From BERT to Custom Embeddings: A Comparative Study on LLM Output Classification

Sandhya Somasundaram, Lakshmi Chandrika Yarlagadda
sms9723, lvy5215
CSE 584 Midterm Project

October 6, 2024

1 Abstract

In this project, we explore the task of classifying Large Language Models (LLM) based on the generated sentence completions for given truncated texts. We utilize a dataset of truncated sentences from the OpenSubtitles corpus and their corresponding completions produced by five different LLMs: GPT-2, Llama 3.2:1b, Tinydolphin:1.1b, Orca-mini, and Mixtral-8x7b-32768. Our approach involves prompting these LLMs to complete the truncated texts, resulting in a diverse dataset of contextually rich completions. We propose and evaluate two deep learning classifiers designed to analyze the input pairs (x_i, x_j)* and accurately predict the LLM responsible for each completion: a fine-tuned BERT model and a neural network using Stella embeddings. The BERT model achieves a test accuracy of 72.62%, while the neural network with Stella embeddings reaches 65.28% accuracy. We conduct a comprehensive error analysis to identify patterns of misclassification and areas for improvement. Our findings contribute to the understanding of LLM behavior and provide insights into distinguishing outputs generated by different models, with implications for applications in different domains.

2 Introduction

Large Language Models (LLMs) have profoundly influenced natural language processing (NLP), enabling more accurate and human-like text generation across a variety of contexts. These models include GPT, Llama and others that are trained on massive datasets, allowing them to learn intricate language structures, word associations, and contextual nuances. While much attention has been paid to improving the fluency and coherence of these models, less focus has been placed on distinguishing the outputs produced by different LLMs. Understanding the unique characteristics of these outputs is crucial for applications that require specific text generation qualities, such as content creation, customer support automation, and data synthesis.

This paper presents a systematic method for classifying the LLM used to complete a truncated text. Given an initial text fragment x_i like "Yesterday I went," different LLMs may generate distinct completions, such as "to Costco and purchased a floor cleaner" or "to the park and enjoyed a picnic." These completions, while

* we use xi, xj interchangeably with truncated_data, generated_data

valid, reflect the unique tendencies of each model based on factors like its architecture, dataset, and training objectives. Our objective is to design a deep learning classifier capable of identifying which LLM generated a specific completion x_j for a given input x_i.

3 Related Work

3.1 Sentence Completion with Large Language Models and Prompt Engineering

Sentence completion using LLMs has seen significant advancements, particularly with models like GPT-2, GPT-3, and other transformer-based architectures. These models are trained on large datasets to predict the next token in a sequence, allowing them to generate coherent and contextually appropriate sentence completions. The work, *"Attention Is All You Need"* by Vaswani et al., introduced the transformer model, which is now foundational for modern LLMs. In sentence completion tasks, models like GPT-3 excel at producing plausible sentence continuations given truncated inputs, showing superior performance in various natural language generation tasks [1].

Prompt engineering has emerged as a crucial technique to optimize the quality of generated completions. By crafting precise and task-specific prompts, users can guide LLMs toward generating relevant and contextually appropriate responses. Brown et al., in *"Language Models are Few-Shot Learners"*, highlighted how LLMs like GPT-3 could be fine-tuned using strategically designed prompts to excel in sentence generation and completion with minimal supervision [2]. Advances in prompt design, such as task-specific prompting or iterative prompting techniques, enable these models to follow complex instructions with high accuracy, enhancing their application in tasks like dialogue generation, summarization, and creative writing [3].

3.2 Text Classification with Large Language Models and Neural Networks

Text classification has significantly benefitted from LLMs like BERT, GPT, and RoBERTa, which can capture intricate contextual dependencies within text. BERT, introduced in *"BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding"* by Devlin et al., provides a robust method for text classification by pre-training on bidirectional representations, making it highly effective in tasks like sentiment analysis, spam detection, and topic categorization [4]. RoBERTa, a variation of BERT, enhances classification performance through optimized pretraining techniques, as seen in the work by Liu et al. [5].

Recent advancements in leveraging LLMs for text classification have drawn substantial attention. Wang et al. (2024), in their *Smart Expert System*, propose the direct application of LLMs for text classification, simplifying the traditional process that usually requires significant preprocessing and domain expertise. This approach aligns closely with our project's aim of classifying text completions generated by different LLMs. Their findings demonstrate that certain LLMs outperform traditional machine learning and neural network models in several text classification tasks, including sentiment analysis and multi-label classification [6].

3.3 LLMDet

The paper "LLMDet: A Third Party Large Language Models Generated Text Detection Tool" presents a relevant approach for the project of identifying which LLM generated a specific text completion. LLMDet is

designed to detect and attribute text to specific large language models, including GPT-2, OPT, and LLaMA, without requiring direct access to the models themselves. The tool uses a novel method of recording next-token probabilities of salient n-grams to calculate a proxy perplexity for each LLM, which is then used to determine the source of the generated text. This approach could be adapted for the project's task of classifying which LLM completed a given truncated text. While LLMDet focuses on full text attribution, its underlying principles of model-specific detection and the use of proxy perplexities could inform the development of a deep learning classifier for the project's paired input (xi, xj) classification task[12].

3.4 Sentence Embeddings

Sentence Transformers are transformer-based models specifically designed to generate high-quality embeddings for sentences. These embeddings are dense, fixed-size vectors that encapsulate the semantic meaning of entire sentences rather than individual words. Unlike traditional word embeddings, such as Word2Vec and GloVe, Sentence Transformers excel at tasks involving sentence-pair relationships, including semantic similarity, entailment, and multi-class classification. The use of Siamese and triplet networks, as demonstrated by Reimers and Gurevych in their paper on Sentence-BERT, enhances these models' performance on tasks that require understanding the relationships between multiple sentences [7].

The architecture of Sentence Transformers enables them to map entire sentences into a multi-dimensional continuous vector space, where semantically similar sentences are located closer together. This property is critical for sentence classification, as the model can efficiently process semantic information. The dense representations facilitate improved accuracy in tasks such as sentence similarity and text classification. Research has shown that these embeddings significantly outperform traditional embeddings in various downstream NLP tasks [7, 4].

Additionally, the **Massive Text Embedding Benchmark (MTEB)** has emerged as a vital framework for evaluating text embeddings across multiple tasks. MTEB highlights the versatility of modern text embeddings and their ability to generalize across various applications, such as information retrieval, classification, and clustering [10]. Hugging Face's Transformers library has made it easier for researchers and practitioners to access and fine-tune embedding models, thus promoting the creation of high-quality sentence embeddings that significantly enhance performance in various NLP tasks [11]. The advances in sentence embeddings and their evaluation through frameworks like MTEB have made a substantial impact on the accuracy of tasks that depend on understanding the relationships between sentences [10, 11].

4 Dataset Curation

4.1 Input data source

As part of this project, we utilized the **OpenSubtitles** dataset from Hugging Face, to extract a diverse collection of truncated sentences. The OpenSubtitles dataset is a repository which contains subtitles from Movies and TV shows. Usage of such colloquial truncated sentences would be a good start to generate different responses from various models. These subtitles reflect informal, conversational speech patterns that are often missing in more formal datasets. To ensure that our data was diverse, we focused on extracting approximately 4,000 sentences from the English-Hindi translation subset.

The OpenSubtitles dataset was chosen not only for its huge size but also for the richness in the types of language it covers—from everyday conversations to complex, context-rich dialogues. This is particularly

useful for training and evaluating large language models (LLMs) to ensure different contexts and nuances of the english language is covered.The diversity of this dataset was a key factor in ensuring that the LLMs we used would be tested on a wide range of sentence types and structures.

4.2 Creation of truncated data

To generate the truncated sentences, we applied several pre-processing steps to the original OpenSubtitles dataset. First, we filtered the English sentences to select only those that contained more than 10 words, ensuring that the sentences provided sufficient context for meaningful completion.

Once the sentences were filtered, they were truncated either at the midpoint or, in cases where the midpoint exceeded 10 words, at the 10-word limit. This process ensured that all selected sentences were cut off in a way that left them incomplete but still coherent, thereby simulating a natural scenario where sentence completion is necessary.

In order to refine the data further, we used a Python script to select the sentences that began with capital letters, as this indicates that they are more likely to represent the beginning of a statement. Additionally, any sentences containing only alphanumeric characters or punctuation were included in the dataset to eliminate noise and irrelevant data. Once the sentences had been filtered, we checked for duplicates, which were removed to ensure that the final dataset contained only unique inputs. These processed and truncated sentences form the truncated_data column of our dataset, providing a diverse and challenging set of prompts for the LLMs to complete.

4.3 Prompt design for sentence completion

After obtaining the truncated texts truncated_data, the next step was to generate their corresponding completions generated_data using a selection of five distinct LLMs. For this project, we carefully selected models that would offer a variety of outputs, based on differences in model architecture, training data, and size. The models included in this task were:

- GPT-2
- Llama 3.2:1b
- Tinydolphin:1.1b
- Orca-mini
- Mixtral-8x7b-32768

These models were accessed through Groq, a third-party API that facilitated efficient interaction with multiple LLMs.

Each model was tasked with completing the truncated sentences using a standardized prompt: *"Complete this in one sentence. Please provide only the remaining half of the sentence as output, without additional information, and use generic information instead of real-time data."* This prompt was carefully designed to ensure consistency across all the models. By restricting the completions to a single sentence and instructing the models to avoid drawing on real-time data, we aimed to reduce variability in the task instructions and focus solely on the models' internal knowledge and language capabilities.

4.4 Data processing

Once the LLMs generated the completions, we conducted a thorough post-processing step to ensure the quality and relevance of the output. During this process, we identified any sentence pairs where the completions generated_data were irrelevant, incoherent, or contained additional information that deviated from the original prompt. Some models, for instance, included extra phrases such as "Note: This could be..." or "Here is the completed version," which were unnecessary and not part of the original sentence completion task.

To address this, we implemented a Python script that automatically detected and removed such extraneous text. In cases where the completions included these unnecessary additions, we regenerated the prompt by rewriting it to strictly match the desired format. The revised prompts ensured that the models would complete the sentences without including any irrelevant content, focusing purely on providing the missing portion of the sentence. This step was crucial for maintaining consistency across all outputs and ensuring that each completion truncated_data adhered to the original instruction to provide only the remaining part of the sentence, without added commentary or explanations. This refined approach allowed us to discard irrelevant completions and improve the overall quality of the dataset.

Ultimately, for every truncated sentence truncated_data in our dataset, we ensured that there were five corresponding completions generated_data, one generated by each of the selected LLMs. This resulted in a balanced dataset that provided multiple completions for the same input, allowing for robust comparisons across models.

4.5 Dataset Description

The final curated dataset is composed of three columns, each capturing a different aspect of the sentence completion task:

- `truncated_data`: This column contains the truncated sentences (x_i, named as truncated_data) that were presented to the LLMs as input prompts. These sentences are typically cut off at the halfway point or after 10 words, and represent incomplete statements that require completion.
- `generated_data`: This column holds the sentence completions (x_j, named as generated_data) generated by the LLMs. These completions were produced in response to the truncated sentences and provide a wide range of possible continuations based on the model's training.
- `LLM`: This column records the name of the specific language model that generated each completion, allowing us to easily identify and compare outputs from different models.

The final dataset contains approximately 21,550 records, with each of the five Large Language Models (LLMs) contributing around 4,310 sentence completions. To prepare the dataset for training and testing, we split the data in an 80-20 ratio, dedicating 80% to training and 20% to testing. Special attention was given to maintaining class balance in both datasets, ensuring that the distribution of different classes remains consistent. Additionally, we implemented measures to ensure that none of the training inputs truncated_data appear in the test set, thereby preventing any leakage of information from the training phase into the evaluation phase. This strict separation reinforces the integrity of the testing process and supports more reliable performance metrics.

5 Model Architecture

5.1 BERT

BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based model designed to understand the context of words in a sentence through bidirectional training. It captures both left and right context in all layers, making it powerful for understanding sentence relationships.

The key feature of BERT is its ability to generate contextualized embeddings, where the same word can have different representations depending on the surrounding text. This property is essential for sentence-pair classification tasks such as textual entailment, semantic similarity, or multi-class classification, which require a deep understanding of sentence context.

For this project, we used the `bert-base-uncased` model from Hugging Face, a pre-trained version of BERT that has 12 transformer layers and a hidden size of 768. To optimize BERT for sentence classification, we fine-tuned it on our specific dataset of truncated and generated sentence pairs.

5.1.1 Input Processing and Tokenization

Before passing the data through the BERT model, several preprocessing steps are performed:

1. **Data Preparation**: The dataset used contains pairs of sentences, where one sentence is truncated text, and the other is generated text from LLMs. Each sentence pair is labeled with the name of the LLM that generated it. We split the dataset into training and test sets using an 80/20 ratio.
2. **Label Encoding**: The LLM labels are converted into numeric form using the `LabelEncoder` from the `sklearn` library, preparing the data for multi-class classification.
3. **Tokenization**: The BERT tokenizer processes each pair of sentences. The tokenizer uses the following features:
   - Adds special tokens, including `[CLS]` and `[SEP]` to mark sentence boundaries.
   - Pads or truncates sentences to a maximum sequence length of 128 tokens.
   - Returns attention masks to indicate which tokens are padding and which are actual words.

   These tokenized sentences are then converted to tensor format, enabling compatibility with TensorFlow for training the model.

5.1.2 Architecture

The architecture is built on top of the pre-trained `bert-base-uncased` model. The following components are used to adapt BERT to this task:

Input Layer: Tokenized sequences (sentences) are passed through BERT's tokenizer, generating input IDs, attention masks, and token type IDs.

BERT Model Layers: The core architecture consists of the BERT Transformer Encoder with:

- Multi-Head Self-Attention Mechanism: Computes attention scores for tokens in the sequence.
- Feedforward Neural Network (FFNN): Refines token representations using a two-layer fully connected network.
- Layer Normalization & Residual Connections: Stabilize and maintain gradients after each attention and FFNN layer.

[CLS] Token Representation:

- The [CLS] token, added at the start of the sequence, represents the entire sequence.
- The output of the [CLS] token from the final BERT layer is fed into the classification head.

Classification Head:

- A fully connected layer (size equal to the number of classes) processes the [CLS] token output.
- Softmax is used as activation function to generate class probabilities.

Loss Function: Cross-Entropy Loss compares predicted class probabilities with actual labels.

Optimizer: Adam optimizer minimizes the loss and updates model weights.

5.2 Neural Net with Stella Embeddings

To classify the text completions generated by LLMs we are using embeddings generated from the **Sentence-Transformer** model. These embeddings capture the semantic meaning of entire sentences, enabling effective classification of sentence pairs produced by various large language models (LLMs). Specifically, we utilize the `stella_en_400M_v5` model from Hugging Face, which excels in classification tasks.

Sentence Transformers are transformer-based architectures that generate dense, fixed-size embeddings for sentences, encapsulating their semantic meanings rather than focusing on individual words. Unlike traditional word embeddings (e.g., Word2Vec, GloVe), Sentence Transformers are tailored for tasks involving sentence-pair relationships, such as semantic similarity and entailment. These embeddings map sentences into a multi-dimensional vector space, where semantically similar sentences are positioned closer together, facilitating effective classification through a compact representation of semantic information.

In this project, we utilized the `stella_en_400M_v5` model developed by Dun Zhang, which is optimized for efficient and accurate sentence classification tasks. This model provides robust embeddings that effectively capture semantic relationships between sentences, achieving around 86% accuracy in text classification tasks. Notably, it has been ranked among the top 10 models in the mTEB (Massive Text Embedding Benchmark) dashboard, highlighting its competitive performance in various NLP applications. By choosing the smallest 400M variant, we were able to maintain computational efficiency while still generating high-quality embeddings, making it a practical choice for our classification tasks.

5.2.1 Input Processing

Before feeding the data into the classification model, several preprocessing steps are performed:

1. **Embedding Generation**: For each sentence pair, the `stella_en_400M_v5` model generates embeddings. The model processes sentences in batches to optimize memory and computational efficiency. Each sentence is transformed into a dense vector, and the embeddings for the two sentences in each pair (`xi` and `xj`) are concatenated to form a single input feature vector.
2. **Concatenation**: The embeddings for each pair of sentences are concatenated along the feature axis, resulting in a combined feature vector of size 2048.

The concatenated feature vectors are then stored and later used as input to the neural network classifier.

5.2.2 Neural Network Architecture

The neural network is designed to process the concatenated sentence embeddings and perform sentence-pair classification. The architecture consists of multiple fully connected layers, as outlined below:

1. **Input Layer**: The concatenated embeddings of size 2048 (1024 for `xi` and 1024 for `xj`) are fed into the neural network.
2. **Hidden Layers**: The network consists of several fully connected hidden layers designed to extract and refine features from the sentence embeddings:
   - **Layer 1**: A dense layer with 1024 units, followed by a ReLU activation function to introduce non-linearity and learn complex patterns in the data.
   - **Layer 2**: A dense layer with 256 units and ReLU activation, further refining the feature space.
   - **Layer 3**: A dense layer with 128 units and ReLU activation, compressing the features into a more compact representation while capturing the most relevant information.
3. **Output Layer**: The final dense layer outputs the number of units equal to the number of LLM classes (the number of distinct models that generated the sentences). A softmax activation function is applied to this layer to convert the raw outputs into probability scores for each class.

The hidden layers utilize the ReLU activation function to introduce the necessary non-linearity, allowing the model to capture complex patterns in the sentence embeddings. The softmax activation in the output layer ensures that the predicted probabilities for each class sum to 1, making the model suitable for multi-class classification.

By leveraging the sentence embeddings generated by the `stella_en_400M_v5` model and utilizing a well-structured neural network, this model effectively classifies sentence pairs based on their relationships. The use of dense sentence embeddings, combined with multiple fully connected layers, allows the model to capture intricate patterns between sentences, resulting in high performance for sentence-pair classification tasks.

6 Experiments

6.1 BERT Model

We trained the model using a the above dataset, fine tuning the pre-trained BERT model to classify the truncated and generated pairs in the dataset. During this fine tuning, the model's weights are updated to reduce the classification error.

Optimizer: We used the Adam optimizer with a small learning rate of 2e-5 to ensure efficient fine-tuning.
Loss Function: The model's performance was improved using Sparse Categorical Cross entropy, a loss function ideal for multi-class classification tasks.

The model was trained for 3 epochs, which provided a good balance between improving the model and preventing overfitting.

6.1.1 Comparison and Analysis of BERT Training across Epochs

Three BERT experiments were conducted over 2, 3, and 4 epochs to evaluate training, validation, and test performance.

Training and Validation Performance Training accuracy improved steadily from 73.90% at 2 epochs to 88.01% at 4 epochs, while validation accuracy peaked at 72.62% after 3 epochs but dropped to 71.74% by the 4th epoch. The increasing validation loss from 0.7363 at 2 epochs to 0.8162 at 4 epochs suggests overfitting in later epochs. Test accuracy followed a similar trend, confirming that the model's generalization declines with additional training.

In the 2-epoch experiment, the model demonstrated balanced performance, with test accuracy (72.04%) closely matching validation results, showing good generalization. By the 3rd epoch, slight overfitting emerged, with an increase in validation loss, yet the test accuracy still remained consistent with validation. By 4 epochs, overfitting became evident, with decreased validation and test accuracy. The same trend can be visualized in Figure 1.

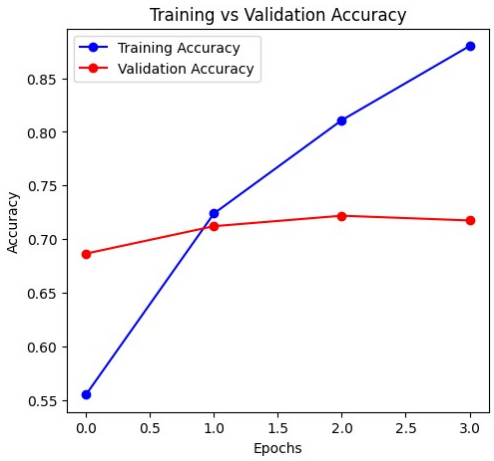

Figure 1: Training vs validation accuracy

6.2 Neural Net with Stella embeddings:

The model is trained to minimize classification loss using the following components:

Optimizer: The Adam optimizer is used for training. Adam is an adaptive optimizer that adjusts learning rates for individual parameters, making it well-suited for complex models like neural networks.
Loss Function: Cross-entropy loss is employed as the loss function for the multi-class classification task.

This loss measures the difference between the predicted class probabilities and the true class labels.

The model is trained over 150 epochs, with each epoch consisting of a forward pass, loss calculation, and backpropagation to update the network weights. During training, the loss is monitored to ensure that the model is converging towards an optimal solution. We are able to achieve an accuracy of **65.28%** with a learning rate if **0.001**.

6.2.1 Effect of hidden layers on the model performance

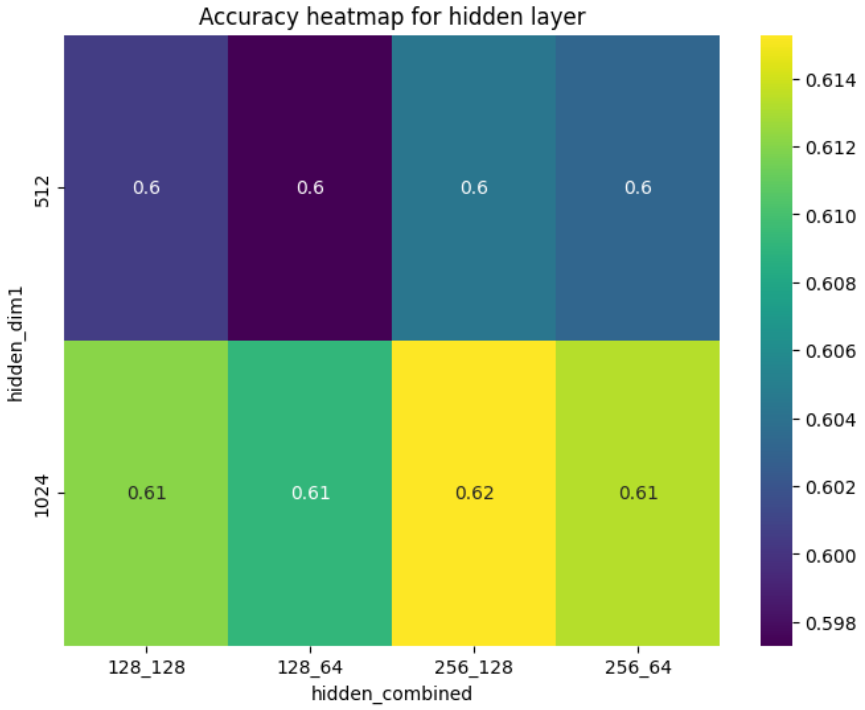

Figure 2: Accuracy heatmap for hidden layer

The heatmap 2 illustrates accuracy as a function of two crucial hyperparameters: **hidden_dim1**, represented on the vertical axis, which indicates the size of the first hidden layer (with values of 512 and 1024), and **hidden_combined**, on the horizontal axis, which combines the sizes of the second and third hidden layers (hidden_dim2 and hidden_dim3). The possible combinations for hidden_combined are labeled as 128_128, 128_64, 256_128, and 256_64. Each cell in the heatmap reflects the accuracy for a specific configuration, with colors denoting accuracy values—yellow signifies higher accuracy, while purple indicates lower accuracy.

Regarding performance trends, models with hidden_dim1 set to 1024 consistently outperform those with a hidden_dim1 of 512, suggesting that a larger first hidden layer enhances the model's capacity to capture complex patterns in the data. The best-performing configurations involve hidden_dim1 = 1024 alongside larger dimensions for hidden_dim2 and hidden_dim3 (e.g., 256_128). Similarly, combinations featuring larger sizes for hidden_dim2 and hidden_dim3 generally yield better performance, with the highest accuracy of 0.62 achieved when hidden_dim1 = 1024, hidden_dim2 = 256, and hidden_dim3 = 128. In contrast, configurations with smaller hidden dimensions, such as 128_128, demonstrate slightly lower accuracy (0.60) when

hidden_dim1 = 512.Thus, larger hidden layer sizes correlate with higher accuracy, particularly for the first hidden layer, as they provide greater capacity for learning complex representations. Finally, smaller hidden layer sizes, like hidden_dim1 = 512 and combinations such as 128_128 or 128_64, yield lower accuracy (around 0.60), implying insufficient capacity to adequately capture the dataset's complexity.

6.2.2 Effect of Learning rate on the model performance

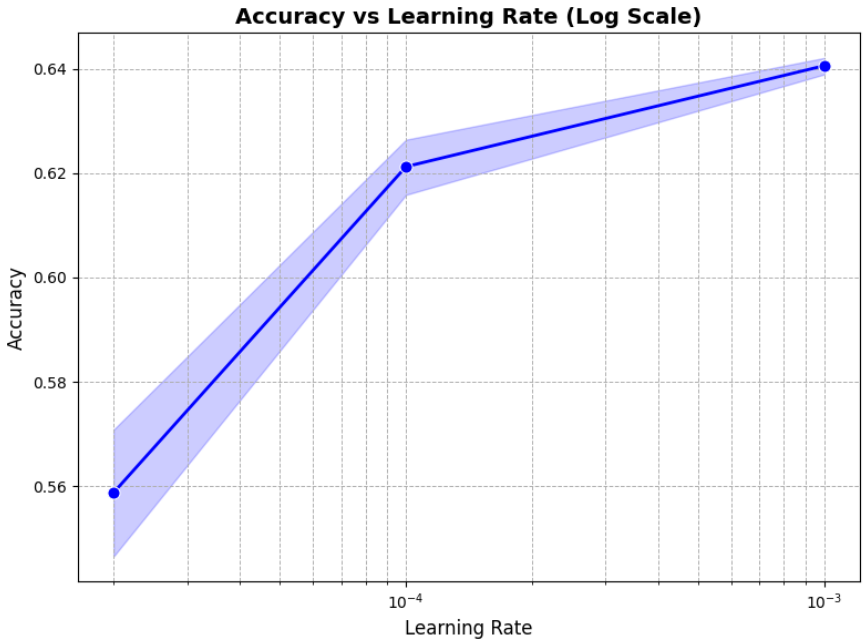

Figure 3: Accuracy vs Learning rate

The plot 3 displays accuracy against learning rate on a log scale, ranging from 10^{-4} to 10^{-3}. The x-axis represents the learning rate, while the y-axis indicates the model's accuracy. As the learning rate increases, accuracy improves, starting at approximately 0.56 for 10^{-4} and peaking at 0.64 for 10^{-3}. The shaded confidence interval illustrates the variability in accuracy across different runs; it is broader at lower learning rates, suggesting instability, and narrows as the learning rate increases, indicating more reliable performance.

From the graph, we can infer that the best accuracy occurs at the highest learning rate of 0.001, reflecting effective convergence. Conversely, the low learning rate of 10^{-4} results in poor performance due to slow convergence, potentially causing the model to get stuck in local minima. Lastly, the narrowing confidence interval with higher learning rates indicates that model performance stabilizes as the learning rate increases, enhancing the optimization process.

6.2.3 Effect of Learning rate and Batch size on the model performance

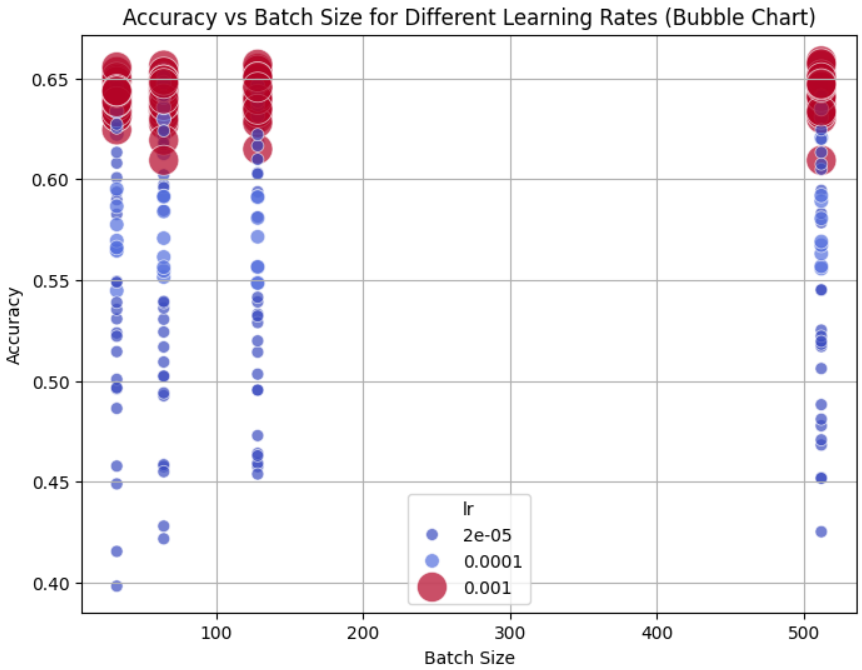

Figure 4: Accuracy vs Batch size for different Learning rates

The bubble chart 4 illustrates accuracy as a function of batch size across different learning rates. The x-axis denotes batch sizes (32, 64, 128, and 512), while the y-axis shows the corresponding accuracy. The color and size of the bubbles indicate learning rates: red bubbles (larger size) represent the highest learning rate (0.001), and blue bubbles (smaller size) correspond to lower rates (0.0001 and 2e-05).

From the plot, we can observe that the highest learning rate (0.001) yields the best accuracy, peaking at around 0.65 with smaller batch sizes (32 and 64). As batch size increases to 128 and 512, accuracy declines slightly but remains superior to that of lower learning rates. In contrast, the lower learning rates generally result in lower accuracy, with the smallest learning rate (2e-05) performing the poorest. For batch sizes 32 and 64, lower rates achieve moderate accuracy (0.55–0.60), but do not match the performance of higher rates.

This indicates that the best performance increases by combining a high learning rate (0.001) with small batch sizes (32 or 64). Conversely, lower learning rates (0.0001 and 2e-05) are less effective, especially with larger batch sizes, suggesting they do not provide sufficient weight updates per epoch for effective learning.

6.2.4 Effect of Learning rate and Epochs on the model performance

This plot 5 illustrates the relationship between accuracy and epochs across different learning rates. The x-axis represents epochs (ranging from 50 to 500), while the y-axis shows model accuracy. Three lines indicate the learning rates: blue for 2e-05, orange for 0.0001, and green for 0.001, with shaded regions denoting confidence intervals.

Figure 5: Accuracy vs Epoch for different Learning rates

The learning rate of 2e-05 (blue line) starts with the lowest accuracy at around 0.45 for 50 epochs, gradually reaching approximately 0.60 by 500 epochs, with a narrow confidence interval indicating consistent performance. In contrast, the learning rate of 0.0001 (orange line) improves steeply from 0.55 to about 0.65, stabilizing after 100 epochs, suggesting early optimization. The 0.001 learning rate (green line) begins at 0.63 after 50 epochs, peaks at 0.65 by 100 epochs, and maintains this level, showcasing effective convergence.

It can be observed that the 0.001 learning rate is the most effective, achieving high accuracy quickly, while 0.0001 performs well but requires more epochs for similar results. The 2e-05 learning rate shows the slowest improvement, suggesting it is too small for effective updates. For the higher learning rates, accuracy peaks by 100 epochs, indicating potential benefits of early stopping, whereas the lowest learning rate continues to improve but at a slower pace.

6.2.5 Effect of different hyper parameters on the model performance

This pairplot 6 representation features a grid of scatter plots, illustrating pairwise relationships between hyperparameters with accuracy encoded as hue. Each scatter plot depicts the interaction between two hyperparameters, with accuracy visualized through color gradients, ranging from dark purple (lowest accuracy, 0.40) to yellow (highest accuracy, 0.65), while dot sizes remain constant to emphasize color.

Specific interactions reveal that higher values of `hidden_dim1` (1024) correlate with improved accuracy, especially when paired with smaller batch sizes or specific learning rates. Combinations of `hidden_dim2` and `hidden_dim3` at higher values (256 and 128, respectively) also yield better accuracy, while lower dimensions correlate with diminished performance. The learning rate of 0.001 consistently produces the highest accuracy, underscoring its importance for model convergence, whereas lower rates (e.g., 2e-5) are linked to poorer accuracy.

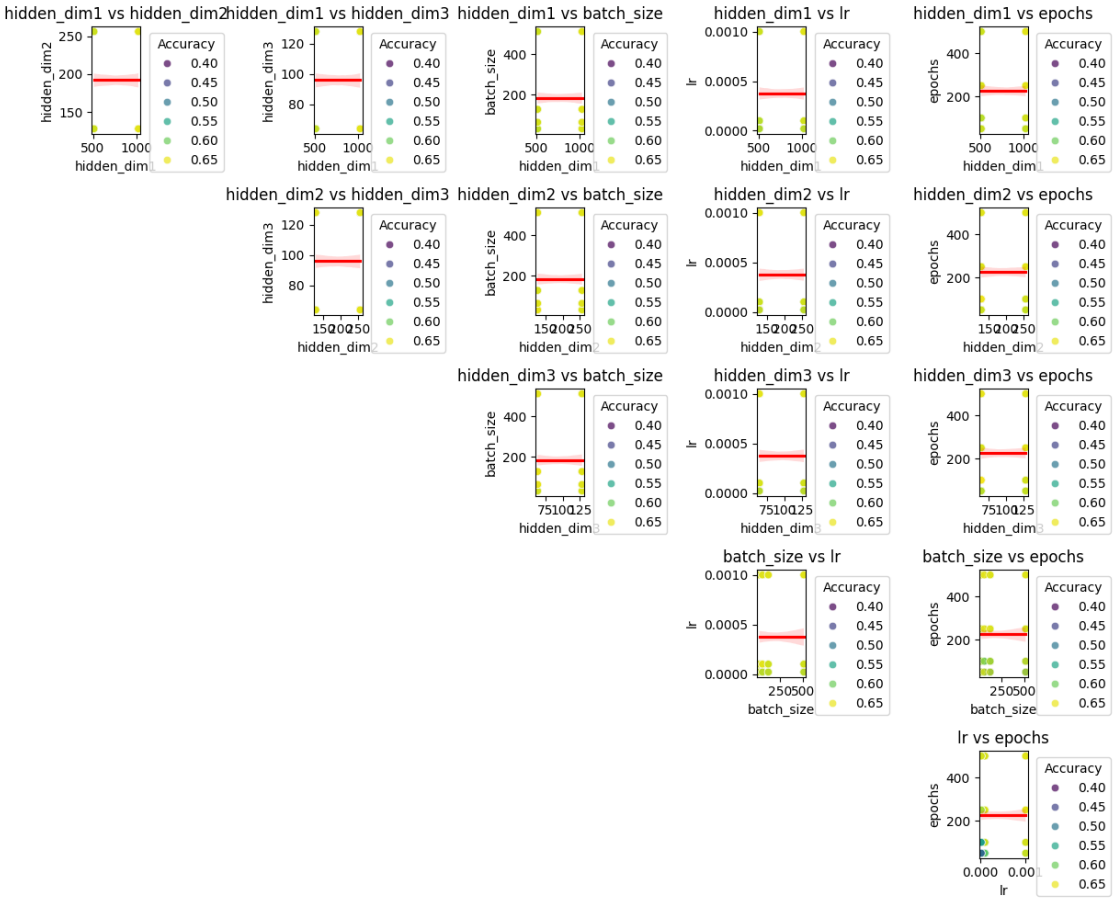

Figure 6: Pair plot showing the relation between hyper parameters

Moreover, smaller batch sizes (32 and 64) combined with larger hidden dimensions and higher learning rates consistently lead to better accuracy. Plots of epochs against `hidden_dim1` and `lr` suggest that configurations around 250 epochs optimize convergence without overfitting.

In summary, the analysis indicates that larger hidden layers, particularly for `hidden_dim1`, and a learning rate of 0.001 enhance performance, while smaller batch sizes prove more effective for training. Future hyperparameter tuning should focus on these optimal configurations, including `hidden_dim1` at 1024, elevated values for `hidden_dim2` and `hidden_dim3`, and maintaining a learning rate of 0.001 with smaller batch sizes. Exploring slight variations in epochs could further refine performance outcomes.

In order to better understand the par plot above, we have represented a correlation matrix that helps us understand the relationship between various hyperparameters. The correlation matrix 7 heatmap illustrates the relationships between hyperparameters and accuracy, with color coding ranging from -1 (dark blue) to 1 (dark red). Diagonal elements show a correlation of 1.00, as expected. `hidden_dim1` has a slight positive correlation (0.10) with accuracy, while `hidden_dim2` and `hidden_dim3` show weak correlations (0.04 and 0.02). The learning rate (`lr`) exhibits a moderate positive correlation (0.46) with accuracy, indicating its importance, while batch size shows no meaningful correlation (0.00). `epochs` has a moderate correlation (0.48) with accuracy, suggesting longer training improves performance. The most influential hyperparameters are the learning rate and epochs, while hidden dimensions and batch size have less impact. Keeping this

in mind, out future tuning focuses on optimizing the learning rate and epochs, using hidden dimensions as secondary adjustments for better model performance.

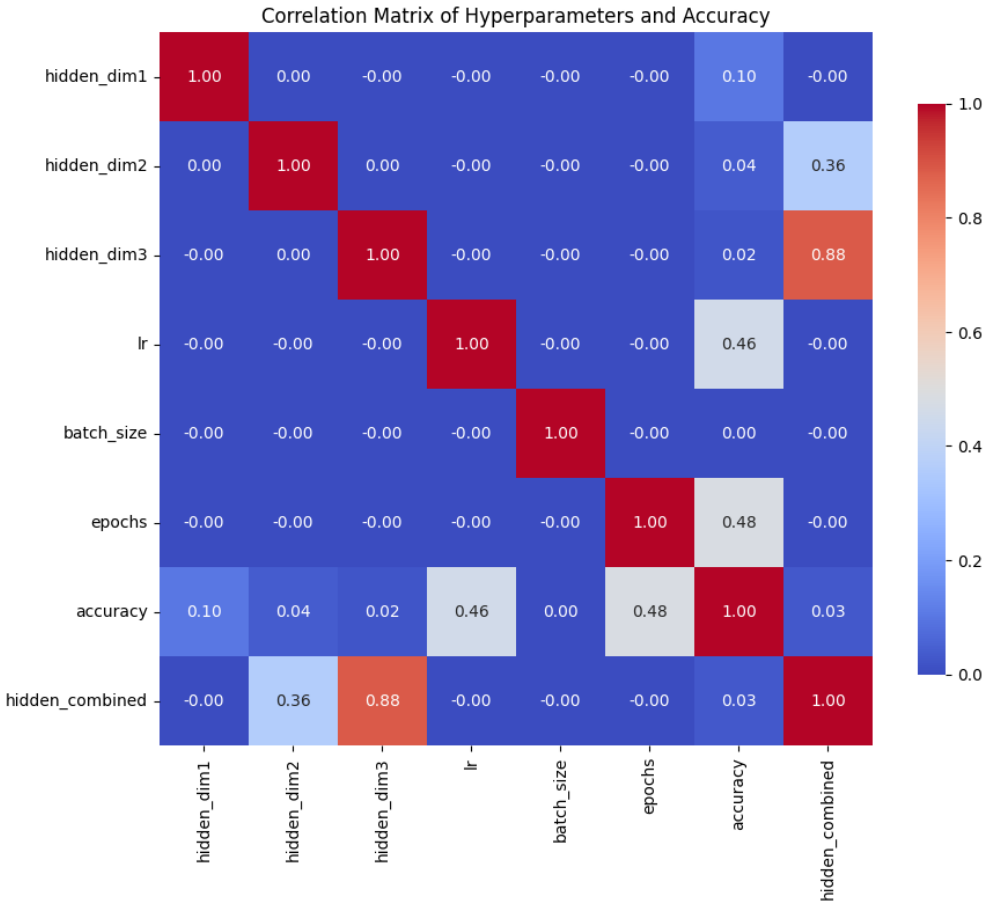

Figure 7: Correlation matrix of hyper parameters and accuracy

7 Results

7.1 BERT

1. Training and Validation Results

The BERT model was fine-tuned over three epochs, with the following results:

Epoch	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
1	1.0375	0.5762 (57.62%)	0.7830	0.6895 (68.95%)
2	0.6866	0.7363 (73.63%)	0.6940	0.7341 (73.41%)
3	0.4756	0.8234 (82.34%)	0.7441	0.7262 (72.62%)

The model shows steady improvement in training accuracy, reaching 82.34% by the third epoch. The validation and test losses remain similar at 0.7441, indicating that the model may have plateaued in its generalization performance.

Final test results show a test loss of 0.7441 and a test accuracy of 72.62%. The final test accuracy is consistent with the validation results, confirming stable performance on unseen data but highlighting room for improvement in generalization.

2. Classification Report

Table 1: Classification Report for BERT

Class	**Precision**	**Recall**	**F1-score**	**Support**
GPT-2	0.61	0.75	0.67	860
Llama3.2:1b	0.81	0.90	0.86	866
Mixtral-8x7b-32768	0.73	0.62	0.67	853
Orca-mini	0.69	0.67	0.68	861
Tinydolphin:1.1b	0.82	0.69	0.75	863
Accuracy		0.73 (4303)		
Macro Avg	0.73	0.73	0.73	4303
Weighted Avg	0.73	0.73	0.73	4303

The overall accuracy is 73% (0.73), with both macro and weighted averages for precision, recall, and F1-score at 0.73.

From 1, We can see that that llama3.2:1b shows the best performance with an F1-score of 0.86, indicating that the model distinguishes this class well. In contrast, classes like gpt-2 and mixtral-8x7b-32768 exhibit lower precision and recall, suggesting confusion between these classes, which is also reflected in the confusion matrix. The balanced macro and weighted averages further indicate consistent performance across classes.

3. Confusion Matrix Analysis

The confusion matrix 8 reveals how the model performs on each class. Diagonal dominance is observed, with the highest number of correct predictions for llama3.2:1b (782 out of 866).

Notable misclassifications include frequent errors between gpt-2 and mixtral-8x7b-32768, suggesting similarity in outputs that the model struggles to differentiate. Similarly, orca-mini and tinydolphin:1.1b exhibit significant cross-misclassification, indicating overlap in their outputs. Overall, the classifier performs reasonably well across classes, with the most misclassification observed between similar models, as highlighted by the confusion matrix.

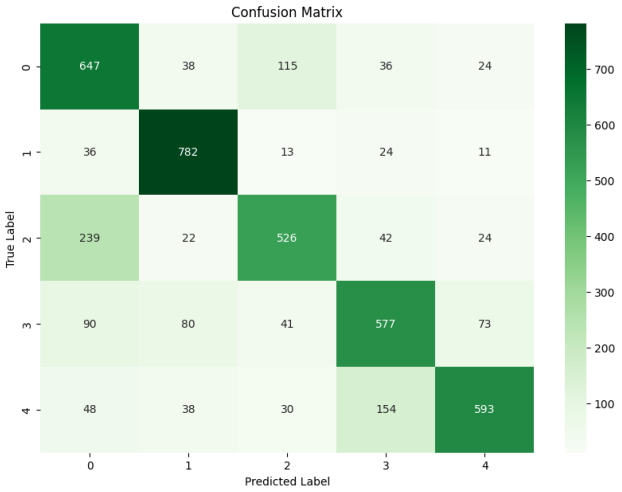

Figure 8: Confusion matrix for Bert

7.2 Neural net with Stella

1. Training and Results

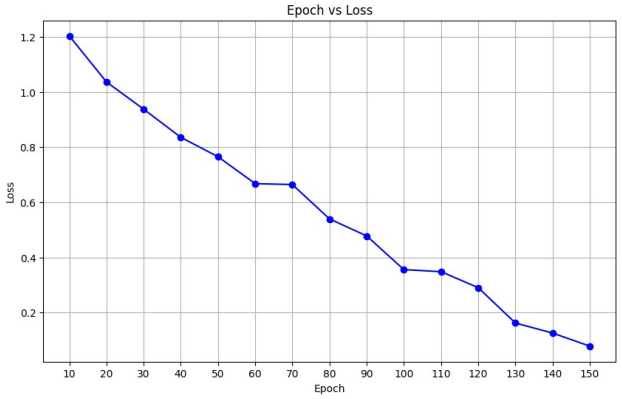

Figure 9: Epoch vs Training loss

The model was trained over 150 epochs, achieving a final test accuracy of 65.28% and we can see a consistent decrease in the loss with the epochs in 9.

2. Classification Report

The classification report for shows precision, recall, and F1-score for each class:

Table 2: Calssfication report for Neural net with Stella embeddings

Class	Precision	Recall	F1-score	Support
GPT-2	0.56	0.58	0.57	860
Llama3.2:1b	0.73	0.78	0.75	866
Mixtral-8x7b-32768	0.62	0.63	0.63	853
Orca-mini	0.60	0.56	0.58	861
Tinydolphin:1.1b	0.73	0.70	0.71	863
Accuracy		0.65 (4303)		
Macro Avg	0.65	0.65	0.65	4303
Weighted Avg	0.65	0.65	0.65	4303

The overall accuracy is 65% (0.65), with both macro and weighted averages for precision, recall, and F1-score at 0.65 each.

It can be inferred from 2 that llama3.2:1b has the highest F1-score (0.75) and performs the best among all classes, indicating that the model identifies this class with greater accuracy. In contrast, gpt-2 shows the lowest precision and recall, leading to a lower F1-score (0.57). This suggests that the model struggles to distinguish gpt-2 outputs from other classes. Tinydolphin:1.1b also performs reasonably well, with balanced precision, recall, and F1-score around 0.71. The balanced macro and weighted averages confirm that the model's performance is consistent across classes but indicates room for improvement, particularly for gpt-2 and orca-mini.

3. Confusion Matrix Analysis

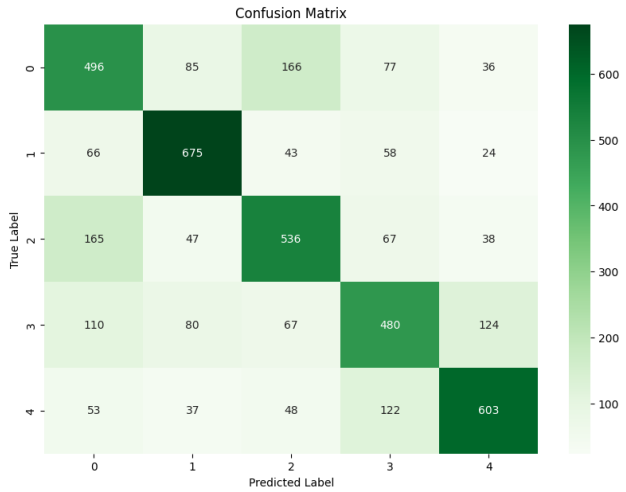

Figure 10: Confusion matrix for Bert

Diagonal dominance is observed in the confusion matrix 10, as the diagonal values show the number of correctly classified instances for each class. Llama3.2:1b shows the highest number of correct predictions (675 out of 866), confirming its better performance.

Common misclassifications reveal that gpt-2 (class 0) is frequently misclassified as mixtral-8x7b-32768 (class 2), with 166 instances misclassified. This indicates that the features of these two classes may overlap, leading to confusion. Orca-mini (class 3) is often misclassified as other classes, with a significant number of instances predicted as tinydolphin:1.1b (class 4). This suggests that the outputs of these two classes may share similarities that the model struggles to distinguish. Additionally, tinydolphin:1.1b (class 4) also shows misclassifications with orca-mini, indicating that these two classes have overlapping characteristics in their embeddings.

The confusion matrix indicates accuracy consistency; the model shows a consistent pattern of correct and incorrect classifications, which aligns with the overall accuracy of 65%. However, there is evident confusion between certain pairs of classes, particularly gpt-2 and mixtral-8x7b-32768, as well as orca-mini and tinydolphin:1.1b.

Class-specific performance suggests that the model performs well for llama3.2:1b, indicating that its features are more distinguishable compared to the other classes. The difficulty in differentiating between gpt-2 and mixtral-8x7b-32768, and between orca-mini and tinydolphin:1.1b, highlights the need for further refinement in feature engineering or data augmentation.

8 Error Analysis

An in-depth error analysis was conducted to examine the specific misclassification patterns observed in the models, helping to identify which classes were most frequently misclassified and why these errors occurred, thereby offering insights for future improvements.

In the case of the BERT model, it generally demonstrated high accuracy but had difficulties distinguishing gpt-2 from mixtral-8x7b-32768. This misclassification indicates that the outputs from these two LLMs may possess similar structural or contextual characteristics, which the BERT model's embeddings struggle to differentiate. Another common misclassification occurred between orca-mini and tinydolphin:1.1b, suggesting these classes share linguistic features that overlap, making accurate separation challenging.

For the neural network with Stella embeddings, significant confusion was observed between gpt-2 and mixtral-8x7b-32768, marking this as the most frequent misclassification in the confusion matrix. This indicates that the Stella embeddings, while beneficial, may lack the specificity required to distinguish subtle differences in outputs generated by these models. Additionally, misclassifications between orca-mini and tinydolphin:1.1b were prominent, highlighting the inherent difficulty in distinguishing these classes without more refined features.

Errors in model performance could stem from several critical factors. Lack of diversity in data might hinder the model's ability to learn essential features, resulting in misclassifications due to insufficient representation of various classes. Overfitting could also be a significant issue, where the model learns noise instead of meaningful patterns, leading to poor generalization on unseen data. Furthermore, the use of low-quality embeddings might prevent the model from effectively distinguishing between classes, impacting overall classification accuracy.

9 Conclusion

This study demonstrates the feasibility and challenges of classifying LLM-generated text completions. The BERT model, trained for three epochs, achieved superior performance with a test accuracy of 72.62% and excelled particularly in classifying outputs from Llama 3.2:1b (F1-score of 0.86). However, it faced difficulties in distinguishing between outputs from GPT-2 and Mixtral-8x7b-32768, as well as between Orca-mini and Tinydolphin:1.1b. The neural network utilizing Stella embeddings, while showing promise, achieved a lower test accuracy of 65.28% after 150 epochs of training, indicating potential overfitting issues.

Our error analysis reveals that both models struggle with similar pairs of LLMs, suggesting inherent similarities in the output patterns of these models that are challenging to differentiate. This highlights the need for more sophisticated feature engineering and possibly the incorporation of additional contextual information to improve classification accuracy.

Future work should focus on refining embedding techniques, particularly for the Stella-based model, to capture more nuanced differences between LLM outputs. Exploring advanced regularization methods, such as L2 regularization or dropout, could enhance the neural network's generalization capabilities. Additionally, investigating other transformer-based architectures or ensemble methods might further improve performance.

The insights gained from this study have significant implications for applications requiring the identification or verification of LLM-generated content. As LLMs become increasingly prevalent in various domains, the ability to accurately classify their outputs will be crucial for tasks such as content moderation, plagiarism detection, and maintaining the integrity of human-authored content.

In conclusion, while our models show promising results in distinguishing between different LLM outputs, there is still room for improvement. This work lays the foundation for future research in LLM output classification and emphasizes the importance of continued development in this rapidly evolving field of natural language processing.

References

[1] Vaswani, Ashish, et al. *Attention Is All You Need*. Advances in Neural Information Processing Systems (NeurIPS), 2017.

[2] Brown, Tom, et al. *Language Models are Few-Shot Learners*. NeurIPS, 2020.

[3] Radford, Alec, et al. *Language Models are Unsupervised Multitask Learners*. OpenAI, 2019.

[4] Devlin, Jacob, et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv, 2018.

[5] Liu, Yinhan, et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. arXiv, 2019.

[6] Wang, J., et al. *Smart Expert System for Text Classification Using LLMs*. Proceedings of the ACL, 2024.

[7] Reimers, Nils, and Iryna Gurevych. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. EMNLP, 2019.

[8] Muennighoff, Nils, et al. *MTEB: Massive Text Embedding Benchmark*. arXiv, 2022.

[9] Hugging Face. (2023). *Stella EN 400M v5*. Retrieved from https://huggingface.co/dunzhang/stella_en_400M_v5.

[10] Thakur, A., et al. (2023). *MTEB: Massive Text Embedding Benchmark*. arXiv preprint arXiv:2303.01645.

[11] Wolf, T., et al. (2020). *Transformers: State-of-the-Art Natural Language Processing*. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations.

[12] *https://ar5iv.labs.arxiv.org/html/2305.15004*

[13] *Github link: https://github.com/sandhyasomasundaram/CSE-584*$_M idtermProject/tree/main$