STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXIX**, Number 2, 2024
DOI: 10.24193/subbi.2024.2.03

DEEP LEARNING APPROACHES FOR DETECTING TEXT GENERATED BY ARTIFICIAL INTELLIGENCE

DAVID BIRIȘ

Abstract. Large language models have been a hot topic for discussion and research for quite a few years, allowing them to infiltrate in many industries, especially education. Their rise in popularity among students was caused by their vast capabilities in giving quick and reliable answers to questions on any topic. The use of these models for the purpose of generating schoolwork can be seen as a challenge to academic integrity. We investigate the development of AI capable of detecting AI-generated texts and explore with training different types of deep learning models, on a mixed dataset, containing essays, both human written and AI-generated, as well as movie reviews and books. We experimented with LSTM (Long short-term memory) and fine-tuning transformer based models. We achieve results close to the state of the art, and, in some particular cases, we surpass a few of these models. For instance, one of our models surpasses a state of the art model on a set of both student written and generated essays, in terms of accuracy by up to 5%, and F1 score by up to 4%, in two different experiments. Furthermore, our another model of ours surpasses a state of the art model on a set of essays, but this time only in terms of precision, by only 1%. These results indicate the potential of properly fine-tuned transformer-based models, as well as the importance of a well-prepared dataset.

1. Introduction

Ever since the revolutionary introduction of the transformer model in 2017 [29], the artificial intelligence industry has experienced a never-before-seen

Received by the editors: 31 July 2024.

2010 *Mathematics Subject Classification.* 68P15, 94A12.

1998 *CR Categories and Descriptors.* I.2.7 [Artificial Intelligence]: Natural Language Processing – Text Analysis; I.2.6 [Artificial Intelligence]: Learning – Deep Learning; H.3.1 [Information Storage and Retrieval]: Content Analysis and Indexing – Content Analysis and Feature Selection .

Key words and phrases. machine learning, chatbot, AIGT, detection.

explosion in both performance and popularity. While the transformer was initially designed for translation tasks, it has since been adapted for a varied range of uses in natural language processing and way beyond. Large language models (LLMs) were introduced the very next year, with pioneers such as OpenAI's Generative Pre-Trained Transformer (GPT) [27] and Google's Bidirectional Encoder Representations from Transformers (BERT) [8]. One more year later, LLMs began to be publicly accessible with the release of GPT-2, and, in no time, people realized the immense potential of these models as chat bots.

Today, these chat bots are everywhere and are immensely more capable than their predecessors. They are very popular and widely used by people in diverse everyday tasks. Moreover, tools like ChatGPT have become helpful allies for students, when trying to do their schoolwork. According to [3], 19% of students aware of ChatGPT admitted to having used it for schoolwork. This number is already high, but we can safely assume that, in reality, it might be a lot higher, since only 25% first-year computer science students that participated in a survey for this study [5] declared to have never used ChatGPT for their assignments.

While these tools might not yet be useful for reliably solving complex problems, they certainly can write a satisfactory essay on any topic, especially for students in earlier grades. Research made by OpenAI [22] shows that GPT-4 on its own can even pass many exams, with high grades. Some people might see this as concerning, since students' usage of these tools lead to inaccuracies in the evaluation of their text comprehension, writing abilities and both logical and critical thinking.

To address these concerns, we create a tool designed to help both students and teachers. We aim not to punish students for using large language models to learn and find ideas for their assignments, which can be a good habit, but to discourage the damaging practice of carelessly copying and pasting entire AI-generated essays and sending them as homework solutions.

In the following, we will present the steps we took to achieve to this finished product in great detail. We will start by presenting three state of the art models, in a section dedicated to related work. The next section will show the employed methodology and will briefly discuss the most important architectural aspects of each of the models. The experiments section focuses on first discussing the content of the datasets and then the processing techniques that have been applied to the data. After that, the process of training and hyperparameter tuning is discussed, and the section is ended with results and a detailed comparison between our models and the state of the art presented in the aforementioned section. The article ends with a section that draws conclusions and brings out possible future improvements.

2. Related Work

2.1. **Sentence-Level AI-Generated Text Detection with SeqXGPT.**

Most AIGT (AI-Generated Text) detection strategies are made with the purpose of detecting, with as high of an accuracy as possible, if an entire document is automatically generated by a LLM, rather than using a sentence-by-sentence approach. While this technique may be useful in many cases, people do not generally rely solely on AI to generate entire documents, and, instead, they often use it to modify or enhance content that was originally written by humans. These AI modifications can be simple enhancements or additions to certain sentences, or entire new AI-generated paragraphs scattered between human-written text. Therefore, using a sentence-level AI generated content detection strategy is crucial in some cases.

SeqXGPT [30] is an open-source, advanced method for sentence-Level AI-Generated text detection. Its approach consists of three parts: Perplexity extraction and alignment, Feature Encoder and Linear Classification Layer.

The AIGT detection tests performed on SeqXGPT show a significant difference when compared to other AIGT detection methods, such as DetectGPT [21] or Sniffer [18]. The LLMs used to construct the dataset are GPT2-x1, GPT-Neo, GPT-J and LLaMA.

2.2. **Zero-Shot AI-Generated Text Detection with Fast-DetectGPT.**

According to a report by OpenAI [28], zero-shot detection uses a pre-trained generative model on text generated either by itself or by other similar models. This is done without subjecting the model to any supplementary training. Zero shot means that access to human written or AIGT samples is not assumed to perform detection. Generally, when using zero-shot AIGT detection, the average per-token log probability of the generated text is evaluated.

This model relies on the hypothesis that language models tend to use tokens with higher statistical probability because they have been pre-trained on lots of human written content. In contrast, humans individually do not display such bias since they compose texts based on contexts, meanings, and intents rather than data and statistics [1]. This hypothesis stems from the fact that language models try to mirror collective human writing behaviour instead of individual human writing behaviour, therefore presenting a contextual difference in the choice of words. This means that the conditional probability function $p(\widetilde{x}|x)$ is much higher for an AIGT x, in comparison with human written text.

Fast-DetectGPT works in these three steps:

- Step 1. Sample alternative word choices $\widetilde{x}_i$ for each token x.

- Step 2. Evaluate the conditional probabilities $p(x|x), p(\widetilde{x}_i|x)$ of these generated samples.
- Step 3. Compare them to arrive at a detection decision: $\frac{1}{n}\sum_i \log \frac{p(x|x)}{p(\widetilde{x}_i|x)} \overset{?}{>} \epsilon$

Here, x represents the entire input text, x_i denotes the i-th token in x, $\widetilde{x}_i$ represents an alternative token generated as a substitute for x_i, and n is the total number of tokens in the input text. The decision to classify the text as AI-generated or human-written depends on whether the aggregated score exceeds the threshold ϵ.

Empirical evidence shows an increase of around 75% in detection accuracy for Fast-DetectGPT over its predecessor, DetectGPT [21].

2.3. **Adversarial learning with RADAR.**

In the context of AIGT detection tools, we refer as *adversarial learning* to the process in which two models are trained at the same time, with two different, even opposing, goals in mind. One of them is a generative model and the other is a discriminative model that tries to determine, with a certain probability, whether some sample came from the generative model or the training data [12]. The generative model's goal is to make the probability of the discriminative model to make a mistake as high as possible.

RADAR: Robust AI-Text Detection via Adversarial Learning is a framework for AIGT detection, that uses adversarial learning. By this method, the discriminator is a "two player game", composed of the paraphraser and the detector. These two "players" have contrasting objectives, because the paraphraser has to generate human-like content that should be able to avoid AI detection, while the detector's job is to enhance AIGT detectability. In the training phase, the paraphraser learns to rewrite text from a dataset generated by a target LLM, while trying to decrease the probability that the detector will be able to discern the difference between the paraphraser's text and human written text. At the same time, the detector learns to compare AIGT from the training dataset and from the paraphraser's output with human written text, in order to improve the detection performance [15].

3. Proposed models

The task at hand is formulated as a binary classification task with two classes: human written, which is considered to be negative and AI-generated, which is considered to be positive. Each input text belongs to one of these two categories and it will be evaluated at document-level, meaning that there will be no sentence-level analysis, such as in the case of SeqXGPT. We fine tune some relatively lightweight transformer-based models, like BERT and

DeBERTa (Decoding-enhanced BERT with Disentangled Attention) [13], since their small size allows them to be directly integrated in applications, without the need of an API, unlike some newer and heavier models, that cannot directly run on users' personal computers. Additionally, they are free to use and open source. For the same reason, we also train an LSTM, which turns out to be even lighter than the transformer models, at the expense of some performance.

3.1. **LSTM.** We start with the implementation of a LSTM model [14]. The text data is tokenized with the *basic_english* tokenizer provided by PyTorch, in the torchtext package [23]. The vocabulary is built from these tokens. We apply padding to the sequences, since the LSTM expects constant length inputs. We use the LSTM class from PyTorch to build the model and try multiple sets of hyperparameters to find the ideal ones, using Wandb [2].

The training is done using K-fold cross validation, splitting the dataset in K parts. One of the K parts is used as a validation dataset, and the other $K-1$ are used for training. This process is repeated until each of the parts has played the role of a validation dataset. We use the *KFold* function from *scikit-learn* [25] to set up the cross validation and we experiment with $3-5$ splits. The data is shuffled every time before splitting to ensure that the model does not make any connections related to the order of the entries, since the order does not matter. We save the model for every fold and pick the one with the highest performance.

As an optimization algorithm for gradient descent we use Adam.

The loss function used is a Binary Cross Entropy Loss, with a sigmoid over the outputs of the model, called *BCEWithLogitsLoss*, in the PyTorch library:

$$l_i = -w_i\big(y_i \cdot \log \sigma(x_i) + (1-y_i) \cdot \log(1-\sigma(x_i))\big) \tag{1}$$

for the ith example in the batch, where x_i is the the i^{th} raw output of the neural network, y_i is the actual i^{th} truth label and w_i is a weight associated with the i^{th} example, and can be useful if we want to give different importance to different examples in the batch.

The overall loss for the batch is the mean if the individual losses:

$$l(x,y) = \frac{1}{N}\sum_{i=1}^{N} l_i, \tag{2}$$

where l_i is defined in Equation 1, N is the batch size and $x = \{x_1, x_2, \ldots, x_N\}$, $y = \{y_1, y_2, \ldots, y_N\}$.

For this LSTM, we create a PyTorch module (a Python class that inherits from the Module class in PyTorch) named *TextLSTM*. Our final model will have type *TextLSTM*. Its constructor initializes the embedding layer, giving as parameters the size of the aforementioned vocabulary and the dimension of the embedding vectors, which we chose to be 100. The constructor also initialized the LSTM class from PyTorch that we have mentioned, as well as a linear layer called *fc* that maps the output of the LSTM (with dimension *hidden_dim*, a hyperparameter we chose to be 256) to the desired output size (which in this case is 1), since our task is binary classification. The *forward* pass function starts by embedding text and then passing it through the LSTM. Then it takes the last layer from the hidden states and passes it through the *fc*, before returning it.

3.2. **BERT.** We also fine-tune multiple transformer-based models, the first one being BERT. We use *bert-base-uncased*, a version of BERT with 110 million parameters. The parameters are basically the total number of weights and biases from the transformer's layers. It has been pre-trained only on English datasets. We choose the uncased version from the presumption that the case of the letters in a text are not significantly relevant in the context of detecting whether a text is AI-generated or not. BERT's tokenizer includes the functionality of encoding the tokens, but has the limitation of admitting a maximum of only 512 tokens per input, which might cause loss of data on entries with texts longer than that. This is why we might want to truncate and split the text into multiple sections of 512 tokens, but then we sometimes run into the problem of losing the necessary context for the transformer to make the needed connections for discriminating between AI and human text. We will handle this problem in the data preprocessing subsection 4.2. If the tokenizer does not find a specific word in the BERT dictionary, it splits it in the largest subwords from the vocabulary. In the rare case when the tokenizer cannot find a subword in the vocabulary, the entire word is tokenized as unknown [9]. If a sequence is already split in 512 tokens (the maximum number), and we need to split a word in subwords, the sequence will be truncated to the maximum length [8].

For training, we split the dataset into three sets: 80% of the data was used for training, 10% for testing, and 10% for validation. The test dataset is used only to evaluate the performance of the final model. The other two sets are used during training, just like in the LSTM: the validation is used to evaluate the model every *evaluation_frequency* steps on data that is new for the model, since it is not included in the training dataset. The number of steps in an epoch is dependent on the batch size, since each epoch is a complete pass through the entire dataset and the batch size indicates how many samples are taken for each forward back propagation of the neural network layer of the transformer. So, the number of steps in an epoch is $S = \frac{\text{dataset size}}{\text{batch size}}$.

We use AdamW as optimization algorithm for gradient descent with learning rate $5 \cdot 10^{-5}$. During the 1500 warm up steps, the learning rate is set to linearly increase to the desired value. The learning rate is set to linearly decrease to 0 by the end of the training process. This learning rate scheduler setup allows for the learning rate to stabilize the training in the early steps, since the model has not yet had the chance to adjust to the dataset and we risk overshooting minima of the loss function. After warm up, the learning rate is set to gradually decrease, and this helps the model tune the weights lightly near the end of the training.

3.3. **DeBERTa.** We can fine-tune more BERT-based transformers models, such as DeBERTa [13]. We use the base version of DeBERTa, *deberta-v3-base*, with around 86 million parameters. It has 12 layers and a hidden size of 768. We split the dataset in the same way we do for the BERT model. Again, we use the AdamW optimizer, with learning rates like $5 \cdot 10^{-5}$ or $3 \cdot 10^{-5}$. The same 1500 warm up steps are followed and the learning rate linearly decays during training.

The improvement DeBERTa brings over BERT is that it separates the content and position information in different embeddings. This approach allows the model to more effectively focus separately on semantics and positions of the tokens. Additionally, DeBERTa has an upgraded mask decoder which gives it better predictions during training. With these perks, DeBERTa is a generally more efficient and better performing model in natural language processing tasks than BERT or RoBERTa [19].

4. Experiments

A crucial step in training a text classification model is thoughtfully compiling a dataset of content with diverse writing styles. The models train their weights to find patterns that discern real texts from fake ones, so it it very important to not accidentally introduce biases or imbalances that could skew the model's performance and to ensure these sources vary in tone, complexity, and subject matter. We gather content from a wide range of sources including books, reviews and especially essays, since our tool is created with the intent of primarily detecting schoolwork.

4.1. Datasets.

4.1.1. *DAIGT-V4.* [17] is a dataset compiled from a number of different sources. The AI-generated section has texts generated by different models: LLaMA - 15,796 texts, Mistral - 13,439 texts, Falcon - 4,536 texts, GPT - 4,161, DaVinci - 2,099 texts, Claude - 2,000 texts, PaLM - 1,733 texts, Babbage - 698 texts, Curie - 696 texts, Ada - 692 texts, Cohere - 350 texts. The human generated content is composed of argumentative essays written by 6th-12th grade students [6]. This dataset contains a total of 27,370 human generated texts and 46,200 AI generated texts.

The dataset contains values under multiple labels, such as the text itself, a value that tells us whether the content is AI or human generated, the name of the model it has been generated by, as well as the prompt used for generation. The topics (prompts) of the essays are the same for both human and AI texts.

4.1.2. *DAIGT Gemini-Pro 8.5K Essays.* This dataset [7] brings 8,500 more essays generated using the same prompts as the ones from DAIGT-V4. They are generated by GeminiPro. The CSV file contains multiple labels, such as the text itself, a value that tells us whether the content is generated by AI or written by humans, and also the prompt used to generate the text.

4.1.3. *IMDB 50K Movie Reviews.* This dataset [24] provides a set of 50,000 movie reviews from IMDB, written by humans. The CSV file provides both the review and the opinion reflected by the person in the review (positive or negative sentiment towards the movie), for sentiment analysis. For our purpose, we will not need to use the sentiment, and we will use this dataset as a collection of human written text.

4.1.4. *ArguGPT.* [20] provides 4,038 argumentative essays, on different topics, written by GPT (7 models). The CSV file contains labels for the text, the prompt, as well as an id for each text and prompt and also the individual GPT model used for generation.

4.1.5. *Raw IELTS essays.* Raw IELTS essays [4] is a collection of student-written essays, from the IELTS test. It provides a valuable amount of 4,158 essays, that should definitely help the model find different human writing patterns, during training.

4.1.6. *SeqXGPT's sentence-level detection dataset.* A section of the document-level detection dataset used for evaluating SeqXGPT [30]. It contains, among human generated texts, content from GPT-2, GPT-3, GPT-J, GPT Neo and Llama. We take the GPT-2, GPT-3 and human texts for training, and leave the rest for subsequent testing. GPT-2 and GPT-3 are extensively studied and thoroughly evaluated models, making them suitable choices for establishing a solid training dataset. We reserve GPT-J, GPT-Neo, and LLaMA for testing, in order to ensure that the trained model is evaluated on texts it has not seen during training phase. We use a total of 600 texts for training, from this dataset, 200 human written and 400 human generated.

4.1.7. *Some books.* We also include some books in the dataset. Books offer very high quality examples of human writing, while also being well edited and well reviewed. This should help the model's ability to detect subtle characteristics of human authored text. For diversity of writing styles, we choose both newer and older books. The books we have included in our dataset are *Crime And Punishment*, *The Great Gatsby*, *The Hunger Games*, *Frankenstein*, *Twilight*, *Harry Potter*, *The DaVinci Code* and *Tarzan Of The Apes*. After splitting these books in sections of maximum 450 words (in order to have similar length texts in the dataset) we get about 2,100 small texts.

4.1.8. *Alpaca GPT4.* Alpaca GPT4 [10] [26] is a collection of instruction-following texts generated by GPT-4. It does not include essays, but these texts might help our model understand some more diverse patterns in AI-generated content, so it might be beneficial to not only include essays in the training dataset. For this reason, we choose a random set of 2,100 samples of the total 52,000. Since these texts are not essays, they have an unusual writing style compared to the other AI-generated texts, so we choose 2,100 in order to match the number of texts from the previous human-written dataset, composed of books. This way, we have a balance between AI-generated and human-written texts with different writing styles.

4.1.9. *AI Vs Human Text.* This dataset is a huge cluster of essays, both AI-generated and human written essays[11]. It includes some of the datasets presented above and many more. Since our tool targets detecting AI-generated content in academic scenarios, this dataset is a very good choice due to its large collection of essays, aligned with academic writing styles. This dataset contains around 300,000 human written and 180,000 AI-generated essays. We have used AI-generated some texts from "AI Vs Human Text" only as a filler, for balancing the training dataset, since we ended up with more human written content.

4.1.10. *Testing Dataset.* This dataset will only be used for testing, so we do not count it in with the other training datasets. We use another IELTS essays dataset [16], different from the one used for training. This dataset contains both the question that the students were asked to write the essay about, and the student essay itself. We take a sample of these questions and ask the most popular AI models to write essays as well. Now we have created our test dataset with some student essays and some generated essays. We have, in total, 1332 pieces of writing created by *gpt-3.5-turbo*, *gpt-4-turbo* and *gpt-4o*, the latest model from OpenAI, as of this writing. We have generated these essays ourselves, using the API that OpenAI has made publicly available.

4.1.11. *Data splitting.* The number of texts in the final dataset and their provenience is described in detail in Table 1. We split the dataset in three sections: the training set (80% of the entire dataset), the validation set and the test set (both 10%). The training set is used for the actual process of adjusting the weights of the model while training. The validation dataset helps assessing the model's performance during training and preventing overfitting by providing a separate set of data to evaluate the immediate performance of the model. The test set is a completely unseen section of the data, that provides an unbiased evaluation of the model's performance after the training is completed.

Datasets				
Dataset Name	Human Written Texts	AI-Generated Texts	Total	AI/Human Ratio
DAIGT-V4	27,370	46,200	73,570	1.68
DAIGT Gemini-Pro 8.5K	0	8,500	8,500	-
IMDB 50K Movie Reviews	50,000	0	50,000	-
ArguGPT	0	4,038	4,038	-
Raw IELTS essays	4,158	0	4,158	-
SeqXGPT's sentence-level detection dataset	200	400	600	0.5
Alpaca GPT4	0	2,100	2,100	-
Books	2,100	0	2,100	-
AI Vs Human Text	0	22,590	22,590	-
Total	83,828	83,828	167,656	1

TABLE 1. Summary of datasets used.

4.2. **Data preprocessing.** Before doing any further processing on our dataset, the texts have been grammatically corrected. Correcting grammar is an important step in processing our datasets. We use a special Python library called *language-tool* to correct all the grammatical errors in all our texts, both AI and human written. We do not want our model to form bias towards labeling a text as AI just because it does not have grammatical errors. After correcting the grammar in the texts, we see that 1,456,283 errors have been corrected in human texts and only 631,083 in AI content. This data supports our previous hypothesis that the model could have been biased towards labeling correctly written text as AI, when trained on an uncorrected dataset.

Since transformer-based models like BERT or DeBERTa have a maximum input size of 512 tokens, we cannot keep texts longer than 512 tokens in our dataset. We could just truncate the longer texts, but this way we could lose meaningful context from those texts. This is why we will just discard the texts with more than 500 words. We choose to count words instead of tokens since the BERT and DeBERTa tokenizers are different, but they both generally split tokens as words. The reason we choose not to split them into chunks, as we have previously done for the books dataset is because there is not enough content that we can work with in these texts. The texts are generally only slightly longer than 500 words, and, by keeping the surplus in a separate chunk, we would have many short pieces of text with no context behind. We will also discard texts with less than 50 words, since they might not provide enough context for the AI model to properly train. The vast majority of the texts lied in the desired range, even before this processing, as can be seen in Figure 1, but, after discarding texts that are too long or too short, Figure 2 shows a nicer, almost Gaussian distribution of text lengths. We have discarded a few too long or too short texts, but we still have 152,386 texts to work with, 76,534 human written and 75,852 AI-generated.

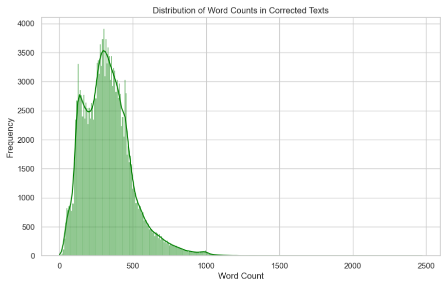

FIGURE 1. Initial distribution of word counts in the dataset.

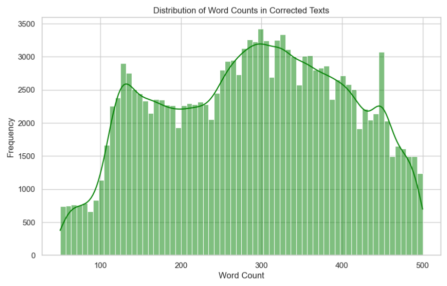

FIGURE 2. Distribution of word counts in the dataset, after discarding too long or too short texts.

4.3. **Metrics.** During training, we compute accuracy, loss, and validation loss every epoch. These values, called metrics, specifically the last two, help us interpret the progress during training. Validation loss is computed on the validation dataset, and loss on the training dataset. If the two loss values become closer and lower, it is a good sign that the training process has steady and good progress. Otherwise, if they are far apart, this might indicate *overfitting*.

We can also track the learning rate and *f1 score*. The f1 score is a function of *precision* and *recall*. The precision is a metric that shows us how accurate the positive predictions are. We will abbreviate true positives by TP, false positives by FP and false negatives by FN. TP are the texts correctly classified as AI-generated, and FN are texts incorrectly classified as human written. Then, the precision has the formula $\text{Precision} = \frac{\text{TP}}{\text{TP+FP}}$. Recall is the ration between true positives and all the actual positives: $\text{Recall} = \frac{\text{TP}}{\text{TP+FN}}$. The f1 score measures the balance between precision and recall and has values between 0 and 1, 1 meaning perfect precision and recall:

$$\text{f1\_score} = \frac{2 \cdot \text{TP}}{2 \cdot \text{TP} + \text{FP} + \text{FN}} \tag{3}$$

4.4. **Training and hyperparameter tuning.** It is good practice to implement multiple models, based on different architectures, and tune their hyperparameters, in order to find the best possible solution. The deep learning library of choice is PyTorch [23]. Hyperparameter tuning is done with Wandb. Wandb *Weights & Biases* is a platform that helps with tracking a history of experiments in machine learning. It provides the tools to log each training run and save the training progress, the hyperparameters and the metrics of the model (loss, accuracy, f1_score, etc.), as well as many details about the system's performance during training. In addition, it creates graphs with these metrics. The *sweep* functionality from Wandb allows us to pre-plan multiple sets of hyperparameters and Wandb will train the model with these multiple possible settings so we can choose the preferred one.

When training the LSTM model, we choose to use 5 folds for k-fold cross validation and experiment with different hyperparameters. The best results have been achieved with the dimension of the vector space in which words are represented (embedding dimension) set to 100. We choose the number of neurons in the hidden layers (hidden dimension) to 256, the learning rate to 10^{-3} and the dropout rate to 20% of the neurons. We train this model with a batch size of 128. The training and validation loss progress for each of these 5 folds can be seen in Figure 3.

When fine-tuning BERT, only 3 epochs are needed, since this is a transformer-based model, pre-trained on massive datasets and has already captured a significant number of natural language features. This is the reason why the learning rate we set is much lower, compared to the LSTM one, specifically 10^{-5}. During fine-tuning we only need to finely adjust the model's weights, to fit our new text classification purpose. We also need to set the batch size to

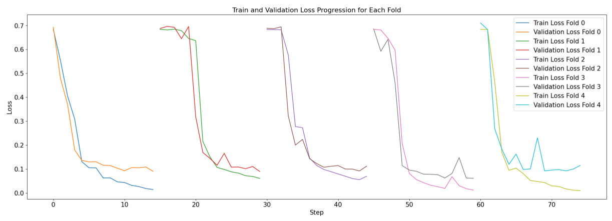

FIGURE 3. BCEWithLogitsLoss (sigmoid wrapped cross entropy loss) progression in the training process of the LSTM model (training vs validation loss), for each of the 5 folds

a lower number, 16, since the multi-head attention mechanisms of the transformer require much more video RAM than the LSTM architecture. The loss progression during the fine-tuning process is displayed in Figure 4.

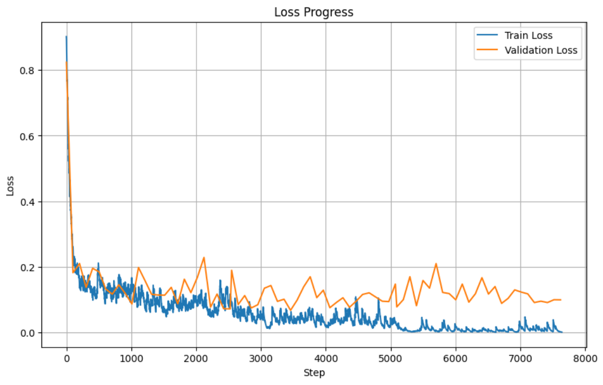

FIGURE 4. Training vs validation loss during the fine-tuning of the BERT model.

We fine-tune DeBERTa for 4 epochs, but this time we use three *sweeps* to automatically find the best hyperparameters, instead of manually changing them and trying again. The best results are achieved with a dropout rate of 0.1, a initial, linearly decaying learning rate of $3 \cdot 10^{-5}$ and a batch size of 16. The first 1000 steps (first 1000 batches) in the first epoch are used as warm up steps, to gradually increase the learning rate to the initial value of $3 \cdot 10^{-5}$. The model is trained for two epochs, and the training progress can be seen in Figure 5.

4.5. **Results.** We will try to find a winner between our models, by subjecting them to a few classification test on our testing dataset. We will go through all

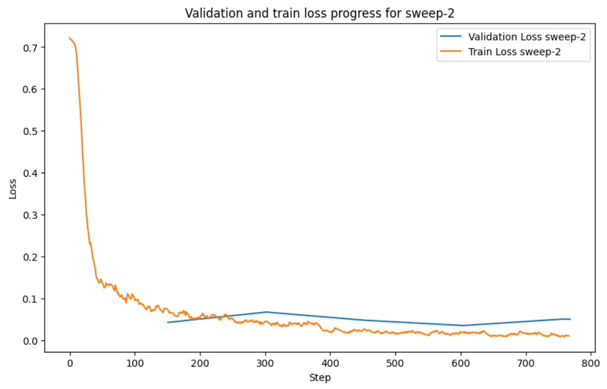

FIGURE 5. Training vs validation loss during the fine-tuning the DeBERTa model.

the sections of the dataset, which, as previously stated, is composed of human written and *gpt-3.5-turbo*, *gpt-4-turbo* and *gpt-4o* generated essays, as well as a section from SeqXGPT's dataset. We will begin by comparing only our three models on the *gpt-3.5-turbo* section, in order to establish a baseline, and then we will continue with a comparison with the state of the art models. We will compare the models by calculating metrics such as accuracy, precision, recall and f1 score for each one of them.

Model	Accuracy	Precision	Recall	F1 Score
Our LSTM model	0.78	0.78	0.78	0.78
Our BERT model	0.86	0.88	0.86	0.86
Our DeBERTa model	**0.90**	**0.91**	**0.90**	**0.90**

TABLE 2. Comparison between results from all our three models on 955 gpt-3.5 turbo vs 955 human texts from our testing dataset.

By doing some tests using our testing dataset we can observe that our DeBERTa model yields the highest performance of all three, as shown in Table 2. This result is expected, since DeBERTa is an improved version of BERT, both in performance and in efficiency, and it also is designed to focus more on semantics and the position of tokens, because of its different approach to embeddings.

Next, we will subject all three models, plus some more, on a series of more tests, and we will see if DeBERTa still retains its performance against our other methods and also against some state of the art methods.

4.6. **Comparison with other methods.** We now make use of the testing dataset that we've compiled to test our models versus some state of the art models presented in the second section. We use different sections of the dataset. Considering which model has generated the text is crucial when evaluating performance, as it allows for an assessment of whether the detection is effective against state-of-the-art generative models or if its capabilities are limited to identifying text generated by older, less advanced models.

Model	Accuracy	Precision	Recall	F1 Score
RADAR	**0.97**	**0.97**	**0.97**	**0.97**
Our LSTM model	0.76	0.78	0.73	0.76
Our BERT model	0.88	0.90	0.88	0.88
Our DeBERTa model	0.93	0.93	0.93	0.93

Table 3. Comparison between results from RADAR vs our models, on a sample of 286 student essays and 286 essays generated by gpt-3.5-turbo.

The AI-generated texts from the dataset for which the models yielded the results presented in Table 3 is different from the one in Table 2, though both are generated by gpt3.5-turbo.

Model	Accuracy	Precision	Recall	F1 Score
RADAR	0.88	0.89	0.88	0.88
Our LSTM model	0.68	0.75	0.53	0.62
Our BERT model	**0.93**	**0.93**	**0.92**	**0.92**
Our DeBERTa model	0.84	0.86	0.84	0.83

Table 4. Comparison between results from RADAR vs our models, on a sample of 94 student and AI essays generated by gpt-4-turbo

This time, in Table 4, the BERT model stands on top, overtaking even RADAR in gpt-4-turbo texts detection.

Again, our BERT model seems to classify texts from OpenAI's latest model, gpt-4-o very well, even slightly surpassing RADAR, as displayed in Table 5.

Model	Accuracy	Precision	Recall	F1 Score
RADAR	0.88	0.89	0.88	0.88
Our LSTM model	0.70	0.73	0.63	0.67
Our BERT model	**0.89**	**0.90**	**0.89**	**0.89**
Our DeBERTa model	0.73	0.79	0.73	0.71

Table 5. Comparison between results from RADAR vs our models, on a sample of 253 student essays and 253 essays generated by gpt-4o

As mentioned in the datasets section, we have used some samples from SeqXGPT's datasets for training, but we have also left some for testing. This time, we will compare our results with the one from the table for document level detection, from SeqXGPT's paper [30], since they have already tested on the same dataset. We will test our models and also the RADAR model on the sections left for testing from this dataset (200 GPT-J texts, 200 GPT-Neo texts and 200 LLaMA texts). Since we have used the human-written texts from this dataset for training the detection models, we cannot fairly compute precision, but we will compare the recall values, since recall is a function of true positives and false negatives and it only deals with truly AI generated texts. True positives are the texts correctly classified as AI-generated, and false negatives are texts incorrectly classified as human written.s

Model	GPT-J	GPT-Neo	LLaMA
Sniffer	0.74	0.83	0.07
Sent-RoBERTa	0.21	0.46	0.10
Seq-RoBERTa	0.26	0.40	0.72
SeqXGPT	**0.96**	**0.99**	**0.90**
RADAR	0.31	0.26	0.23
Our LSTM model	0.27	0.33	0.30
Our BERT model	0.95	0.97	0.89
Our DeBERTa model	0.73	0.81	0.67

Table 6. Comparison between recall values from models compared in SeqXGPT's paper, RADAR and our models, on three of the document-level datasets from SeqXGPT's testing sets.

RADAR and our LSTM seem to be performing particularly poorly on this specific dataset. SeqXGPT though has outstanding performance when compared to all other models in this case. Our bert model comes very close to SeqXGPT's performance, falling behind with only 1% accuracy when it comes to classifying the texts from these datasets as AI generated.

Next up, we compare Fast-DetectGPT [1] with our models, on a section of 80 texts from our IELTS student and gpt-3.5-turbo test dataset. We are constrained to reduce the size of the test dataset for this particular experiment due to the very heavy workload Fast-DetectGPT demands during execution. Fast-DetectGPT yields really good results, with an impressive perfect recall, meaning it correctly guessed all the AI generated texts, as can be seen in Table 7. Our DeBERTa model comes really close, followed by our BERT, and then the LSTM.

Model	Accuracy	Precision	Recall	F1 Score
Fast-DetectGPT	**0.97**	0.95	**1**	**0.97**
Our LSTM model	0.74	0.76	0.70	0.73
Our BERT model	0.89	0.91	0.89	0.88
Our DeBERTa model	0.96	**0.96**	0.96	0.96

Table 7. Comparison between results from Fast-DetectGPT vs our models, on a sample of 80 texts from our IELTS student and gpt-3.5-turbo dataset.

5. Conclusions and Future Work

To conclude, we have focused on developing a tool that aims to diminish academic dishonesty caused by the use of large language models. This dishonesty is not caused by many educationally appropriate use cases of generative pre-trained transformers, such as researching, searching for ideas, finding answers to problems in order to learn solving methods or even receiving feedback for one's own work. However, a problem could arise when students claim entire AI works or very big chunks of generated content as being their own. This is where our tool proves to be useful. We have trained multiple models with multiple architectures, on various datasets, to find, to the best of our ability, the best configuration for creating a tool specialized to detect essays, documents or stories generated by AI. Specifically, we have created an LSTM model, a BERT and a DeBERTa model, which are all lightweight, free to use and open source, so that they can all be run locally on any user's personal computer. Based on the comparison in the previous section, it is hard to pick a winner between BERT and DeBERTa, since they both perform the best between the 3 models developed by us in 3 out of 6 experiments. However, BERT surpasses a state of the art model, RADAR in all metrics in two of our experiments, whereas DeBERTa only manages to achieve a slightly better precision than Fast-DetectGPT, and, therefore, we will declare the BERT model our best.

With some future improvements, these models could become part widely-used tools in schools and universities all around the world. They could benefit from an even larger and more diverse dataset to be trained on, which would require much more computational power, but would also yield much better results. Experimenting with many different other transformer based models and different hyperparameters definitely brings potential for achieving a much higher accuracy. Another potentially big improvement would be creating custom embeddings for specializing models in particular detection applications, meaning detecting generated text for each school subject in particular. We would have, for example, a model specially designed to detect biology essays, another for history, and so on. By using our custom embeddings instead of pre-trained ones, we could much easier train a transformer for subject-specific tasks.

References

[1] Bao, G., Zhao, Y., Teng, Z., Yang, L., and Zhang, Y. Fast-detectgpt: Efficient zero-shot detection of machine-generated text via conditional probability curvature, 2024.

[2] Biewald, L. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

[3] Center, P. R. About 1 in 5 u.s. teens who've heard of chatgpt have used it for schoolwork, November 2023.

[4] Cheplukov, A. Raw ielts essays, 2024. Retrieved May 10, 2024 from `https://www.kaggle.com/datasets/arsenycheplukov/raw-ielts-essays`.

[5] Cipriano, B. P., and Alves, P. "chatgpt is here to help, not to replace anybody" – an evaluation of students' opinions on integrating chatgpt in cs courses, 2024.

[6] Crossley, S. A., Baffour, P., Tian, Y., Picou, A., Benner, M., and Boser, U. The persuasive essays for rating, selecting, and understanding argumentative and discourse elements (persuade) corpus 1.0. *Assessing Writing 54* (2022). https://doi.org/10.1016/j.asw.2022.100667.

[7] Demir, E. Daigt gemini-pro 8.5k essays, 2023. Retrieved May 10, 2024 from `https://www.kaggle.com/datasets/datafan07/daigt-gemini-pro-8-5k-essays`.

[8] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[9] Face, H. Natural language processing course, chapter 6. `https://huggingface.co/learn/nlp-course/en/chapter6/6`, 2023.

[10] Gallego, V. Alpaca-gpt4 dataset, 2023. Retrieved May 10, 2024 from `https://huggingface.co/datasets/vicgalle/alpaca-gpt4`.

[11] Gerami, S. Ai vs human text, 2024. Retrieved May 10, 2024 from `https://www.kaggle.com/datasets/shanegerami/ai-vs-human-text`.

[12] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial networks, 2014.

[13] He, P., Liu, X., Gao, J., and Chen, W. Deberta: Decoding-enhanced bert with disentangled attention, 2021.

[14] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780.

[15] Hu, X., Chen, P.-Y., and Ho, T.-Y. Radar: Robust ai-text detection via adversarial learning, 2023.

[16] Ibrahim, M. Ielts writing scored essays dataset, 2023. Retrieved May 10, 2024 from `https://www.kaggle.com/datasets/mazlumi/ielts-writing-scored-essays-dataset`.

[17] Kłeczek, D. Daigt-v4-train-dataset, 2024. Retrieved May 10, 2024 from `https://www.kaggle.com/datasets/thedrcat/daigt-v4-train-dataset/data`.

[18] Li, L., Wang, P., Ren, K., Sun, T., and Qiu, X. Origin tracing and detecting of llms, 2023.

[19] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach, 2019.

[20] Liu, Y., Zhang, Z., Zhang, W., Yue, S., Zhao, X., Cheng, X., Zhang, Y., and Hu, H. Argugpt: evaluating, understanding and identifying argumentative essays generated by gpt models, 2023.

[21] Mitchell, E., Lee, Y., Khazatsky, A., Manning, C. D., and Finn, C. Detectgpt: Zero-shot machine-generated text detection using probability curvature, 2023.

[22] OpenAI. Gpt-4. `https://openai.com/index/gpt-4-research/`, 2023.

[23] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library, 2019.

[24] Pathi, L. N. Imdb dataset of 50k movie reviews, 2019. Retrieved May 10, 2024 from `https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews`.

[25] 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., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[26] Peng, B., Li, C., He, P., Galley, M., and Gao, J. Instruction tuning with gpt-4, 2023.

[27] Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. Improving language understanding with unsupervised learning. *OpenAI Blog* (June 2018).

[28] Solaiman, I., Brundage, M., Clark, J., Askell, A., Herbert-Voss, A., Wu, J., Radford, A., Krueger, G., Kim, J. W., Kreps, S., McCain, M., Newhouse, A., Blazakis, J., McGuffie, K., and Wang, J. Release strategies and the social impacts of language models, 2019.

[29] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need, 2023.
[30] Wang, P., Li, L., Ren, K., Jiang, B., Zhang, D., and Qiu, X. Seqxgpt: Sentence-level ai-generated text detection, 2023.

Babeș-Bolyai University, Faculty of Mathematics and Computer Science, 1 Mihail Kogălniceanu, Cluj-Napoca 400084, Romania
Email address: `david.biris1@stud.ubbcluj.ro`