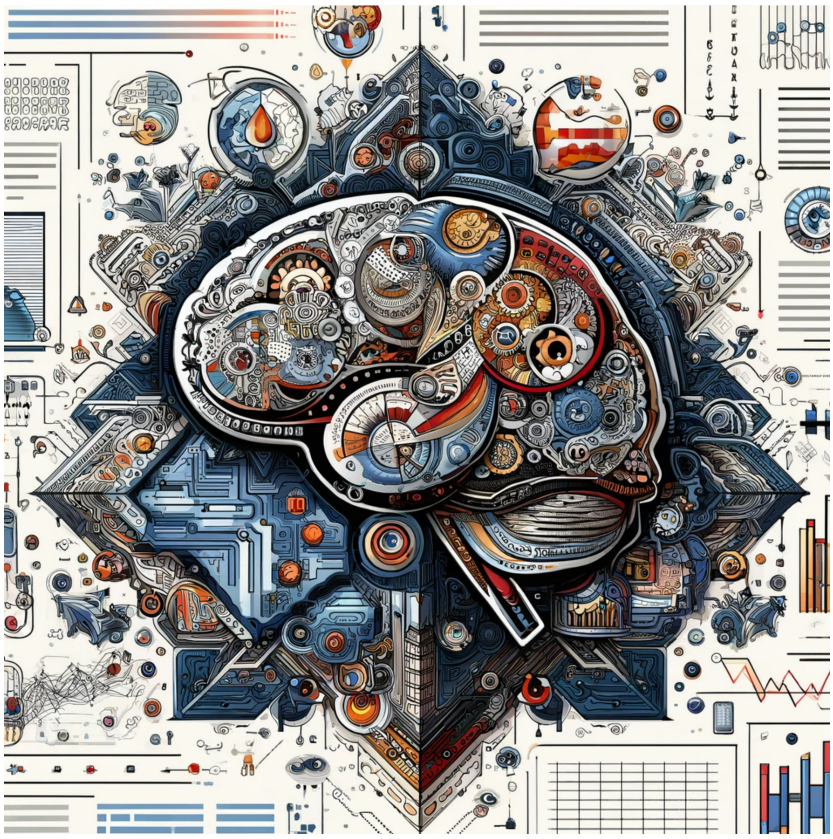

Oppositional Thinking Analysis

Héctor García Hernández y David Andreu Roqueta

May 23, 2024

Contents

1 Objective

The objective of this study is to develop and fine-tune Transformer-based models for the task of analyzing oppositional thinking, specifically distinguishing between conspiracy and critical narratives. This research aims to address one critical challenges in the field of Natural Language Processing (NLP).

Distinguishing Conspiracy from Critical Narratives: This task involves identifying whether a given text reflects a conspiracy mentality or merely questions major decisions without promoting a conspiracist viewpoint. Accurately differentiating between these two is essential for effective content moderation, as mislabeling critical thought as conspiratorial can inadvertently push individuals towards conspiracy communities:

The motivation behind this research stems from the complexity of conspiracy theories, which are intricate narratives suggesting that significant events are the result of plots orchestrated by secret, powerful, and malevolent groups. Understanding and identifying these narratives using NLP models is challenging but vital, especially in the context of automatic content moderation. This research is significant as it addresses the potential harm of misclassifying oppositional narratives and contributes to the broader field of NLP by exploring advanced Transformer-based models.

1.1 Scope

The scope of this study includes:

- Developing separate models for English and Spanish datasets.
- Utilizing Transformer-based models such as BERT, RoBERTa, and DeBERTa, and their language-specific variants (e.g., BETO for Spanish).
- Implementing a binary classification task to distinguish between critical and conspiracy texts, with the goal of achieving a high Matthews Correlation Coefficient (MCC).

1.2 Modeling Approach

We have chosen Transformer-based models due to their proven effectiveness in various NLP tasks. Traditional methods have been explored previously; now, we aim to leverage the power of advanced models to achieve better performance. Specifically, we will focus on BERT, RoBERTa, and DeBERTa models, which have demonstrated strong capabilities in understanding complex textual data.

1.3 Goals and Outcomes

The primary goal is to develop robust models capable of accurately classifying texts as either conspiracy or critical. We aim to achieve the highest possible MCC to ensure reliable model performance. Additionally, we will explore the identification of key narrative elements within texts, providing a deeper understanding of how oppositional thinking is structured.

1.4 Broader Impact

This research has significant implications for content moderation on digital platforms. By accurately distinguishing between conspiracy and critical thinking, our models can help prevent the unintended promotion of conspiracy theories. Furthermore, understanding the elements of oppositional narratives can provide valuable insights into how these narratives spread and influence public opinion.

In conclusion, this study aims to explore and evaluate various models and hyperparameters to develop effective tools for analyzing oppositional thinking. While this report represents a sample of the possible configurations, it sets the foundation for future research that may explore additional parameters and models, subject to computational constraints.

2 Data

The dataset provided for the "Analysis of Oppositional Thinking: Conspiracy versus Critical Narratives" task consists of JSON files containing texts with their respective annotations. Each text entry in the training dataset is represented as a dictionary with the following fields:

- **id:** A unique identifier for the text.
- **text:** The raw text content to be analyzed.
- **category:** A binary label indicating whether the text is a "CONSPIRACY" or "CRITICAL" narrative.
- **annotations:** A list of dictionaries, each corresponding to an annotated span within the text. Each annotation includes:
  - **span_text:** The text span that has been annotated.
  - **category:** The category of the annotated span (e.g., "NEGATIVE_EFFECT," "VICTIM," "AGENT").
  - **start_char:** The starting character index of the span within the text.
  - **end_char:** The ending character index of the span within the text.

An example of a fully annotated text entry is shown below:


```
{
    "id": "91221",
    "text": "\" Scientism backed ... not \" \" science - backed \" \" .
    There is nothing scientific about the Covid or childhood
    vaccine quackery that is slowly but surely killing ,
    maiming and neurologically injuring the next generation -
    just as the Clintons and other Moloch worshippers want . \" ",
    "category": "CONSPIRACY",
    "annotations": [
        {
            "span_text": "that is slowly but surely killing , maiming and
            neurologically injuring the next generation",
```

```
            "category": "NEGATIVE_EFFECT",
            "start_char": 128,
            "end_char": 219
        },
        {
            "span_text": "the next generation",
            "category": "VICTIM",
            "start_char": 200,
            "end_char": 219
        },
        {
            "span_text": "the Clintons",
            "category": "AGENT",
            "start_char": 230,
            "end_char": 242
        },
        {
            "span_text": "other Moloch worshippers",
            "category": "AGENT",
            "start_char": 247,
            "end_char": 271
        }
    ]
}
```


During the testing phase, the dataset will only include the "id" and "text" fields. The required output is a JSON file with each text annotated with its category and span annotations.

Before feeding the text data into the models, a tokenizer is used to convert the raw text into a format suitable for model processing. The tokenizer from the `transformers` library by Hugging Face is utilized, as shown in the following code snippet:

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained(model_name)
```

Here, `model_name` refers to the specific pre-trained Transformer model being used (e.g., `bert-base-uncased`, `roberta-base`). The tokenizer performs several tasks:

- **Tokenization:** Splits the text into individual tokens.
- **Padding:** Ensures all sequences have the same length by adding padding tokens where necessary.
- **Truncation:** Trims sequences that exceed the maximum length allowed by the model.
- **Encoding:** Converts tokens into their corresponding numerical IDs required by the model.

This tokenization process is essential for preparing the text data, ensuring it is in the correct format for input into Transformer-based models. This enables the use of models like BERT, RoBERTa, and DeBERTa for the binary classification and span annotation tasks in this study.

3 Model Selection and Architecture Details

In this study, various Transformer-based models tailored to English and Spanish texts are utilized. The models chosen are:

- **English:**
  - `roberta-base`
  - `microsoft/deberta-base`
- **Spanish:**
  - `dccuchile/bert-base-spanish-wwm-uncased`
  - `PlanTL-GOB-ES/roberta-base-bne`
  - `bert-base-multilingual-uncased`

The selected models are robust and well-suited for the respective languages. For English, RoBERTa and DeBERTa are used as they are enhancements of BERT and offer improved performance. For Spanish, we selected RoBERTa and BERT variants that are trained in Spanish text, additionally we will try the multilingual model.

3.1 Theory Behind BERT, RoBERTa, and DeBERTa

BERT (Bidirectional Encoder Representations from Transformers) is a Transformer-based model that uses a bidirectional approach to pre-train a language model. Unlike traditional models, which read text input sequentially (left-to-right or right-to-left), BERT reads the entire sequence of words at once. This approach allows BERT to understand the context of a word based on its surroundings, making it highly effective for a variety of NLP tasks.

RoBERTa (Robustly optimized BERT approach) builds on BERT by improving the training methodology. RoBERTa removes the Next Sentence Prediction (NSP) task used in BERT's pre-training, increases the batch size and learning rate, and trains on a larger dataset with longer sequences. These modifications enhance RoBERTa's ability to understand complex language patterns and improve performance on downstream tasks.

DeBERTa (Decoding-enhanced BERT with disentangled attention) further improves upon BERT by introducing two key innovations: disentangled attention and an enhanced mask decoder. Disentangled attention separates content and position information, allowing the model to process each independently. The enhanced mask decoder refines the masked language model pre-training task, leading to better performance on NLP benchmarks.

3.2 From Embedding to Classification

To implement the chosen models for sequence classification, the `AutoModelForSequenceClassification` class from the `transformers` library is used. This class converts a pre-trained embedding model into a sequence classifier with minimal additional coding.

The architecture for sequence classification involves adding a classification layer on top of the pre-trained embedding model. The pre-trained model generates contextualized embeddings for the input text, which capture the semantic meaning of each token based on its context. The classifier layer, a dense (fully connected) layer, takes these embeddings and computes the final prediction.

- **Embedding Layer:** The pre-trained model's output embeddings are representations of the input text.
- **Pooling Layer:** The [CLS] token embedding (representing the entire sequence) is used.
- **Classification Layer:** A dense layer with a softmax activation function is applied to the [CLS] token embedding to produce probabilities for each class.

The overall process can be summarized as:

1. Input text is tokenized and converted to embeddings by the pre-trained model.
2. The [CLS] token embedding is extracted from the model's output.
3. The classification layer processes the [CLS] embedding to generate a probability distribution over the classes.
4. The class with the highest probability is assigned as the predicted label for the input text.

This approach leverages the rich contextual information captured by Transformer-based models, enabling accurate and efficient text classification.

4 Fine-tuning

Parameter fine-tuning is an optimization technique used to update the weights and biases of a pre-trained model, typically a deep neural network, using a smaller dataset specific to the target task. The goal is to specialize the model's knowledge and representations learned during the initial pre-training phase to better fit the new task or domain, while leveraging the general features and patterns learned from the larger pre-training dataset.

4.1 Utilizing Discriminative Fine-Tuning for Model Optimization

For our project, we are implementing Discriminative Fine-tuning, a refined variant of fine-tuning that involves recalibrating the weights learned by the model based on training examples from the specific task to be solved. This approach allows for more nuanced adjustments of the model parameters at different layers, enhancing its ability to adapt to the nuances of the new domain.

4.2 Discriminative Fine-Tuning Process

Discriminative Fine-Tuning involves several key steps:

- **The Dataset Class**: Essential for handling data operations such as loading, processing, and preparing datasets for both training and validation phases.
- **Model Configuration**: Setting up the model with a sequence classification head appropriate for the task.
- **Training Loop**: Involves forward and backward propagation of the model on the training data, adjusting weights as necessary.
- **Validation Loop**: Regular evaluation of the model's performance on a validation dataset to ensure the model generalizes well and isn't overfitting.
- **Main Code Implementation**: Integrating all components into a cohesive application for training and subsequent prediction on test data.

4.3 Testing Configurations and Hyperparameters

We are testing various model configurations and hyperparameters to identify the optimal settings for our tasks. The model configurations cover multiple languages, as specified in the previous section.

Hyperparameters:

Hyperparameter	**Values**
Optimizers	Adam, RMSprop
Learning Rates	0.5e-5, 1e-6
Learning Rate Schedule	Linear, cosine
Patience	5, 10 epochs
Training Epochs	5, 10
Performance Measure	Matthews Correlation Coefficient (MCC)
Batch Size	64
Maximum Sequence Length	128 tokens

Table 1: Overview of hyperparameters used in model fine-tuning.

For each hyperparameter configuration, we have implemented a 5-fold cross-validation process. This method involves dividing the complete dataset into five equal parts, using each part in turn as a test set while the remaining four parts serve as the training set. This approach ensures that each segment of the dataset is used for both training and validation, thereby providing a robust measure of model performance across different subsets of data. The cross-validation process helps in identifying the most effective hyperparameters by evaluating the model's stability and performance consistency across different folds.

4.4 Experiment Tracking with Weights & Biases

To systematically track and manage our experiments, we utilize Weights & Biases (wandb). This tool enables us to monitor the training progress in real-time, compare different runs, and optimize hyperparameters effectively, facilitating a more structured and data-driven approach to model development.

5 Results

5.1 English Language Model Performance

To predict conspiracy and critical narratives in English, we utilized two transformer-based models: `roberta-base` and `microsoft/deberta-base`. Each model underwent fine-tuning with 19 distinct hyperparameter configurations. Initially, we planned to test 32 configurations. However, after evaluating the first five configurations with a learning rate of 1×10^{-6}, it became apparent that these models performed significantly worse, leading us to exclude these settings (refer to Figure 1).

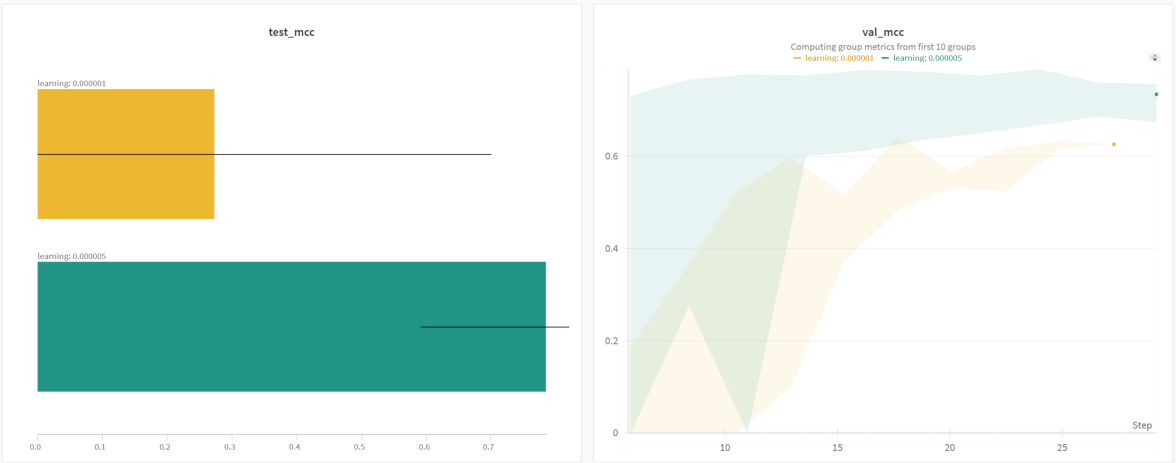

Figure 1: This is a caption for the image.

5.2 Performance Comparison: RoBERTa vs. DeBERTa

We compared the performance of the two models using the Matthews Correlation Coefficient (MCC) and the different hyperparameters we enanunced on the last section. The results demonstrated that `roberta-base` consistently outperformed `microsoft/deberta-base` if we focus on the mean of every variant.

The comparison between RoBERTa and DeBERTa models for predicting conspiracy and critical narratives in English reveals notable differences in their performance and stability (refer to Figure 2). The bar plot on the left illustrates the mean Matthews Correlation Coefficient (MCC) for both models. RoBERTa achieves a mean MCC of 0.78, while DeBERTa lags behind with a mean MCC of 0.68. This clearly indicates that RoBERTa is more effective in accurately distinguishing between conspiracy and critical texts.

Furthermore, the line plot on the right, depicting the validation MCC over different training steps, highlights the stability of each model. The standard deviation of MCC for RoBERTa is significantly lower at 0.07 compared to DeBERTa's 0.20. This suggests that RoBERTa's performance is more consistent across various hyperparameter configurations, whereas DeBERTa exhibits greater variability.

These results indicate that not only does RoBERTa achieve a higher mean MCC, but it also exhibits greater stability across different hyperparameter configurations compared to DeBERTa.

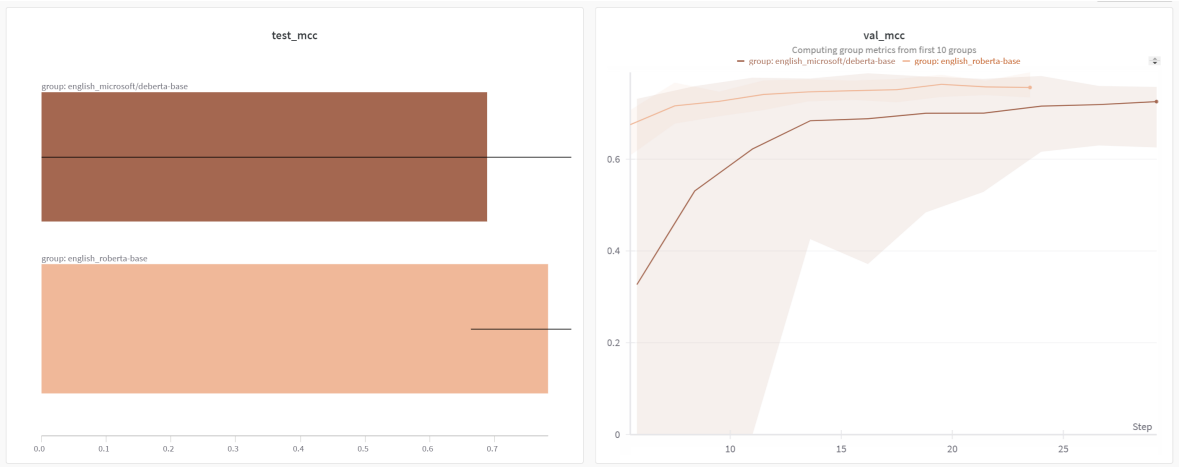

Figure 2: This is a caption for the image.

5.3 Selection of the Best Model

English

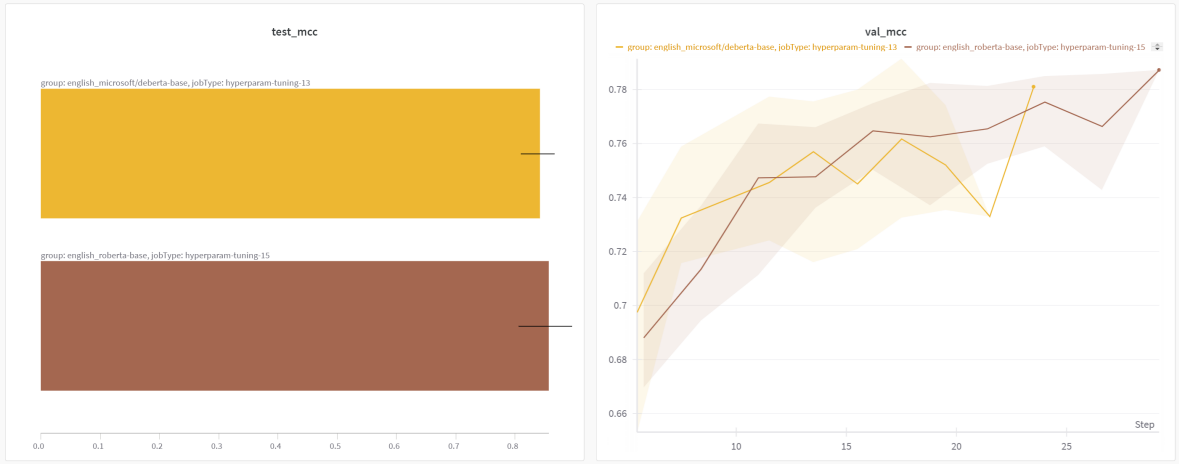

Figure 3: Best english models metrics.

The table comparison reveals that the roberta-base model (represented in brown) outperforms the deberta-base model (represented in yellow) in several key aspects. One of the primary indicators of this superior performance is the higher Matthews Correlation Coefficient (MCC) score. The roberta-base model achieves an MCC of 0.8563, while the deberta-base model scores 0.8419. The MCC is a robust metric that balances precision and recall, providing a comprehensive measure of classification quality. Thus, a higher MCC score suggests that the roberta-base model is more effective at correctly classifying both classes in the dataset.

Moreover, while the roberta-base model has a slightly higher standard deviation in its MCC score (0.04948 compared to 0.0283 for the deberta-base model), the overall higher mean MCC indicates that the benefits of improved accuracy outweigh the slightly increased variability. This level of deviation

Parameter	Model 1: deberta-base	Model 2: roberta-base
Optimizer	RMSprop	RMSprop
Learning Rate	0.5e-5	0.5e-5
Learning Rate Schedule	Linear	Linear
Patience	5	10
Training Epochs	10	10
Performance Measure	MCC	MCC
Batch Size	64	64
Maximum Sequence Length	128 tokens	128 tokens
MCC Score	0.8419	0.8563
MCC Deviation	0.0283	0.04948

Table 2: Comparison of Hyperparameters for the Two Best Models

is still within an acceptable range, suggesting that the performance of the roberta-base model is consistently high across different folds of cross-validation. Additionally, the score of the worst fold for roberta, 0.805, is similar to the 0.809 of deberta's worst.

Spanish

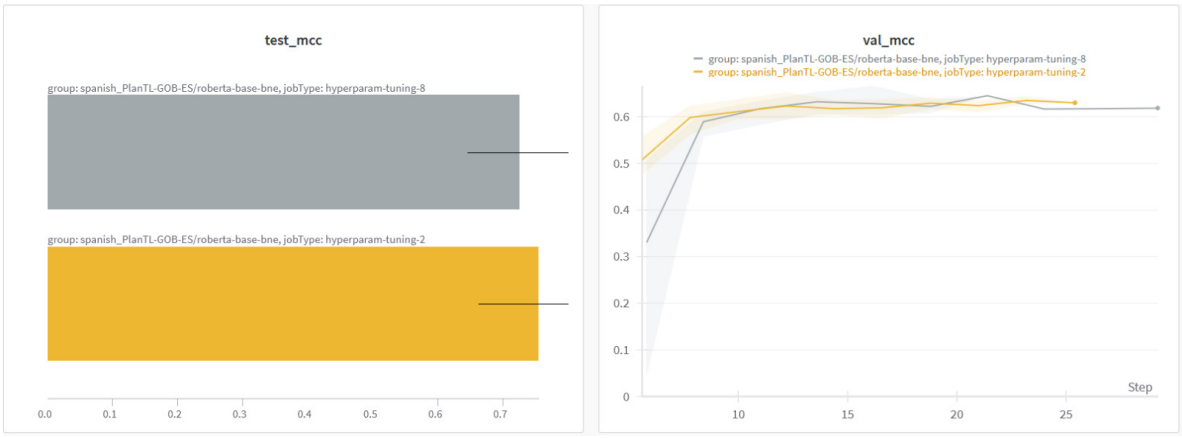

Figure 4: Best Spanish models metrics.

The comparison between the two models, roberta-base-bne with linear learning rate scheduling (Model 1) and roberta-base-bne with cosine learning rate scheduling (Model 2), clearly demonstrates the superiority of the first model (depicted in yellow). The primary metric used for evaluating these models is the Matthews Correlation Coefficient (MCC), a robust measure that takes into account true and false positives and negatives, providing a balanced view of model performance.

Model 1 achieved an MCC score of 0.75, which is higher than the 0.72 achieved by Model 2. This indicates that Model 1 has a better overall performance in correctly classifying the data, balancing the trade-off between precision and recall more effectively. The higher MCC score is a strong indicator that Model 1 is more reliable in distinguishing between the categories of interest, which is crucial for the task at hand.

Parameter	**Model 1: roberta-base-bne**	**Model 2: roberta-base-bne**
Optimizer	Adam	Adam
Learning Rate	0.5e-5	0.5e-5
Learning Rate Schedule	Linear	Cosine
Patience	5	10
Training Epochs	10	10
Performance Measure	MCC	MCC
Batch Size	64	64
Maximum Sequence Length	128 tokens	128 tokens
MCC Score	0.75	0.72
MCC Deviation	0.11	0.09

Table 3: Comparison of Hyperparameters for the Two Best Models

Moreover, the MCC deviation for Model 1 is 0.11, which, although slightly higher than the 0.09 deviation for Model 2, still falls within an acceptable range. This slight increase in variability is outweighed by the overall higher performance. The deviation value indicates that while Model 1 has a slightly broader range of performance across different folds, its superior mean MCC score suggests that it consistently performs better on average, making it a more dependable choice.

The hyperparameter settings for Model 1 also contribute to its superior performance. The use of a linear learning rate schedule, combined with a learning rate of 0.5e-5 and a patience of 5 epochs, suggests a more stable and controlled training process. This setting allows the model to adjust its learning gradually, avoiding the potential pitfalls of rapid learning rate changes that might occur with a cosine schedule, which could lead to underfitting or overfitting.

In terms of training epochs, both models are set to 10, ensuring that the comparison is fair and the observed differences in performance are not due to differences in training duration. The batch size of 64 and the maximum sequence length of 128 tokens are also identical for both models, further validating that the differences in performance are due to the learning rate schedule and not other factors.