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 to classify text completions generated by different Large Language Models. Utilizing the Neural Network based classifiers and BERT-based architectures, in our quest to develop a classifier that can distinguish between outputs generated by distinct LLMs. The dataset curated for this task consisted of text completion pairs for which embeddings were created to perform the classification. It was trained based on the cross-entropy loss function optimized with the Adam optimizer, with pretty good accuracy on LLM outputs. Results are analyzed in terms of accuracy and loss metrics, demonstrating our approach. Finally, a comparative analysis with related works is presented, discussing the evolution and challenges in the field of text generation classification.

Related Work:

1. **Large Language Models and Text Generation** – GPT, BERT, and all their variants started a new era for large language models in NLP, especially in the domain of text generation. This class of large language models was basically developed on the transformer architecture that was proposed by Vaswani et al. in the paper titled "Attention is All You Need," 2017. Transformers follow the principle of self-attention whereby the models can process long texts and give coherently human-like completions. Recently, several LLMs have been trained for text generation applications, each with their strengths. As an example, GPT-3, developed by OpenAI, set the next generation of benchmarks for a wide variety of generative tasks and with remarkable performance both at sentence completion and also at contextualized understanding. Large-scale text generation has also been developed with models such as LLaMA and Qwen. These models, by different counts of parameters, have been benchmarked on tasks such as language modeling, sentence completion, and knowledge-based reasoning. Such evaluation and comparison of their outputs remain an active line of research, and this project presents itself in line with that effort since it will be reviewing the outputs from six different 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 most basic elements of natural language understanding is to represent text in a manner that models can handle efficiently. Sentence embeddings are crucial to that end. Methodas like Sentence-BERT by Reimers & Gurevych (2019) have provided methods for creating dense, contextual sentence embeddings that perform very well in many downstream applications like text classification. In our project, the stella_en_1.5B_v5 SentenceTransformer was used to create embeddings of the text completions generated by different LLMs. This model, similar to Sentence-BERT, enables richer semantic comparisons by representing each sentence with a vector of fixed size while conveying its meaning.
3. **Prompt Engineering and its Impact** - Prompt engineering plays an important function in evaluating large language models. The variety of approaches towards prompt-based learning, including but not limited to GPT-3 and BERT, exhibit that it is possible to control the nature of the model completions based on prompt design or to drive models toward accurate or more specific outputs. This project tested prompts for sentence completion tasks and found that, indeed, introducing a prompt improved the classification accuracy remarkably. In fact, the prompt given was designed in a way that it would not require the LLM to seek current or real-time data, but instead, general knowledge would complete the sentence. While the accuracy of the BERT classifier on the HellaSwag dataset was earlier 82%, with the introduction of the prompt, this went up to 89.92%. This is in consonance with the consensual result of previous studies, in which well-designed prompts had been used 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 of the pre-trained models on specific tasks is therefore considered a very established practice within the NLP community, given it allows any model to adapt to certain datasets while retaining

knowledge acquired through large-scale pre-training. Several works demonstrated the effectiveness of BERT in downstream tasks like sentence pair classification, semantic similarity, and text categorization. Our results, where a fine-tuned BERT classifier achieved near 90% accuracy on the HellaSwag dataset, are in line with the high performance usually achieved by BERT-based models when fine-tuned on task-specific data.

5. **LLM Evaluation on Benchmarks (HellaSwag, Wikipedia)** - The HellaSwag dataset presented by Zellers et al. in 2019 has been a standard benchmark to evaluate the model's abilities in sentence completion and to reason out a scenario with common sense. This benchmark is considered a very difficult task for both humans and machines because, in the provided dataset, there are instances where completing sentences requires plenty of contextual understanding and commonsense knowledge. In this work, we have applied both Wikipedia datasets with 60,000 sentences and the HellaSwag dataset to test the classifier models. As in prior work, LLMs produced plausible completions for both datasets. However, by applying prompt engineering techniques, we improved the performance of classification on harder tasks posed by HellaSwag.
6. **Model Size and Performance Trade-offs** - There has been extensive research into how performance on different tasks varies with model size in parameters. Scaling laws indicate that larger models generally perform better on a wide range of tasks but at a certain cost with respect to the computational resources and complexity.
   We consider models in our work that are from 1 billion to 3 billion parameters. We notice the larger ones, such as Qwen 2.5 (3B), Phi-3.5 (3B), performing well compared to the smaller models like LLaMA 3.2 (1B) and nemotron-mini (1B).

Data Curation:

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 below 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.

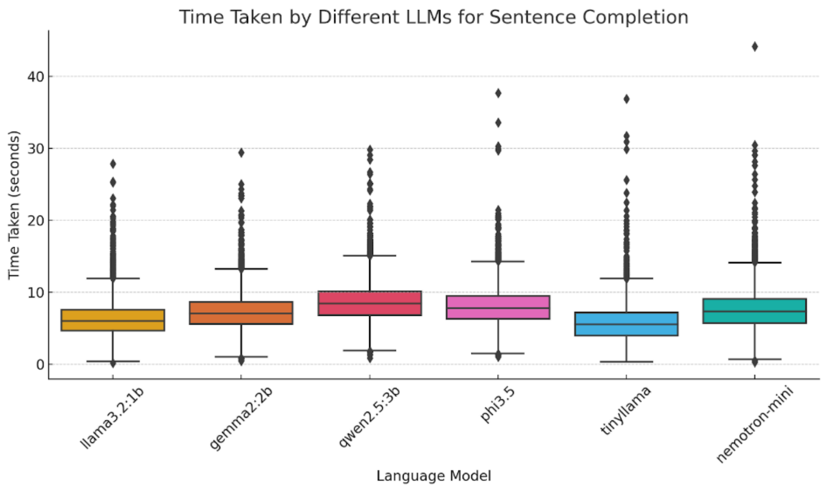


Figure 1: Time Distribution Box Plot

We concentrated on creating a dataset for the data curation process using several language models with fewer than 3 billion parameters. The HellaSwag dataset was our input source, and we managed these models and performed sentence completions on it using the Ollama package. By producing text

completions and timing how long it takes for each model to react, this code compares many Large Language Models (LLMs). The code communicates with these LLMs by using the langchain and langchain_ollama libraries, enabling users to provide text prompts and assess the model's output. Since all operations were conducted on a local macOS computer, the models were chosen based on their size to guarantee effectiveness and compatibility with local hardware resources.
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")

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:

To ascertain which Large Language Model (LLM) produced a pair of text completions (xi for input prompt and xj for LLM output), two classification models were used: the BERT-Based Classifier with Fine-Tuning (BERTClassifier) and the Fully Connected Neural Network Classifier (LLMClassifier).
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.

The "**dunzhang/stella_en_1.5B_v5**" SentenceTransformer model is imported in order to produce embeddings from the text pairs that are input (xi) and output (xj). 512 tokens is the maximum sequence length that can be used. Prior to settling on this model, the NV-Embed-v2 model was chosen; however, the stella_en_1.5B_v5 was chosen instead because of its high memory consumption (up to 30GB).

Pandas is used to import a CSV file (llm_outputs_ollama_with_time.csv) that contains input prompts (xi), LLM outputs (xj), and the labels that correspond to them (Used LLM). To indicate the conclusion of the sequence, EOS tokens were attached to the text of both the input and output. Using pd.Categorical, LLM labels were converted into number codes.

The SentenceTransformer model is used by the 'generate_embeddings' function to encode the input (xi) and output (xj) texts into high-dimensional embeddings. The classifier will get the combined embedding that was created by concatenating the embeddings of xi and xj for each pair. To be used again, the concatenated embeddings are stored in a.pkl file. The train_test_split function from scikit-learn divides the dataset into 80% training and 20% testing. This guarantees that the models are tested on unseen samples to assess their generalization ability and trained on most of the data.

Classifier 1: Text Embedding and Fully Connected Neural Network (LLMClassifier)

1. Architecture:
   - There are two completely linked layers in the LLMClassifier.
   - A Linear layer with 256 hidden units and ReLU activation.
   - The last linear layer, which converts the 256 hidden units into the total number of output classes (or LLMs).
   - The outcome of concatenating two 1024-dimensional embeddings yields an input size of 2048.
2. Training:
   - Data Splitting: Using an 80-20 split (train test split function), the data is divided into training and testing sets, with 80% of the data being utilized for training and 20% for testing.
   - The loss function for classification is the CrossEntropyLoss function.
   - The optimizer used is Adam with a learning rate of 1e-4.
   - The model is trained for 50 epochs on batches of 16 samples.
   - The model is given embeddings during training, and the loss is backpropagated following each batch. Gradient descent is used to update the model parameters using the Adam optimizer.
3. Evaluation:
   - The test set is used to evaluate the model after training. The anticipated and actual labels are compared to determine the accuracy.

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

1. Architecture:

- The xi and xj texts are encoded independently by this model using a pre-trained BERT model (bert-base-uncased).
- For both the input prompt and the LLM output, the CLS token output (a summary of the whole sequence) is taken from BERT.
- A fully connected layer that maps the concatenated embeddings to the number of LLM classes receives the two concatenated CLS embeddings.
- The BERT model can be selectively fine-tuned thanks to the design. To maintain the pre-trained weights, BERT's remaining layers are frozen and only the final four are fine-tuned.

2. Training:
   - Data Splitting: Same 80-20 split as the first classifier.
   - The xi and xj texts are tokenized using the BERT tokenizer. Token IDs and attention masks are generated from these, which are required for the BERT model to handle 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: Only the final four layers of BERT are updated while the model is being trained; the remaining layers stay unchanged. This limits the amount of trainable parameters, which helps minimize overfitting and accelerates training.
3. Evaluation:
   - Following testing, the real labels and predicted labels are used to create a confusion matrix, which illustrates the classifier's effectiveness in differentiating between various LLM classes.
   - The evaluation loop computes the model's accuracy on the test set.
   - The frequency of projected labels is displayed in a prediction distribution plot, which sheds light on the model's prediction process.

Key Differences Between the Classifiers

1. Model Architecture:
   - LLMClassifier: LLMClassifier is a straightforward two-layer fully connected neural network that processes text pair concatenated embeddings.
   - BERTClassifier: An advanced model with selective BERT layer fine-tuning that encodes texts using BERT before concatenating the CLS embeddings.
2. Training Approach:
   - LLMClassifier: Trained directly using the concatenated embeddings produced by the SentenceTransformer that has already been trained.
   - BERTClassifier: This model fine-tunes its layers based on which pre-trained BERT is used.
3. Input Size:
   - LLMClassifier: Concatenates two 1024-dimensional vectors to produce 2048-dimensional embeddings as input.
   - BERTClassifier: This algorithm runs tokenized text sequences via BERT.

Main Results:

BERT-Based Classifier:

The classifier based on BERT was trained over three epochs with a learning rate of 2e-5 using the AdamW optimizer, which achieved a satisfactory balance. The use of a comparatively modest learning rate (2e-5) allowed for steady performance gains by preventing abrupt updates throughout the fine-tuning phase. The input prompts and LLM-generated outputs, which were tokenized independently and supplied into the BERT model, made up the dataset. Two distinct BERT embeddings were applied to each input pair (prompt and output), which were subsequently concatenated and fed via a fully connected classification layer.

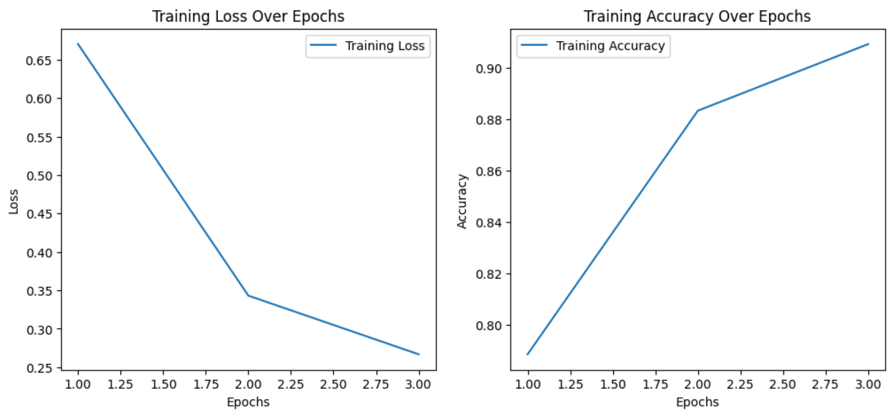


Figure 1: Loss and Accuracy of BERT-based Classifier over epochs

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%

The model steadily improved in terms of both accuracy and loss after training. For example, after the last epoch, the accuracy on the training data was 90.52%, but the accuracy on the test set was 88.78%. This shows that the model could accurately identify and efficiently learn to differentiate patterns amongst the LLM-generated outputs.

The final four layers of the BERT encoder were selectively fine-tuned, which was a crucial component of the model design. Just the final four BERT layers were unfrozen for fine-tuning at first; all other layers remained frozen. In order to balance using BERT's pre-trained language comprehension with tailoring it to the particular job of differentiating LLM-generated outputs, this strategy was used.

Performance was clearly affected by the fine-tuning of these layers. A few carefully chosen layers might be updated, allowing the model to become more task-specific while retaining the majority of BERT's pre-trained information. On the comparatively smaller dataset, overfitting may have occurred from fine-tuning every layer, while underfitting might have resulted from fine-tuning too few layers. By optimizing the last four layers, the model was able to concentrate on the key elements from both the produced text and the prompt.

A confusion matrix was constructed, as seen in Figure 2 below, in order to assess the model's performance. The matrix shows which LLM outputs were harder for the model to recognize correctly and the locations of incorrect classifications. Most predictions clustered correctly along the diagonal, indicating that the model performed well on the majority of the test cases.

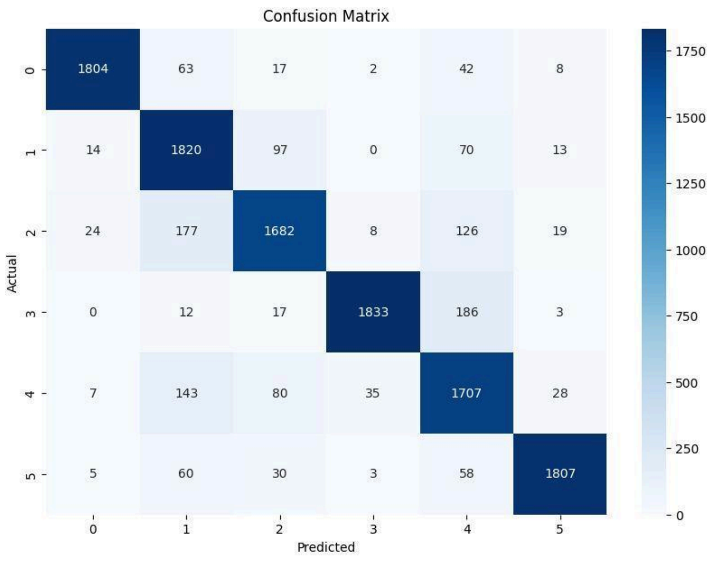


Figure 2. Confusion Matrix of BERT-based Classifier

The confusion matrix indicates that the model's ability to differentiate between certain LLM-generated outputs was probably impacted by the class distribution in the training data. Good overall performance is shown by the diagonal elements, which reflect correct predictions, being much bigger than off-diagonal components. A class imbalance is also evident in the confusion matrix, where model LLaMA 3.2, Gemma, and TinyLlama have much more occurrences than the rest.

The most examples, LLaMA 3.2 (class 0), has a significant diagonal value and very tiny off-diagonal values, suggesting that it is properly categorized. Gemma (class 1) performs well with a high diagonal value and low off-diagonal values, much as LLaMA 3.2. There are discernible off-diagonal values, indicating possible misunderstanding with other classes, even if the diagonal value of Qwen 2.5 (class 2) is still substantial. Phi-3.5 (class 3) shows high classification accuracy with a significant diagonal value and relatively modest off-diagonal values. With low off-diagonal values and a high diagonal value, TinyLlama (class 4) performs admirably. Class 5, Nemototron-mini, has the fewest examples and may be more difficult to categorize.

There are distinct off-diagonal values and a very tiny diagonal value, which may indicate possible class confusion. The frequency of expected labels is represented by the distribution of predictions, which is illustrated in Figure 3 below and closely resembles the distribution of actual labels. The findings from the confusion matrix are consistent with the forecast distribution.

In the bar graph, the classes with greater diagonal values in the confusion matrix often have higher frequencies. A class's high frequency in the prediction distribution frequently correlates with a class's high diagonal value in the confusion matrix. This suggests that the model is correctly categorizing instances that fall within that particular class.

It is possible that the model is confusing two classes if there are high off-diagonal values in the confusion matrix between them. It may be because of the model's propensity to incorrectly categorize examples from other classes into this class if one of these classes has a larger frequency in the prediction distribution. The disparity in class is brought to light by the confusion matrix and bar graph, underscoring the importance of evaluating performance of the model with caution.

The distribution is somewhat tilted to the right, suggesting that the nemotron-mini and TinyLlama models have more instances. With far more instances of models LLaMA 3.2, Gemma, and TinyLlama than the rest, the bar graph supports the class imbalance shown in the confusion matrix. Given their comparatively high frequencies, LLaMA 3.2 and Gemma appear to be well-represented in the sample.

Qwen 2.5 has a moderate frequency, which suggests a fair representation. The frequency of Phi-3.5 is somewhat lower than that of the LLaMA 3.2, Gemma, and Qwen 2.5 models. TinyLlama appears to be overrepresented in the sample, as seen by its greatest frequency. Nemototron-mini has the lowest frequency, which suggests that the dataset contains less of it.

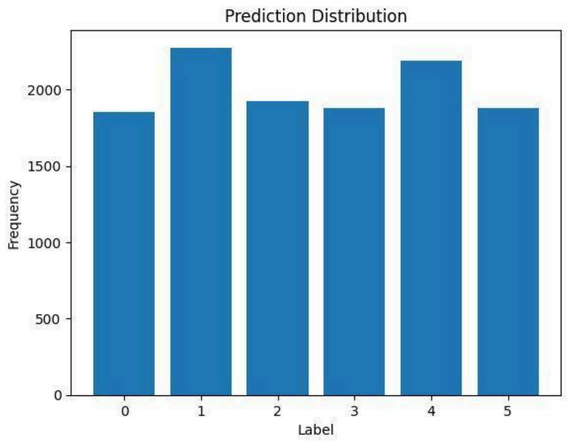
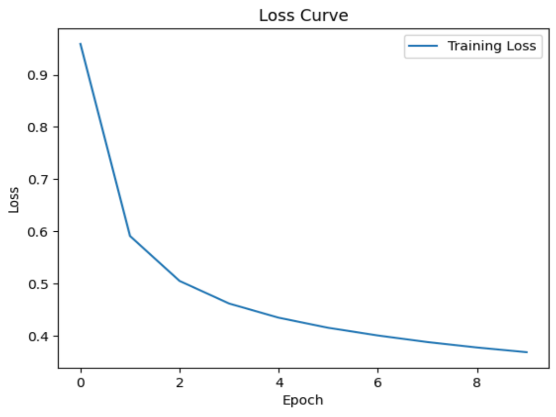


Figure 3: Prediction Distribution of BERT-based Classifier

Fully Connected Neural Network LLM Classifier:

The neural network classifier for LLM-generated text completion categorization was trained and evaluated, and the findings demonstrate a steady improvement in accuracy and loss reduction over the course of the 10 training epochs. The training loss, training accuracy, and test accuracy are the main measures. Furthermore, we offer graphical depictions of the model's performance, such as prediction distributions, confusion matrices, and loss and accuracy curves.

The model's performance increase during training is clearly shown by the plotted loss and accuracy curves (Figures 1 and 2). Cross-Entropy Loss, which calculates the discrepancy between true and predicted labels, is the loss function that is employed. Effective learning is shown by both the accuracy and loss curves, which exhibit a continuous upward ascent and a smooth decreasing trend, respectively.



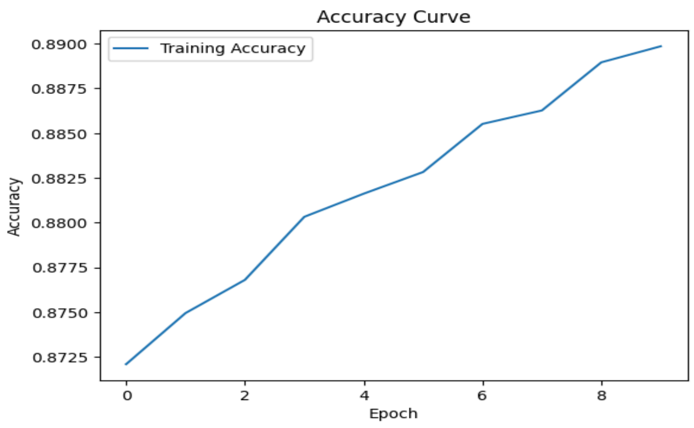
Epoch 1, Loss: 0.9590461246867975
Epoch 2, Loss: 0.5909971048583587
Epoch 3, Loss: 0.5049030164188395
Epoch 4, Loss: 0.46161705132201314
Epoch 5, Loss: 0.4344852385893464
Epoch 6, Loss: 0.41514970873296 26
Epoch 7, Loss: 0.40042777959691983
Epoch 8, Loss: 0.3878069570722679
Epoch 9, Loss: 0.3774289210420102
Epoch 10, Loss: 0.368410297545294

Figure 1: Loss vs epochs for NN Classifier

Over the course of ten epochs, the training loss dropped progressively from an initial loss of 0.96 to a final loss of 0.37. This suggests that as training went on, the model successfully learnt from the data and enhanced its predictions. The decrease in loss indicates a progressive improvement in the model's predictions' agreement with the real labels. The loss curve's general decreasing tendency suggests that the model is picking up new skills and becoming more proficient across the training epochs. This is encouraging since it shows that the model is becoming better at minimizing the error between its predictions and the actual labels. It appears that the curve has peaked at the end, indicating that the model may have achieved a point of convergence. This suggests that more training epochs could not result in appreciable performance gains. Furthermore, the model is still learning and might benefit from more training if the slope keeps going down without plateauing.

The learning rate can have an impact on the curve's form. A low learning rate might lead to sluggish convergence, whereas a high learning rate can cause the curve to vary or even diverge. Determining the ideal rate of learning is essential to effective training. The optimal learning rate may be found with the use of strategies like learning rate scheduling or grid search. It's critical to monitor both the training and validation losses in order to evaluate the model's capacity for generalization.

The training accuracy also improved with time, going from 87.21% in the first epoch to 88.99% in the last, demonstrating a steady improvement in the model's capacity to accurately categorize the text completions supplied by LLM. The classifier's accuracy in predicting the right labels from the embeddings is monitored throughout training.



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 accuracy curve's general increasing trend suggests that the model is doing better generally over the training epochs. This is encouraging as it shows that the model is becoming better at categorizing the data. It appears that the curve has peaked at the end, indicating that the model may have achieved a point of convergence. This suggests that further training epochs could not result in appreciable accuracy gains. Furthermore, if the curve keeps growing without reaching a plateau, the model is still developing and may use more practice. The result at the last epoch can be used to evaluate the model's final accuracy. Better overall performance is indicated by a greater final accuracy.

A reasonably flat curve with few significant oscillations indicates training-induced steady progress. The learning rate can have an impact on the curve's form. A low learning rate might lead to sluggish convergence, whereas a high learning rate can cause the curve to vary or even diverge. Tracking the validation accuracy in addition to the training accuracy is crucial for evaluating the model's capacity for generalization. Overfitting may be indicated if the validation accuracy plateaus or declines while the training accuracy keeps rising. Overfitting can be lessened by employing strategies like early halting and regularization.
The test set was used to evaluate the model once training was finished.

While marginally less than the training accuracy, the test accuracy of 84.68% is still a strong indicator that the model generalizes well to new data. Although there may be some overfitting based on the discrepancy between test and training accuracy, overall test performance is still strong.

Test Accuracy: 84.68%

A confusion matrix was created in order to evaluate the model's effectiveness in categorizing certain LLMs (Figure 3). The confusion matrix indicates regions where the model produced inaccurate predictions and shows the frequency with which each LLM was properly categorized. Correct classifications are represented by diagonal entries, whereas incorrect classifications are shown by off-diagonal entries. Good overall performance is shown by the diagonal elements, which reflect correct predictions, being much bigger than off-diagonal components. 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.

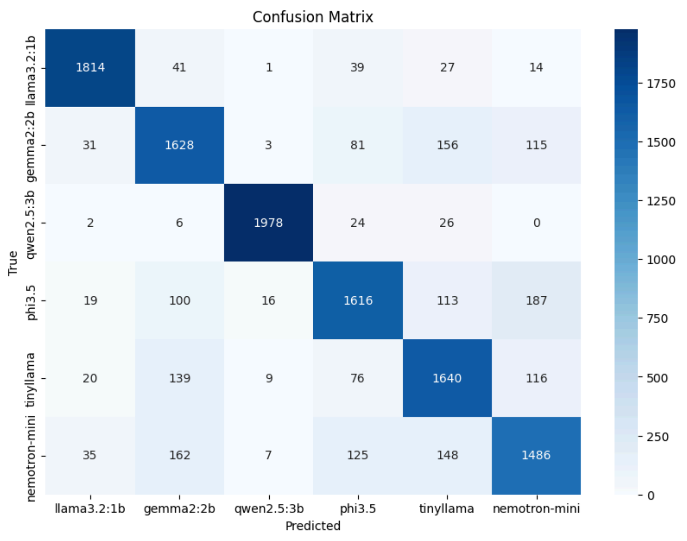


Figure 3: Confusion Matrix of NN Classifier

The confusion matrix indicates a class imbalance, with LLaMA 3.2, Gemma, and TinyLlama having much more occurrences than the other classes. With its large diagonal value and very small off-diagonal values, the model LLaMA 3.2 looks to be correctly classified and contains the greatest number of cases. Similar to LLaMA 3.2, Gemma 2B performs well with low off-diagonal values and a high diagonal value. The diagonal value of Qwen 2.5 is still very important, but there are some noticeable off-diagonal values that might cause confusion with other classes. Phi-3.5 shows high classification accuracy with a strong diagonal value and relatively modest off-diagonal values. Similar to LLaMA and Gemma, TinyLlama performs well with a high diagonal value and low off-diagonal values. nemotron-mini has the lowest number of instances and might be more challenging to classify. An extremely small diagonal value and several separate off-diagonal values suggest that there could be class misunderstanding.

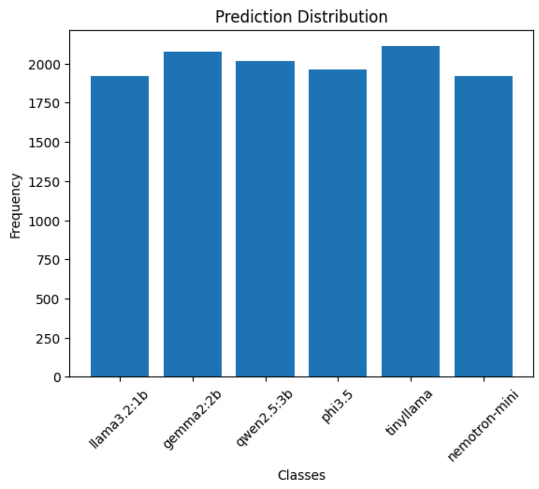


Figure 4. Prediction Distribution of NN Classifier

The classifier was not biased toward any one model, as seen by the balanced distribution of predictions across all six LLMs in a bar plot of the predicted labels (Figure 4). With far more instances of models LLaMA 3.2, Gemma, and TinyLlama than the rest, the bar graph validates the class imbalance shown in the confusion matrix. The distribution has a small rightward skew, suggesting a greater number of cases in the higher-order models, TinyLlama and Nemotron-mini. The findings from the confusion matrix are consistent with the forecast distribution. In the bar graph, the classes with greater diagonal values in the confusion matrix often have higher frequencies.

Other In-depth analysis and experimentations:

We ran a preliminary experiment using a Wikipedia dataset before testing the model with the HellaSwag dataset. We prepared a dataset of 10,000 phrases for each of the six Large Language Models (LLMs) that would be employed in this research during the initial stage. This produced 60,000 sentences in all, representing a wide variety of sentence completions in the various LLMs. We conducted experiments with the following six LLMs: Phi-3.5 (3B), LLaMA 3.2 (1B), Gemma 2b (2B), Qwen 2.5 (3B), TinyLlama (1.1B), and nemotron-mini (1B). These LLMs offer a rich number of embeddings for training and assessment, and were selected due to their diversity in parameter sizes and structures.

The Wikipedia dataset was embedded using the stella_en_1.5B_v5 SentenceTransformer model for this experiment. Our neural network (NN) classifier was then given these embeddings and given the job of guessing which LLM produced the completion for a given text. Using the Wikipedia dataset, the neural network classifier that was trained using these embeddings had an accuracy of 81%. This was a promising beginning, indicating that the model could successfully distinguish, using phrase embeddings, between the output styles of various LLMs.

In this experiment, the LLMs were asked to complete sentences using the following cue, which had the following structure:

```
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}"')
    ]
)
```

The purpose of this prompt is to motivate the LLMs to provide answers solely based on their innate knowledge and sentence completion skills, without depending on any external inquiries or real-time data. For this stage of the project, it was applied consistently to all LLMs to guarantee uniformity in sentence completions.
Following our studies with the Wikipedia dataset, we turned our attention to the HellaSwag dataset, which offers a more complex and difficult collection of text completion challenges. utilizing this dataset, we ran two sets of experiments: one utilizing the above-described prompt and the other without.

In the first trial, we found that the model's accuracy on the HellaSwag dataset was 82% without the use of the prompt. This was already better than what was seen in the Wikipedia sample, indicating that even in the absence of a prompt, the model was more adept in differentiating between LLM completions on more difficult tasks. Next, using the stella_en_1.5B_v5 embeddings, we applied the prompt to the HellaSwag dataset and tested with two classifiers: the BERT classifier and the NN classifier. With the unshuffled dataset, the accuracy rose to 89.92% when the BERT classifier was used. However, the BERT classifier's accuracy decreased to 89.35% when we shuffled the dataset before to training. This suggests that how well the model learns to identify LLM outputs may depend on the sequence in which the data are presented.

Using the stella_en_1.5B_v5 SentenceTransformer embeddings, the NN classifier obtained an accuracy of 87% on the HellaSwag dataset when the same prompt was applied. Even while this was somewhat worse than the BERT classifier, it was still a significant improvement above the accuracy obtained without prompting, confirming the usefulness of employing a prompt to direct LLM sentence

completions.
We also ran a quick test on another model, DeepSeek-Coder (1B), during our investigations, mostly to see how well it performed with code-related results. In a coding setting, we generated sentence completions using the model. However, the findings from DeepSeek-Coder were excluded from the final experimental analysis because the project's main focus was on general LLM sentence completions. In spite of this, it offered intriguing insights into the ways in which various models manage particular task domains, such as coding, and it may be investigated further in further research.

By conducting methodical experiments on the Wikipedia and HellaSwag datasets, we were able to show the value of employing prompts and the efficacy of our NN classifier. The prompt considerably improved classification performance, with both BERT and NN classifiers showing large improvements in accuracy when it was implemented. The studies show how various datasets and task complexity may affect model performance and how important it is to select the right LLM and classifier architecture to get the desired results.

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 brief error examination of our BERT-based classifier model's confusion matrix reveals:

Certain patterns of confusion can be found by analyzing the off-diagonal components. The model may be having trouble telling these two LLMs apart, for instance, if there are high values between classes 2 and 3.

The properties of the incorrectly categorized cases can shed light on the BERT-based classifier's shortcomings. For example, if the model consistently misclassifies texts with specific subjects or linguistic traits, it may indicate that the model is having difficulty capturing these subtleties.

This classifier model might be further improved by:

Enhancement of Data: Data augmentation methods can be used to produce additional instances of underrepresented classes in order to redress the imbalance in class representation.

Fine-tuning the model: Trying out other hyperparameters, architectures, or pre-trained models may help you get better results.

Feature engineering: By adding characteristics other than textual data, such stylistic or semantic details, the model may be better able to distinguish between different LLMs.

The BERT-based classifier model's prediction distribution indicates the following:

Outliers: Extremely high or low frequencies might be a sign of abnormalities or outliers in the data.

Data Preprocessing: By knowing the causes of the class imbalance, data preprocessing methods like oversampling or undersampling may be more effectively used to solve the problem.

Furthermore, based on our observations of the classifier models (NN and BERT-based), we were able to draw the general conclusion that the following factors might affect the accuracy and loss metrics as they were seen during training and evaluation:

Data Imbalance: The classifier's capacity to discriminate between LLMs may be impacted if certain LLMs were either over- or underrepresented in the dataset. An imbalance of data might cause the model to prefer the classes that occur more frequently.

Embedding Quality: The neural network's ability to learn the connections between LLMs is influenced by the quality of the embeddings, which are created from BERT-like architectures. Noisy or inaccurate embeddings may result in increased loss and decreased accuracy.

Model Complexity: The two fully connected layers of the neural network classifier model may not be adequate for more intricate patterns between the text completions of the LLMs, particularly those with bigger parameters like Phi-3.5 and Qwen 2.5.

Batch Size: The gradient estimations could have been affected by the training's 16-person batch size. While a lower batch size may introduce more noise and hinder convergence, a bigger batch size may yield more steady updates.

Learning Rate: The model's rate of learning is determined by the learning rate (1e–4). While a learning rate that is too low might slow down learning and avoid attaining the optimal loss, a learning rate that is too high could lead to the model overshooting ideal weights.

Overfitting: The model may have trained to perform well on training data but finds it difficult to generalize to new test data, as indicated by the small difference between training and test accuracies.

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