Group: JAIDEEP KUKKADAPU, AISHWARYA KULKARNI, DIVYA NAVULURI

CSE 584: Machine Learning
Mid-term Project: Creating a classifier to identify which LLM generated a given text

Abstract:

This report presents a deep learning-based approach for classifying text completions generated by different Large Language Models (LLMs). Utilizing the Neural Network model and BERT-based architectures, we aimed to develop a classifier capable of distinguishing between outputs generated by distinct LLMs. The dataset curated for this task consisted of pairs of text completions, which were processed to generate embeddings for classification. The classifier was trained using a cross-entropy loss function, optimized with the Adam optimizer, achieving significant accuracy in distinguishing LLM outputs. The results are analysed in terms of accuracy and loss metrics, demonstrating the efficacy of our approach. Additionally, a comparative analysis with related works is presented, highlighting the advancements and challenges in the domain of text generation classification.

Related Work:

1. **Large Language Models and Text Generation** - Large language models (LLMs) like GPT, BERT, and their successors have revolutionized natural language processing tasks, including text generation. The development of these models has primarily been based on the transformer architecture, introduced by Vaswani et al. (2017) in their influential paper "Attention is All You Need." Transformers rely on self-attention mechanisms, allowing models to process vast amounts of text and generate coherent, human-like completions.

   In recent years, numerous LLMs have been trained for text generation tasks, each with its specific strengths. For example, GPT-3, developed by OpenAI, set a new standard for generative tasks, showcasing impressive abilities in sentence completion and contextual understanding. Similarly, models like LLaMA and Qwen have also been developed for large-scale text generation. These models, with varying numbers of parameters, have been benchmarked against tasks such as language modeling, sentence completion, and knowledge-based reasoning. Evaluating and comparing their outputs remains a crucial research topic, and this project aligns with that effort by investigating the outputs from six distinct LLMs: LLaMA 3.2 (1B), Gemma 2b, Qwen 2.5 (3B), Phi-3.5 (3B), TinyLlama (1.1B), and nemotron-mini (1B).
2. **Sentence Embeddings and Feature Representations** - One of the fundamental components of natural language understanding is representing text in a way that models can process efficiently. Sentence embeddings are crucial for this purpose. Techniques like Sentence-BERT (Reimers & Gurevych, 2019) have introduced ways to generate dense, contextual embeddings for sentences, which are particularly useful in downstream tasks like text classification. In our project, we leveraged DunZhang's stella_en_1.5B_v5 text embeddings to generate embeddings for the text completions produced by different LLMs. This model, like Sentence-BERT, enables better semantic understanding by representing each sentence as a fixed-size vector while capturing its meaning.
3. **Prompt Engineering and its Impact** - Prompt engineering plays a crucial role in determining the performance of large language models. Prompt-based learning techniques, such as those used in GPT-3 and BERT, show that prompt design can control the nature of model completions and steer models towards more accurate or specific outputs.

   In this project, we experimented with prompts for sentence completion tasks, observing notable improvements in classification accuracy when a prompt was used. Specifically, the prompt used was designed to encourage LLMs to complete sentences based on general knowledge without requesting real-time or current data. Without the prompt, the accuracy of the BERT classifier on the HellaSwag dataset was 82%, while adding the prompt increased it to 89.92%. This is consistent with findings from prior research, where well-designed prompts have been shown to guide LLMs toward more relevant and contextually appropriate responses.
4. **Neural Network and BERT-based Classifiers** - Neural networks, particularly transformer-based architectures like BERT (Devlin et al., 2018), have become the go-to models for text classification tasks. In our project, we utilized a BERT-based classifier and fine-tuned its last four layers to classify which LLM generated the given text completions. Fine-tuning pre-trained models for specific tasks is a well-established practice in the NLP

community, as it allows models to adapt to particular datasets while retaining the knowledge gained from large-scale pre-training.

Previous works have demonstrated the effectiveness of BERT in downstream tasks like sentence pair classification, semantic similarity, and text categorization. Our results—where the fine-tuned BERT classifier achieved nearly 90% accuracy on the HellaSwag dataset—are in line with the high performance that BERT-based models typically achieve when fine-tuned on task-specific data.

5. **LLM Evaluation on Benchmarks (HellaSwag, Wikipedia)** - The HellaSwag dataset, introduced by Zellers et al. (2019), has become a standard benchmark for evaluating models' abilities to complete sentences and reason about scenarios with commonsense knowledge. This benchmark poses a challenging task for both humans and machines, as the dataset contains instances where completing sentences requires a strong understanding of context and commonsense knowledge.

   In this project, we tested our classifier models using both the Wikipedia dataset, containing 60,000 sentences, and the HellaSwag dataset. Similar to previous research, we observed that LLMs could generate plausible completions for both datasets, but adding prompt engineering techniques improved classification accuracy on more challenging tasks like those posed by HellaSwag.
6. **Model Size and Performance Trade-offs** - There has been significant research into how the size of a model, in terms of parameters, affects its performance on various tasks. Scaling laws for neural language models suggest that larger models tend to perform better on a wide range of tasks, but this comes at the cost of increased computational resources and complexity. In our project, we explored models ranging from 1 billion to 3 billion parameters, noting that the larger models like Qwen 2.5 (3B) and Phi-3.5 (3B) performed competitively compared to smaller models such as LLaMA 3.2 (1B) and nemotron-mini (1B). To generate a dataset that is atleast 60k we got restricted to use <3B parameters as large models taking longer time to complete the text.

Data Curation:

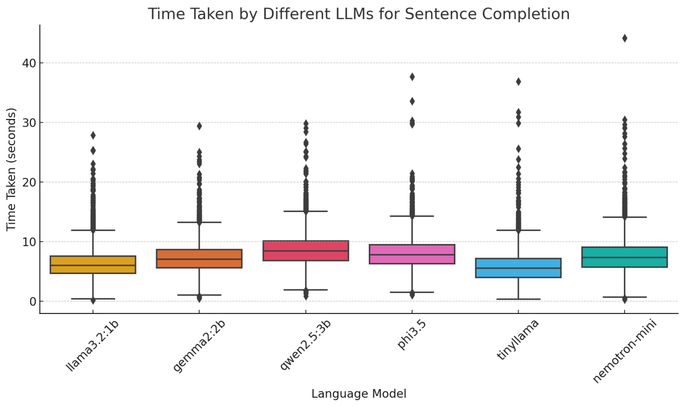
For the data curation process, we focused on generating a dataset using several language models, all under 3 billion parameters. We utilized the Ollama library to manage these models and perform sentence completions on the *HellaSwag* dataset, which served as our input source. This code is designed to compare multiple Large Language Models (LLMs) by generating text completions and measuring the time it takes for each model to respond. The code utilizes the *langchain* and *langchain_ollama* libraries to interact with these LLMs, allowing users to input text prompts and evaluate the model's output. The models were selected based on their size to ensure efficiency and compatibility with local hardware resources, as all operations were run on a local macOS machine.
The models used for this task included:

1. LLaMA 3.2 (1B): ("llama3.2:1b")
2. Gemma 2b: ("gemma2:2b")
3. Qwen 2.5 (3B): ("qwen2.5:3b")
4. Phi-3.5 (3B): ("phi3.5")
5. TinyLlama (1.1B): ("tinyllama")
6. nemotron-mini (1B): ("nemotron-mini")

We curated a dataset of **60,000 text completions**, with **10,000 completions from each model**. The data was generated using two main sources:

1. **Wikipedia:** Used to provide a diverse set of inputs based on general knowledge and facts**.**
2. **HellaSwag Sentence Completion Dataset:** Used to evaluate the models' ability to complete sentences in a challenging and contextually complex manner.

   But we eventually chosen hellaswag due to its correct sentence completion inputs.



The box plot above shows the distribution of time taken by each language model to generate sentence completions. This visualization helps us understand which models are more efficient in terms of response time and also highlights any variability or outliers in their performance.

The primary goal was to process a dataset of text prompts from the HellaSwag dataset and generate completions from these models, storing the results alongside performance metrics (time taken) in a CSV file for later analysis. The prompt used for each LLM was consistent to maintain fairness across models. It asked the LLMs to complete a sentence based on a partial input, ensuring responses were generated purely from the models' training data without accessing real-time information.

The prompt ensured consistency in how the models generated their outputs, establishing a uniform framework to fairly compare completions. It restricted the models to sentence completion, reducing irrelevant content and allowing for consistent comparisons across models.

The process began by loading the LLM models using the load_ollama_model function, which utilized the Ollama API to specify parameters like the model's name and the number of tokens for prediction.

The dataset was loaded using the load_hellaswag_dataset function, which read a .jsonl file containing structured data in JSON format. Each line contained a context (prompt) and possible sentence endings, which were appended to the truncated_texts list for generating text completions. This dataset is commonly used for benchmarking language models' ability to predict plausible text completions from partial inputs.

The generate_text_with_ollama function handled text generation and performance tracking. It used the ChatPromptTemplate class from langchain_core to format the prompt, specifying the system message (instructing the model to complete the text) and the human message (containing the actual prompt). The time taken for generating a response was measured using Python's time.time() function, capturing start and end times. The function returned the original prompt, the generated text, the model's name, and the time taken, providing comprehensive performance results.

To handle a large number of prompts concurrently, Python’s ThreadPoolExecutor was used for parallel execution. Each prompt was processed by all models, with up to 10,000 prompts handled across 10 parallel threads, significantly improving efficiency. The results (input prompt, model output, model name, and time taken) were saved in a CSV file (llm_outputs_ollama_with_time.csv) for easy inspection and comparison of the LLMs' performance.

We also created a dataset from Wikipedia by truncating the text to 20 tokens and generating sentence completions. However, this dataset is not ideal for sentence completion as it may contain dates and numbers. Therefore, we also used the Stella dataset.

Our Classifiers and their Optimization:

Two types of classification models were implemented for determining which *Largw* Language Model (LLM) generated a pair of text completions (xi for input prompt and xj for LLM output), the Fully Connected Neural Network Classifier (LLMClassifier) and the BERT-Based Classifier with Fine-Tuning (BERTClassifier). The key libraries used were:

1. Sentence Transformers: Provides pre-trained language models for sentence-level embeddings.
2. BERT (from Hugging Face's transformers): A pre-trained model for text classification tasks.

To generate embeddings from the input ("xi") and output ("xj") pairs of text, the SentenceTransformer model "dunzhang/stella_en_1.5B_v5" is used. Both stella_en_1.5B_v5 and NV-Embed-v2 were taken from the Hugging Face MTEB leaderboard. We took the best 1st and 3rd model and wanted to try them. This model has a maximum sequence length of 512 tokens. Initially, the NV-Embed-v2 model was chosen, but due to its large memory usage (up to 30GB), it was replaced by stella_en_1.5B_v5 for better efficiency. A CSV file (llm_outputs_ollama_with_time.csv) is loaded using pandas, containing input prompts (xi), LLM outputs (xj), and their associated labels (Used LLM). EOS tokens were appended to both the input and output text to mark the end of the sequence. LLM labels were encoded into numerical codes using *pd.Categorical*.

A function named *'generate_embeddings'* uses the SentenceTransformer model to encode the input (xi) and output (xj) texts into high-dimensional embeddings. The embeddings of xi and xj were then concatenated for each pair to form a combined embedding, which will serve as input to the classifier. The combined embeddings are saved into a *.pkl* file for reuse. The dataset is split into 80% training and 20% testing using the *train_test_split* function from scikit-learn. This ensures that the models are trained on the majority of the data and tested on unseen samples to evaluate their generalization performance.

Classifier 1: Fully Connected Neural Network (LLMClassifier)

1. Architecture:
   - The LLMClassifier consists of two fully connected layers.
   - A Linear layer with 256 hidden units and ReLU activation.
   - A final Linear layer that maps the 256 hidden units to the number of output classes (the number of LLMs).
   - The input size is 2048, which is the result of concatenating two 1024-dimensional embeddings.
2. Training:
   - Data Splitting: The data is split into training and testing sets using an 80-20 split (*train_test_split* function), 80% of the data is used for training and 20% for testing.
   - The CrossEntropyLoss function is used as the loss function for classification.
   - The optimizer used is Adam with a learning rate of 1e-4.
   - The model is trained for 50 epochs on batches of 16 samples.
   - During training, embeddings are fed into the model, and the loss is backpropagated after each batch. The model parameters are updated using gradient descent via the Adam optimizer.
3. Evaluation:
   - After training, the model is evaluated on the test set. The accuracy is calculated by comparing the predicted and actual labels.

Classifier 2: BERT-Based Classifier with Fine-Tuning (BERTClassifier)

1. Architecture:

- This model uses a pre-trained BERT model (bert-base-uncased) to encode the xi and xj texts separately.
- The CLS token output from BERT (a summary representation of the entire sequence) is extracted for both the input prompt and LLM output.
- The two CLS embeddings are concatenated and passed through a fully connected layer that maps the concatenated embeddings to the number of LLM classes.
- The architecture allows selective fine-tuning of the BERT model. Only the last 4 layers of BERT are fine-tuned, while the rest are frozen to preserve the pre-trained weights.

2. Training:
    - Data Splitting: Same 80-20 split as the first classifier.
    - The BERT tokenizer is used to tokenize the xi and xj texts. These are encoded into token IDs and attention masks, which are necessary for the BERT model to process the sequences.
    - Training Strategy:
        - → The CrossEntropyLoss function is used for classification.
        - → The optimizer is AdamW, which is an improved version of Adam for weight decay. The learning rate is set to 2e-5.
        - → The model is trained for 3 epochs on batches of 64 samples.
        - → Selective Fine-Tuning: During training, only the last 4 layers of BERT are updated, while the rest of the model remains frozen. This helps reduce overfitting and speeds up training by limiting the number of trainable parameters.
3. Evaluation:
    - The evaluation loop computes the model's accuracy on the test set.
    - After testing, the true labels and predicted labels are used to generate a confusion matrix, which visualizes the classifier's performance in distinguishing between different LLM classes.
    - A prediction distribution plot shows the frequency of predicted labels, providing insight into how the model is making predictions.

Key Differences Between the Classifiers

1. Model Architecture:
    - LLMClassifier: A simple two-layer fully connected neural network operating on concatenated embeddings of text pairs.
    - BERTClassifier: A more complex model using BERT to encode texts before concatenating the CLS embeddings, with selective fine-tuning of BERT layers.
2. Training Approach:
    - LLMClassifier: Directly trained on the concatenated embeddings generated by the pre-trained SentenceTransformer.
    - BERTClassifier: Uses pre-trained BERT as part of the model and selectively fine-tunes its layers.
3. Input Size:
    - LLMClassifier: Takes 2048-dimensional embeddings as input (from concatenating two 1024-dimensional vectors).
    - BERTClassifier: Takes tokenized text sequences as input and processes them through BERT.

Main Results:

BERT-Based Classifier:

The BERT-based classifier was trained over **three epochs** using the AdamW optimizer, which struck a good balance, with a learning rate of **2e-5**. The choice of a relatively small learning rate (2e-5) helped ensure that the fine-tuning process didn’t lead to sudden updates, allowing for gradual improvements in performance. The dataset consisted of input prompts and LLM-generated outputs, which were tokenized separately and fed into the BERT model. Each input pair (prompt and output) was processed through two separate BERT embeddings, which were then concatenated and passed through a fully connected classification layer.

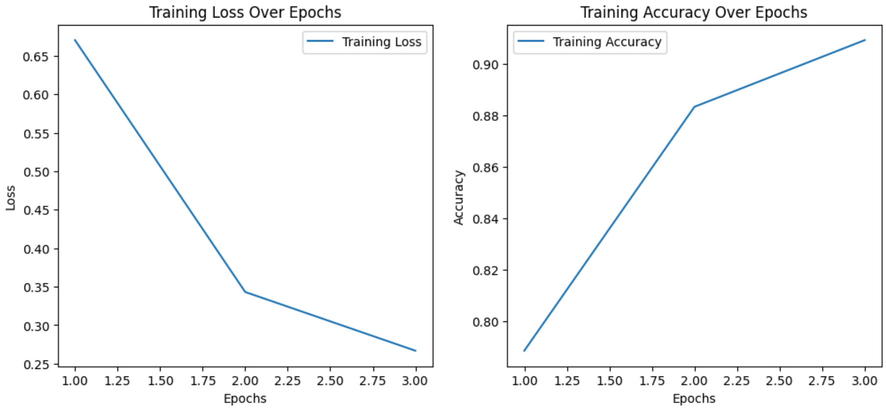


Figure 1: Loss and Accuracy of BERT-based Classifier over epochs

Epoch 1, Loss: 0.6849, Accuracy: 78.19%
Epoch 2, Loss: 0.3515, Accuracy: 88.14%
Epoch 3, Loss: 0.2761, Accuracy: 90.52%
Test Accuracy: 88.78%

During training, the model achieved a steady improvement in both accuracy and loss. For instance, the accuracy on the training data after the final epoch was **90.52%**, while the test set accuracy reached **88.78%**. This demonstrates that the model could effectively learn distinguishing patterns between the LLM-generated outputs and correctly classify them.

An essential aspect of the model architecture was the selective fine-tuning of the last four layers of the BERT encoder. Initially, all BERT layers were frozen, and only the last four layers were unfrozen for fine-tuning. This approach was chosen to strike a balance between leveraging BERT’s pre-trained language understanding and adapting it to the specific task of distinguishing LLM-generated outputs.

The fine-tuning of these layers had a noticeable impact on performance. By selectively updating only a few layers, the model could retain most of BERT’s pre-trained knowledge while becoming specialized for the task at hand. Fine-tuning all layers might have led to overfitting on the relatively smaller dataset, while too few layers could have resulted in underfitting. Fine-tuning the last four layers allowed the model to focus on the most important features from both the prompt and the generated text.

To evaluate the model's performance, a confusion matrix was plotted, as shown in **Figure 2** below. The matrix highlights which LLM outputs were more challenging for the model to classify and where misclassifications occurred. Most predictions clustered correctly along the diagonal, indicating that the model performed well on the majority of the test cases.

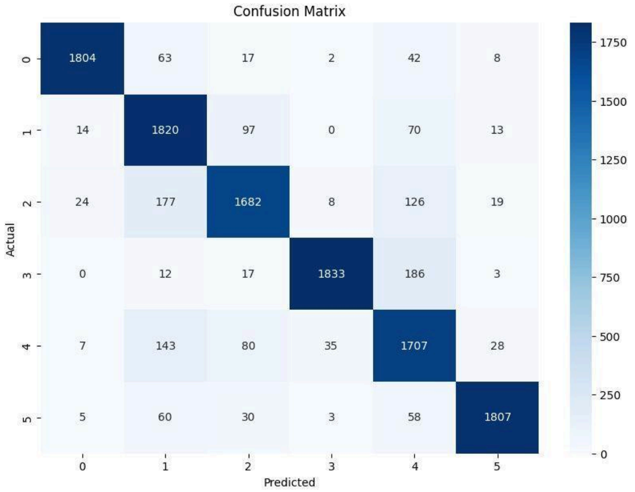


Figure 2. Confusion Matrix of BERT-based Classifier

The class distribution in the training data likely had an effect on the model’s ability to distinguish between certain LLM-generated outputs, as evidenced by the confusion matrix. The diagonal elements, representing correct predictions, are significantly larger than off-diagonal elements, indicating good overall performance. The confusion matrix also reveals a class imbalance, with model LLaMA 3.2, Gemma, and TinyLlama having significantly more instances than the others. LLaMA 3.2 (class 0) has the highest number of instances and seems to be accurately classified, with a large diagonal value and relatively small off-diagonal values. Similar to LLaMA 3.2, Gemma (class 1) also shows good performance with a high diagonal value and low off-diagonal values. While the diagonal value of Qwen 2.5 (class 2) is still significant, there are noticeable off-diagonal values, suggesting potential confusion with other classes. Phi-3.5 (class 3) exhibits a strong diagonal value and relatively low off-diagonal values, indicating good classification accuracy. TinyLlama (class 4) demonstrates strong performance with a high diagonal value and low off-diagonal values. nemotron-mini (class 5) has the lowest number of instances and might be more challenging to classify. The diagonal value is relatively small, and there are noticeable off-diagonal values, suggesting potential confusion with other classes.

The distribution of predictions, visualized in **Figure 3** below, shows the frequency of predicted labels, which corresponds closely with the distribution of actual labels. The prediction distribution aligns with the observations from the confusion matrix.

The classes with higher diagonal values in the confusion matrix tend to have higher frequencies in the bar graph. A high diagonal value in the confusion matrix for a class often corresponds to a high frequency for that class in the prediction distribution. This indicates that the model is accurately classifying instances belonging to that class. High off-diagonal values in the confusion matrix between two classes might suggest that the model is confusing these classes. If one of these classes has a higher frequency in the prediction distribution, it could be due to the model's tendency to misclassify instances from other classes into this class. Both the confusion matrix and bar graph highlight the class imbalance, emphasizing the need for careful consideration when interpreting the model's performance.

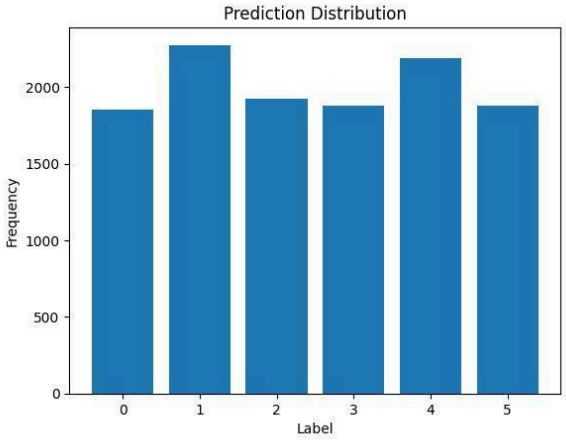


Figure 3: Prediction Distribution of BERT-based Classifier

The distribution is slightly skewed to the right, indicating that there are more instances in the models TinyLlama and nemotron-mini. The bar graph confirms the class imbalance observed in the confusion matrix, with models LLaMA 3.2, Gemma, and TinyLlama having significantly more instances than the others. LLaMA 3.2 and Gemma have a relatively high frequency, suggesting that they are well-represented in the dataset. The frequency of Qwen 2.5 is moderate, indicating a reasonable representation. Phi-3.5 has a slightly lower frequency than models LLaMA 3.2, Gemma, and Qwen 2.5. TinyLlama has the highest frequency, suggesting that it is overrepresented in the dataset. The frequency of nemotron-mini is the lowest, indicating that it is underrepresented in the dataset.

Fully Connected Neural Network LLM Classifier:

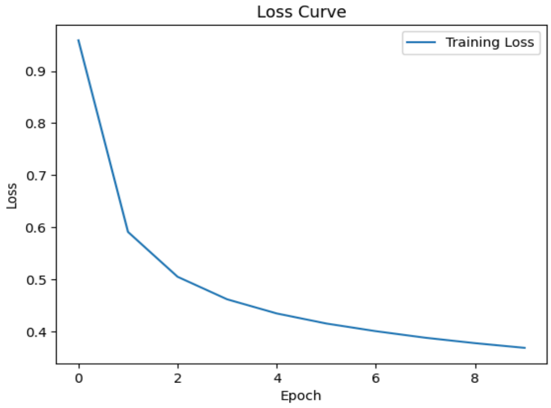
The results from training and evaluating the neural network classifier for LLM-generated text completion classification show a consistent improvement in both loss reduction and accuracy over the 10 epochs of training. The primary metrics include the training loss, training accuracy, and test accuracy. Additionally, we provide visual representations of the model's performance, including loss and accuracy curves, confusion matrices, and prediction distributions.

The plotted loss and accuracy curves (**Figure 1 and Figure 2**) visually represent the improvement in the model’s performance during training. The loss function used is Cross-Entropy Loss, which measures the difference between predicted and true labels. The loss curve shows a smooth, downward trend, while the accuracy curve demonstrates a steady upward climb, both of which are indicative of effective learning.

The training loss steadily decreased over the course of 10 epochs, from an initial loss of 0.96 to a final loss of 0.37. This indicates that the model effectively learned from the data and improved its predictions as the training progressed. The reduction in loss suggests that the model's predictions were gradually becoming more aligned with the true labels.

The overall trend of the loss curve is downward, indicating that the model is learning and improving over the training epochs. This is a positive sign as it suggests that the model's ability to minimize the error between its predictions and the true labels is increasing. The curve seems to plateau towards the end, suggesting that the model might have reached a convergence point. This means that further training epochs might not lead to significant improvements in performance. And further, if the curve continues to decrease without plateauing, it indicates that the model is still learning and could benefit from additional training.

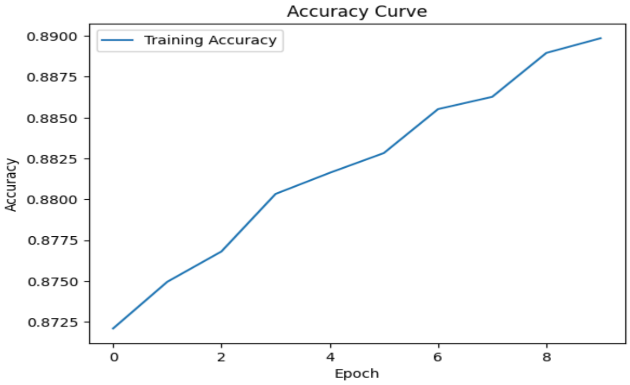
The shape of the curve can be influenced by the learning rate. A high learning rate might cause the curve to fluctuate or even diverge, while a low learning rate might result in slow convergence. Finding the optimal learning rate is crucial for efficient training. Techniques like grid search or learning rate scheduling can help determine the best learning rate. To assess the model's generalization ability, it's important to track the validation loss in addition to the training loss.



Epoch 1, Loss: 0.9590461246867975
Epoch 2, Loss: 0.5909971048583587
Epoch 3, Loss: 0.5049030164188395
Epoch 4, Loss: 0.46161705132201314
Epoch 5, Loss: 0.4344852385893464
Epoch 6, Loss: 0.4151497087329626
Epoch 7, Loss: 0.40042777959691983
Epoch 8, Loss: 0.3878069570722679
Epoch 9, Loss: 0.3774289210420102
Epoch 10, Loss: 0.368410297545294

Figure 1: Loss vs epochs for NN Classifier

Similarly, the training accuracy increased from 87.21% in the first epoch to 88.99% by the final epoch, showing a consistent improvement in the model's ability to correctly classify the LLM-generated text completions. Accuracy is tracked during training to measure the classifier's ability to predict the correct labels from the embeddings.



Epoch 1, Accuracy: 87.21%
Epoch 2, Accuracy: 87.50%
Epoch 3, Accuracy: 87.68%
Epoch 4, Accuracy: 88.03%
Epoch 5, Accuracy: 88.16%
Epoch 6, Accuracy: 88.28%
Epoch 7, Accuracy: 88.55%
Epoch 8, Accuracy: 88.63%
Epoch 9, Accuracy: 88.90%
Epoch 10, Accuracy: 88.99%

Figure 2: Accuracy vs Epochs for NN Classifier

The overall trend of the accuracy curve is upward, indicating that the model's performance is improving over the training epochs. This is a positive sign as it suggests that the model is becoming more accurate in classifying the data. The curve seems to plateau towards the end, suggesting that the model might have reached a convergence point. This means that further training epochs might not lead to significant improvements in accuracy. And further, if the curve continues to increase without plateauing, it indicates that the model is still learning and could benefit from additional training.

The final accuracy achieved by the model can be assessed by examining the value at the last epoch. A higher final accuracy indicates better overall performance. A relatively smooth curve without large fluctuations suggests consistent improvement during training. The shape of the curve can be influenced by the learning rate. A high learning rate might cause the curve to fluctuate or even diverge, while a low learning rate might result in slow convergence. To assess the model's generalization ability, it's important to track the validation accuracy in addition to the training accuracy. If the training accuracy continues to increase while the validation accuracy plateaus or decreases, it might be a sign of overfitting. Techniques like early stopping or regularization can help mitigate overfitting.

After completing training, the model was evaluated on the test set. The test accuracy reached **84.68%**, which is slightly lower than the training accuracy but still a strong indication that the model generalizes well to unseen data. This difference between training and test accuracy suggests minor overfitting, but the test performance remains robust overall.

Test Accuracy: 84.68%

To analyse the model's performance in classifying specific LLMs, a confusion matrix was generated (**Figure 3**). The confusion matrix reveals how often the model correctly classified each LLM and highlights areas where it made incorrect predictions. The diagonal entries represent correct classifications, while off-diagonal entries indicate misclassifications. The diagonal elements, representing correct predictions, are significantly larger than off-diagonal elements, indicating good overall performance. The matrix shows that the model performed well in distinguishing between different LLMs, though some overlap between certain models suggests areas where the classifier might struggle.

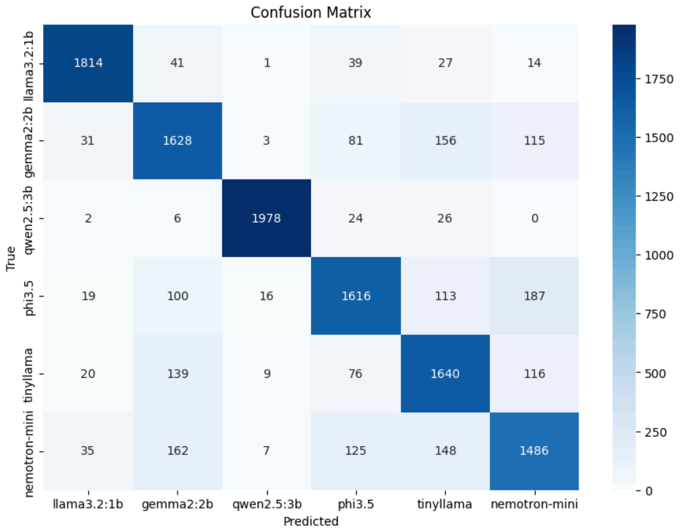


Figure 3: Confusion Matrix of NN Classifier

There is a class imbalance seen in the confusion matrix, with classes LLaMA 3.2, Gemma, and TinyLlama having much more occurrences than the rest. The model LLaMA 3.2, with a big diagonal value and relatively tiny off-diagonal values, appears to be properly identified and has the maximum number of examples. Gemma 2B has strong performance with low off-diagonal values and a high diagonal value, much like LLaMA 3.2. Qwen 2.5 's diagonal value is still crucial, but there are some discernible off-diagonal values that raise the possibility of misunderstanding with other classes. Phi-3.5 exhibits a strong diagonal value and relatively low off-diagonal values, indicating good classification accuracy. TinyLlama has good performance with a high diagonal value and low off-diagonal values, much like LLaMA 3.2 and Gemma 2B. nemotron-mini has the lowest number of cases and could be more complex to categorize. There are distinct off-diagonal values and a very tiny diagonal value, which may indicate possible class confusion.

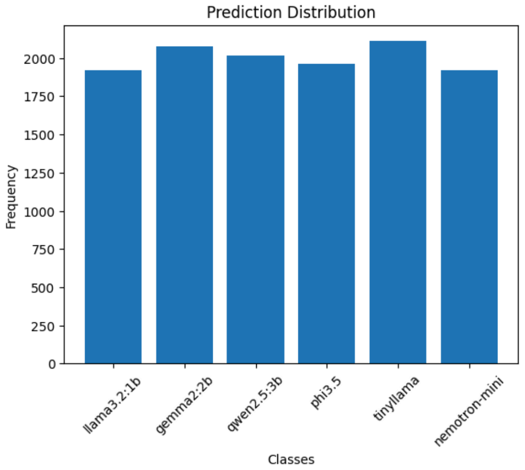


Figure 4. Prediction Distribution of NN Classifier

A bar plot of the predicted labels (**Figure 4**) revealed a balanced distribution, with predictions spread across all six LLMs, demonstrating that the classifier was not biased toward any particular model. The bar graph confirms the class imbalance observed in the confusion matrix, with models LLaMA 3.2, Gemma and TinyLlama having significantly more instances than the others. The distribution is slightly skewed to the right, indicating that there are more instances in the higher models TinyLlama and nemotron-mini. The prediction distribution aligns with the observations from the confusion matrix. The classes with higher diagonal values in the confusion matrix tend to have higher frequencies in the bar graph.

Other in-depth analyses/experiments:

Before we moved on to testing the model with the HellaSwag dataset, we conducted an initial experiment using a **Wikipedia** dataset. In this preliminary phase, we curated a dataset with 10,000 sentences for each of the six Large Language Models (LLMs) used in this project. This resulted in a total of 60,000 sentences, covering a diverse range of sentence completions across the different LLMs. The six LLMs we experimented with are: LLaMA 3.2 (1B), Gemma 2b (2B), Qwen 2.5 (3B), Phi-3.5 (3B), TinyLlama (1.1B), and nemotron-mini (1B). These LLMs were chosen for their variety in parameter sizes and architectures, providing a rich set of embeddings for training and evaluation.

For this experiment, we utilized the stella_en_1.5B_v5 SentenceTransformer model to generate embeddings from the Wikipedia dataset. These embeddings were then fed into our neural network (NN) classifier, which was tasked with predicting which LLM generated the completion for a given sentence. The neural network classifier, trained on these embeddings, achieved an 81% accuracy on the Wikipedia dataset. This was a strong starting point, suggesting that the model could effectively differentiate between the output styles of different LLMs based on sentence embeddings.

The prompt used to generate sentence completions from the LLMs during this experiment was structured as follows:

```
prompt_template = ChatPromptTemplate.from_messages(
  [
       ("system", "You are an assistant for sentence completion. Please complete the sentence
based on your general knowledge, and do not request current or real-time data."),
     ("human", 'Complete the following sentence: "{input}"')
  ]
)
```

This prompt was designed to encourage the LLMs to generate responses purely based on their inherent knowledge and sentence completion capabil ities, without relying on any real-time data or external queries. It was used uniformly across all LLMs for this phase of the project to ensure consistency in sentence completions.

After the Wikipedia dataset experiments, we shifted focus to the HellaSwag dataset, which presents a more challenging and nuanced set of text completion tasks. We conducted two rounds of experiments on this dataset—one with the prompt described above, and one without using the prompt.

In the first experiment, without applying the prompt, we observed that the accuracy of the model on the HellaSwag dataset was 82%. This was already an improvement compared to the Wikipedia dataset, suggesting that the model was better at distinguishing between LLM completions on more complex tasks, even without the assistance of a prompt.

Next, we introduced the prompt to the HellaSwag dataset and experimented with two classifiers: the BERT classifier and the NN classifier using the stella_en_1.5B_v5 embeddings. When using the BERT classifier on the unshuffled dataset, the accuracy increased significantly to 89.92%. However, when we shuffled the dataset before training, the accuracy of the BERT classifier slightly dropped to 89.35%. This indicates that the ordering of the data might play a role in how effectively the model learns to classify LLM outputs.

When applying the same prompt to the NN classifier, which leveraged the stella_en_1.5B_v5 SentenceTransformer embeddings, the accuracy achieved on the HellaSwag dataset was 87%. While this was slightly lower than the BERT classifier, it was still a notable improvement over the unprompted accuracy, further validating the effectiveness of using a prompt to guide LLM sentence completions.

Additionally, during the course of our experiments, we briefly tested another model, **DeepSeek-Coder (1B)**, specifically to observe its performance on code-related outputs. We used the model to generate sentence completions in a coding context. However, since the primary focus of the project was on general LLM sentence completions, the results from DeepSeek-Coder were not included in the final experimental analysis. Despite this, it provided interesting insights into how different models handle specific task domains like coding, and could be explored further in future work.

Through our systematic experimentation with both the Wikipedia and HellaSwag datasets, we were able to demonstrate the effectiveness of our NN classifier and the impact of using prompts. The prompt significantly improved classification performance, with both BERT and NN classifiers showing notable gains in accuracy when it was applied. The experiments highlight how different datasets and task complexities can influence model performance, and how the choice of LLM and classifier architecture plays a crucial role in determining the outcome.

Deeper Analysis of our classifier models:

Classification Report:

	precision	recall	f1-score	support
0	0.93	0.95	0.94	1936
1	0.85	0.76	0.80	2014
2	0.97	0.98	0.97	2036
3	0.80	0.81	0.81	2051
4	0.84	0.76	0.80	2000
5	0.71	0.82	0.76	1963

A small error analysis of the confusion matrix of our BERT-based classifier model suggests:

Confusion Patterns: Analyzing the off-diagonal elements can help identify specific confusion patterns. For example, if there are high values between classes 2 and 3, it might indicate that the model is having difficulty distinguishing between these two LLMs.

Error Sources: Understanding the characteristics of the misclassified instances can provide insights into the limitations of the BERT-based classifier. For instance, if the model frequently misclassifies texts with certain linguistic features or topics, it might suggest that the model is struggling to capture these nuances.

Further Improvements of this classifier model:

Data Augmentation: To address class imbalance, data augmentation techniques can be employed to generate more instances of underrepresented classes.

Model Fine-tuning: Experimenting with different hyperparameters, architectures, or pre-trained models might lead to improved performance.

Feature Engineering: Incorporating additional features beyond the text itself, such as stylistic or semantic information, could enhance the model's ability to discriminate between LLMs.

The prediction distribution of the BERT-based classifier model suggests:

Outliers: If there are any extremely high or low frequencies, they might indicate outliers or anomalies in the data.

Data Preprocessing: Understanding the reasons for the class imbalance can help inform data preprocessing techniques, such as oversampling or under-sampling, to address this issue.

Additionally, from our observations on both the classifier (NN and BERT-based) models, we could conclude that in general, there were several factors that can influence the loss and accuracy metrics, as observed in the training and evaluation:

Data Imbalance: If some LLMs were over- or under-represented in the dataset, this could affect the classifier's ability to distinguish between them. Data imbalance could lead to the model favoring the more frequent classes.

Embedding Quality: The quality of the embeddings, derived from BERT-like architectures, affects how well the neural network learns the relationships between LLMs. Inaccurate or noisy embeddings could lead to higher loss and lower accuracy.

Model Complexity: The neural network classifier model consists of two fully connected layers, which might not be sufficient for more complex patterns between the text completions of the LLMs, especially those with larger parameters, such as Qwen 2.5 and Phi-3.5.

Batch Size: The training was conducted with a batch size of 16, which may have impacted the gradient estimates. A larger batch size might provide more stable updates, while a smaller one might introduce more noise, affecting convergence.

Learning Rate: The learning rate (1e-4) controls how fast the model learns. A too-high learning rate might cause the model to overshoot optimal weights, while a too-low rate could slow down learning and prevent reaching the optimal loss.

Overfitting: The slight gap between training and test accuracies suggests potential overfitting, meaning the model might have learned to perform well on the training data but struggles to generalize to unseen test data.

References:

1. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805.
2. Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. arXiv preprint arXiv:1908.10084.
3. Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., & Choi, Y. (2019). HellaSwag: Can a Machine Really Finish Your Sentence? Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics.
4. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017). Attention is All You Need. Advances in Neural Information Processing Systems, 30, 5998–6008.
5. ChatGPT and ChatGPT Canvas
6. https://ollama.com/library
7. https://huggingface.co/ and https://huggingface.co/spaces/mteb/leaderboard