybrid methods seem to be a very promising idea.

In (Singh et al., 2022) the authors proposed a semantic feature selection process based on GloVe word embeddings for eliminating redundant features. Their method is called redundant feature removal (rRF) and it provided a considerable improvement in classification accuracy, beyond reducing dimensionality. Besides that, they introduced a novel performance metric (NPM) for balancing between evaluation of feature-reduction effectiveness and classification accuracy, which could serve as a benchmark for improvements in efficiency in AI text detection.

PCA and Autoencoders are proven to be good dimensionality reduction techniques for increasing classification accuracy and computational efficiency of classic machine learning models (Khan et al., 2022). This approach showed that PCA-based preprocessing yielded significant improvements in training and inference times, while also showing that AutoEncoders (AE) could provide a viable alternative for unsupervised data compression. Indeed, their findings reveal that preprocessing is essential to speeding up model convergence, improving feature interpretability, and ensuring robustness on various text datasets. The application of AE's alongside techniques like PCA, TSVD or FastICA could achieve greater efficiency gains, particularly in real-time AI text classification tasks.

In order to identify the optimal number of features to use (Moulik et al., 2023) performed a systematic analysis of feature dimensionality reduction through PCA and decision-tree-based classifiers (AdaBoost). Their results show that reducing the feature space to 3-6 dimensions can greatly speed up training time without sacrificing classification performance - an insight directly applicable to AI-generated text detection. Also, the use of Kernel PCA (KPCA) with an RBF kernel gives a valuable alternative to using regular PCA, particularly in domains where modelling non-linear relationships is crucial – in our case, preserving semantic integrity is of utmost importance.

For the task of detecting AI-generated texts supervised learning is the main technique used by many authors. In a nutshell, various machine learning classifiers are trained on labelled datasets (comprising both human-written and AI-produced texts) and used for predicting the class of unlabelled data. For example, CamemBERT, CamemBERTa and XLM-R are used in (Antoun et al., 2023); ChatGPT was put up to this task in (Bhattacharjee & Liu, 2023); a logistic regression model and two deep classifiers based on RoBERTa are tested by (Guo et al., 2023); Bag-of-Words with a logistic regression classifier and a fine-tuned BERT model are used in (Ippolito et al., 2020). Building up in complexity, the RADAR framework is proposed by (Hu et al., 2023), then in a recent concurrent work the OUTFOX framework is introduced by (Koike et al., 2024). Also, human-assisted detection methods were proposed in (Dou et al., 2022).

Commonly, such classifiers use sentence embeddings for the role of (text-extracted) features. Those multi-dimensional vectors supposedly describe the text's semantic content and can easily be fed into the classifiers.

Fine-tuned transformers, as for example RoBERTa (Liu et al., 2019), turned out to be very successful at detection when trained on appropriate datasets. Another possible approach is using stylometric features for detecting AI-generated text – the authors use differences and inconsistencies in the writing style as markers of generated text (Kumarage et al., 2023).

Finally, we also note that two recent surveys on detecting of AI-generated texts are also available, that is (Jawahar et al., 2020) and (Tang et al., 2024).

3 METHODOLOGY

3.1 Problem Statement

Assume we are presented with a two-column dataset, the first column contains a "text" and the second column contains a "label" which is 0 or 1 (in our case 0 is the label for human-generated text and 1 is the label for AI-generated text). For the associated binary classification problem the problem is to find an *efficient* algorithm, that it we would like to trade a little from the classification performance

in order to obtain a significant reduction in the computational effort involved. In order to investigate this problem we experiment with three text embeddings algorithms, four machine learning binary classification algorithms, as well as with three dimensionality reduction algorithms.

3.2 Dataset

We have created a new dataset by merging several publicly available labelled sets (from GitHub, HuggingFace, and Kaggle). We ensured consistent labelling schemes using Pandas. In our approach to merging we also took care to remove the duplicate entries. Next, we manually scanned samples that were flagged by first-order heuristics: repeated words, odd punctuation patterns, or really awkward semantic structures. Simple examples of "unnatural" samples include placeholder-like tokens (like "XXXXXXXX"), repeated nonsensical phrases, or alignment issues seen in pairs of text and labels. Although subjective, this split was consistent in its rule: if the text was obviously incomplete, silly, or contained placeholder tokens, it was removed. Then, spelling errors were rectified across the corpus with the use of specialized libraries (SymSpell and Neuspell). We found this step justified because typos can introduce a large set of near-duplicate terms in the TF-IDF vectors – diluting its power. Also, LLMs rarely generate typos, so leaving such mistakes in our dataset could introduce an (exploitable) classifier bias. Typographical variance should not be a good feature in this case.

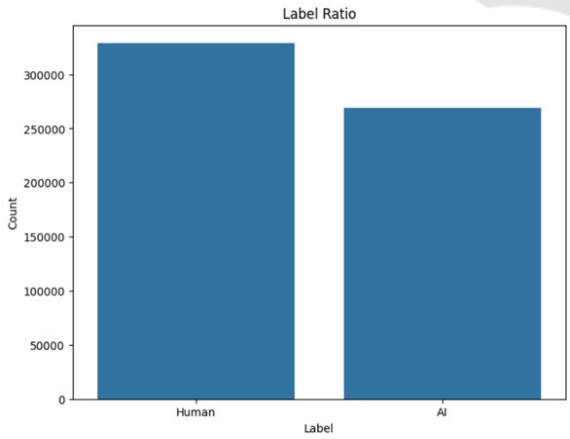

Figure 1. The balance between human and AI-generated text classes.

As can be seen in Figure 1, the resulting new prepared dataset is quite balanced between the human and AI-generated text classes. It contains 598441 samples that were further split as 80% train dataset and 20% test dataset.

3.3 Technical Implementation

We have used three approaches in order to generate text embeddings:

- TF-IDF representations (TfidfVectorizer);
- SBERT embeddings;
- SimCSE embeddings.

As output we get matrices of sizes (598441, 1000) for TF-IDF and (598441, 768) for SBERT and SimCSE. (In fact this CUDA-accelerated process yielded gigabytes of data.) The reason we included TF-IDF representations was that we wanted to see how relatively easily calculable embeddings can stack up against transformer-based embeddings. We note that TF-IDF representations took significantly less time to generate – they were finished in a few minutes, compared to a few hours for SBERT and SimCSE. And to our surprise – in several experiments, they deliver quite similar performance.

Term Frequency-Inverse Document Frequency representations (TF-IDF) is a statistical method that is frequently used in natural language processing tasks for representing sentences. It was introduced as a response to the traditional Bag-of-Words approaches in order to account for the importance of a word in a document relative to a corpus. As a refresher, the TF-IDF value is the product of two values – the term frequency and the inverse document frequency. Term frequency reflects the number of times a word appears in a document from a collection (which can be interpreted as the relevance of the word within the document itself), while inverse document frequency expresses how common or rare a word is across all documents in the collection (the assumption here is that less frequent words are of higher importance). By combining these two metrics through multiplication, the TF-IDF value effectively highlights words that are significant in a specific document similar to a Bag-of-Words approach, but the key difference lies in downplaying common words that appear across many documents.

Sentence-BERT (SBERT) embeddings – were derived from BERT embeddings and are a way of extracting text features at the sentence-level, essentially creating embeddings for whole sentences. While BERT alone excels at producing contextualized word embeddings, it cannot create fixed-sized vectors for entire sentences, which is a crucial requirement for tasks like semantic similarity

and clustering. SBERT approaches this issue by fine-tuning BERT on sentence pairs (via a siamese network). During inference, SBERT encodes each sentence independently into a fixed-dimensional vector (768 in this case), preserving the semantic relationships between sentences. As a result, we can use these vectors to do things like semantic search and question answering with a significantly reduced computational cost when compared to traditional models.

Our last type of text embeddings, Simple Contrastive Sentence Embeddings (SimCSE), is a recent and more sophisticated approach to generating sentence embeddings. It enhances the semantic content of the sentences through contrastive learning. The main point is training the model to distinguish between similar and dissimilar pairs of sentences. During training, the model is provided with sentences and their slightly perturbed versions (with the help of data augmentation techniques) as positive pairs, and unrelated sentences as negative pairs. Of course, the model builds upon pre-trained transformers like BERT or RoBERTa and is fine-tuned to bring the embeddings of positive pairs closer together while pushing apart the embeddings of negative pairs in the vector space. This supposedly helps capture fine-grained semantic nuances. There are many possible variants of loss functions for this task – for example, TripletLoss, MultipleNegativesSymmetricalLoss, and Softmax to name a few – all of them perform well for various use-cases.

For the binary classification task, the following well-known machine learning algorithms were used:

- Logistic Regression;
- K-Nearest-Neighbors (KNN);
- Naïve Bayes;
- Neural Network.

These classification algorithms were chosen almost arbitrarily for our experiments, therefore we do not further elaborate on how each algorithm works.

The dimensionality-reduction algorithms chosen were

- PCA;
- TSVD;
- FastICA.

All these algorithms are available in Python from scikit-learn (Pedregosa et al., 2011).

Principal Component Analysis (PCA) is a well-known linear dimensionality reduction technique. In fact, the principal components are just the eigenvectors of the input data covariance matrix ordered by the size of the associated eigenvalues. By selecting the top k principal components we can effectively reduce the data dimension – while simultaneously maintaining most of the data variance. The implicit assumption is that the variance captures the most significant information in the data.

Truncated Singular Value Decomposition (TSVD) is a variant of Singular Value Decomposition (SVD) and another popular technique. It is most commonly used when we want to reduce the dimensionality of sparse matrices (as for example CSR matrices).

Finally, Fast Independent Component Analysis (FastICA) is a faster way of doing Independent Component Analysis (ICA). In short, it is a computational way of separating multivariate signals into additive, independent components. In other words, its scope is to transform multivariate data points into statistically independent components. The first step in FastICA is to center the data (subtract the mean) and whiten it, that is alter the data points to become uncorrelated and have unitary variance. Whitening is typically achieved through Principal Component Analysis or Singular Value Decomposition – in a sense up to this point this technique it is a combination of the previous two methods. After this pre-processing step, the algorithm iteratively adjusts a set of weights in order to maximize the statistical independence of the rotated components. This is a very strong condition requiring infinite data to check; therefore a proxy called “maximal non-Gaussianity” is used. The algorithm stops when certain convergence criteria are met (for example, the difference from the changed weights to the initial is smaller than an epsilon).

4 EXPERIMENTS

The experiments were performed using Jupyter Notebook (The Jupyter Development Team, 2015), scikit-learn (Pedregosa et al., 2011) and PyTorch (Paszke et al., 2019).

4.1 Performance Metrics

Since our dataset is balanced and we are interested in a binary classification task, we considered (test) accuracy to be good as a base metric for evaluating the performance.

$$accuracy = \frac{number\ of\ correct\ predictions}{total\ number\ of\ predictions}$$

The accuracy takes values between 0 and 1 and the bigger the accuracy, the better the model at the classification task.

4.2 Tuning Model Parameters

For hyperparameter tuning (see Table 1), we have chosen to use Optuna (Akiba et al., 2019). This is a Bayesian hyperparameter optimizer that is very popular in machine learning competitions (for example the Kaggle competitions). Optuna basically is an open-source framework for hyperparameter optimization designed to automate and streamline the process of finding the best hyperparameters for various machine learning models. In a nutshell, the user needs to define the hyperparameter space and then Optuna repeats the process of suggesting hyperparameters by evaluating model performance and learning from the results. It dynamically adjusts its search strategy based on past trials to find better configurations more efficiently. Its algorithms intelligently navigate the hyperparameter space, reducing the time and resources required for manual tuning – it also helps that it is framework-agnostic, meaning it works with any possible algorithm – the user just needs to specify the value to be maximized (which is in our case accuracy).

4.3 Performance Results

The results obtained in our experiments for the performance of the machine learning models are contained in Table 2, in which we use the abbreviations TF for TF-IDF, SB for SBERT and SC for SimCSE. Is easy to see that the neural network classifier delivers the best accuracy with all embedding types and dimensionality-reduction algorithms. This came as expected. It's worth mentioning that TF-IDF representations performed even better than the transformers in some cases on some reduced datasets. Naïve Bayes was clearly the worst performing by far – it had the best accuracy of only 79% and the worst one of 67%. In all experiments done KNN showed an edge over Logistic Regression.

Regarding the performance of the text embeddings, while SimCSE embeddings are clearly delivering the best overall performance, we note that TF-IDF and SimCSE are delivering quite similar performance after the dimensionality-reducing algorithms are applied to the datasets – the performance difference between them is not significant. This is a quite surprising result and great news since it is much easier computationally to compute TF-IDF embeddings. SBERT embeddings were systematically the worst-performing embeddings.

Table 1: Hyperparameter Tuning Summary.

Model	Hyperparameters tuned with Optuna
Logistic Regression	C (Inverse of regularization strength), penalty ('l1', 'l2', 'elasticnet'), solver ('newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga')
K-Nearest Neighbors	number of neighbors, weights ('uniform', 'distance'), algorithm ('ball_tree', 'kd_tree', 'brute'), leaf_size
Naïve Bayes	var_smoothing (represents the portion of the largest variance of all features that is added to variances for calculation stability)
Neural Network	number of hidden layers, number of neurons per layer, activation functions ('relu', 'sigmoid', 'tanh'), optimizer ('adam', 'sgd'), learning rate, batch size, dropout rate

Table 2: Experimental Results – Test Accuracies (% of correct predictions on test).

Dimension Reduction	Logistic Regression			K-Nearest Neighbors			Naïve Bayes			Neural Network		
	TF	**SB**	**SC**	**TF**	**SB**	**SC**	**TF**	**SB**	**SC**	**TF**	**SB**	**SC**
None	89%	86%	91%	92%	91%	93%	79%	67%	76%	95%	93%	97%
PCA	84%	74%	82%	92%	90%	93%	75%	68%	72%	94%	91%	94%
TSVD	83%	74%	82%	91%	90%	93%	74%	67%	72%	93%	91%	94%
FastICA	84%	74%	82%	91%	90%	93%	74%	70%	74%	93%	92%	95%

Table 3: Computational costs – Time to finish training (minutes).

	Logistic Regression			K-Nearest Neighbors			Naïve Bayes			Deep Neural Network		
	TF	**SB**	**SC**	**TF**	**SB**	**SC**	**TF**	**SB**	**SC**	**TF**	**SB**	**SC**
None	6.75	100	10	0.16	0.12	0.1	4.75	1	1.25	197	210	110
PCA	0.21	0.25	0.12	0.01	<.01	0.01	0.06	0.06	0.06	62	27	20
TSVD	1	0.41	0.25	0.01	<.01	<.01	0.06	0.07	0.06	20	73	19
FastICA	0.06	0.06	0.19	0.01	<.01	0.01	0.06	0.06	0.06	48	50	19

Table 4: Computational costs – Inference on test set (seconds).

	Logistic Regression			K-Nearest Neighbors			Naïve Bayes			Deep Neural Network		
	TF	**SB**	**SC**	**TF**	**SB**	**SC**	**TF**	**SB**	**SC**	**TF**	**SB**	**SC**
None	0.92	0.25	0.11	<.01	0.01	0.03	5.54	5	4.74	5.47	6.78	5.81
PCA	0.01	0.01	0.01	<.01	<.01	<.01	0.30	0.26	0.28	4.90	0.85	0.36
TSVD	0.01	<.01	0.01	<.01	<.01	<.01	0.28	0.32	0.28	1.29	0.33	2.42
FastICA	0.01	<.01	<.01	<.01	<.01	<.01	0.29	0.30	0.29	1.85	1.26	2.52

Finally, we note that applying the dimension reduction techniques in order to significantly decrease the computational effort seems only to lead to a relatively minor decrease in the performance of the classification task.

4.4 Computational Effort

The text encodings and the training of the machine learning models were accelerated with CUDA – we have used an RTX 4060 Mobile graphics chip made by NVIDIA for this. Generating the full-text encodings took around 4 hours in total. Then performing the dimensionality reduction took around 20 minutes for each of the three algorithms – FastICA was (ironically) the slowest to finish – this makes sense since at the processing step FastICA includes the previous two. All processes, even the CPU-bound ones (like, for example, PCA and KNN) benefited from parallelization. The maximum amount of RAM memory needed was 10 GB. We completed a total of 720 training cycles (that is 48 Optuna runs with 15 trials each).

The training and inference times are presented in Tables 3 and 4, in which we use the abbreviations TF for TF-IDF, SB for SBERT and SC for SimCSE.

The reader should note that training the models after a dimensionality reduction algorithm was applied is **5** to **500** times faster than on the full dataset. In Table 4 we can see a similar trend for inference time as well. (The times presented in Table 4 are for the test dataset inference).

4.5 Threats to Validity

Internal Validity: The preparation procedure for the dataset relies on subjective decisions about what counts as "unnatural" texts. While we attempted to systematically codify guidelines, there may remain some bias regarding which texts were filtered out. Although hyperparameter tuning with Optuna is robust, it can still overfit to the training split if not carefully cross-validated.

External Validity: The data were sourced from publicly available sources (GitHub, HuggingFace, Kaggle), which harbour potential biases with respect to domain coverage. Therefore, caution must be exercised in generalizing to highly professional or domain-specific texts, for example, in the case of legal or medical texts. The dimension-reduction methods could perform differently when analysing extremely large corpora or new languages.

Construct Validity: The metric chosen for performance evaluation was classification accuracy. While it covers much ground in terms of statistical performance, other metrics such as F1, precision, recall could prove useful for a more complete analysis. Future studies could measure robustness to adversarial attacks, interpretability, or real-time adaptiveness, but this is beyond the scope of analysis in this work.

Robustness: There was no systematic analysis on how dimension-reduced embeddings stand up to adversarial examples specifically engineered to fool AI detectors. The preliminary findings imply that, due to a domain shift or the introduction of synthetic perturbations, the detection accuracy drops.

Therefore future works should explore the intricacies of the vulnerabilities in detail.

5 CONCLUSIONS

In this study we tackled the idea of applying dimensionality reduction techniques (PCA, TSVD, and FastICA) to multidimensional text embeddings vectors generated via different approaches. Our experiments show that across the board, irrespective of the type of text embeddings or dimension reduction technique, applying such methods to text embeddings vectors may lead to an important reduction in the computational effort without a corresponding significant reduction in performance.

In particular, applying dimensionality reduction techniques can drastically improve the running times and overall efficiency of AI text detectors. Through all experiments, such techniques not only accelerated training and inference times but also preserved performance. By using dimension reduction techniques on can successfully manage the trade-off between reducing model complexity and maintaining the accuracy of the predictions made. This makes them perfectly suitable for deployment, especially in resource-constrained environments.

Regarding the text embeddings we also note that SimCSE consistently delivered the best accuracy, with the lightweight TF-IDF proving to be a fierce competitor beating the SBERT on multiple trials.

We think that incorporating these methods provides a viable pathway towards optimizing large-scale AI detectors, contributing to both their scalability and efficiency. Other important practical implications might be speeding up real-time systems like social-media and content monitoring which proves to be an increasingly important problem. Also, applying dimensionality reduction in resource-constrained environments can reduce the memory and energy requirements enough to make detection significantly cheaper, but also feasible on smaller devices. There also are potential industrial applications – dimension reduction enables the handling of high-throughput demands, from massive-scale news verification to customer-service chat filtering, without necessarily losing serious performance.

Future research could look into the integration of these techniques in advanced language models while investigating their capacity to stand against potential attacks – which a key point in this field. Having the capacity to detect generated text in a context in which some user included adversarial tactics in order to fool potential detectors (e.g. including subtle changes and aiming for the text to sound more human) is a very important aspect that should be further investigated. Another future direction may examine adversarial robustness to check whether reduced embeddings are easier or harder to fool. Finally, hybrid methods integrating statistical dimensionality reduction techniques with semantic feature pruning could prove useful in order to further improve the results.

REFERENCES

Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M. (2019). Optuna: A Next-generation Hyperparameter Optimization Framework. In *KDD 2019, Proceedings of the 25th ACM International Conference on Knowledge Discovery & Data Mining*, pages 2623 – 2631. https://doi.org/10.1145/3292500.3330701

Antoun, W., Mouilleron, V., Sagot, B., Seddah, D. (2023). Towards a Robust Detection of Language Model Generated Text: Is ChatGPT that Easy to Detect? In *Actes de CORIA-TALN 2023*, pages 14 – 27.

Bender, E. M., Gebru, T., McMillan-Major, A., Shmitchell, S. (2021). On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? In *FAccT 2021, Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pages 610 – 623. https://doi.org/10.1145/3442188.3445922

Bhattacharjee, A., Liu, H. (2024). Fighting Fire with Fire: Can ChatGPT Detect AI-generated Text? *ACM SIGKDD Explorations Newsletter* 25, pages 14 – 21. https://doi.org/10.1145/3655103.3655106

Casal, J. E., Kessler, M. (2023). Can linguists distinguish between ChatGPT/AI and human writing? A study of research ethics and academic publishing. *Research Methods in Applied Linguistics* 2, 100068. https://doi.org/10.1016/j.rmal.2023.100068

Dou, Y., Forbes, M., Koncel-Kedziorski, R., Smith, N., Choi, Y. (2022). Is GPT-3 Text Indistinguishable from Human Text? Scarecrow: A Framework for Scrutinizing Machine Text. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers),* pages 7250 – 7274. https://doi.org/10.18653/v1/2022.acl-long.501

Guo, B., Zhang, X., Wang, Z., Jiang, M., Nie, J., Ding, Y., Yue, J., Wu, Y. (2023). How Close is ChatGPT to Human Experts? Comparison Corpus, Evaluation, and Detection. *Preprint* arXiv: 2301.07597.

Hu, X., Chen, P.-Y., Ho, T.-Y. (2023). RADAR: Robust AI-Text Detection via Adversarial Learning. *Preprint* arXiv: 2307.03838.

Ignat, O., Xu, X., Mihalcea, R. (2024). MAiDE-up: Multilingual Deception Detection of GPT-generated Hotel Reviews. *Preprint* arXiv:2404.12938.

Ippolito, D., Duckworth, D., Callison-Burch, C., Eck, D. (2020). Automatic Detection of Generated Text is Easiest when Humans are Fooled. In *Proceedings of*

the 58th Annual Meeting of the Association for Computational Linguistics, pages 1808 – 1822. https://doi.org/10.18653/v1/2020.acl-main.164

Jakesch, M., Hancock, J. T., Naaman, M. (2023). Human heuristics for AI-generated language are flawed. *Proceedings of the National Academy of Sciences* 120, e2208839120. https://doi.org/10.1073/pnas.2208839120

Jawahar, G., Abdul-Mageed, M., Lakshmanan, L. V. S. (2020). Automatic Detection of Machine Generated Text: A Critical Survey. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2296 – 2309.

Khan, W., Turab, M., Ahmad, W., Ahmad, S. H., Kumar, K., Luo, B. (2022). Data Dimension Reduction makes ML Algorithms efficient. In *Proceedings of the 2022 International Conference on Emerging Technologies in Electronics, Computing and Communication (ICETECC)*, pages 1 – 7. https://doi.org/10.1109/ICETECC56662.2022.10069527

Koike, R., Kaneko, M., Okazaki, N. (2024). OUTFOX: LLM-Generated Essay Detection Through In-Context Learning with Adversarially Generated Examples. In *AAAI 2024, Proceedings of 38th AAAI Conference on Artificial Intelligence*, pages 21259 – 21266. https://doi.org/10.1609/aaai.v38i19.30120

Kumarage, T., Garland, J., Bhattacharjee, A., Trapeznikov, K., Ruston, S., Liu, H. (2023). Stylometric Detection of AI-Generated Text in Twitter Timelines. *Preprint* arXiv: 2303.03697.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. *Preprint* arXiv:1907.11692.

Moulik, R., Phutela, A., Sheoran, S., & Bhattacharya, S. (2023). Accelerated Neural Network Training through Dimensionality Reduction for High-Throughput Screening of Topological Materials. *Preprint* arXiv: arXiv:2308.12722.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems* 32, pages 8024 – 8035.

Patel, P., Choukse, E., Zhang, C., Shah, A., Goiri, I., Maleki, S., Bianchini, R. (2024). Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *ISCA 2024, Proceedings of the 51st ACM/IEEE Annual International Symposium on Computer Architecture*, pages 118–132. https://doi.org/10.1109/ISCA59077.2024.00019

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12, 2825 – 2830.

Rojas-Simón, J., Ledeneva, Y., García-Hernández, R. A. (2024). A Dimensionality Reduction Approach for Text Vectorization in Detecting Human and Machine-generated Texts. *Computación y Sistemas* 28, pages 1919 – 1929. https://doi.org/10.13053/cys-28-4-5214

Singh, K. N., Devi, S. D., Devi, H. M., Mahanta, A. K. (2022). A novel approach for dimension reduction using word embedding: An enhanced text classification approach. *International Journal of Information Management Data Insights* 2, 100061. https://doi.org/10.1016/J.JJIMEI.2022.100061

Tang, R., Chuang, Y.-N., Hu, X. (2024). The Science of Detecting LLM-Generated Texts. *Communications of the ACM* 67, pages 50 – 59.

The Jupyter Development Team. (2015). Project Jupyter. Jupyter Notebook. *Available at* https://jupyter.org/.