CSE 584, Machine Learning
The Pennsylvania State University
Fall 2024

Midterm Project

Jeongwon Bae (945397461)
Individual participation

Oct. 6, 2024

1 Introduction

In recent years, advancements in natural language processing (NLP) have been driven by two main types of models: Large Language Models (LLMs) primarily designed for text generation, and powerful encoder models focused on language understanding. Notable examples include OpenAI's Generative Pre-trained Transformer (GPT) series [1] for generation, and Google's Bidirectional Encoder Representations from Transformers (BERT) [2] for understanding tasks. While both types have revolutionized NLP, this study focuses primarily on generative LLMs and their distinctive text generation patterns.

Despite their impressive performance, different LLMs exhibit distinct patterns and styles in their text generation processes. These variations arise from differences in training data, model architectures, and optimization techniques employed during their development. Consequently, when presented with the same input prompt, different LLMs may generate different continuations, even if the resulting texts are semantically similar.

This project aims to explore and exploit these differences by building a deep learning classifier capable of identifying the specific LLM that generated a given text completion. Specifically, given a set of truncated text inputs x_i (e.g., "Yesterday I went"), various LLMs are tasked with completing these texts by appending different continuations x_j (e.g., "to Costco and purchased a floor cleaner."). The resulting complete texts, such as "Yesterday I went to Costco and purchased a floor cleaner," are then used as input pairs (x_i, x_j) for the classifier.

The primary objectives of this project are:

(1) Dataset Construction: Curate a comprehensive dataset comprising truncated text inputs (x_i) from the BookCorpus dataset [3] and their corresponding completions (x_j) generated by seven different LLMs. This dataset serves as the foundation for training and evaluating the classifier.

(2) Classifier Development: Design and implement a deep learning-based classification model, specifically the DeBERTaV3Classifier, capable of accurately determining which LLM produced a given (x_i, x_j) pair. This involves leveraging the DeBERTaV3 architecture [4], implementing multi-head attention mechanisms, and optimizing the model using advanced techniques such as AdamW optimizer and linear learning rate scheduling with warmup.

(3) Performance Evaluation: Assess the classifier's effectiveness using various performance metrics, including Accuracy and micro-F1 score, and provide insights into its strengths and limitations through confusion matrix analysis. Additionally, conduct in-depth analyses by

applying the classifier to alternative datasets and extending the generation of x_j to five sentences to understand the impact of text length on classification accuracy. Perform LLM characteristic analysis using N-gram analysis and topic modeling techniques to elucidate the underlying factors contributing to the classification patterns and to gain insights into each model's unique generative tendencies and thematic preferences.

(4) Related Work Analysis: Situate this project within the broader context of existing research on LLMs, text classification, and model attribution, highlighting the unique contributions in multi-class LLM attribution and the potential applications of the developed classifier in areas such as AI-generated text detection and extreme multi-label classification.

By completing this project, we aim to enhance the understanding of subtle differences between various LLMs and demonstrate the feasibility of attributing generated texts to their source models.

2 Dataset Construction

2.1 Construction of x_i

The dataset for this project was constructed using the BookCorpus dataset, which comprises over 11,000 unpublished books across various genres, providing a diverse range of linguistic patterns. The primary goal was to generate 5,000 truncated text inputs x_i to serve as prompts for LLMs. Initially, the text data was segmented into individual sentences using NLTK's sentence tokenizer [5]. Each sentence was then filtered to exclude those starting with double backticks (“) and those with fewer than 5 or more than 30 words to ensure appropriate length and formatting. To ensure grammatical completeness, sentences were analyzed for the presence of both a subject and a verb using part-of-speech tagging.

Qualified sentences were truncated at the first verb, resulting in coherent prompts such as “Yesterday I went.” These truncated texts underwent normalization, including the expansion of contractions, conversion to lowercase, and the removal of excessive whitespace. Additionally, sentences with an odd number of quotation marks were excluded to maintain text integrity.

To promote diversity, duplicate entries were removed, and a semantic similarity check was performed using the SentenceTransformer model [6]. Sentences with a similarity score above 0.8 were discarded to ensure a varied dataset. The final set of 5,000 unique and diverse x_i samples was then saved in a CSV file for subsequent use in training and evaluating the classifier.

2.2 Description of the Utilized LLMs

We generated the text completions x_j using a selection of state-of-the-art LLMs. These models were chosen not only for their diverse architectures and capabilities, ensuring a comprehensive representation of various text generation styles, but also with consideration for GPU resource constraints and API processing speeds. As a result, we selected lighter and faster versions of each representative model, striking a balance between performance diversity and computational efficiency. The list of models utilized includes:

- Claude-3-haiku
- Falcon-7b
- Qwen2.5-3B-Instruct
- Phi-3-Mini-4K
- GPT-4o-mini
- Llama-2-7b-chat
- Llama-3.2-3B-Instruct

Claude-3-haiku [7] is the fastest and most compact model in the Claude 3 family, generating near-instantaneous outputs. In addition to a 200k token context window, the Claude 3 Haiku model can produce human-like outputs and is ideal for automating basic repetitive tasks. It costs $0.25 per million input tokens and $1.25 per million output tokens.

Falcon-7b [8] is a 7 billion parameter model developed by the Technology Innovation Institute (TII) in Abu Dhabi. It is the first large-scale pure Mamba model, designed to overcome sequence scaling limitations without compromising performance. Based on State Space Language Models (SSLMs), it can process sequences of arbitrary length with constant memory usage and generation time. The model is open-access, available through the Hugging Face [9] ecosystem, and can run on a single A10 24GB GPU, making it highly efficient for various research and application purposes.

Qwen2.5 [10] is the latest addition to the Qwen family, part of a series ranging from 0.5B to 72B parameters. 3B-Instruct model is a 3.09 billion parameter causal language model from the Qwen2.5 family, featuring both pretraining and post-training stages. It uses a transformer architecture with Rotatory Position Embedding (RoPE) [11], SwiGLU [12][13], Root Mean Square Layer Normalization (RMSNorm), Attention Query, Key, Value (QKV) bias, and tied word embeddings. The model has 36 layers with 16 attention heads for Q and 2 for KV (Grouped Query Attention [14]). It supports a full context length of 32,768 tokens and can generate up to 8,192 tokens. This instruct variant offers improved performance in instruction following, long text generation, structured data understanding, and JSON output. It's designed for enhanced role-play and chatbot applications, supporting over 29 languages and demonstrating increased resilience to diverse system prompts.

The Phi-3 family [15] offers highly capable and cost-effective small language models (SLMs). Phi-3-Mini-4K-Instruct has 3.8B parameters and supports a context length of 4K tokens. It's trained on the Phi-3 dataset, combining synthetic data with filtered publicly available website data, focusing on high-quality and reasoning-dense properties. The model has undergone supervised fine-tuning and direct preference optimization for instruction following and safety measures, making it suitable for various language, reasoning, coding, and math tasks.

GPT-4o-mini [16] is a compact version of GPT-4o, the multilingual, multimodal generative pre-trained transformer by OpenAI. It's designed for high-volume API usage, costing $0.15 per million input tokens and $0.6 per million output tokens, with prices doubling after fine-tuning. The model supports text, image, and audio processing and generation, with a context length of 128k tokens and an output limit of 4,096 tokens (later updated to 16,384). GPT-4o has knowledge up to October 2023 but can access the internet for up-to-date information. GPT-4o-mini is significantly more capable than GPT-3.5 Turbo, which it replaced on the ChatGPT interface. It serves as the default model for ChatGPT guest users and those who've reached their GPT-4o usage limit. Also, GPT-4o mini will become available in fall 2024 on Apple's mobile devices and Mac desktops.

Llama-2-7b-chat [17] is part of the Llama 2 family of LLMs released by Meta AI in 2023. This 7 billion parameter model is a fine-tuned "chat" variant, designed for dialogue and instruction-following tasks. Llama 2 models are offered with 7B, 13B, or 70B parameters, focusing on advancing performance in smaller models rather than increasing parameter count. The model is available free of charge for both research and commercial use, enabling deployment on local infrastructure without extensive computing resources. Llama 2 models are capable of various natural language processing tasks, including text generation and programming code tasks, and aim to democratize access to state-of-the-art language models for a wider range of organizations and researchers.

Llama-3.2-3B-Instruct [18] is a lightweight model from the Llama 3.2 family, designed for on-device inference. This 3 billion parameter model supports BFloat16 numerics and is compatible with Android and iOS devices. It offers tool-calling capabilities through custom functions defined in either the system or user prompt. Unlike larger Llama models, it does not support built-in tools like Brave Search or Wolfram. The model is optimized for efficient deployment across various edge devices, including wearables, embedded devices, and microcontrollers, making it suitable for mobile and edge computing applications.

2.3 Generation of x_j using LLMs

The following description outlines the process using one of these models as an illustrative example, with similar procedures applied to other models. Our approach began with crafting detailed prompts for each X_i sample, instructing the model to complete the given sentence fragment without altering its original structure. By providing identical prompts to each LLM, we minimized the potential impact of prompt variation on the generated completions, ensuring a more standardized basis for comparison across models. By employing identical prompts across all models, we were able to isolate and examine the unique generative patterns of each LLM. This approach revealed subtle yet significant differences in their text production mechanisms. Figure 1 illustrates an example of such a prompt:

```
Complete the following sentence by adding words to form a complete sentence ending
with a period.
Do not change or rephrase the beginning of the sentence.
Only provide the continuation of the sentence; do not include any additional comments,
emoticons, explanations, or notes.
Examples:
Sentence: Yesterday I went
Completion: to Costco and purchased a floor cleaner.
Sentence: She decided to
Completion: take a shower.
Sentence: {x_i}
Completion:
```

Figure 1: Example of a Prompt Used for x_j Generation

We generated the text completions in batches of 32 to enhance efficiency and control API usage. For each batch, we followed a series of key steps to ensure consistent and accurate results:

First, we submitted the prepared prompts to each model's API, with parameters set to a maximum of 100 tokens and a temperature of 0.7 to balance creativity and coherence in the outputs. For Claude-3-haiku and GPT-4o-mini, we utilized their respective paid API clients to generate completions. For the other LLMs, we implemented the generation process using free APIs through the Hugging Face Transformers library, specifically employing the AutoTokenizer, AutoModelForCausalLM, and pipeline packages.

Following this, the raw outputs underwent a thorough cleaning process, which included removing extraneous whitespace, ensuring proper sentence structure and punctuation, and using sequence matching to align the generated text with the original X_i.

Next, each cleaned X_j underwent a series of validation checks. These checks verified that the completion started with the original X_i (case-insensitive), ensured a minimum of 3 and a maximum of 50 words were added, checked for proper sentence endings (period, question mark, or exclamation mark), detected and removed unnecessary starting phrases, verified matching quotation marks, and checked for repetition of 3 or more consecutive words.

In cases where the generated text failed validation, we implemented a retry mechanism, allowing up to five attempts to generate a valid completion for each X_i. To prevent API overload, we also implemented a 0.5-second delay between API calls.

Table 1 below presents sample pairs of X_i and their corresponding X_j completions:

X_i	$X_i + X_j$	**LLM**
in a panic , she demanded	in a panic , she demanded to speak to the manager.	Claude-3-haiku
her father smiled	her father smiled and nodded approvingly.	Claude-3-haiku
what she was	what she was thinking about during the meeting.	Claude-3-haiku
all her relatives were	all her relatives were invited to the family reunion.	Claude-3-haiku

Table 1: Sample Generated x_j Completions

This detailed process was applied consistently across all selected LLMs, resulting in a diverse set of X_j completions. In our testing environment, which consisted of an Intel Xeon Gold 5315Y (45GB CPU RAM) and an Ampere A4000 (16GB GPU RAM), the processing time for each LLM varied significantly. We utilized Python version 3.11.7 and the PyTorch version 2.1.1 deep learning framework for the experiments. Our transformer-based models and APIs were implemented using the Transformer 4.45.1 library. The dataset generation time for each LLM took anywhere from a minimum of 1.5 hours to a maximum of 3.5 hours. (For the 5-sentence dataset, which will be discussed later, the processing time extended to approximately 7 hours.) Our methodical approach aimed to create a robust dataset, providing a foundation for the development of a deep learning classifier. The final dataset composition after applying our generation and validation process is presented in Table 2.

LLM Model	Train	Validation	Test	Total
Claude-3-haiku	3179	795	993	4967
Falcon-7b	3168	792	990	4950
GPT-4o-mini	3198	800	1000	4998
Llama-2-7b-chat	3182	796	994	4972
Llama-3.2-3B-Instruct	3198	799	1000	4997
Phi-3-Mini-4K	3197	800	1000	4997
Qwen2.5-3B-Instruct	3178	794	993	4965
Total	22300	5576	6970	34846

Table 2: Final dataset composition after processing and validation. These numbers reflect the samples that successfully passed our validation checks, forming the basis for our subsequent classifier development and evaluation.

3 Model Design and Training

To develop an effective classifier capable of attributing text completions (X_i, X_j) to their respective LLMs, a comprehensive approach was adopted. This involved evaluating a variety of baseline machine learning and neural network models before selecting the most suitable deep learning architecture.

3.1 Baseline Models

Initially, several traditional machine learning and neural network models were implemented to establish performance benchmarks. These models included Machine Learning Models such as Multinomial Naive Bayes, K-Nearest Neighbors (KNN), Logistic Regression, Decision Tree, Random Forest, and Extreme Gradient Boosting (XGBoost) [19].

Neural Network Models were also implemented, including Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), Bidirectional LSTM (BiLSTM) [20], and Convolutional Neural Network with BiLSTM (CNNBiLSTM) [21]. These models provided a diverse set of approaches, ranging from probabilistic classifiers to ensemble methods and deep learning architectures, facilitating a comprehensive comparison of their efficacy in the classification task.

3.2 Our Deep Learning Classifier: DeBERTaV3Classifier

After evaluating various baseline models, we selected the The Decoding-enhanced BERT with Disentangled Attention V3 (DeBERTaV3 architecture) [22] for our classifier due to its strong performance in natural language understanding tasks. The DeBERTaV3Classifier utilizes the enhanced capabilities of DeBERTaV3, building upon the foundational BERT architecture [2] to achieve improved contextual representations.

BERT is a transformer-based model designed to understand the context of words in a sentence by considering both their left and right surroundings. Its architecture consists of multiple layers of bidirectional transformers, enabling it to capture deep semantic relationships within the text. The key components of BERT include an embedding layer that converts input tokens into dense vectors, incorporating token embeddings, positional embeddings, and segment embeddings. Transformer encoder layers, comprising self-attention mechanisms and feed-forward neural networks, allow the model to weigh the importance of different words relative to each other in a sentence. A pooling layer then aggregates the information from the transformer layers to produce a fixed-size representation for classification tasks.

DeBERTa improves upon traditional BERT and RoBERTa [23] models by introducing a disentangled attention mechanism and an enhanced mask decoder. The disentangled attention mechanism separates content and position information within the attention layers, allowing the model to better capture relationships between words irrespective of their positions in the sequence. The enhanced mask decoder refines the decoding process during pre-training, improving the model's ability to predict masked tokens and leading to more accurate and coherent text representations.

Building upon these innovations, DeBERTaV3 further enhances the model's capabilities by improving the efficiency of DeBERTa using ELECTRA-style [4] pre-training with Gradient-Disentangled Embedding Sharing. This approach allows the model to utilize training data more effectively and achieve better performance with less computational cost. The DeBERTaV3 base model comes with 12 layers and a hidden size of 768, comprising 86 million backbone parameters. It features an expanded vocabulary containing 128K tokens, introducing 98 million parameters in the embedding layer.

The DeBERTaV3Classifier combines the pretrained DeBERTaV3 model with a specially designed classification layer, making it well-suited for our specific classification tasks. This model is then fine-tuned on our specific dataset to adapt it for LLM attribution. The architecture is structured as follows:

1. Pretrained DeBERTaV3 Backbone:
    - Utilizes `microsoft/deberta-v3-base` from the Hugging Face Transformers library.
    - Processes input text to generate detailed contextual embeddings.
    - Fine-tuned on our dataset to adapt to the specific task of LLM attribution.
2. Multi-Head Attention Layer:
    - Applies a multi-head attention mechanism to the sequence output from DeBERTaV3, enabling the model to focus on different parts of the input dynamically.
    - Fine-tuned to capture LLM-specific attention patterns.
3. Pooling Mechanism:
    - Implements mean pooling over the attention-enhanced embeddings to obtain a fixed-size representation of the input sequence.
4. Fully Connected Layers:
    - Dropout Layer: Applies a configurable dropout rate (defaulted to 0.26) to prevent overfitting during fine-tuning.
    - First Linear Layer: Reduces the dimensionality of the pooled output to 128 units, followed by a ReLU activation function to introduce non-linearity.
    - Second Linear Layer: Maps the 128-dimensional representation to the number of target classes, producing the final logits for classification.
    - These layers are trained from scratch during the fine-tuning process to adapt to the LLM attribution task.

This architecture, when fine-tuned on our dataset, effectively captures the nuanced differences in text generation styles across different LLMs, enabling accurate attribution through the classifier. The fine-tuning process allows the model to adapt its pretrained knowledge to the specific characteristics of various LLMs, enhancing its ability to distinguish between them.

```python
class DeBERTaV3Classifier(nn.Module):
 def __init__(self, num_classes, dropout_rate=0.26):
  super(DeBERTaV3Classifier, self).__init__()
  self.deberta = DebertaV2Model.from_pretrained('microsoft/deberta-v3-base')
  self.dropout = nn.Dropout(dropout_rate)
  self.attention = nn.MultiheadAttention(embed_dim=self.deberta.config.hidden_size, num_heads=8)
  self.fc1 = nn.Linear(self.deberta.config.hidden_size, 128)
  self.relu = nn.ReLU()
  self.fc2 = nn.Linear(128, num_classes)

 def forward(self, input_ids, attention_mask):
  outputs = self.deberta(input_ids=input_ids, attention_mask=attention_mask)
  sequence_output = outputs.last_hidden_state
  sequence_output = sequence_output.permute(1, 0, 2)
  attn_output, _ = self.attention(sequence_output, sequence_output, sequence_output)
  attn_output = attn_output.permute(1, 0, 2)
  pooled_output = torch.mean(attn_output, dim=1)
  x = self.dropout(pooled_output)
  x = self.fc1(x)
  x = self.relu(x)
  x = self.dropout(x)
  return self.fc2(x)

```

Figure 2: DeBERTaV3Classifier Implementation

The code above demonstrates the implementation of the DeBERTaV3Classifier, showcasing the architectural components described in the previous section.

3.3 Additional Classifiers for Performance Comparison

To demonstrate the superiority of the DeBERTaV3Classifier and provide a comprehensive evaluation, additional classifiers were implemented and assessed. These classifiers were based on architectures similar to DeBERTaV3Classifier, including:

- BERT (Fine-tuned): Utilizing the standard BERT architecture
- XLNet (Fine-tuned): Leveraging the XLNet [24] model
- RoBERTa (Fine-tuned): Based on the RoBERTa model

By implementing these additional classifiers, we aimed to establish a robust comparison baseline and validate the performance advantages of the DeBERTaV3Classifier across various text generation patterns. Through this comparative analysis, we gained valuable insights into how different architectures perform in the nuanced task of text attribution. Notably, some models exhibited unexpected strengths, challenging our initial hypotheses about their capabilities.

3.4 Training Methodology and Optimization

The training process for the DeBERTaV3Classifier involved several critical steps to optimize performance and ensure effective learning.

3.4.1 Data Preparation

- The dataset was split into training, validation, and testing subsets to facilitate unbiased evaluation.
- Text $(X_i + X_j)$ inputs were tokenized using the appropriate tokenizer corresponding to each model, ensuring consistency in input representation.

3.4.2 Hyperparameter Tuning with Optuna

- Hyperparameters such as learning rate, batch size, weight decay, and dropout rate were meticulously tuned using the Optuna library [25].
- Optuna's efficient hyperparameter optimization algorithms enabled the identification of optimal configurations, significantly improving the classifier's accuracy and robustness.
- The hyperparameter tuning process involved defining the search space, creating an objective function that trains the model with a given set of hyperparameters, and utilizing Optuna's Tree-structured Parzen Estimator (TPE) to navigate the hyperparameter space effectively.

3.4.3 Optimizer and Learning Rate Scheduler

- AdamW Optimizer [26]
  Employed for its effectiveness in handling sparse gradients and regularization through weight decay. AdamW is an extension of the Adam optimizer that correctly implements weight decay regularization. It addresses the issue in Adam where L2 regularization behaves differently from weight decay, leading to suboptimal performance in some cases. The key innovation of AdamW is decoupling the weight decay from the gradient-based update, allowing for more effective regularization. This decoupling enables the optimizer to maintain the benefits of adaptive learning rates while properly implementing weight decay. AdamW has been shown to improve generalization, especially in large models with many parameters, by preventing over-fitting more effectively than standard Adam. The optimizer adapts the learning rate for each parameter independently, which is particularly useful for models with diverse parameter sensitivities, like transformer-based architectures.
- Learning Rate Scheduler
  Utilized a linear schedule with warmup, gradually increasing the learning rate at the start of training and then decreasing it linearly, which aids in stable convergence. The warmup phase helps to prevent early overfitting and unstable gradients at the beginning of training by slowly increasing the learning rate from a very small value. After the warmup phase, the learning rate decreases linearly, which allows the model to make larger updates initially and then finer adjustments as it approaches convergence. This schedule is particularly effective for transformer-based models, as it helps in managing the complex loss landscape typical in these architectures. The combination of warmup and linear decay helps in achieving a

balance between exploration of the parameter space early in training and exploitation of good parameters later on.

3.4.4 Training Loop

- The model was trained for a maximum of 20 epochs with an early stopping mechanism set to halt training if no improvement was observed in the validation loss for three consecutive epochs.
- Gradient Clipping: Applied to prevent exploding gradients, ensuring stable training dynamics. Gradient clipping is a technique that limits the maximum norm (magnitude) of gradients during backpropagation. When the gradient norm exceeds a specified threshold (in this case, 1.0), the gradients are scaled down proportionally to match this threshold. This technique is particularly important for deep networks and recurrent neural networks, where gradients can grow exponentially and lead to numerical instability. By constraining the gradients, it prevents drastic updates to the model parameters, which can derail the training process or cause the model to diverge. Gradient clipping helps maintain a more stable and controlled optimization process, especially in the early stages of training when gradients can be particularly large and unstable. It allows the use of larger learning rates and can help the model traverse sharp minima in the loss landscape more effectively.

By using a comprehensive training methodology, thorough hyperparameter tuning, and advanced optimization techniques, we ensured that the DeBERTaV3Classifier performed robustly in attributing text completions to their respective LLMs.

Figure 3 showcases the main functions used in the training and evaluation process of the model. The `train_model` function handles the training loop, including early stopping. The `evaluate_model` function performs the final evaluation on the test set, calculating various metrics. Finally, the `train_and_evaluate` function orchestrates the entire process, from data preparation to model evaluation.

```
def train_model(model, train_loader, val_loader, criterion, optimizer,
 scheduler, device, num_epochs=10, patience=3):
 for epoch in range(num_epochs):
  # Training loop
  model.train()
  for batch in train_loader:
   # Forward pass, loss calculation, backward pass, optimization
   ...
  # Validation loop
  model.eval()
  with torch.no_grad():
   for batch in val_loader:
    # Validation steps
    ...
  # Early stopping check
  ...

 return train_losses, val_losses

def evaluate_model(model, test_loader, device, label_encoder,
 train_losses, val_losses):
 model.eval()
 with torch.no_grad():
  for batch in test_loader:
   # Prediction and evaluation steps
   ...
 # Calculate metrics (accuracy, F1 score, confusion matrix)
 ...
 # Visualize results
 ...
 return results

def train_and_evaluate(model_class, X_train, X_val, X_test, y_train,
 y_val, y_test, label_encoder, tokenizer, ...):
 # Prepare data loaders
 ...

 # Define model and optimizer
 model = define_model(model_class, num_classes)
 optimizer = AdamW([...], weight_decay=0.02)

 # Set up loss function and scheduler
 criterion = nn.CrossEntropyLoss()
 scheduler = get_linear_schedule_with_warmup(...)

 # Train model
 train_losses, val_losses = train_model(...)

 # Evaluate model
 evaluation_results = evaluate_model(...)

 return evaluation_results

```

Figure 3: Key Functions for Model Training and Evaluation

4 Experimental Results

4.1 Evaluation Metrics

To assess the performance of the various models employed in our study, we utilized three primary evaluation metrics: Accuracy, F1 Score, and Confusion Matrix. These metrics provide a comprehensive understanding of the models' effectiveness in correctly classifying text data.

4.1.1 Accuracy

Accuracy measures the proportion of correctly predicted instances across all classes out of the total number of instances evaluated. It provides a straightforward assessment of a model's overall correctness in a multi-class setting. The formula for calculating accuracy in a multi-class problem is as follows:

$$\text{Accuracy (ACC)} = \frac{\text{Number of correctly classified samples}}{\text{Total number of samples}}. \tag{1}$$

4.1.2 F1 Score

In a multi-class classification problem, the F1 Score is typically calculated for each class separately and then averaged. For this study, we use the micro-averaged F1 Score (micro-F1), which is particularly useful when dealing with imbalanced datasets. The micro-F1 is calculated as follows:

$$\text{Micro-Precision} = \frac{\sum_{i=1}^{k} TP_i}{\sum_{i=1}^{k}(TP_i + FP_i)} \tag{2}$$

$$\text{Micro-Recall} = \frac{\sum_{i=1}^{k} TP_i}{\sum_{i=1}^{k}(TP_i + FN_i)} \tag{3}$$

$$\text{Micro-F1} = 2 \cdot \frac{\text{Micro-Precision} \cdot \text{Micro-Recall}}{\text{Micro-Precision} + \text{Micro-Recall}} \tag{4}$$

where k is the number of classes, TP_i, FP_i, and FN_i are the true positives, false positives, and false negatives for class i, respectively. In our study, although the class distribution is relatively balanced, we opted to use the micro-F1, which tends to give more weight to the performance on minor classes. This choice ensures that our evaluation metric is sensitive to the model's performance across all classes, including those with slightly fewer instances.

4.1.3 Confusion Matrix

A confusion matrix is a tabular summary of a classifier's performance, showing the counts of correct and incorrect predictions for each class. It provides a detailed breakdown of the model's performance across all classes. The confusion matrix is particularly useful for multi-class classification problems, as it reveals not just the overall accuracy but also which classes are being confused with each other.

4.2 Experimental Results

The following table presents the experimental results for all the models evaluated in this study. The models are categorized into traditional machine learning algorithms, neural network Architectures, and transformer-based encoder only fine-tuned model, and DeBERTaV3Classifier. The table includes the ACC and micro-F1 for each model, allowing for a direct comparison of their performance on the classification tasks.

Model	**ACC(↑)**	**micro-F1(↑)**
Multinomial Naive Bayes (MNB)	0.3154	0.3062
K-Nearest Neighbors (KNN) @k=7	0.1687	0.1646
Logistic Regression (LR)	0.3976	0.3923
Random Forest (RF)	0.2634	0.2648
Extreme Gradient Boosting (XGBoost)	0.3581	0.3523
RNN (Tokenizer-DeBERTaV3)	0.4258	0.4183
LSTM (Tokenizer-DeBERTaV3)	0.4204	0.4162
GRU (Tokenizer-DeBERTaV3)	0.4268	0.3983
BiLSTM (Tokenizer-DeBERTaV3)	0.4446	0.4433
CNNBiLSTM (Tokenizer-DeBERTaV3)	0.3914	0.3938
BERT (Fine-tuned)	0.5372	0.5369
XLNet (Fine-tuned)	0.5108	0.5139
RoBERTa (Fine-tuned)	0.5202	0.5182
DeBERTaV3Classifier	**0.5478**	**0.5396**

Table 3: Performance Comparison of Models on Our Dataset

The DeBERTaV3Classifier achieved the highest performance with an Accuracy of 0.5478 and a micro-F1 score of 0.5396. While these scores indicate room for improvement, they establish a baseline for the challenging task of LLM attribution. To provide a more detailed view of the DeBERTaV3Classifier's performance, we present its confusion matrix:

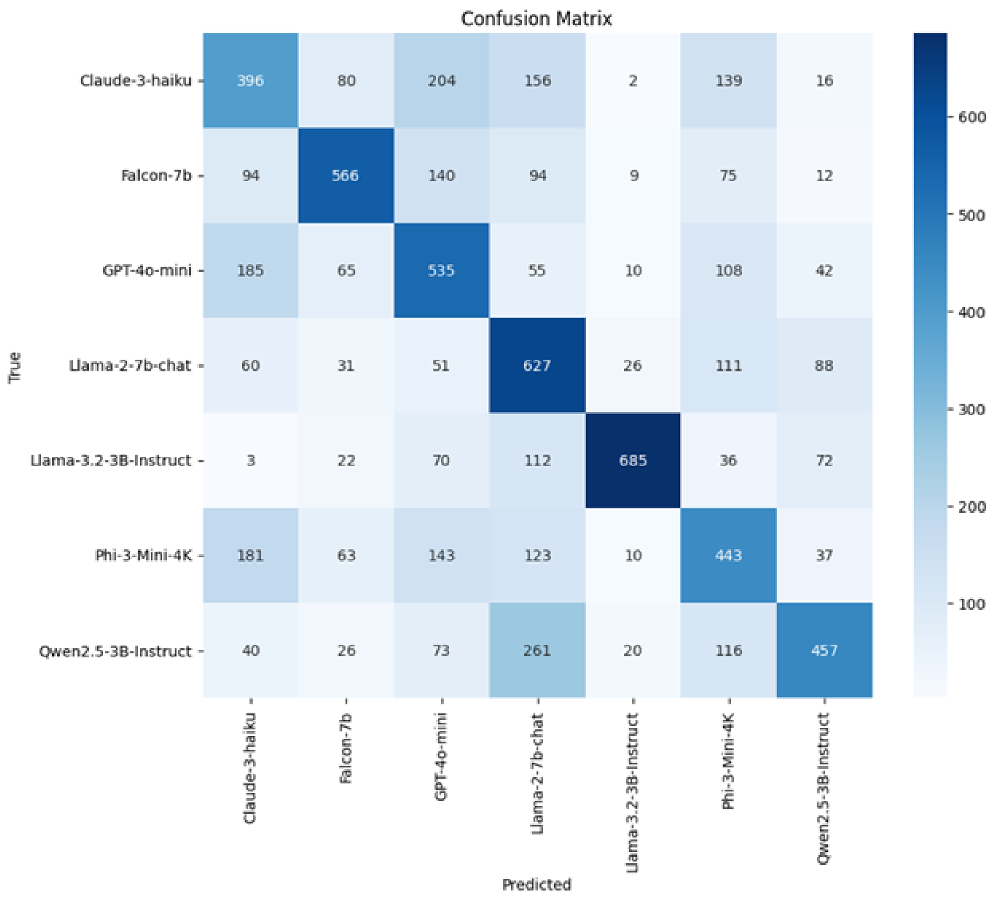

Figure 4: Confusion Matrix of DeBERTaV3Classifier on Our Dataset.

Figure 4 offers insights into the model's classification performance across different LLM classes, highlighting areas of strength and difficulty in distinguishing between certain LLMs.

A comprehensive analysis of these results, including an in-depth examination of the confusion matrix, exploration of factors contributing to misclassifications, and strategies for improvement, will be presented in the subsequent sections. This analysis will delve into the nuances of LLM attribution and discuss the implications for future research in this field.

5 In-Depth Analysis and Additional Experiments

In this section, we delve deeper into the experimental results and conduct additional experiments to further evaluate the robustness and effectiveness of the DeBERTaV3Classifier. Specifically, we apply the classifier to alternative datasets to assess its generalizability and explore the impact of expanding the generated text from one sentence to five sentences.

5.1 Applying DeBERTaV3Classifier to Other Datasets

To determine whether the observed lower performance of the DeBERTaV3Classifier on our primary dataset was due to model misconfiguration or inherent data characteristics, we applied the classifier to two additional standard datasets: the BBC-text Multi-Class dataset and the Tweet Emotion dataset.

5.1.1 BBC-text Multi-Class Dataset

The BBC-text Multi-Class dataset [27] comprises 2,225 news articles categorized into five distinct classes: Business, Entertainment, Politics, Sport, and Technology. This dataset is widely used for benchmarking text classification models and is known for its relatively balanced class distribution. Traditional machine learning algorithms typically achieve around 0.97 accuracy on this dataset, providing a reliable benchmark for model evaluation.

The DeBERTaV3Classifier was applied to the BBC-text Multi-Class dataset, achieving an accuracy of **0.9752** and a micro-F1 score of **0.9753**. These results are consistent with the performance of traditional machine learning methods on the same dataset, indicating that the classifier is well-configured for multi-class classification tasks. Consequently, the lower performance observed on the primary dataset is likely attributable to its specific characteristics rather than any deficiencies in the model's architecture.

5.1.2 Tweet Emotion Dataset

The Tweet Emotion dataset [28] consists of 20,000 English-language tweets, each annotated with one of six basic emotions: Anger, Fear, Joy, Love, Sadness, and Surprise. This dataset is commonly used for emotion detection and sentiment analysis tasks. Notably, models achieving 0.938 accuracy and micro-F1 on this dataset are considered competitive, often ranking highly on platforms. The DeBERTaV3Classifier was evaluated on the Tweet Emotion dataset, resulting in an accuracy of **0.9363** and a micro-F1 score of **0.9359**. These performance levels are comparable to the second-place standings on the "Papers with Code Public Leaderboad" [29], underscoring the classifier's proficiency in emotion detection tasks. The high performance on this dataset reinforces the model's reliability and effectiveness in multi-class text classification scenarios.

5.1.3 Extending Generation of x_j to Five Sentences

Building upon the success observed with single-sentence completions, we expanded the text generation process by increasing the number of generated sentences from one to five. The revised prompt for generating x_j is illustrated in Figure 5. Table 4 presents sample pairs of X_i and their corresponding X_j completions:

X_i	$X_i + X_j$	**LLM**
in a panic, she demanded	in a panic, she demanded to speak with the manager. The situation had escalated quickly, and she felt her concerns were not being addressed. The manager listened patiently and tried to find a reasonable solution. After a brief discussion, they were able to reach a compromise that satisfied both parties. Ultimately, the issue was resolved, and the customer left feeling heard and respected.	Claude-3-haiku

Table 4: Sample Generated Five-Sentences x_j Completions

```
Complete the following sentence by adding words to form five complete sentences,
including the initial one.
Do not change or rephrase the beginning of the first sentence.
Ensure that the five sentences are coherent and flow logically.
Only provide the continuation of the text; do not include any additional comments,
emoticons, explanations, or notes.
Examples:
Sentence: Yesterday I went
Completion: to Costco and purchased a floor cleaner.
The store was bustling with shoppers looking for deals.
I spent an hour browsing through various aisles.
In addition to the floor cleaner, I picked up some groceries and household items.
Overall, it was a productive shopping trip.
Sentence: x_i
Completion:
```

Figure 5: Modified Prompt for Five-Sentences X_j Generation

After applying this prompt, we excluded the Llama-2-7b-chat and Falcon-7b models as they did not consistently adhere to the five-sentence structure. The evaluation was conducted on the remaining five LLM models. The composition of the five-sentence dataset is presented in Table 5.

LLM Model	Train	Validation	Test	Total
Claude-3-haiku	3102	775	969	4846
GPT-4o-mini	3047	762	952	4761
Llama-3.2-3B-Instruct	3174	794	993	4961
Phi-3-Mini-4K	3198	800	1000	4998
Qwen2.5-3B-Instruct	2979	745	931	4655
Total	15500	3876	4845	24221

Table 5: Final dataset composition after processing and validation. These numbers reflect the samples that successfully passed our validation checks, forming the basis for our subsequent classifier development and evaluation.

The DeBERTaV3Classifier was re-evaluated using the five-sentence dataset. We observed a significant improvement in performance compared to the single-sentence dataset. For the five LLM models, the accuracy on the single-sentence dataset was 0.6074, with a micro-F1 score of 0.5954. In contrast, the five-sentence dataset yielded an accuracy of **0.8844** and a micro-F1 score of **0.8838**, demonstrating a substantial increase in performance. These results suggest that the classifier requires approximately five sentences to effectively capture the meaning and make accurate classifications. This indicates that longer text samples provide sufficient context for the model to distinguish between different LLMs more effectively.

5.2 LLM Characteristic Analysis

The confusion matrices presented in Figure 6 reveal varying degrees of accuracy in our classifier's ability to distinguish between different LLMs. To elucidate the underlying factors contributing to these classification patterns, we conducted a comprehensive analysis of each model's text generation characteristics, focusing on lexical choices, thematic tendencies, and stylistic nuances.

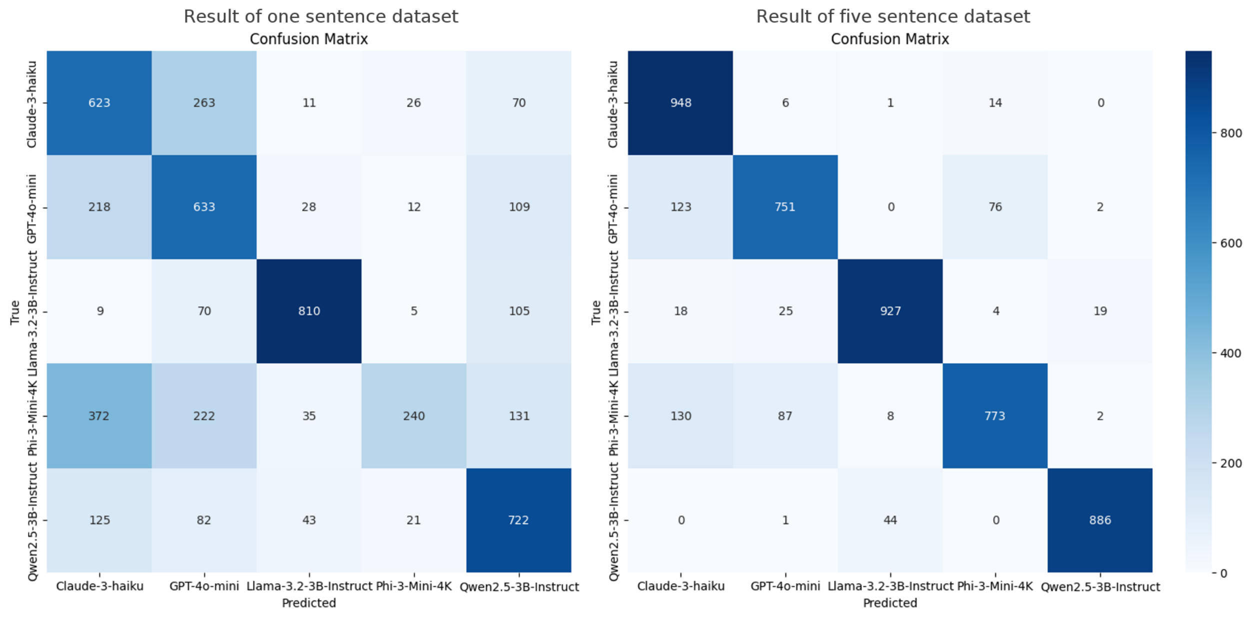

Figure 6: Confusion Matrix of DeBERTaV3Classifier on 1 Sentence Dataset vs 5 Sentences Dataset.

5.2.1 N-gram Analysis

N-grams are contiguous sequences of N items (typically words) from a given text. This analysis method is useful for understanding patterns and structures in language models. Our initial approach involved examining the top bigrams (N=2) for each LLM model, as presented in Table 6.

This analysis unveiled intriguing patterns in the linguistic structures employed by various LLMs. Notably, certain bigrams such as "deep breath", "felt sense", and "living room" emerged as common across multiple models. The prevalence of these shared linguistic constructs suggests a degree of homogeneity in the generative patterns of different LLMs, potentially contributing to the reduced discriminative power of our classifier in certain instances.

Conversely, we observed model-specific bigrams that could serve as distinctive fingerprints. For instance, the Qwen2.5-3B-Instruct demonstrated a pronounced tendency to employ temporal markers like "yesterday evening" and "yesterday morning". While these unique patterns offer potential for model differentiation, it is crucial to note that they appear to be heavily influenced by the structure of our prompts. This prompt dependency potentially limits their utility as robust discriminative features and underscores the importance of diverse and carefully crafted prompts in future iterations of this study.

LLM Model	**Single Sentence Dataset**		**Five Sentence Dataset**	
	Top Bigrams	**Frequency**	**Top Bigrams**	**Frequency**
Claude-3-haiku	late meeting	33	deep breath	199
	surprise party	25	felt sense	195
	deep breath	22	moving forward	131
	beautiful sunset	21	hard work	127
	felt sense	18	couldn help	126
GPT-4o-mini	felt sense	30	deep breath	292
	beautiful sunset	29	took deep	176
	finish homework	22	felt sense	172
	deep breath	22	filled air	127
	finish project	20	hard work	123
Llama-3.2-3B-Instruct	high school	35	couldn help	225
	best friend	35	felt like	116
	living room	34	living room	94
	birthday party	32	best friend	85
	felt like	31	coffee shop	74
Phi-3-Mini-4K	living room	25	felt sense	131
	grocery store	22	living room	119
	cup coffee	15	couldn help	115
	walk park	15	end day	109
	deep breath	14	floor cleaner	93
Qwen2.5-3B-Instruct	yesterday evening	84	felt like	288
	yesterday morning	28	yesterday evening	286
	dinner tonight	19	yesterday morning	184
	work today	18	yesterday afternoon	112
	late appointment	15	feels like	92
Falcon-7b	walked away	30		
	deep breath	27		
	gravity situation	25		
	sight behold	23		
	realized gravity	18		
Llama-2-7b-chat	home work	23		
	late work	21		
	finished homework	19		
	video games	19		
	walked away	16		

Table 6: Top 5 Bigrams and Their Frequency for Each LLM Model (Single Sentence and Five Sentence Datasets). Blue bigrams are common across multiple models, while red bigrams are specific to a particular model.

5.2.2 Topic Modeling

Topic modeling is a statistical model for discovering abstract "topics" that occur in a collection of documents. This technique is useful for uncovering hidden semantic structures in large text data. To delve deeper into the thematic proclivities of each LLM, we employed Latent Dirichlet Allocation (LDA) for topic modeling [30]. LDA is a generative probabilistic model that assumes documents are mixtures of topics, and topics are probability distributions over words. This method allowed us to uncover the underlying thematic structure of the LLM outputs. By treating the entire dataset as a unified corpus, we derived common topics that span the outputs of all LLMs. Subsequently, we calculated the average topic distribution for each model to discern patterns in vocabulary usage and thematic focus. The results, based on the five-sentence dataset, revealed three primary topics as shown in Table 7.

Topic	**Keywords**
Topic 1	home, yesterday, friend, today, soon, just, work, quickly, time, like
Topic 2	clear, people, challenges, sense, life, felt, experience, mind, despite, ultimately
Topic 3	help, make, home, family, work, friends, decided, day, time, new

Table 7: Keywords for each topic derived from LDA analysis.

The average topic distribution for each LLM is presented in Table 8. These results illuminate distinct thematic tendencies among the LLMs. Claude-3-haiku and GPT-4o-mini exhibit a stronger affinity for Topic 2, which encompasses abstract concepts and emotional states. This predilection for more nuanced and introspective content aligns with the sophisticated capabilities often attributed to these models. In contrast, Qwen2.5-3B-Instruct shows a marked preference for Topic 1, which is characterized by everyday activities and temporal references. This bias towards more concrete, time-bound narratives may reflect the model's training focus or inherent limitations. Interestingly, Llama-3.2-3B-Instruct and Phi-3-Mini-4K demonstrate a more balanced distribution across topics, with a slight inclination towards Topic 3. This topic, centered around social interactions and decision-making, suggests these models may excel at generating content that blends personal narratives with broader societal themes.

LLM	**Topic 1**	**Topic 2**	**Topic 3**
Claude-3-haiku	0.1637	0.3265	0.2452
GPT-4o-mini	0.1411	0.2917	0.2239
Llama-3.2-3B-Instruct	0.2731	0.1707	0.2965
Phi-3-Mini-4K	0.1854	0.2135	0.3061
Qwen2.5-3B-Instruct	0.3938	0.1321	0.2755

Table 8: Average topic distribution for each LLM.

The variation in topic distribution across LLMs reflects their distinctive generative tendencies. This diversity likely contributes to our classifier's improved performance when using the five-sentence dataset, as evidenced by the increase in accuracy from 0.6074 to 0.8844. Longer text samples appear to provide more pronounced thematic patterns, enabling more effective differentiation between LLMs. This finding highlights the importance of using extended text samples for accurate LLM attribution, as they better capture each model's unique characteristics.

6 Related Work

Recent studies have explored various approaches to text classification using LLMs and deep learning techniques. This section reviews notable works and highlights how our approach differs and builds upon existing research.

6.1 Deep Learning for Text Classification

Alqahtani et al. [31] presented an efficient approach for textual data classification using deep learning, specifically employing LSTM networks. Their work achieved 0.92 accuracy for binary classification tasks. While this demonstrates the efficacy of LSTM for structured classification, our approach extends to multi-class classification, offering broader application and comparison between different LLMs.

Recent advancements in deep learning approaches include Zhou et al.'s [32] optical character recognition and classification method for cigarette laser code recognition, using a convolutional recurrent neural network and BiLSTM for text classification. Raza et al. [33] proposed the Nbias framework for detecting and eliminating biases in text recognition. Zhou et al. [34] introduced a semi-supervised generative adversarial learning method to improve classification performance with limited annotated data. Jamshidi et al. [35] developed a hybrid model combining BERT, LSTM, and Decision Templates for IMDB and Drug Review classification. Our work performs more precise classification by specifically identifying the source of generated text, rather than focusing on simpler binary classification tasks.

6.2 Direct Application of LLMs in Text Classification

Wang et al. [36] proposed a "Smart Expert System" that utilizes LLMs as text classifiers. While their abstract suggests multi-label classification capabilities, the actual implementation focused on binary classification tasks. Their approach demonstrates the potential of LLMs for classification tasks, primarily relying on zero-shot or few-shot learning. However, they identified several limitations when using LLMs directly for classification, including lack of consistency in output formats, limitations on the types of content that can be classified, and time-consuming processing.

Recent studies have begun to explore the practical applications of LLMs in specialized fields. For instance, Chae & Davidson [37] study the application of LLMs in sociological text classification, demonstrating their potential in social science research. Loukas et al. [38] examine the performance and cost trade-offs when employing LLMs for text classification, focusing on financial intent detection datasets. Wei et al. [39] investigates the effect of fine-tuning LLMs on text classification tasks within legal document review, highlighting how domain-specific adjustments can enhance model performance.

In contrast, our work focuses on developing a dedicated classifier to distinguish texts from various LLMs, addressing these limitations and allowing for more nuanced multi-class categorization.

6.3 AI-Generated Text Detection

Prova [40] explored the use of BERT architecture along with Support Vector Machine (SVM) and XGBoost classifiers for distinguishing between AI-generated and human-written texts. This study aligns closely with our work in detecting machine-generated texts. However, our research extends this approach by utilizing the more advanced DeBERTaV3 architecture, suggesting the potential for improved performance in this domain. Furthermore, our classifier broadens the scope to include text originating from multiple LLMs, giving it a wider utility range and the ability to distinguish between various AI-generated texts, not just between AI and human-written content.

6.4 Extreme Multi-Label Classification

Zhang et al. [41] addressed the challenge of extreme multi-label text classification using a multi-resolution transformer fine-tuning approach. While our current work does not tackle such extreme scenarios, their research points to future directions where our deep learning classifier could potentially be adapted. The recursive methodology introduced in this study might be valuable for future iterations of our classifier, particularly in handling more nuanced classification tasks, such as recognizing subtle stylistic variations between outputs of very similar LLMs.

In summary, our work builds upon these recent advancements in text classification by focusing on multi-class classification tasks specifically aimed at distinguishing between texts generated by different LLMs. We address limitations identified in direct LLM classification approaches, such as inconsistencies in outputs and lengthy processing times. By leveraging advanced architectures like DeBERTaV3, our method aims to demonstrate enhanced potential across a broader range of text classification scenarios, including more accurate detection of AI-generated content from various sources. Furthermore, our research lays the groundwork for potential future applications in more complex classification tasks, such as extreme multi-label classification or fine-grained stylistic analysis of LLM outputs.

7 Conclusion

This project has successfully developed and evaluated a novel approach for attributing text completions to their respective LLMs. Our main contributions include the construction of a comprehensive dataset using truncated text inputs from the BookCorpus and completions generated by seven different LLMs, providing a robust foundation for training and evaluation. We designed and implemented the DeBERTaV3Classifier, a deep learning-based model leveraging the advanced DeBERTaV3 architecture, multi-head attention mechanisms, and state-of-the-art optimization techniques.

Our performance evaluation demonstrated the classifier's effectiveness, achieving an accuracy of 0.5478 and a micro-F1 score of 0.5396 on the primary dataset. Moreover, further experiments on alternative datasets (BBC-text Multi-Class and Tweet Emotion) validated the model's robustness and generalizability. We conducted an in-depth analysis by extending the generation of text completions to five sentences, which significantly improved performance (accuracy of 0.8844 and micro-F1 score of 0.8838), highlighting the importance of context in LLM attribution tasks. Additionally, our LLM characteristic analysis, utilizing N-gram analysis and topic modeling techniques, provided valuable insights into the unique generative tendencies and thematic preferences of each model, further elucidating the factors contributing to successful attribution. Through the discussion of related works, these findings contribute to the broader field of LLM research, text classification, and model attribution. Also, the DeBERTaV3Classifier shows promise for applications in AI-generated text detection and could potentially be adapted for (extreme) multi-label classification tasks.

Our findings contribute to the growing field of model attribution, offering a tool that can aid in ensuring AI accountability and detecting AI-generated content in various applications. Future work may explore expanding the range of LLMs included in the dataset, refining classification techniques, and applying this approach to real-world scenarios such as plagiarism detection in academic writing and authenticity verification in media.

References

[1] O. J. Achiam, S. Adler, S. Agarwal, *et al.*, “Gpt-4 technical report,” 2023.

[2] J. Devlin, M.-W. Chang, K. Lee, *et al.*, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186.

[3] Y. Zhu, R. Kiros, R. Zemel, *et al.*, “Aligning books and movies: Towards story-like visual explanations by watching movies and reading books,” in *The IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015.

[4] K. Clark, M.-T. Luong, Q. V. Le, *et al.*, *Electra: Pre-training text encoders as discriminators rather than generators*, 2020. arXiv: `2003.10555 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2003.10555`.

[5] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit.* " O’Reilly Media, Inc.", 2009.

[6] N. Reimers and I. Gurevych, “Sentence-bert: Sentence embeddings using siamese bert-networks,” in *Conference on Empirical Methods in Natural Language Processing*, 2019.

[7] “The claude 3 model family: Opus, sonnet, haiku. ”[Online]. Available: `https://api.semanticscholar.org/CorpusID:268232499`.

[8] J. Zuo, M. Velikanov, D. E. Rhaiem, *et al.*, “Falcon mamba: The first competitive attention-free 7b language model,” 2024.

[9] T. Wolf, L. Debut, V. Sanh, *et al.*, *Huggingface’s transformers: State-of-the-art natural language processing*, 2020. arXiv: `1910.03771 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/1910.03771`.

[10] A. Yang, B. Yang, B. Hui, *et al.*, “Qwen2 technical report,” *arXiv preprint arXiv:2407.10671*, 2024.

[11] J. Su, Y. Lu, S. Pan, *et al.*, *Roformer: Enhanced transformer with rotary position embedding*, 2023. arXiv: `2104.09864 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2104.09864`.

[12] P. Ramachandran, B. Zoph, and Q. V. Le, *Searching for activation functions*, 2017. arXiv: `1710.05941 [cs.NE]`. [Online]. Available: `https://arxiv.org/abs/1710.05941`.

[13] N. Shazeer, *Glu variants improve transformer*, 2020. arXiv: `2002.05202 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/2002.05202`.

[14] J. Ainslie, J. Lee-Thorp, M. de Jong, *et al.*, *Gqa: Training generalized multi-query transformer models from multi-head checkpoints*, 2023. arXiv: `2305.13245 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2305.13245`.

[15] M. Abdin, J. Aneja, H. Awadalla, *et al.*, *Phi-3 technical report: A highly capable language model locally on your phone*, 2024. arXiv: `2404.14219 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2404.14219`.

[16] OpenAI, J. Achiam, S. Adler, *et al.*, *Gpt-4 technical report*, 2024. arXiv: `2303.08774 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2303.08774`.

[17] H. Touvron, L. Martin, K. Stone, *et al.*, *Llama 2: Open foundation and fine-tuned chat models*, 2023. arXiv: 2307.09288 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2307.09288.

[18] Meta, *Llama 3.2*, Oct 2, 2024, 2024. [Online]. Available: https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_2.

[19] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16, San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 785–794.

[20] Z. Huang, W. Xu, and K. Yu, *Bidirectional lstm-crf models for sequence tagging*, 2015. arXiv: 1508.01991 [cs.CL].

[21] J. P. Chiu and E. Nichols, "Named entity recognition with bidirectional LSTM-CNNs," *Transactions of the Association for Computational Linguistics*, vol. 4, pp. 357–370, 2016.

[22] P. He, J. Gao, and W. Chen, *Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing*, 2023. arXiv: 2111.09543 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2111.09543.

[23] Y. Liu, M. Ott, N. Goyal, *et al.*, *Roberta: A robustly optimized bert pretraining approach*, 2019. arXiv: 1907.11692 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1907.11692.

[24] Z. Yang, Z. Dai, Y. Yang, *et al.*, *Xlnet: Generalized autoregressive pretraining for language understanding*, 2020. arXiv: 1906.08237 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1906.08237.

[25] T. Akiba, S. Sano, T. Yanase, *et al.*, *Optuna: A next-generation hyperparameter optimization framework*, 2019. arXiv: 1907.10902 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1907.10902.

[26] I. Loshchilov and F. Hutter, *Decoupled weight decay regularization*, 2019. arXiv: 1711.05101 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1711.05101.

[27] A. Kachkach, "Problem-solving with ml: Automatic document classification," Oct 6, 2024, 2018. [Online]. Available: https://cloud.google.com/blog/products/ai-machine-learning/problem-solving-with-ml-automatic-document-classification?hl=en.

[28] E. Saravia, H.-C. T. Liu, Y.-H. Huang, *et al.*, "CARER: Contextualized affect representations for emotion recognition," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 3687–3697.

[29] "Text classification on emotion," Oct 6, 2024, 2021. [Online]. Available: https://paperswithcode.com/sota/text-classification-on-emotion.

[30] H. Jelodar, Y. Wang, C. Yuan, *et al.*, *Latent dirichlet allocation (lda) and topic modeling: Models, applications, a survey*, 2018. arXiv: 1711.04305 [cs.IR]. [Online]. Available: https://arxiv.org/abs/1711.04305.

[31] A. Alqahtani, H. Ullah Khan, S. Alsubai, *et al.*, "An efficient approach for textual data classification using deep learning," *Frontiers in Computational Neuroscience*, vol. 16, p. 992 296, 2022.

[32] W. Zhou, L. Zheng, X. Li, *et al.*, "Clcrnet: An optical character recognition network for cigarette laser code," *IEEE Transactions on Instrumentation and Measurement*, vol. 73, pp. 1–11, 2024.

[33] S. Raza, M. Garg, D. J. Reji, *et al.*, "Nbias: A natural language processing framework for bias identification in text," *Expert Systems with Applications*, vol. 237, p. 121 542, 2024, ISSN: 0957-4174.

[34] N. Zhou, N. Yao, N. Hu, *et al.*, "Cdgan-bert: Adversarial constraint and diversity discriminator for semi-supervised text classification," *Knowledge-Based Systems*, vol. 284, p. 111 291, 2024, ISSN: 0950-7051.

[35] S. Jamshidi, M. Mohammadi, S. Bagheri, *et al.*, "Effective text classification using bert, mtm lstm, and dt," *Data and Knowledge Engineering*, vol. 151, p. 102 306, 2024, ISSN: 0169-023X.

[36] Z. Wang, Y. Pang, and Y. Lin, *Smart expert system: Large language models as text classifiers*, 2024. arXiv: `2405.10523 [cs.CL]`.

[37] Y. Chae and T. Davidson, "Large language models for text classification: From zero-shot learning to instruction-tuning," *SocArXiv*, Aug. 2023, Web.

[38] L. Loukas, I. Stogiannidis, O. Diamantopoulos, *et al.*, "Making llms worth every penny: Resource-limited text classification in banking," in *Proceedings of the Fourth ACM International Conference on AI in Finance*, ser. ICAIF '23, Brooklyn, NY, USA: Association for Computing Machinery, 2023, pp. 392–400, ISBN: 9798400702402.

[39] F. Wei, R. Keeling, N. Huber-Fliflet, *et al.*, "Empirical study of llm fine-tuning for text classification in legal document review," in *2023 IEEE International Conference on Big Data (BigData)*, 2023, pp. 2786–2792. DOI: `10.1109/BigData59044.2023.10386911`.

[40] N. Prova, *Detecting ai generated text based on nlp and machine learning approaches*, 2024. arXiv: `2404.10032 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/2404.10032`.

[41] J. Zhang, W.-C. Chang, H.-F. Yu, *et al.*, "Fast multi-resolution transformer fine-tuning for extreme multi-label text classification," in *Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang, *et al.*, Eds., 2021.