CSE 584 Midterm Project

October 7, 2024

Name	PSU ID	Email
Rohan Prasad	980707395	rpp5524@psu.edu
Kush Harish Vora	946282438	kkv5177@psu.edu
Varunsai Alaparthi	917181259	vsa5067@psu.edu

Problem Statement

Given a set of truncated texts, for each piece of text Xi, such as "Yesterday I went", ask different Large Language Models (LLMs) to complete it by appending Xj ="to Costco and purchased a floor cleaner." so you get a complete text like "Yesterday I went to Costco and purchased a floor cleaner." from each LLM. The same Xi leads to different Xj. Now please build a deep learning classifier to figure out, for each input (Xi, Xj), which LLM was used for this pair.

Dataset Curation

Data Sources

To provide a rich and varied input for testing the language models, we carefully selected the dataset for our experiment from a variety of textual sources. Three main sources were used by us:

- **Stanford Corpus**: We took **2,000** sentence starts, or Xi values, from of the **Stanford Corpus** on **Kaggle**. To mimic real-world situations where language models would need to produce continuations from partial texts, these phrases were shortened at random. Stanford Corpus
- **Recently Published Research Papers**: We selected sentence starters from a variety of recently published research papers across many disciplines in order to include a touch of formal and academic language. We took sentences from medical papers like "*A historical review of classic articles in surgery field*" [8] and "*Lifelong Learning*".
- **Novels, Books and Articles**: In order to capture language that is more casual and narrative in nature, we also went through a number of current articles and contemporary novels. In keeping with the Stanford Corpus, sentences were arbitrarily shortened in these sources. We derived truncated sentences from books like "*Introduction to Machine Learning Alex Smola and S.V.N. Vishwanathan*" Textbook.

Language Model Utilized

We used many cutting-edge language models to construct the Xj values, or the AI-generated continuations of the Xi sentences, ensuring a thorough evaluation across various architectures and capabilities:

- GPT4 (32k context window)
- Mistral-7B-Instruct-v0.3
- Meta-Llama-3-8B-Instruct
- Llama 3.2 3B Instruct
- Llama 3.1 70B-instruct
- Qwen2.5 7B Instruct
- Gemini 1.5 Flash

Automation and Data generation Process

We wrote a script that interacts with each language model to automate the process of creating Xj. The script runs with a prompt appended to the sentence with few shot examples in the prompt.

The automation script records the output of each model (Xj) after transmitting the abbreviated text (Xi) to them. To keep the produced outputs consistent, we made sure every model ran under the same parameters. To investigate how these variables affect the text production quality and style, variables including temperature, top p, and max tokens were carefully adjusted and modified.

Figure 1 shows a preview of how our dataset looks like for mistral.

Locally hosted models:

- Mistral-7B-Instruct-v0.3
- Meta-Llama-3-8B-Instruct
- Llama-3.2-3B Instruct
- Qwen2.5-7B-Instruct
- Llama 3.1-70B-Instruct

Ollama Endpoint: Ollama REST API

```
# Define the URL for the API endpoint
url = 'http://localhost:11434/api/generate'

# Generate completion with a local LLM
response = requests.post(
    url,
    json=data
)
```

And we used REST API for:

- GPT4-32K via OpenAI OpenAI Documentation
- Gemini-Flash-1.5 Google Gemini API

LLM Configurations

```
data = {
    "model": $LLM
    "prompt": $PROMPT,
    "stream": False,
    "eval_count": 8192, # Default
    "options": {
        "temperature": 0.2,
        "top_k": 0,
        "top_p": 0.9,
        "presence_penalty": 0,
        "frequency_penalty": 1,
    },
}
```

Prompt Used

```
def get_prompt(sentence):
    prompt = You are an expert sentence completion bot. I will provide you with incomplete
         sentences. Your job is to complete these sentences in 1 or 2 lines. Also, the
         output should just be the remaining part of the sentence and not the entire
         sentence. I am providing you with a few examples of input and expected output.
         Example 1:
         input: The rain was
         output: going to flood the entire city
         Example 2:
         input: The party was about to end after
         output: the birthday cake was distributed
         Example 3:
         input: Jack fought with him because
         output: he was insecure and jealous
         Now it is your turn, complete this sentence and provide me only the remaining part
             of the sentence:

    prompt += sentence
    return prompt
```

Classifier and HOW we trained it

Our primary goal is to train/use classifiers capable of detecting data distribution shifts in texts generated by models like Mistral, LLaMA, and Qwen. BERT and RoBERTa are particularly suitable for this task due to their robust ability to capture deep contextual representations and generalize well across a range of NLP tasks.

	index	Xi	Xj	model
1	0	Shady man waiting	...in a dark alley for his next move.	mistral:latest
2	1	People moving by a	were ignoring social distancing guidelines	mistral:latest
3	2	A contact juggler performing	at a bustling festival	mistral:latest
4	3	a man is carrying a load	of firewood on his back	mistral:latest
5	4	Two people ride their yellow bikes	down a winding country road	mistral:latest
6	5	A city street shows cars moving along the road, a man in black clothing walking,	...and a group of children playing nearby.	mistral:latest
7	6	Blond lady with apron inquires about	the ingredients for a specific recipe	mistral:latest
8	7	A black-blue insect flying	around the room	mistral:latest
9	8	An old woman with a mustard sweater and white skirt	sat on a park bench feeding pigeons.	mistral:latest
10	9	Three people are looking at a	map trying to find their next destination.	mistral:latest
11	10	A woman with glassing working to	at a high-powered law firm.	mistral:latest
12	11	Two girls, one wearing a red shirt and one wearing a gray and	were arguing over who had the newest smartphone.	mistral:latest

Figure 1: Dataset Preview

Classifier

BERT:

- BERT's [4] *bidirectional architecture* allows it to capture intricate relationships between tokens in both directions.
- This bi-directionality is crucial for detecting *subtle changes in structure or semantics* in text generated by different language models.
- The use of *Masked Language Modeling (MLM)* makes BERT effective at predicting missing parts of a sentence.
- This capability enables BERT to understand *nuanced shifts in token distributions*, which is essential for identifying *data drift* in generated text.

RoBERTa:

- RoBERTa[6] is built on BERT but *removes the Next Sentence Prediction (NSP) task*, focusing entirely on token-level understanding.
- It is trained with *larger datasets* and *longer sequences*, making it more robust in capturing *distributional changes* across different text sources.
- RoBERTa's ability to handle more data during pre-training results in *better sensitivity to shifts* in language patterns, especially in generated text.
- This model is particularly well-suited for detecting *fine-grained distribution shifts*, making it ideal for distinguishing texts generated by various LLMs.

Both models are well-suited for identifying changes in generated text because they are pre-trained on large, diverse datasets, allowing them to generalize well across various domains and capture complex linguistic shifts. By fine-tuning these models on the specific LLM-generated texts,

we aim to leverage their capacity to distinguish variations and adapt to different data distributions effectively.

We have loaded the pre-trained weights for both models from Hugging Face. For BERT, we have used the `'bert-base-cased'` model, and for RoBERTa, we selected the `'roberta-base'` model, which was fine-tuned for sequence classification from the Hugging Face model hub. Since we are leveraging pre-trained models, it is crucial to use the respective tokenizers specifically designed for these models to ensure that the input data is processed in a compatible manner.

The tokenizers play a critical role in transforming raw text into the input format expected by the models. Both BERT and RoBERTa use sub word tokenization strategies, but with slight variations:

- **BERT**: The `BertTokenizer` uses WordPiece tokenization, which splits words into sub words or characters when an unknown word is encountered. This helps BERT handle rare or unseen words more effectively.
- **RoBERTa**: The `RobertaTokenizer` is a modified version of BERT's tokenizer, based on Byte-Pair Encoding (BPE). It doesn't rely on next sentence prediction, and the tokenizer is optimized for a larger training corpus.

```
1
2 tokenizer_roberta = RobertaTokenizer.from_pretrained('roberta-base')
3
4 tokenizer_bert = BertTokenizer.from_pretrained('bert-base-cased')
```

By using these tokenizers, we ensure that the text is tokenized into a format that aligns with the pre-trained models' vocabulary and input requirements. This step is crucial for maintaining compatibility with the models' training process and optimizing the classification task for detecting distribution shifts in text.

Optimization Details

We have further tried various optimizations to reach optimal performance by experimenting with different hyperparameters such as learning rate, batch size, and the number of hidden layers. Due to GPU limitations, we could not conduct extensive experiments, but below are some key observations.

- **Learning Rate:** We initially experimented with a range of learning rates, starting from 1×10^{-2} to 5×10^{-5}. A smaller learning rate resulted in more stable convergence, whereas a larger learning rate led to fluctuating training losses. The optimal learning rate was determined to be 3×10^{-5}, which provided a good balance between speed and convergence stability.
- **Number of Epochs:** We conducted training for up to 15 epochs. However, we observed diminishing returns after around 10 epochs, as the validation loss began to increase, indicating potential overfitting. Hence, we decided to implement early stopping based on the validation loss, which helped in maintaining generalization.
- **Weight Decay:** To prevent overfitting, we implemented weight decay regularization with a coefficient of 0.01. This helped reduce model complexity and improved generalization on the validation set. Through experimentation, we noted a slight improvement in validation accuracy with this regularization technique.

- **Batch Size:** We experimented with various batch sizes, including 16, 32, and 64. A batch size of 32 yielded the best performance in terms of both training speed and model accuracy. Larger batch sizes reduced the variability of gradient estimates but due to gpu limitations we couldn't continuously keep 64.
- **Number of Hidden Layers:** We tested models with varying depths, including 2, 4, and 6 hidden layers. Increasing the number of hidden layers improved the model's capacity to learn complex patterns, but after reaching 4 hidden layers, the performance plateaued. Thus, we found that 4 hidden layers provided sufficient complexity without significant overfitting for Bert and Roberta.

Results

How Results are presented

We conducted several experiments to evaluate the performance of our models across multiple metrics. The following sections present visualizations and key performance measures obtained during these experiments.

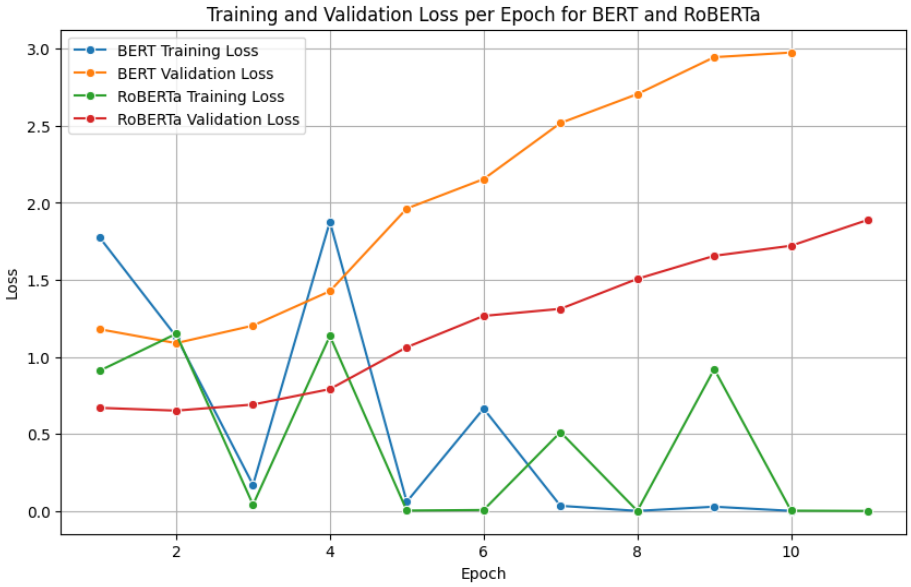

Figure 2: Training and Validation Loss Curves for the BERT and RoBERTa Models. The curves illustrate the model's performance over epochs, with separate lines representing the training and validation losses for each model.

Analyzing these curves in the below figure provides insights into the model's learning process, indicating areas of potential overfitting and guiding adjustments to improve generalization.

Confusion Matrix

The confusion matrix in Figure 3 provides a detailed breakdown of the model's performance by showing the number of true positive, true negative, false positive, and false negative predictions. This allows for a more in-depth analysis of model errors, particularly in distinguishing between different classes. We observed that our models generally performed well, though certain classes posed more challenges in terms of mis-classification.

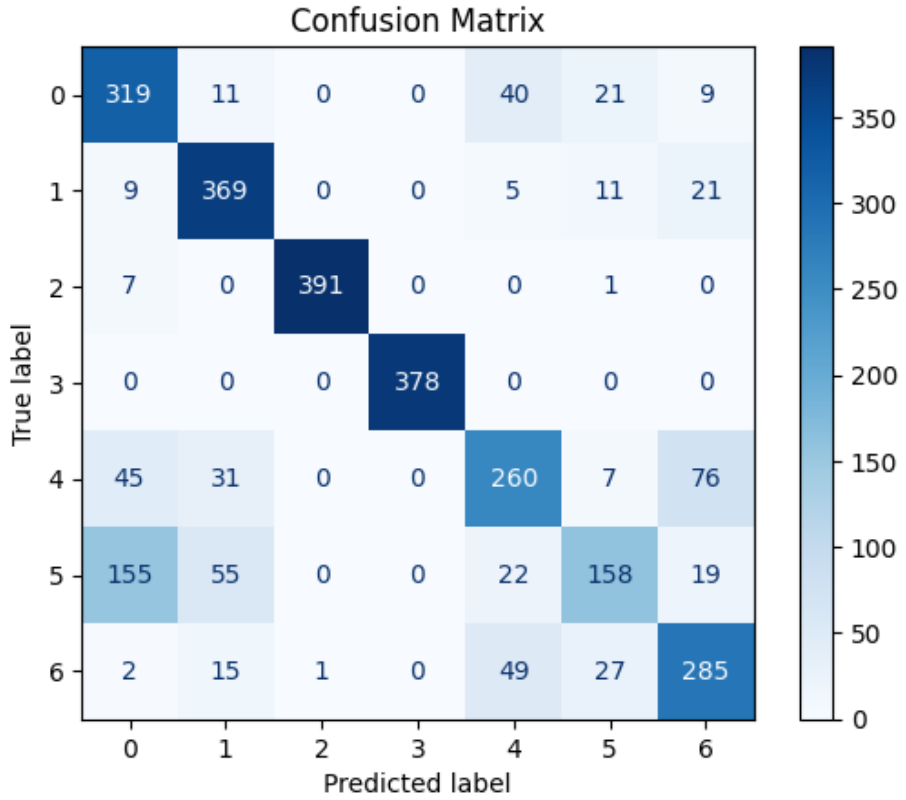

Figure 3: Confusion Matrix for Model Predictions. The labels correspond to the following models: {llama3:latest: 0, llama3.2:latest: 1, mistral:latest: 2, gemini-1.5-flash: 3, llama-3.1-70b-instruct: 4, qwen2.5:latest: 5, gpt-4-32k: 6}.

Training Accuracy and Validation accuracy

We evaluated the performance of our models over multiple epochs to observe how both training accuracy and validation accuracy evolved (Figure 4). Tracking accuracy over epochs helps to identify potential issues like underfiting or overfitting.The following figure illustrates the relationship between validation accuracy and the number of epochs, providing insights into the learning process:

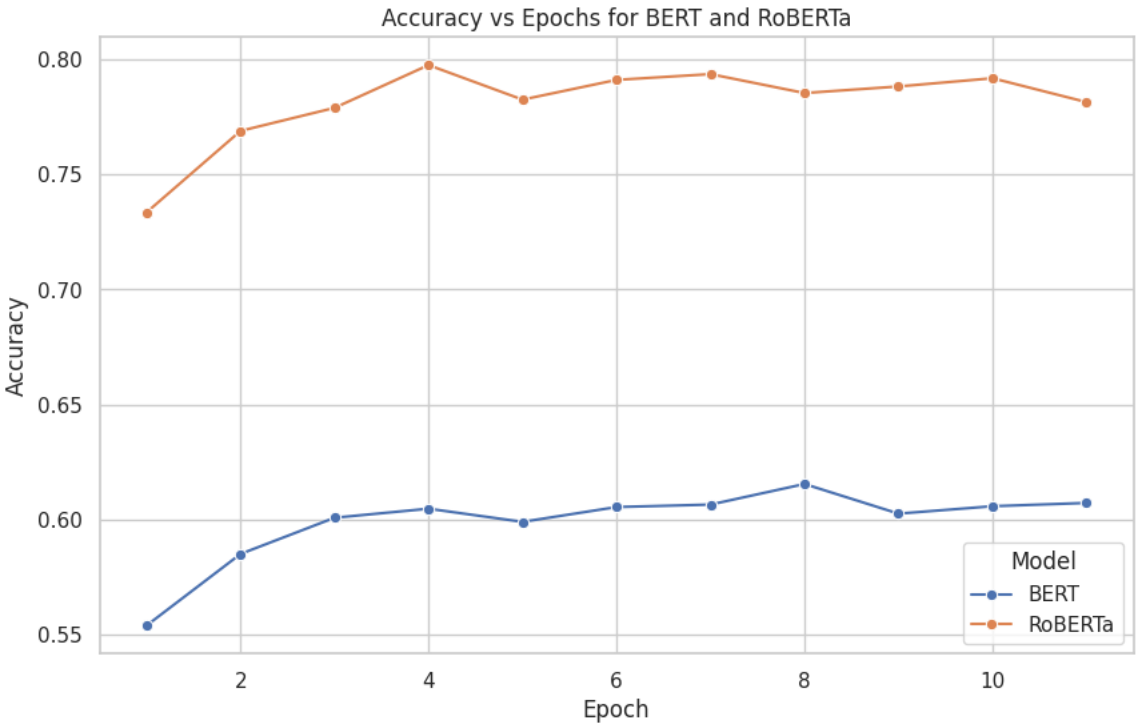

Figure 4: Validation accuracy vs Epochs

F1-Score, Precision, and Recall

In evaluating the performance of our models, we considered three key metrics: **F1-Score**, **Precision**, and **Recall**. These metrics in Table 1 provide a comprehensive view of the model's effectiveness, balancing both precision (the accuracy of positive predictions) and recall (the model's ability to capture all relevant instances). The **F1-Score** represents the harmonic mean of precision and recall, offering a single measure that accounts for both false positives and false negatives.

The table below presents the F1-Score, Precision, and Recall values for the **BERT** and **RoBERTa** models:

Model	Precision	Recall	F1-Score
BERT	0.609	0.608	0.608
RoBERTa	0.783	0.789	0.78

Table 1: F1-Score, Precision, and Recall values for BERT and RoBERTa

In this table:

- **Precision** represents the proportion of correctly predicted positive instances out of all instances predicted as positive.
- **Recall** measures the proportion of correctly predicted positive instances out of all actual positive instances.
- **F1-Score** balances both precision and recall, providing a more nuanced metric for model performance, especially in cases of imbalanced data.

Class-based Accuracy

Class-based accuracy provides insights into the performance of the model for each individual class. It helps in identifying how well the model performs across different categories, allowing for a more nuanced understanding of its strengths and weaknesses. A high accuracy in a specific class indicates that the model is effectively capturing the characteristics of that category, while low accuracy may suggest potential areas for improvement.

The following bar plot illustrates the class-wise accuracies achieved by the model. Each bar represents the accuracy for a specific model, based on the predictions made during the validation phase. This information is crucial for evaluating the reliability of the model's predictions and can guide future enhancements in the model training process.

Experiments and In-depth Analysis

Experiment 1 - Effect of Top p on Validation Accuracy

In this experiment, we analyze how the varying degrees of randomness in token selection, specified by different top-p values, affect the validation accuracy of language model outputs. We adjust top-p at levels 0.8, 0.6, 0.4, and 0.2 while keeping other parameters constant:

```
temperature = 0.2
top-k = 0
AI-generated content = 1-2 lines
maximum evaluation token limit = 8192 (default)
```

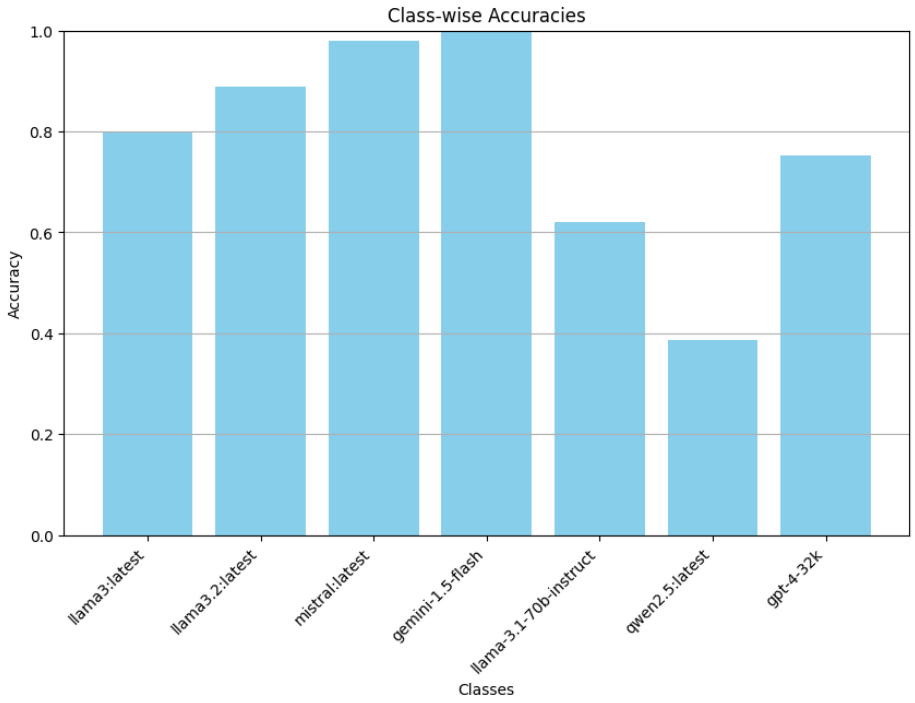

Figure 5: Class-wise Accuracies for the Model Predictions. Each bar represents the accuracy of a specific model in the classification task. This analysis highlights the performance variations across different classes, enabling targeted improvements in the classification process.

The purpose of this setup is to investigate how the model's capacity to predict accurate continuations under limited randomness is affected by the narrowness or broadness of the probability distribution from which the next word is taken.

The results of the experiment show the following validation accuracies for the respective top-p values: 0.6 (top-p = 0.8), 0.7 (top-p = 0.6), 0.6 (top-p = 0.4), and 0.65 (top-p = 0.2). These findings suggest that a moderate level of top-p (specifically, 0.6) leads to the highest validation accuracy, indicating that a balanced approach to randomness may enhance the model's performance. At higher (0.8) and lower (0.4) top-p values, the accuracy tends to decrease, which implies that either excessive randomness or overly constrained sampling can hinder the model's ability to generate coherent and contextually appropriate outputs.

The relationship between the varying top-p values and validation accuracy is illustrated in Figure 6.

Experiment 2 - Effect of Temperature on Validation Accuracy

This experiment focuses on the impact of the softmax temperature on validation accuracy across a range of temperature values: 0.2, 0.4, 0.6, 0.8, and 1. By keeping the following parameters constant:

```
top-k = 0
top-p = 0.9
AI-generated content = 1-2 lines
```

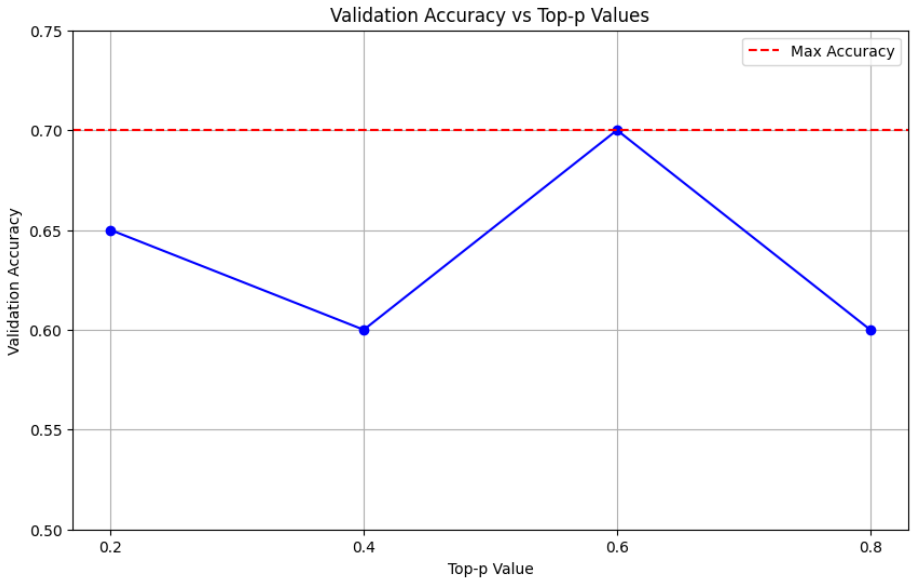

Figure 6: Validation Accuracy vs Top-p Values. The plot demonstrates how varying top-p values impact the validation accuracy of the model's predictions.

```
maximum evaluation token limit = 8192 (default)
```

The changing temperatures will show how the accuracy of the language model's predictions is affected by increasing randomness (higher temperature) versus aiming for more predictable outputs (lower temperature).

The results indicate that at a temperature of 0.6, the model achieved the highest validation accuracy of 0.72. This suggests that this temperature strikes an optimal balance between creativity and coherence in the generated text. Conversely, at lower temperatures (0.2 and 0.4), the model's accuracy remained stagnant at 0.6, indicating that overly deterministic outputs may limit the model's ability to capture the nuances in the data. Higher temperatures (0.8 and 1.0) resulted in decreased accuracy (0.68 and 0.64, respectively), which implies that the increased randomness may lead to less relevant outputs for the classification task.

The relationship between the varying temperature values and the corresponding validation accuracy is illustrated in Figure 7. The plot demonstrates how temperature influences the validation accuracy of the model's predictions:

Experiment 3 - Effect of Percent of AI-generated Words on Validation Accuracy

In this test, we explore the influence of the length of AI-generated content on the validation accuracy by changing the number of words in the prompts—ranging from 5 to 25 words. The constants in this experiment are:

```
temperature = 0.2
top-k = 0
top-p = 0.9
```

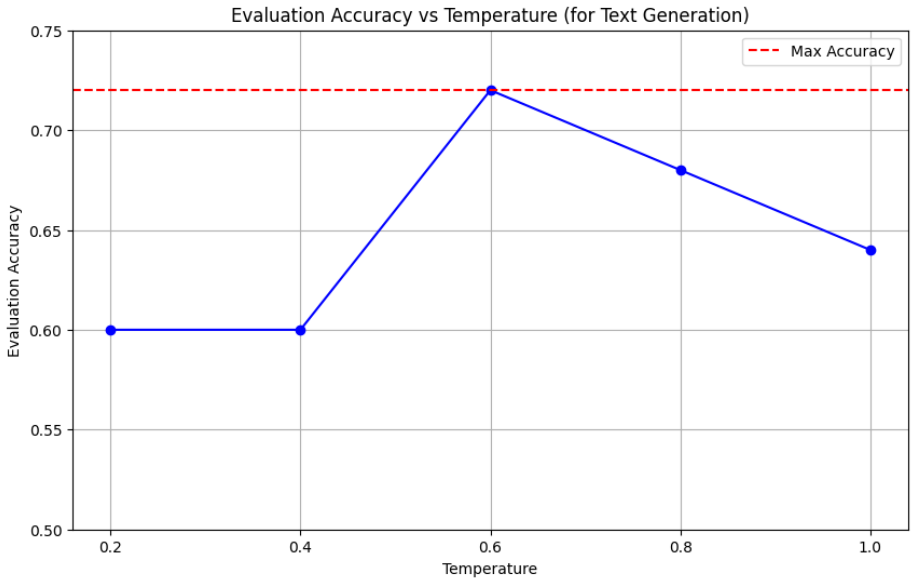

Figure 7: Evaluation Accuracy vs Temperature for LLM-Generated Content. The plot demonstrates how temperature influences the validation accuracy of the model's predictions.

```
maximum evaluation token limit = 8192 (default)
```

This investigation will provide insights into the ideal length for accurate AI-generated text under controlled situations by determining whether longer or shorter AI-generated responses have a more substantial impact on retaining contextual accuracy and coherence.

The results show a decreasing trend in validation accuracy as the percentage of AI-generated content increases. Specifically, the validation accuracies observed are as follows: 0.6 (5%), 0.56 (10%), 0.56 (15%), and 0.48 (20%). This suggests that a moderate level of AI-generated content may be more beneficial for maintaining accuracy, whereas excessive AI-generated text could introduce noise or reduce coherence, leading to diminished performance in the classification task.

The relationship between the percentage of AI-generated words and validation accuracy is illustrated in Figure 8.

Examples

In this section, we present some text samples along with their predicted and original labels. The table below illustrates the model's performance on various inputs.

Original Text	**Predicted Label**	**Original Label**
Two people ride their yellow bikes on water	Llama3	Human
People moving by a busy street often seem like blurs.	gemini	gemini
a man is carrying a load on his back, struggling to maintain his balance.	gpt4	gpt4

Table 2: Text Samples with Predicted and Original Labels

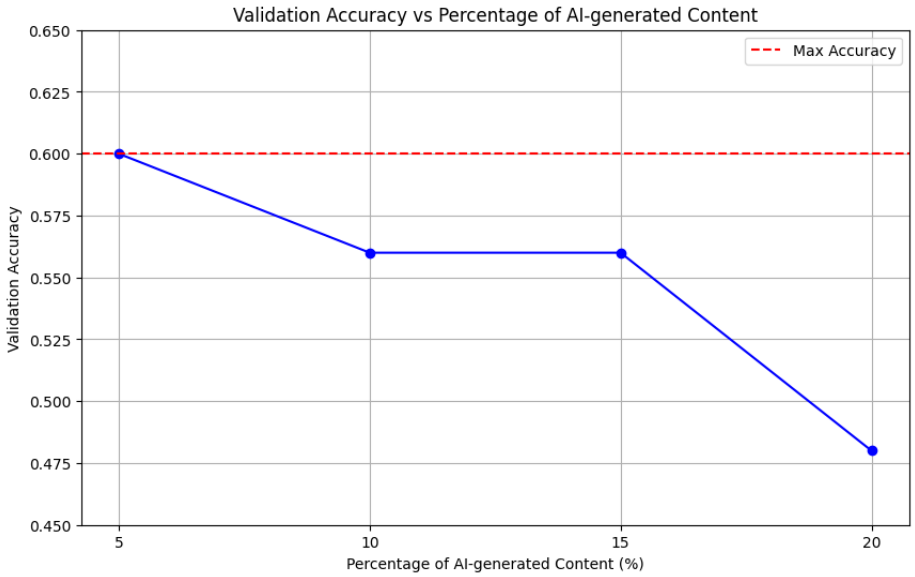

Figure 8: Validation Accuracy vs Percentage of AI-generated Words. The plot demonstrates how varying percentages of AI-generated content impact the validation accuracy of the model's predictions.

We have also observed that when sentences didn't make much sense like the example in the first row in Table2, it's always trying to classify the class to Llama3.

Discussion of Related work

The goal of our project—to classify which LLM generated a text completion given a truncated text (Xi) and its corresponding completion (Xj)—aligns closely with ongoing research in AI-generated text detection. The key challenge here is not just identifying whether a text is generated by an LLM (or AI), but determining which specific model produced the text, based on linguistic, syntactical, or probabilistic patterns that are inherent to different LLMs.

One of the notable works in the literature that focuses on something similar is DetectGPT by Mitchell et al. [1]. Their work is a zero-shot detection method that utilizes log probability curvature that can distinguish between human-written and LLM-generated texts. The primary advantage of DetectGPT is its ability to perform detection without the need for labeled datasets or fine-tuning, making it highly adaptable to new models. In their experiments, DetectGPT significantly outperformed other zero-shot approaches, achieving an AUROC of 0.95 when detecting fake news articles generated by GPT-NeoX. However, while their approach was effective in zero-shot settings, DetectGPT was primarily focused on binary classification (human v/s AI generated), and does not specifically address the task of identifying which LLM produced a given text. This limitation suggests the need for classification approaches that can distinguish between the LLM used for the content generation.

Building on this work, Mo et al.[2] proposed a fine-tuned approach with their Transformer-based

model that combines CNN with LSTMs. Their method also incorporated transformer layers that enhanced the detection accuracy of AI-generated text. Unlike prior work like DetectGPT, which performed remarkably well in zero-shot scenarios, this work was trained on labeled datasets and reaches a remarkable accuracy of 99%. This demonstrates the power of supervised learning in detecting subtle patterns in AI generated text. Due to a combination of CNN and LSTM layers, the model was able to capture local and sequential dependencies. While this makes their model effective against complex detection tasks, it suffers from generalizing across different tasks. Hence, it was seen that their in-domain training did not suffice when the model was tested on out-of-domain data, limiting their applicability to texts from other domains. To ensure that our work does not suffer from this problem, we made sure that our training dataset included data from a variety of sources. Specifically, we used formal language from research papers, informal language from novels, and everyday language from the Stanford Corpus.

To address the limitations of out-of-domain detection, Wang et al. [3] introduced the LLM-Detector, which leverages instruction tuning to align LLMs with specific text detection tasks. LLM-Detector excels in both document-level and sentence-level classification, achieving 98.52% accuracy in in-domain detection and maintaining a high 96.70% accuracy in out-of-domain detection. This robust performance, especially in sentence-level detection, makes LLM-Detector an ideal candidate for tasks where text fragments or short completions need to be classified, as is the case in our project. Wang et al.'s instruction-tuned model is particularly well-suited for handling the kind of sentence-level distinctions that may be required when differentiating between text completions generated by different LLMs.

References

[1] Eric Mitchell et al., *DetectGPT: Zero-shot machine-generated text detection using probability curvature*, International Conference on Machine Learning, PMLR, 2023. `https://proceedings.mlr.press/v202/mitchell23a/mitchell23a.pdf`

[2] Yuhong Mo et al., *Large language model (LLM) AI text generation detection based on transformer deep learning algorithm*, arXiv preprint arXiv:2405.06652, 2024.

[3] Rongsheng Wang et al., *LLM-Detector: Improving AI-Generated Chinese Text Detection with Open-Source LLM Instruction Tuning*, arXiv preprint arXiv:2402.01158, 2024.

[4] Devlin, Jacob et al., *Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).*

[5] Junchao Wu, Shu Yang, Runzhe Zhan, Yulin Yuan, Derek Fai Wong, and Lidia Sam Chao, *A Survey on LLM-Generated Text Detection: Necessity, Methods, and Future Directions*, NLP2CT Lab, Faculty of Science and Technology, Institute of Collaborative Innovation, Department of Chinese Language and Literature, Faculty of Arts and Humanities, State Key Laboratory of Internet of Things for Smart City, University of Macau, 2023.

[6] Yinhan Liu, *RoBERTa: A Robustly Optimized BERT Pretraining Approach*, arXiv preprint arXiv:1907.11692, 2019.

[7] Long X, Huang JZ, Ho YS, *A historical review of classic articles in surgery field*, Am J Surg, vol. 208, no. 5, pp. 841-849, Nov 2014. doi: 10.1016/j.amjsurg.2014.03.016. Epub 2014 Jul 30. PMID: 25167972.

[8] X. Long, J.Z. Huang, Y.S. Ho, *A historical review of classic articles in surgery field*, Am J Surg, vol. 208, no. 5, pp. 841–849, Nov. 2014, doi: 10.1016/j.amjsurg.2014.03.016, Epub 2014 Jul 30, PMID: 25167972.