

Embedding Techniques to Distinguish Chatbot from Human

CS 271 Sec 02 Machine Learning Project

Presented to

Dr. Mark Stamp

Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Class

CS 271 Sec 02

By

Christofer Washington Berruz Chungata and Jakob Kauffmann

December 2024

TABLE OF CONTENTS

Table of Contents.....	2
I. Introduction.....	1
A. Background.....	1
B. Goal.....	1
C. Dataset and Setup.....	2
II. Methodology.....	4
A. Methodology Overview.....	4
B. Word Embeddings.....	4
C. PCA Analysis and Dimensionality Reduction for Text Embeddings.....	6
D. Architectures.....	8
E. Adapting Embeddings to Images.....	11
F. Tracking of Experiment Metrics.....	15
G. Experimentation workflow.....	15
III. Results.....	17
IV. Conclusion.....	23
V. Future Work.....	24
References.....	25

I. INTRODUCTION

A. Background

As Artificial Intelligence (AI) continues to advance, large language models (LLMs) like OpenAI's GPT series and Meta's LLaMA have achieved remarkable success in producing coherent and human-like text. This capability has opened new possibilities across various fields, but it also raises significant concerns, such as detecting and differentiating between human-written and machine-generated text. The ability to identify chatbot-generated content is becoming increasingly important for ethical considerations, content verification, and mitigating misuse in areas like education, content moderation, and cybersecurity.

Previous work by Godghase et al. [1] explored methods to distinguish text origins using advanced feature analysis and embeddings. While their research focused on classifying text generated by GPT, the rapid evolution of AI models and the introduction of new systems like LLaMA warrant further investigation. The diversity in generative model architectures and their varying linguistic patterns require refined detection approaches that adapt to this evolving landscape. By building on prior research, this project incorporates additional models and explores a simplified yet robust approach to text classification.

B. Goal

The primary goal of this project is to extend Godghase's framework by introducing LLaMA 3.2, a state-of-the-art LLM developed by Meta, as an additional text generator for comparison. Instead of relying on feature analysis, we simplify the classification pipeline by embedding entire sentences to capture semantic nuances directly. This embedding-based approach is designed to streamline processing while maintaining accuracy.

Our experiments focus on classifying the text's origin in three contexts: Human vs. GPT, Human vs. LLaMA, and Human vs. Mixed Chatbot (GPT + LLaMA). By evaluating classification accuracy across these scenarios, we aim to gain insight into the generalizability of embedding-based techniques for AI detection.

C. Dataset and Setup

The dataset in [1] consists of approximately 1.4 million text entries evenly split into human and GPT classes. The human-written texts, 700,000 in total, originate from a WikiHow dataset used in prior research by Koupaei and Wang [2]. The GPT entries were generated by querying the GPT model using structured prompts based on the title, length, headline, and section label fields in the WikiHow dataset using the structure in Figure 1 to form queries.

I am writing an article titled {title} for a WikiHow page. Write a paragraph of length {length} whose title is {headline} for the {sectionLabel} section of this article.

Figure 1: ChatGPT prompt [1]

While we utilized the dataset from [1], Godghase et al. did not provide the query dataset they used to generate the GPT text. As a result, we had to recreate these queries, which we did using the same dataset from [2] and the same structure as in Figure 1. We generated the LLaMA responses by running LLaMA 3.2-3b locally. To do this, we used the Ollama framework and downloaded the open-source model from Hugging Face.

To make the dataset manageable, we worked with a subset of 100,000 entries from Godghase et al.'s dataset and generated an additional 50,000 articles using LLaMA 3.2-3b. These entries were combined into three datasets tailored to our experimental needs:

1. HumanGPT.csv – Contains 100,000 text entries, evenly split between human and GPT articles.

2. HumanLlama.csv – Contains 100,000 text entries, evenly split between human and LLaMA articles.
3. Combined.csv – Contains 150,000 text entries, evenly distributed across three classes: human, GPT, and LLaMA.

Additionally, we balanced these datasets in word length to ensure that the models weren't learning to distinguish on length. Figure 2 is a histogram of the lengths for each class:

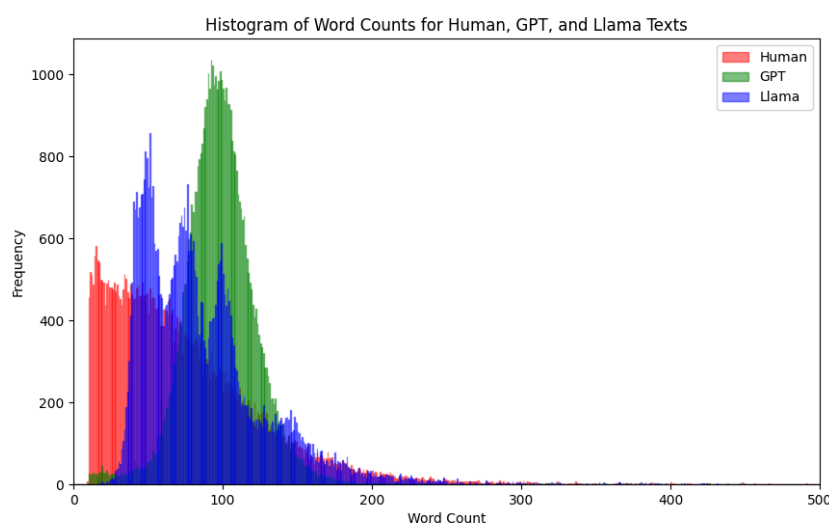


Figure 2: Histogram of Word Counts for Human, GPT, and Llama Texts

These datasets allowed us to perform comprehensive comparisons, exploring the effectiveness of embedding-based classification techniques for distinguishing between human-written text and text generated by different LLMs, individually and in a mixed setting. Furthermore, the structured datasets enabled us to experiment with various classifiers, including AdaBoost, a simple Multilayer Perceptron (MLP), and Convolutional Neural Networks (CNN) applied to 2D image representations of the embeddings, providing additional insights into the performance of these models in identifying text origins.

II. METHODOLOGY

A. Methodology Overview

This project aims to compare and contrast how different machine learning models perform at distinguishing human-written text from chatbot-written text. All these machine learning models solve a classification problem. Plain text is not directly fed into these models. Instead, we use different word embeddings. Figure 3 illustrates the word embeddings and architectures used for the experiments.

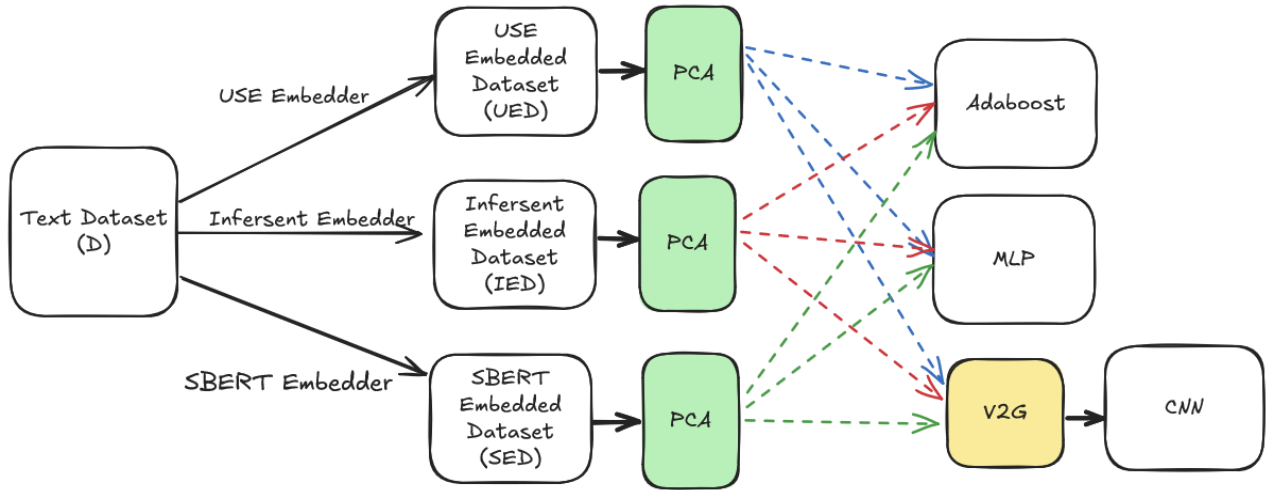


Figure 3: Methodology Overview

Note that for any text dataset, we perform 9 experiments.

B. Word Embeddings

We used three pre-trained sentence embedding techniques to create numerical embeddings of the raw text data: InferSent, Sentence-BERT (SBERT), and Universal Sentence Embedder (USE). These methods were selected for their demonstrated ability to capture sentence-level semantics and their broad application in text classification tasks.

1. *InferSent*: InferSent, developed by Meta and pre-trained on the Stanford Natural Language Inference (SNLI) dataset, generates 4096-dimensional embeddings using a max-pooling operation over BiLSTM hidden states. For this study, we used the FastText-based version (V2), which leverages subword information through character-level n-grams, offering better handling of rare or out-of-vocabulary words compared to the GloVe-based version (V1). The model was initialized with a pre-trained checkpoint (``infsent2.pkl`` [3]), 300-dimensional word vectors from the FastText Common Crawl embeddings (``crawl-300d-2M.vec`` [3]), and a vocabulary size of 100,000 words to balance coverage and computational efficiency. Max pooling was chosen over mean pooling to highlight the most prominent features in the embeddings, preserving critical stylistic and linguistic markers important for distinguishing between human and machine-generated text. No fine-tuning was applied, ensuring the embeddings maintained their original robust pre-trained properties for text analysis.
2. *Sentence-level Bidirectional Encoder Representations from Transformers (SBERT)*: SBERT, an extension of the BERT architecture fine-tuned for sentence-level tasks, is highly effective at capturing contextual nuances in text through its Siamese network structure. For our experiments, we employed the all-MiniLM-L6-v2 model, which generates 384-dimensional embeddings optimized for semantic similarity tasks. The model was sourced from the Hugging Face repository [4] and used without fine-tuning to ensure consistency with other embedding methods. To create a compact yet contextually rich representation of each article, SBERT embeddings were produced by averaging the sentence-level embeddings of all sentences within the article.

3. *Universal Sentence Encoder (USE)*: The Universal Sentence Encoder (USE), developed by Google, produces 512-dimensional embeddings using either a deep averaging network (DAN) or a Transformer-based architecture. For this study, we utilized the Transformer-based version from TensorFlow Hub [5], recognized for its strong contextual representations and versatility in semantic tasks. Embeddings were generated by processing each article as a single text input, yielding a unified representation that encapsulates the text’s overall meaning and structure. Consistent with our approach for InferSent and SBERT, USE was employed without fine-tuning to preserve its general-purpose capabilities and ensure methodological uniformity across experiments.

These embedding methods offer diverse and complementary text representations, each tailored to the classification task. InferSent, utilizing FastText word vectors and max-pooling, excels at handling rare words while highlighting key features; SBERT provides compact, contextually rich embeddings optimized for semantic similarity; and USE combines efficiency and robust contextual understanding with its general-purpose Transformer-based architecture. By employing these pre-trained models without fine-tuning, we ensured methodological consistency and reproducibility, establishing a solid foundation for assessing our classification models’ effectiveness in distinguishing between human and machine-generated text.

C. PCA Analysis and Dimensionality Reduction for Text Embeddings

To prepare text embeddings for efficient model training, Principal Component Analysis (PCA) was applied to reduce the dimensionality of the InferSent (IED), USE (UED), and SBERT (SED) embeddings. PCA was chosen to retain the majority of the variance while reducing computational costs and mitigating the risk of overfitting. The optimal number of components

for each embedding type was determined by analyzing cumulative explained variance, balancing computational efficiency with representation quality. Figure 4 is a sample PCA variance graph.

1. *InferSent*: The InferSent embeddings, originally 4096-dimensional, underwent PCA analysis. The cumulative explained variance plots for both human and GPT-generated text showed that approximately 200 components captured most of the variance. Based on these findings, the IED embeddings were reduced to 200 dimensions, significantly lowering computational overhead while preserving critical information for classification tasks.
2. *USE*: The USE embeddings, with an initial dimensionality of 512, were analyzed using PCA. The analysis revealed that retaining 250 components was sufficient to preserve nearly all the variance in the embeddings for both human and GPT text. This dimensionality struck an ideal balance between preserving information and reducing computational complexity, ensuring the robustness of the embeddings for downstream tasks.
3. *SBERT*: The SBERT embeddings, starting at 384 dimensions, required a relatively modest reduction. PCA analysis indicated that 250 components captured the vast majority of the variance for both human and GPT-generated text. This reduction ensured the embeddings maintained their contextual richness while improving computational efficiency.

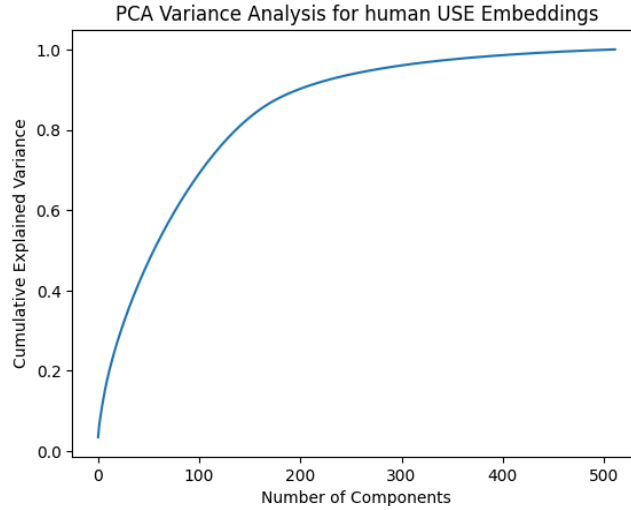


Figure 4: Example PCA Variance Analysis Graph

After determining the target dimensionality for each embedding type, PCA was applied to transform the embeddings, which were then saved in new datasets tailored to each classification task (e.g., human vs. GPT or human vs. LLaMA). This methodology ensured consistency and scalability across all experiments, providing embeddings that were both computationally efficient and representative for machine learning model training. Overall, the embeddings with reduced dimensions and retained 95-99% of the variance in the original set of embeddings.

D. Architectures

Three machine learning techniques were used: Adaboost, a simple Multilayer Perceptron (MLP), and a Convolutional Neural Network (CNN). The input size of each network was dynamic, depending on the embedding or image size. All neural networks used CrossEntropy as a loss function. Note that due to the size of the experiments, hyper-parameter tuning was limited. Furthermore, the same architecture was used for all experiments.

Adaboost

For all embeddings, we used Adaboost with 30 decision trees.

CNN

Two convolutional layers followed by a fully connected layer. Softmax was used after the output of the neural network given the classification task. The number of neurons in the last layer was variable depending on the number of classes to distinguish.

The first convolution layer has:

- 3, 3x3 kernels with a stride of 1. Sigmoid as activation function.
- MaxPooling layer with 2x2 kernels and stride = 2.
- Dropout with $p = 0.3$.

The second convolutional layer has:

- 6, 5x5 kernels with a stride of 1. Sigmoid as activation function.
- MaxPooling layer with 2x2 kernels and stride = 2.
- Dropout with $p = 0.3$

The fully connected layer has:

- 1 hidden layer with 32 neurons. Sigmoid as activation function.

Note that RELU was initially tested for the CNN, but it performed poorly compared to Sigmoid.

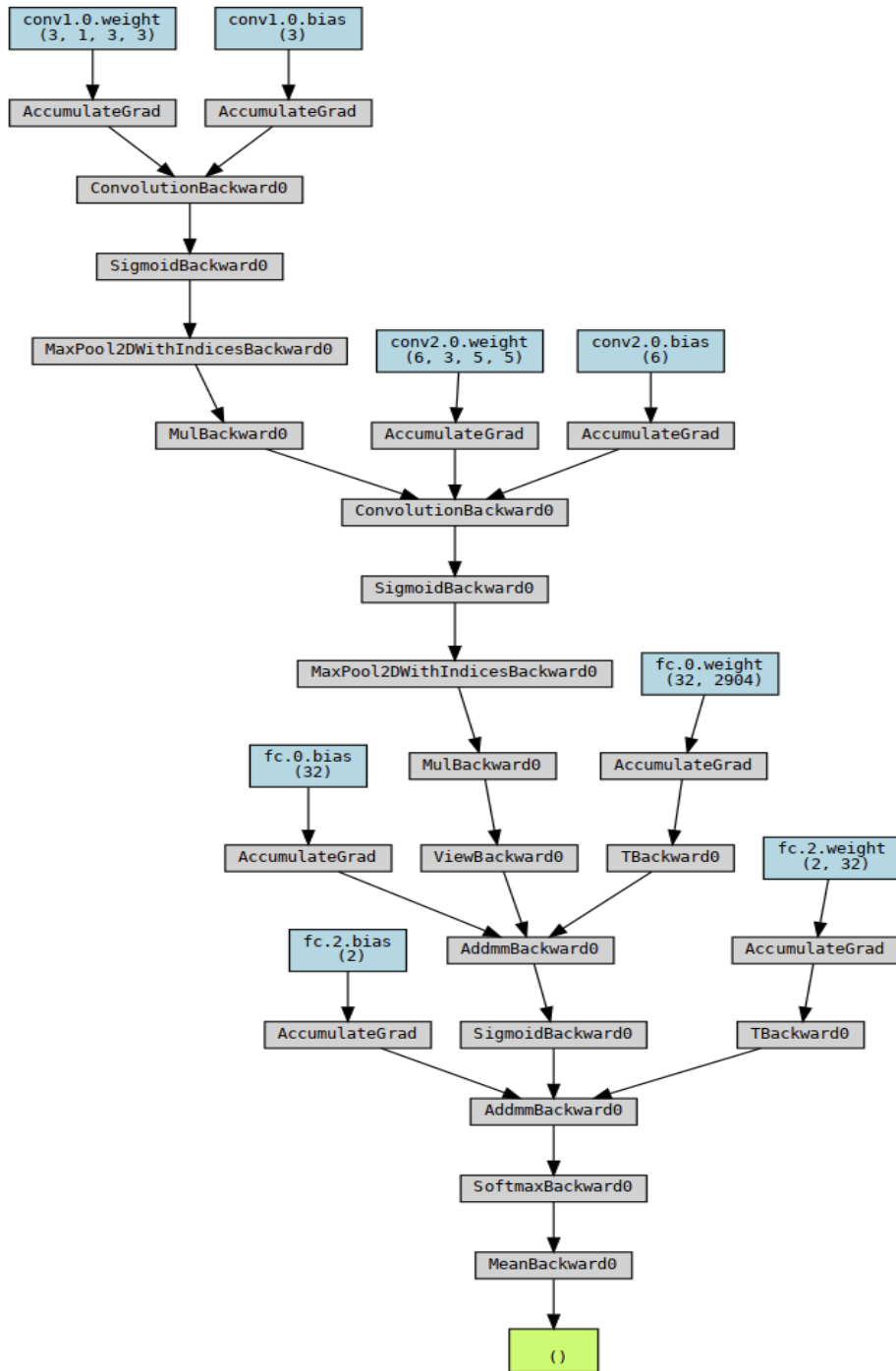


Figure 5. Computational graph of the CNN. Made using torchviz.

Simple Multi-Layer Perceptron

An MLP with one hidden layer of 128 neurons followed by the output layer with M neurons, depending on the number of classes. The SimpleMLP uses ReLU as an activation function in the hidden layer, and it applies Softmax to the output of the network.

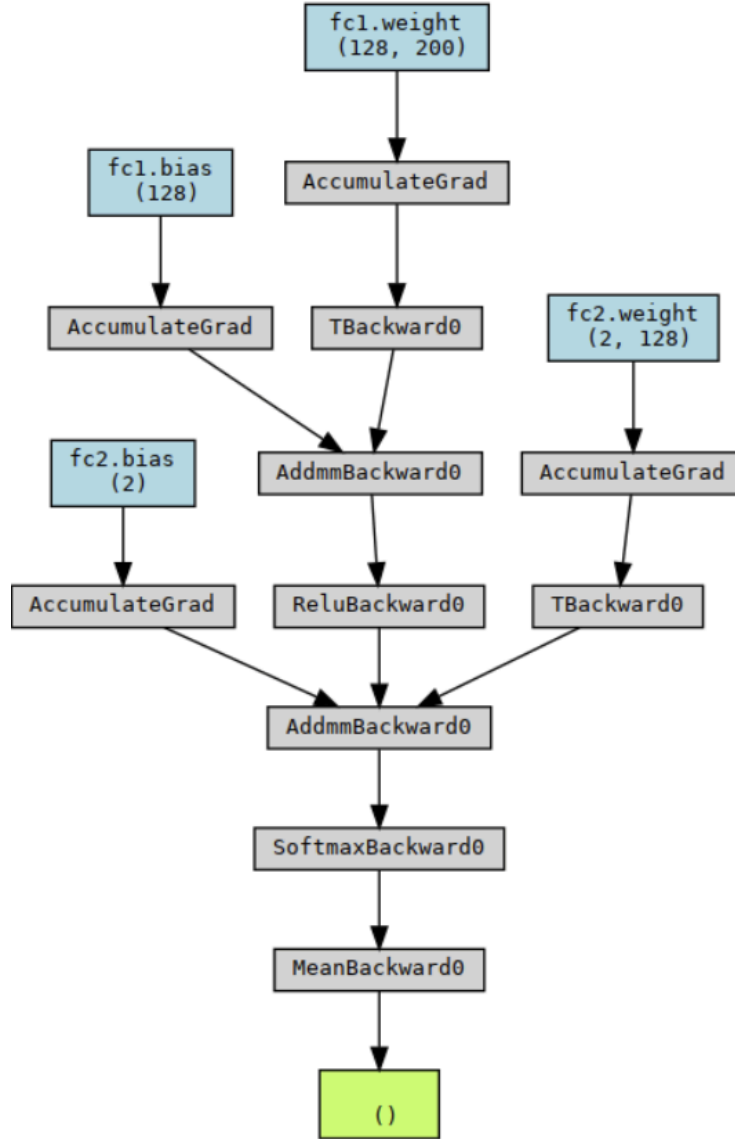


Figure 6. Computational graph of the simple MLP. Made using torchviz.

E. Adapting Embeddings to Images

To generate an image from an embedding vector V , we first normalized the vector as follows:

$$V' = \frac{V - \min(V)}{\max(V) - \min(V)}$$

As a result, all features are between $[0, 1]$. Assume the embedding vector is a row vector of dimensions $(1, n)$. To create a grayscale image we can compute

$$I = 255 V'^T V'$$

The resulting image has dimension (n, n) . We scale up the pixels from $[0, 1]$ to $[0, 255]$ by multiplying with 255, which is the desired range for a grayscale image.

The intuition behind this image generation technique, which we call Vector-to-Grayscale (V2G), is that it encodes the relationship between all features. Unlike a simple reshape, no locality is enforced on the image, allowing a CNN to discover important localities through training.

The following tables contain examples of the same human, GPT, and LLaMA text sample expressed through different embedding techniques under the V2G transformation.

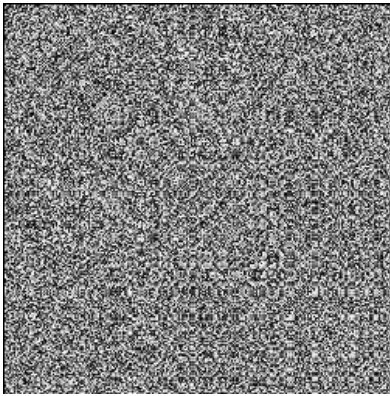
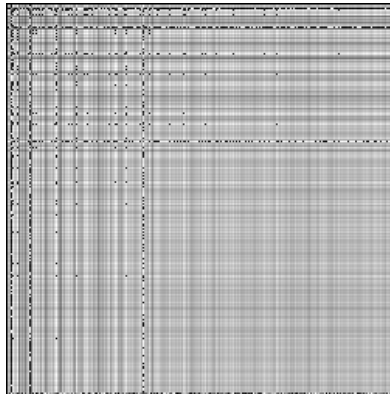
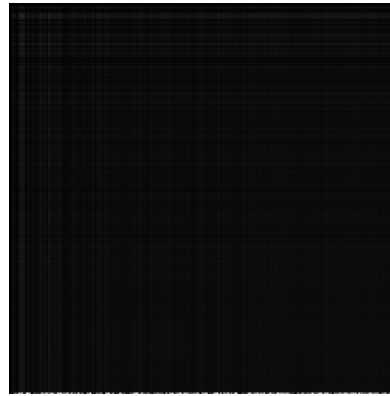
		
Human	GPT	LLaMA

Table 1. Examples of Human, GPT, and LLaMA SBert embeddings under V2G

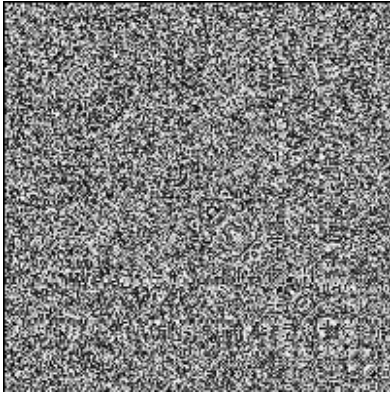
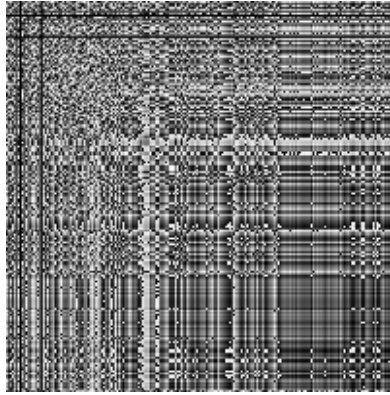
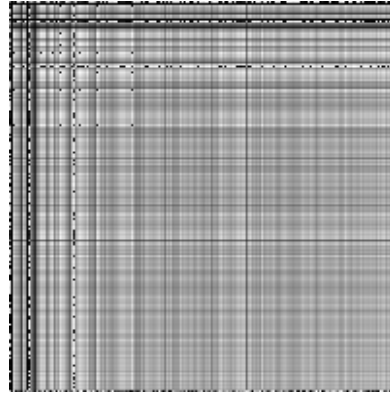
		
Human	GPT	LLaMA

Table 2. Examples of Human, GPT, and LLaMA Infsent embeddings under V2G

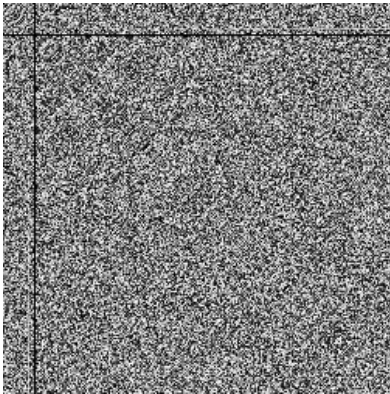
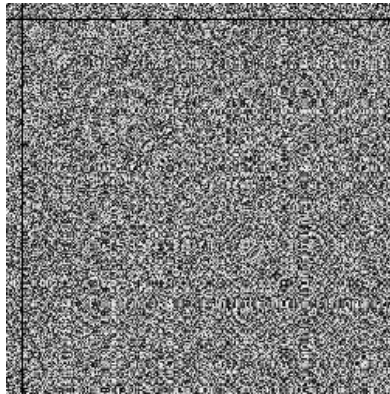
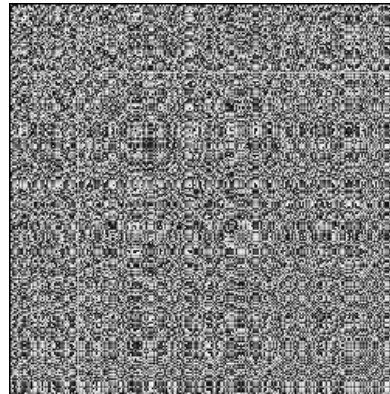
		
Human	GPT	LLaMA

Table 3. Examples of Human, GPT, and LLaMA USE embeddings under V2G.

From the grayscale images, it is clear that there is a stark difference between Human, GPT, and LLaMA text samples across all embedding techniques.

However, the simplicity of the V2G techniques comes at the expense of image size. For an embedding vector of size N , the image has dimensions (N, N) . For large embedding vectors, training a CNN becomes computationally expensive for two reasons. First, the memory required

to hold these images on RAM forces training to load the images from the disk. Even on Solid State Disks (SSDs), doing I/O is significantly slower than using images loaded on memory. Second, the number of convolution operations required to pass this image through the CNN increases with image size.

As a result, during the training of the CNN, we used **resized** images. We resized all images to be of dimensions 100x100 using bilinear interpolation, regardless of the embedding feature space. The following are examples of the same V2G images from Table [1-3], resized.

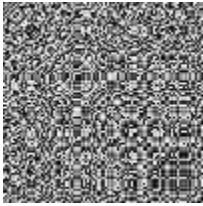
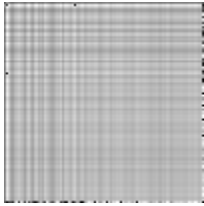

		
Human	GPT	LLaMA

Table 4. Examples of Human, GPT, and LLaMA SBERT embeddings under V2G resized to 100x100 using Bilinear interpolation.

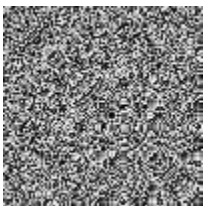

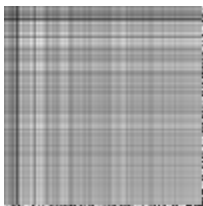
		
Human	GPT	LLaMA

Table 5. Examples of Human, GPT, and LLaMA Infsent embeddings under V2G resized to 100x100 using Bilinear interpolation.

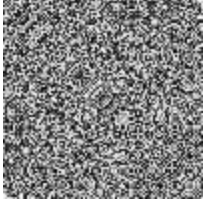
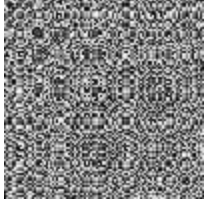

		
Human	GPT	LLaMA

Table 6. Examples of Human, GPT, and LLaMA USE embeddings under V2G resized to 100x100 using Bilinear interpolation.

F. Tracking of Experiment Metrics

For all neural networks, we tracked the following per epoch:

- Training and testing loss
- Training and testing accuracy
- Classification results. For each predicted class i , we tracked how many elements from true class j were classified as i . We used this information to generate confusion matrices and F-1 scores during the analysis of results phase.

For AdaBoost, we tracked:

- Training and testing accuracy
- Classification results.

Given the number of experiments, all neural networks were trained for only 15 epochs. No cross-validation was done as our datasets were balanced.

G. Experimentation workflow

The following workflow summarizes how we generated all the metrics specified above.

1. For each dataset in (combined, humanLlama, humanGPT) do:
 - a. Embed using InferSent
 - b. Embed using SBERT

- c. Embed using USE
2. For each embedded dataset do:
 - a. Apply PCA as dimensionality reduction
3. For each reduced embedded dataset do:
 - a. Generate V2G images with a bilinear interpolation resize to 100x100
4. For all reduced embedded datasets do
 - a. Apply an 80-20 split using a random seed; we used seed = 42.
 - b. Train Adaboost with 30 decision trees
5. For all reduced embedded datasets do
 - a. Apply an 80-20 split using a random seed; we used seed = 42.
 - b. Train an MLP for 15 epochs with Adam optimizer and a learning rate of 0.001
6. For all V2G datasets do
 - a. Apply an 80-20 split using a random seed; we used seed = 42.
 - b. Train a CNN for 15 epochs with Adam optimizer and a learning rate of 0.001
7. Compile all metrics and generate graphs.

III. RESULTS

This section presents a detailed analysis of the evaluation outcomes for Adaboost, CNN, and Simple MLP models, specifically aimed at distinguishing between human-generated and chatbot-generated content, including GPT-3.5 and LLaMA 3.2 outputs. The models were assessed using critical metrics such as accuracy, precision, recall, and F1 score, providing a comprehensive understanding of their respective strengths and weaknesses in various classification settings.

A. Model Analysis

1) *Adaboost*: The Adaboost model demonstrated reasonable effectiveness in binary classification tasks, such as distinguishing between human-generated content and either GPT or LLaMA outputs. However, its performance declined significantly in multiclass classification scenarios involving Human vs. GPT vs. LLaMA, exhibiting a notably high rate of false positives. When GPT and LLaMA were combined into a single chatbot class, the performance of Adaboost improved markedly, indicating its limitations in handling finer distinctions between multiple chatbot sources. The heatmap table below (Table 7) illustrates Adaboost's performance metrics compared to the other models.

2) *CNN*: The CNN model exhibited robust performance across both binary and multiclass classification tasks. It achieved high precision, recall, and accuracy, consistently exceeding 0.999 in simpler binary classification tasks. However, its ability to distinguish between GPT and LLaMA outputs in multiclass settings presented challenges, as reflected by a slight decline in precision. False positives were most prevalent when distinguishing between GPT and LLaMA,

suggesting an overlap in the learned features between these chatbot-generated contents. In the binary classification context, CNN performed exceptionally well, achieving near-perfect recall despite minor occurrences of false positives. The heatmap table below (Table 7) illustrates CNN's performance metrics compared to other models across the different embeddings. As shown, combining GPT and LLaMA into a single chatbot class led to a significant improvement in CNN's performance metric. The learning curves (Figure 7) demonstrate rapid convergence with minimal overfitting, underscoring the model's efficient training dynamics.

3) Simple MLP: The Simple MLP model achieved perfect classification performance across all tasks. Regardless of the scenario, the model consistently attained precision, recall, and F1 scores of 1.0, indicating its ability to flawlessly distinguish between human and chatbot-generated content. No errors were recorded across any of the tasks, further emphasizing the efficacy of the model's learning capabilities. Table 7 displays the performance metrics indicative of the Simple MLP's perfect classification performance across all subsets and embedding techniques. The learning curves for Simple MLP (Figure 7) also indicate rapid convergence with no sign of overfitting, demonstrating both robustness and generalizability.

4) Comparative Analysis: Among the models evaluated, Simple MLP emerged as the most effective, achieving perfect classification metrics across all tasks and scenarios. CNN also demonstrated strong performance, particularly in binary classification tasks, but it struggled to differentiate between GPT and LLaMA in the more complex multiclass setting. Adaboost, while effective in simpler binary scenarios, faced significant challenges with multiclass tasks due to its higher rate of false positives.

	Subset	Class	Model											
			Adaboost				CNN				Simple MLP			
			Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score
InferSent	Human vs. Combined Chatbot	Chatbot	0.823	0.868	0.863	0.866	0.999	0.999	1.000	0.999	1.000	1.000	1.000	1.000
		Human	0.823	0.734	0.742	0.738	0.999	0.999	0.998	0.999	1.000	1.000	1.000	1.000
		Overall	0.823	0.801	0.803	0.802	0.999	0.999	0.999	0.999	1.000	1.000	1.000	1.000
	Human vs. GPT	GPT	0.865	0.867	0.865	0.866	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
		Human	0.865	0.863	0.865	0.864	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
		Overall	0.865	0.865	0.865	0.865	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	Human vs. GPT vs. LLaMA	GPT	0.605	0.592	0.588	0.590	0.965	0.948	0.945	0.947	1.000	1.000	1.000	1.000
		Human	0.605	0.734	0.742	0.738	0.965	0.999	0.998	0.999	1.000	1.000	1.000	1.000
		Llama	0.605	0.487	0.484	0.486	0.965	0.947	0.950	0.949	1.000	1.000	1.000	1.000
		Overall	0.605	0.604	0.605	0.605	0.965	0.965	0.965	0.965	1.000	1.000	1.000	1.000
	Human vs. Llama	Human	0.783	0.782	0.782	0.782	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
		Llama	0.783	0.785	0.785	0.785	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
		Overall	0.783	0.783	0.783	0.783	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
SBERT	Human vs. Combined Chatbot	Chatbot	0.866	0.880	0.924	0.901	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
		Human	0.866	0.834	0.752	0.791	1.000	1.000	0.999	1.000	1.000	1.000	1.000	1.000
		Overall	0.866	0.857	0.838	0.846	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	Human vs. GPT	GPT	0.895	0.877	0.920	0.898	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
		Human	0.895	0.914	0.869	0.891	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
		Overall	0.895	0.896	0.894	0.894	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	Human vs. GPT vs. LLaMA	GPT	0.643	0.601	0.605	0.603	0.894	0.838	0.840	0.839	1.000	1.000	1.000	1.000
		Human	0.643	0.834	0.752	0.791	0.894	1.000	0.999	1.000	1.000	1.000	1.000	1.000
		Llama	0.643	0.522	0.570	0.545	0.894	0.842	0.841	0.841	1.000	1.000	1.000	1.000
		Overall	0.643	0.652	0.642	0.646	0.894	0.893	0.893	0.893	1.000	1.000	1.000	1.000
	Human vs. Llama	Human	0.821	0.823	0.814	0.819	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
		Llama	0.821	0.819	0.828	0.823	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
		Overall	0.821	0.821	0.821	0.821	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
USE	Human vs. Combined Chatbot	Chatbot	0.800	0.836	0.870	0.852	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
		Human	0.800	0.721	0.663	0.691	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
		Overall	0.800	0.778	0.766	0.772	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	Human vs. GPT	GPT	0.895	0.903	0.887	0.895	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
		Human	0.895	0.887	0.903	0.895	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
		Overall	0.895	0.895	0.895	0.895	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	Human vs. GPT vs. LLaMA	GPT	0.597	0.623	0.600	0.611	0.999	0.997	1.000	0.999	1.000	1.000	1.000	1.000
		Human	0.597	0.721	0.663	0.691	0.999	1.000	1.000	1.000	1.000	1.000	1.000	1.000
		Llama	0.597	0.471	0.526	0.497	0.999	1.000	0.997	0.999	1.000	1.000	1.000	1.000
		Overall	0.597	0.605	0.596	0.600	0.999	0.999	0.999	0.999	1.000	1.000	1.000	1.000
	Human vs. Llama	Human	0.811	0.809	0.810	0.810	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
		Llama	0.811	0.813	0.812	0.812	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
		Overall	0.811	0.811	0.811	0.811	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000

Figure 7: Heatmap Table of Performance Metrics by Model, Embedding, Subset, and Class

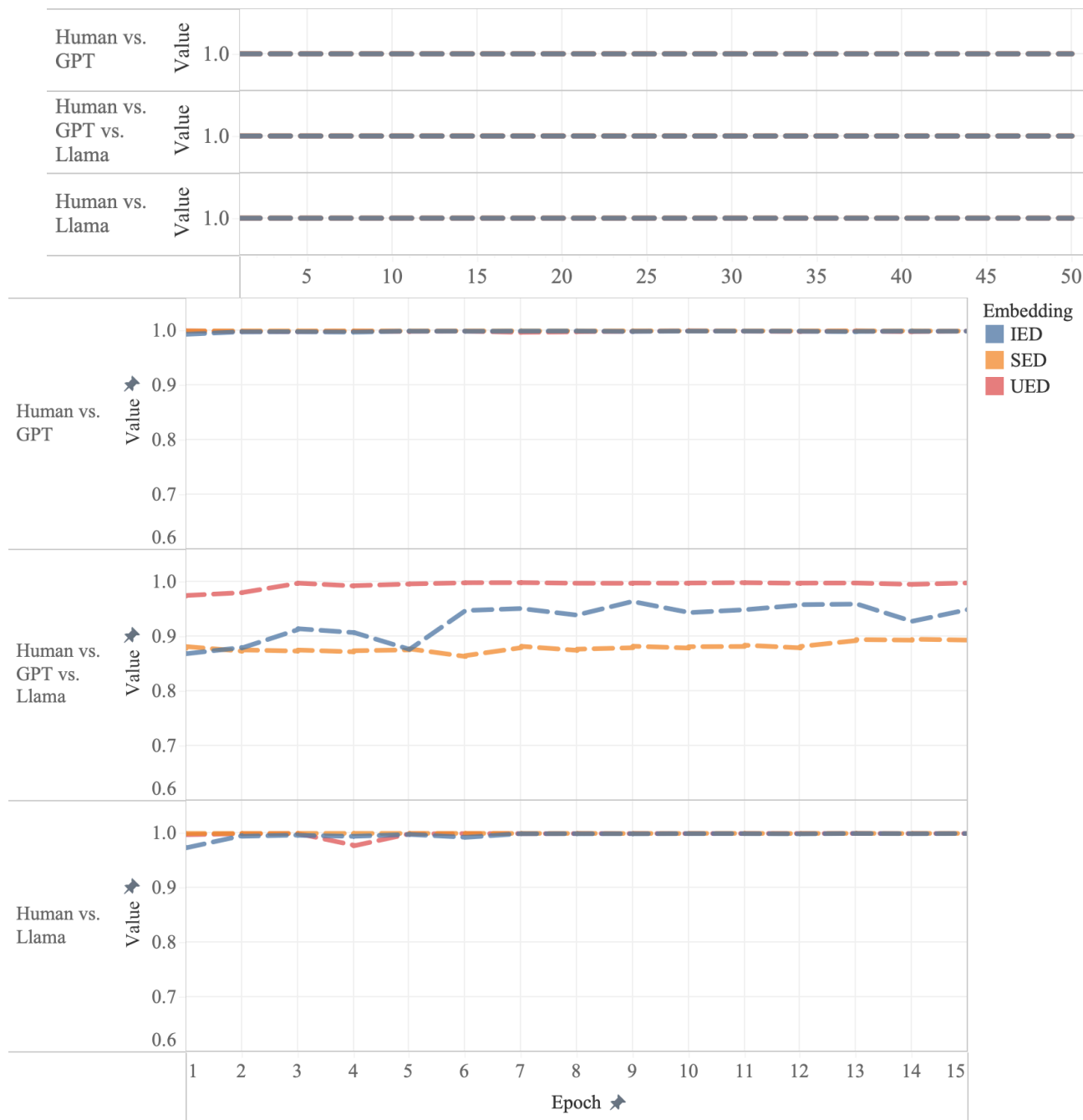


Figure 8: Simple MLP (top) and CNN (bottom) Testing and Training Learning Curves by Subset and Embedding

B. Analysis of Word Embeddings

1) *InferSent*: The performance of InferSent was inconsistent across models. While it showed reasonable efficacy when used with CNN and Simple MLP, it struggled notably with Adaboost, particularly in multiclass classification settings. The high computational demands, coupled with limited performance gains, rendered InferSent less practical for large-scale deployment.

2) *SBERT*: SBERT consistently delivered superior performance across all models. It outperformed both InferSent and USE in accuracy, precision, recall, and F1 score, making it the most reliable choice for high-stakes classification tasks. SBERT's computational efficiency also makes it particularly suitable for large-scale or real-time applications..

3) *USE*: The Universal Sentence Encoder (USE) performed reliably for CNN and Simple MLP, yielding moderate performance gains for Adaboost. While USE offered a balanced trade-off between computational efficiency and classification performance, it did not match SBERT's robustness or efficiency in most scenarios.

4) *Comparative Analysis*: In comparative terms, SBERT consistently outperformed InferSent and USE, particularly in conjunction with the Adaboost model. SBERT's combination of high performance and computational efficiency makes it highly advantageous for applications requiring both accuracy and scalability. The bar graph below (Figure 8) shows overall classification accuracy, F1 score, precision, and recall across different embeddings by subset, providing a visual summary of their respective impacts on model outcomes.

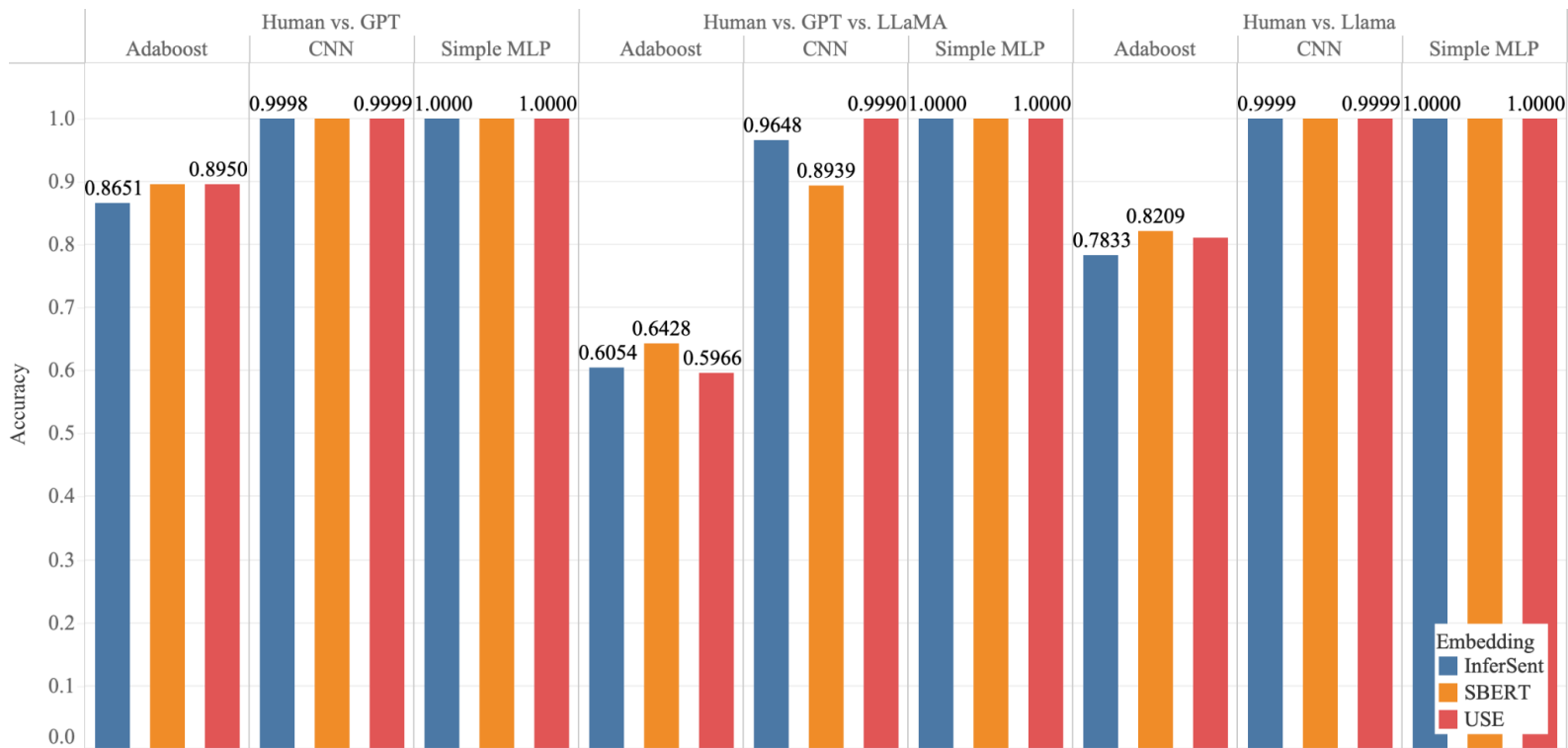


Figure 8: Accuracies by Embedding

IV. CONCLUSION

This project explored the effectiveness of using word embeddings and machine learning to distinguish between human, GPT3.5, and LLaMA3.2-generated text. This project used three machine learning models: Adaboost, an MLP, and a CNN. Furthermore, three embedding models were used: Infsent, Sentence-BERT (SBERT), and Universal Sentence Encoder (USE). Finally, three text datasets were considered: humanGPT (100,000 text entries, evenly split between humans and GPT), humanLLaMA (100,000 text entries, evenly split between human and LLaMA), and combined (150,000 text entries, evenly split between human, GPT, and LLaMA). A total of 27 experiments were performed: 3 datasets x 3 embeddings x 3 machine learning models. For all experiments, accuracy, precision, and recall were tracked. For the neural networks, the loss function across 15 epochs was collected.

Furthermore, PCA was used for dimensionality reduction of the embedded vectors, and a technique called Vector-to-Grayscale (V2G) was used to generate images fed into the CNN.

The MLP emerged as the most effective model across all tasks with perfect accuracy, precision, and recall scores. As a result, we can argue that text written by humans is significantly different from chatbot-generated text.

The CNN model demonstrated strong performance, particularly in binary tasks, though it encountered challenges in differentiating between GPT and LLaMA in multiclass settings. Adaboost, while effective in simpler tasks, struggled significantly in multiclass classifications, primarily due to its susceptibility to false positives.

The analysis of word embeddings underscored SBERT as the most robust and computationally efficient option, consistently enhancing model performance across all settings. InferSent and USE showed moderate effectiveness but fell short of SBERT's reliability and scalability. Given the smaller computational cost of generating embedding vectors using SBERT, SBERT has the potential to be deployed in systems where efficiency is key.

V. FUTURE WORK

While this project successfully demonstrated the effectiveness of using word embedding techniques and machine learning models to distinguish between human and chatbot written text, several areas can be expanded upon. For example, it is reasonable to argue that pre-trained models have an inherent bias towards human text as they were trained before the wide use of Large Language Models (LLMs). Therefore, it would be interesting to see the performance of embedding techniques directly trained, or fine-tuned, on human and chatbot text.

Similarly, hyperparameter tuning and the number of epochs used were limited during the training of the neural networks. Given that the MLP performed perfectly across all tasks, it is worth exploring hyper-parameter tuning and increasing the number of epochs for the CNN. By doing so, it is possible to find any inherent limitation in the V2G technique or the CNN architecture used. Additionally, other vector-to-image techniques could be explored when training the CNN.

Finally, SBERT emerged as the most computationally efficient and best embedding technique (to differentiate human text from chatbot text) across all experiments. This raises the following question: is there an embedding technique that is more computationally efficient than SBERT while maintaining the same, or similar performance?

REFERENCES

- [1] G. A. Godghase, R. Agrawal, T. Obili, and M. Stamp, “Distinguishing Chatbot from Human,” arXiv preprint arXiv:2408.04647, Aug. 2024. Available: <https://arxiv.org/abs/2408.04647>.
- [2] Mahnaz Koupaei and William Yang Wang. WikiHow: A large scale text summarization dataset. <https://arxiv.org/abs/1810.09305>, 2018.
- [3] A. Conneau, D. Kiela, H. Schwenk, L. Barrault, and A. Bordes, “Supervised learning of universal sentence representations from natural language inference data,” Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP), Copenhagen, Denmark, Sep. 2017, pp. 670-680. Available: <https://github.com/facebookresearch/InferSent>.
- [4] N. Reimers and I. Gurevych, “Sentence-BERT: Sentence embeddings using Siamese BERT-Networks,” Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), Hong Kong, China, Nov. 2019, pp. 3973-3983. Available: <https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>.
- [5] Universal Sentence Encoder: D. Cer, Y. Yang, S. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, Y.-H. Sung, B. Strope, and R. Kurzweil, “Universal sentence encoder,” arXiv preprint arXiv:1803.11175, 2018. Available: <https://tfhub.dev/google/universal-sentence-encoder/4>.
- [6] Meta AI, “Llama 3.2 3B: Model Card,” GitHub repository, 2024. [Online]. Available: https://github.com/meta-llama/llama-models/blob/main/models/llama3_2/MODEL_CARD.md.
- [7] Llama 3.2 3B Hugging Face Repository: Meta AI, “Llama 3.2 3B,” Hugging Face repository, 2024. [Online]. Available: <https://huggingface.co/meta-llama/Llama-3.2-3B>

