Report- Final Project Advances ML and DL Course

Toxic & Harmful Text Classifier using Traditional NN, Mistral, and Llama LLMs

By: Raz Graider, Dana Braynin, Sophie Margolis and Ran Asanta

Introduction

In recent years, the rapid expansion of social media platforms and online communities has not only facilitated greater connectivity and communication but has also revealed a darker side: hate speech and online harassment.

In this project, we explored the challenge of detecting hate speech using machine learning, NLP techniques, and LLMs for sentiment analysis. Our focus was on comparing a **Keras** neural network model to two advanced Large Language Models (LLMs): **Mistral-7b** and **Llama-2-7b**. The objective was to classify text as offensive or non-offensive (a binary classification task), leveraging three distinct datasets.

Our approach involved training a Keras neural network on the entire dataset, which comprised 140,218 rows of text collected from various sources, including Wikipedia forums. However, **the LLMs were fine-tuned using only 0.08% of the training data**, utilizing LoRa (Low-Rank Adaptation), a technique for optimizing model performance. Additionally, we explored the performance of the LLMs with prompt engineering.

The results presented below offer insights into hate speech detection. Through our exploration, we gained a deeper understanding of the challenges, strengths, and limitations of each model or approach for this specific task.

Data Preparation

We used 3 data sets we took from Hugging Face 🤗:

1. stormfront_dataset: Contains data related to hate speech.
2. wiki_dataset: Consists of toxic comments from Wikipedia.
3. jigsaw_dataset: Includes multilingual toxicity data from Jigsaw.

We removed duplicated rows from each data set and performed a train-validation-test split on each one. For the training data, we combined the 3 training data sets and used it for Keras. For the LLMs we used only 0.08% of it.

Exploratory Data Analysis

By observing the EDA we did, we learned that we have 140,102 rows in our combined train data set (1.0 - Data train information) and two types of labels: 0 and 1. We discovered that we have 15,834 samples of offensive comments and 124,268 samples of non-offensive comments (1.1 - Label count).

We also explored the top 20 most common words, with and without stop words, and none of these words are offensive, which aligns with the fact that our data has much more non-offensive comments than offensive. Then, we explored the most common words only from the offensive comments - we can tell that most of them are just hate words and words that indicate racial and sexual insults (1.2 - top 20 most common from offensive comments). Moreover, we checked the average length of each type of comment and also checked the top 15 3-grams-words (1.3 - Top 15 3-grams).

Keras NN

Firstly, performed preprocessing on the train data by removing special characters, removing stop words and performing lemmatization. After that, we also applied the preprocessing on the validation and test data.
We performed tokenization on all the data sets and afterwards, we used a Keras NN model, and checked its outcome on our binary classification task. We built an architecture for our Keras model (2.0 - Keras architecture) and we explored various configurations of NN layers and their parameters:

1. We tried both `MaxPooling1D` and `AveragePooling1D` layers, but the first yielded better results.
2. We attempted to add a `BatchNormalization` layer, but it negatively affected the results.
3. We also determined 0.5 to be the final value for the Dropout regularization layer .We've done so in order to synchronize all Dropout values through our project, from Keras to LLM and fine-tuning.
4. Our last layer, we used sigmoid, which is common for binary classification tasks. In addition, the ReLu activation function is used between the hidden layers.
5. The model was compiled using the `binary_crossentropy` loss function and the adam optimizer, which is commonly used for binary classification tasks.

Overall, this architecture was chosen and tuned based on experimentation to achieve optimal performance for classifying text into offensive or non-offensive categories.
We observe the results of the Keras model in the comparison section in this report.

LLM- Theoretical overview

In this section, we researched about the theoretical overview of the LLM models we use in the project: Mistral and Llama-2. We gave a theoretical overview on LLMs in general and also did a comparison table between Mistral-7b and Llama-2-7b in particular (2.1 - Mistral-7b vs. Llama-2-7b).

For each of the LLM models, we explored prompt engineering and fine-tuning for our task, investigating the strengths and limitations of each model. In our project, one of the main goals is to compare the accuracy provided by prompt engineering to that of fine-tuning, as both strategies play crucial roles in enhancing the performance of AI models.

LLM - Preprocessing

Before moving to the models, we performed preprocessing on the data. We did tokenization for both the train data for Mistral and train data for Llama-2. Both models use the same tokenization from Hugging Face's 🤗 `AutoTokenizer` class in the `Transformers` library (`LlamaTokenizer`) . We also gave an in depth explanation about the tokenization used and showed how it works. After that we handled the imbalanced data by assigning higher weight to the minority class (offensive) and lower weight to the majority class (not offensive). The weights are proportional to the frequency of the classes.

In this section we also defined the performance metric we will use to compare the models which is accuracy.

LLM – Prompt Engineering and modeling

In the prompt engineering section, we developed two functions: `generate_test_prompt` (3.0 - Prompt generate function) and `predict` (3.1 - Prompt predict function). The `generate_test_prompt` function creates a test prompt for sentiment analysis tasks, taking a comment as input and generating a formatted prompt with instructions for analyzing the sentiment of the comment. The predict function predicts the sentiment of a comment. Essentially, for each comment, it generates a prompt asking to analyze the sentiment and returns the label. We utilized some Hugging Face's 🤗 `Transformers` functions to streamline the process, such as `pipeline()`, which generates text from the language model using the prompt. At the end of the process, the model gave a classification for the comment: offensive or not offensive.

In the notebook itself, we dived deeper into the concept of prompt engineering, discussing its advantages and disadvantages. The results of this section are shown below in the section comparing the models.

LLM – Fine Tuning

In this section, we experimented with the fine-tuning strategy, which involves training a subset of parameters of a pre-trained model on a specific dataset to enhance its performance for a particular task or domain. As models grow larger, full fine-tuning, which entails retraining all the model's parameters, becomes less feasible due to the required time, cost, and resources.
We delved into Parameter-Efficient Fine-Tuning (PEFT) techniques, which make fine-tuning more manageable in terms of memory and computational resources. For this project, we fine-tuned the models using the LoRa (Low-Rank Adaptation) PEFT technique from Hugging Face's 🤗 `PEFT` library. Further explanations are provided in the notebook.
The LoRa training arguments provided to both Mistral and Llama were the same, with the Hugging Face's 🤗 `TrainingArguments` class that serves as a container for all hyperparameters and configurations related to the training models. (4.0 - Chosen LoRa parameters; 4.1 - Lora parameters explain; 4.2 - Chosen TrainingArguments; 4.3 - TrainingArguments explain).
We trained 0.025% of the parameters in Mistral and 0.032% of the parameters in Llama.

Finally we used this Trainer class `WeightedCELossTrainer`, which utilizes class weights previously defined to address the challenge of handling unbalanced data (4.4 - Chosen WeightedCELossTrainer ; 4.5 - WeightedCELossTrainer explain).

After that, we were ready to run the trained fine-tuned models. We examine their results, as well as the previous parts, in the next comparison part.

Comparison

For this part we connected to WandB's reports (5.0 - Train reports) to explore the trainer better and these are our conclusions (more in depth explanations are in the notebook):

- **Runtime**
  In both Mistral and Llama-2 models, the time taken for training exceeds that of evaluation. Moreover, it's observed that both training and evaluation times for Llama-2 surpass those of Mistral. This observation reinforces the notion of Llama-2's greater hardware demand explained in the theoretical section.
- **Train & Evaluation Accuracy**
  Consistent accuracy across all epochs in the Mistral model were observed. This prompted us to re-run the training in a separate notebook, yielding identical results. Despite efforts, no exact explanation for this issue could be found. However, it's noteworthy that both models achieved evaluation accuracies exceeding 90%.
- **GPU Power Usage**
  Both Mistral and Llama-2 models heavily utilize the GPU, often reaching approximately 100% usage.

In this part we also examined the results from all the tests we ran during the project (5.1 - Test reports) and these are the conclusions (more in depth explanations are in the notebook):

- **Results for Jigsaw Dataset**
  Mistral-7b Prompt Engineering yielded the highest accuracy (0.9326), while Llama-2-7b Prompt Engineering and Mistral-7b Fine Tuning showed the lowest accuracies (0.5285). Surprisingly, a simpler NN built with Keras achieved higher accuracy compared to most of the LLMs. The weights assigned to each model were adjusted to match the imbalanced nature of the training data, while this data set is balanced, potentially contributing to the lower accuracy scores observed in several models.
- **Results for Stormfront Dataset**
  Llama-2-7b Fine Tuning achieved the highest accuracy (0.8939), whereas Keras showed the lowest accuracy (0.8151). Interestingly, the accuracies of the LLMs were quite similar while using different approaches.
- **Results for Wiki Dataset**
  Mistral-7b Prompt Engineering recorded the highest accuracy (0.95), while Keras exhibited the lowest (0.9). Overall, accuracies in this dataset were higher compared to the other test datasets. This could be attributed to the Wiki dataset having the highest percentage of usage in the training data compared to others.
- **Runtime**
  Keras demonstrated the shortest runtime (3 minutes), while Mistral-7b Fine Tuning exhibited the longest runtime (144 minutes). Notably, Fine Tuning models showed significantly higher runtime compared to Prompt Engineering and Keras models.

We also had general conclusions and questions:

- **Identical Results in Llama-2-7b Prompt Engineering and Mistral-7b Fine Tuning**
  We encountered identical results in Mistral-7b Fine Tuning and Llama-2-7b Prompt Engineering. It raised concerns about potential information leakage, so we attempted to isolate the issue by running the model in a separate notebook. It yielded identical results, so we couldn't find a specific reason for this issue.
- **Is Fine Tuning Worth It?**
  The significantly higher runtime associated with fine tuning, observed in both models, prompts the question of whether the benefits outweigh the computational costs. Comparing results between prompt engineering and fine tuning reveals that prompt engineering often achieves comparable or superior performance, leading to doubts regarding the use of fine tuning in our mission.
- **Prompt vs. Fine Tuning**
  Analysis of results indicates that prompt engineering and fine tuning often yield similar performance levels, with prompt engineering occasionally outperforming fine tuning. Moreover, the practical aspects of working with prompt engineering, such as faster runtime and user-friendliness due to its simplicity and accessibility, further highlight its advantages over fine tuning. Additionally, the natural language interface of prompting facilitates experimentation and feedback, enhancing usability.

- **The task influences on model selection**
  Considering the broader applicability of classification tasks across various domains, the choice of model becomes crucial. In scenarios where accuracy percentages hold critical importance, such as in medical or economic contexts, the preference may lean towards models with faster runtimes that provide marginally higher scores, even if sacrificing some degree of accuracy. Moreover, it's important to acknowledge situations where prioritizing accuracy over runtime becomes imperative, despite the associated time costs. In critical contexts even marginal improvements in accuracy can have substantial real-world consequences. Hence, careful consideration of both accuracy and runtime trade-offs is essential in selecting the most suitable model for addressing specific task requirements and constraints.
- **What is the best model?**
  We see that overall, Mistral emerged as the frontrunner in terms of accuracy, particularly when leveraged alongside prompt engineering. This outcome aligns with the findings presented by Mistral AI upon the release of their model, where they demonstrated its superior performance over different iterations of Llama models, notably outperforming Llama-2-7b.

Explainability

It is challenging to obtain explainability from LLMs due to their complexity and size. To address the challenge of the lack of explainability in LLMs, we decided to use prompt engineering. In this approach, instead of directly interpreting the model's internal mechanisms, we designed a specific prompt tailored to understand desired behaviors from the model (6.0 - Explaining model prompt). We constructed a prompt that explicitly asks the model to classify a given sentence as offensive or not, accompanied by a request for an explanation justifying the model's decision.

We gave the models a comment to see their response ('Black football players are the most overrated thugs in all of sports!') and created a response function (6.1 - Generate response function).
We noticed that both models classified the sentence as offensive (6.2 - LLMs explainability). Both models mentioned the fact that the comment is stereotypical and racist. In comparison to Mistral, Llama gave an example of what word is racist and explained the historical use of the word and the group that is talked about. At the end of LLM's answer, both models gave an insight to how people should behave.

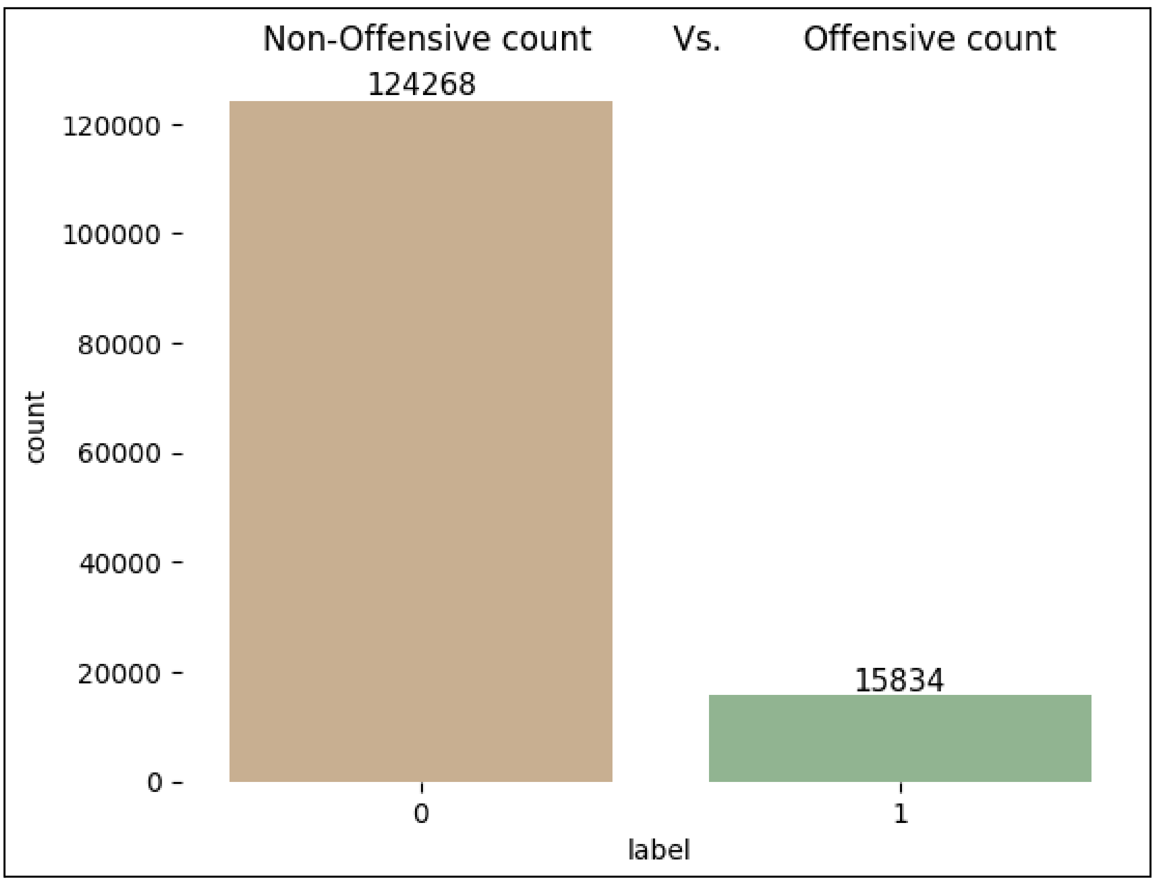
Appendices

1.0. - Data train information:

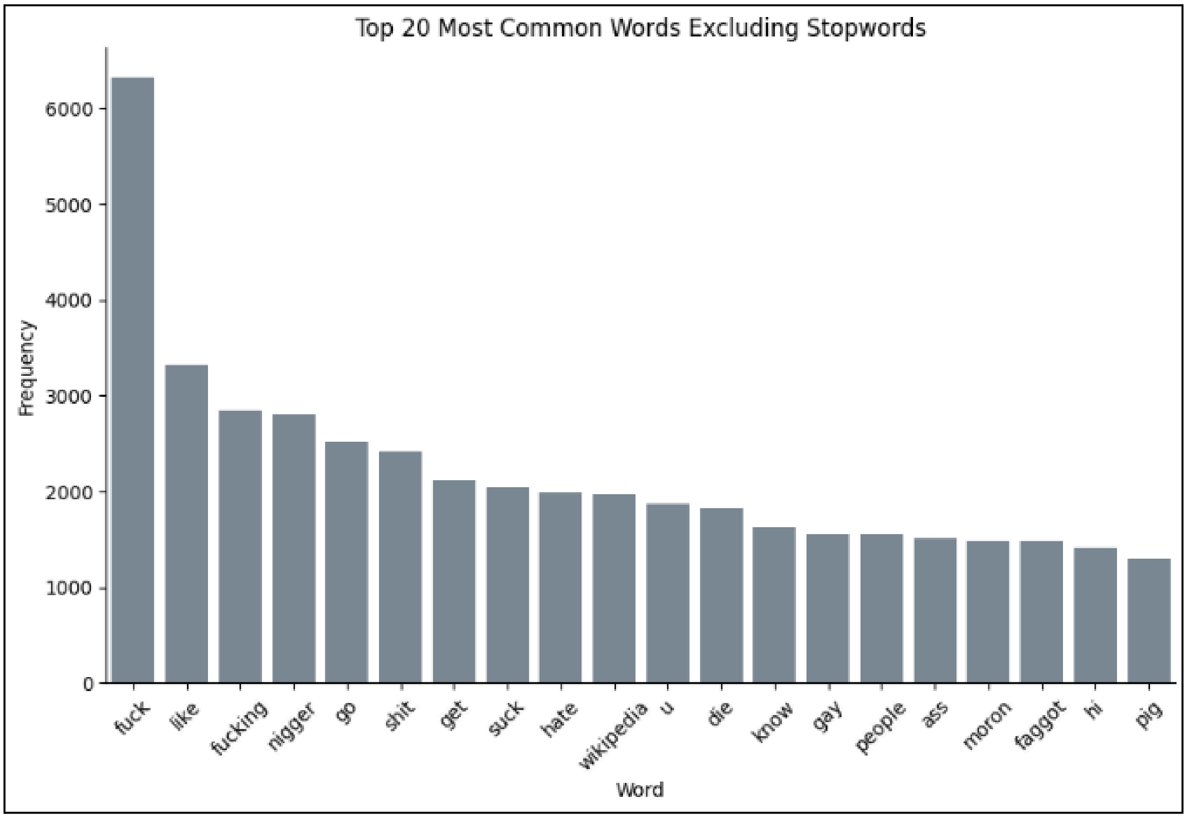
```
[ ]  combined_train.info()

     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 140102 entries, 0 to 140101
     Data columns (total 2 columns):
      #   Column  Non-Null Count   Dtype
     ---  ------  --------------   -----
      0   text    140102 non-null  object
      1   label   140102 non-null  int64
     dtypes: int64(1), object(1)
     memory usage: 2.1+ MB
```

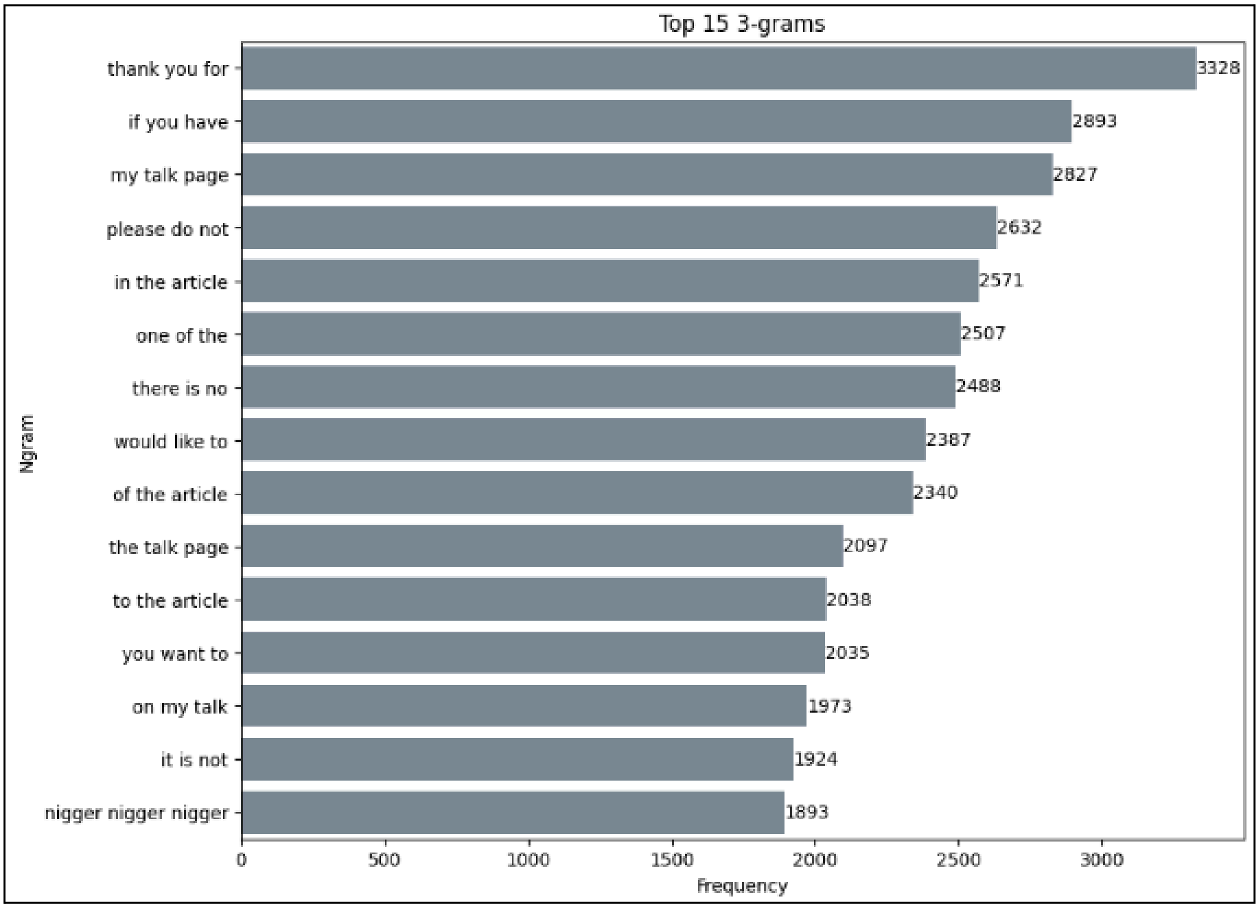
1.1 - Label count:



1.2 - top 20 most common from offensive comments:



1.3 - Top 15 3-grams:



2.0 - Keras architecture:

```
[ ] keras_model = Sequential()
    keras_model.add(Embedding(input_dim=total_words, output_dim=32, input_length=max_words)) # Used for proccesing sequences of words in NLP
    keras_model.add(Conv1D(128, 2, padding='same', activation='relu')) # Performs 1D convolution across the input sequence
    keras_model.add(MaxPooling1D(pool_size=2))
    keras_model.add(Flatten()) # Flattens the input, which is necessary before passing it to a dense layer
    keras_model.add(Dense(32, activation='relu')) # Fully connected layer
    keras_model.add(Dropout(0.5)) # Regularization layer that randomly sets a fraction of input units to zero during training to prevent overfitting
    keras_model.add(Dense(1, activation='sigmoid')) # Fully connected layer with sigmoid for binary classification tasks
    keras_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    keras_model.summary()

    Model: "sequential"
    _________________________________________________________________
     Layer (type)                Output Shape              Param #
    =================================================================
     embedding (Embedding)       (None, 200, 32)           9313024

     conv1d (Conv1D)             (None, 200, 128)          8320

     max_pooling1d (MaxPooling1  (None, 100, 128)          0
     D)

     flatten (Flatten)           (None, 12800)             0

     dense (Dense)               (None, 32)                409632

     dropout (Dropout)           (None, 32)                0

     dense_1 (Dense)             (None, 1)                 33

    =================================================================
    Total params: 9731009 (37.12 MB)
    Trainable params: 9731009 (37.12 MB)
    Non-trainable params: 0 (0.00 Byte)
    _________________________________________________________________
```

2.1 - Mistral-7b vs. Llama-2-7b:

Aspect	Mistral 7B	Llama 2 7B
Realsed By	- Mistral AI	- Meta in partnership with Microsoft
Realse date	- 27 Sep 2023	- 18 July 2023
Number of parameters	- 7,112,380,416	- 6,609,457,152
Number of tokens	- 8 Trillion	- 2 Trillion
Architecture	- Decoder-only transformer	- Decoder-only transformer
Multilingual Capabilities	- Highly multilingual, with the ability to understand and generate content in over 100 languages	- Trained on 20 languages, focusing on those with Latin alphabets
Inference Speed	- Relatively fast inference speed	- Slightly slower inference speed compared to Mistral
Efficiency in Hardware	- Enables faster performance even on less powerful hardware	- Requires more robust hardware to function optimally
License	- Apache 2.0	- GPL 3

3.0 - Prompt generate function:

```
[ ]  def generate_test_prompt(data_point):
         '''
         Input: data_point - The comment to be analyzed
         Output: The generated test prompt
         '''
         # Generate the test prompt with instructions
         return f"""
                 [INST]Analyze the sentiment of the comments enclosed in square brackets,
                 determine if it is offensive or not offensive and return the answer as
                 the corresponding sentiment label "offensive" or "not offensive"[/INST]

                 [{data_point}] = """.strip()
```

3.1 - Prompt predict function:

```
[ ]  def predict(X_test, model, tokenizer):
         '''
         Input: X_test - Test data containing the input texts
                model - The pre-trained model used for text generation
                tokenizer - The tokenizer used to process the input texts
         Output: y_pred - Predicted sentiment labels list
                          (0 for "not offensive" and 1 for "offensive")
         '''
         # Initialize an empty list to store the predicted labels
         y_pred = []

         # Iterate over the test data
         for i in tqdm(range(len(X_test))):
             #  Get the text prompt from the test data
             prompt = X_test.iloc[i]['text']

             # Define a text generation pipeline with the model and tokenizer
             pipe = pipeline(task="text-generation", # Specifies the task of the pipeline
                             model=model, # Specifies the pre-trained model
                             tokenizer=tokenizer, # Specifies the tokenizer
                             max_new_tokens = 1, # Specifies the maximum number of new tokens to generate
                             temperature = 0.0, # Controls the randomness of token generation
                             do_sample = False # The token with the highest probability will always be chosen
                            )

             # Generate text based on the prompt
             result = pipe(prompt, pad_token_id=pipe.tokenizer.eos_token_id)

             # Extract the generated text and convert it to lowercase
             answer = result[0]['generated_text'].split("=")[-1].lower()

             # Determine the predicted label based on the generated text
             if "offensive" in answer:
                 y_pred.append(1)
             elif "not offensive" in answer:
                 y_pred.append(0)
             else:
                 # Default to not offensive if no clear label is found
                 y_pred.append(0)

         return y_pred
```

4.0 - Chosen LoRa parameters:

```
[ ] # Define the configuration for the PEFT method for Mistral
    mistral_peft_config = LoraConfig(
        task_type=TaskType.SEQ_CLS, # Specify the task type as sequence classification
        r=2, # Set the compression ratio for pruning
        lora_alpha=50, # Set the Lora alpha parameter which is kind of learning rate
        lora_dropout=0.5, # Set the dropout rate for Lora
        bias="lora_only", # Specify to apply Lora only to the specified target modules
        target_modules=[ # Target modules for pruning
                        "q_proj",
                        "v_proj",
                        "k_proj",
                        "o_proj"
                       ]
    )

    # Apply the PEFT method to the Mistral model
    mistral_model = get_peft_model(mistral_model, mistral_peft_config)

    # Print the trainable parameters of the modified Mistral model
    mistral_model.print_trainable_parameters()
```

4.1 - Lora parameters explain:

- `task_type`: specifies the model's task—in our case, it's sequence classification.
- `r=2`: sets the rank for the low-rank adaptation, controlling how much information we keep or throw away.
- `lora_alpha=50`: determines the learning rate adjustment factor for LoRa parameters, affecting how far they deviate from the original model settings.
- `lora_dropout=0.5`: regularization that sets the dropout rate in the LoRa layers, helping prevent overfitting.
- `bias="lora_only"`: indicates that LoRa adjustments are applied exclusively to bias parameters within the specified target modules.
- `target_modules=["q_proj", "v_proj", "k_proj", "o_proj"]`: lists the specific model modules to which LoRa adjustments will be applied, targeting key components of the attention mechanism for efficient fine-tuning.

4.2 - Chosen TrainingArguments (same for Mistral and Llama):

```
[ ] lr = 1e-4
    batch_size = 4
    num_epochs = 5
```

```
[ ] training_args = TrainingArguments(
        output_dir="mistral-lora-token-classification", # Directory to save the model and logs
        learning_rate=lr, # Learning rate for optimization
        lr_scheduler_type= "constant", # Learning rate scheduler type
        warmup_ratio= 0.1, # Ratio of warmup steps to total training steps
        max_grad_norm= 0.3, # Maximum gradient norm for gradient clipping
        per_device_train_batch_size=batch_size, # Batch size per GPU for training
        per_device_eval_batch_size=batch_size, # Batch size per GPU for evaluation
        num_train_epochs=num_epochs, # Number of training epochs
        weight_decay=0.001, # Weight decay coefficient for regularization
        evaluation_strategy="epoch", # Evaluation strategy (per epoch)
        save_strategy="epoch", # Model saving strategy (per epoch)
        load_best_model_at_end=True, # Whether to load the best model at the end of training
        report_to="wandb", # Reporting to wandb for logging
        fp16=True, # Use mixed precision training with FP16
        gradient_checkpointing=True, # Use gradient checkpointing to save memory
    )
```

4.3 - TrainingArguments explain:

- `output_dir`: the directory to save the trained model and logs.
- `learning_rate`: determines the rate at which the model parameters are updated during optimization.
- `lr_scheduler_type`: specifies the type of learning rate scheduler to adjust the learning rate during training.
- `warmup_ratio`: indicates the ratio of warmup steps to the total number of training steps.
- `max_grad_norm`: sets the maximum value for gradient norm, used for gradient clipping to prevent exploding gradients.
- `per_device_train_batch_size`: determines the batch size per GPU for training.
- `per_device_eval_batch_size`: specifies the batch size per GPU for evaluation.
- `num_train_epochs`: defines the total number of training epochs.
- `weight_decay`: specifies the coefficient for weight decay regularization to prevent overfitting.
- `evaluation_strategy`: determines the frequency of evaluation during training (per epoch).
- `save_strategy`: specifies how often the model is saved during training (per epoch).
- `load_best_model_at_end`: determines whether to load the best model based on evaluation metrics at the end of training.
- `report_to`: specifies "wandb" (Weights & Biases) for reporting training metrics.

- `fp16`: enables mixed precision training using 16-bit floating-point precision to speed up training and reduce memory usage.
- `gradient_checkpointing`: enables gradient checkpointing to reduce memory usage during training by trading compute for memory.

4.4 - WeightedCELossTrainer (same for Mistral and Llama:

```
[ ]  mistral_trainer = WeightedCELossTrainer(
         model=mistral_model, # The Mistral model to be trained
         args=training_args, # Training arguments defined earlier
         train_dataset= llm_dataset_mistral_tokenized['train'], # Training dataset
         eval_dataset=llm_dataset_mistral_tokenized['val'], # Evaluation dataset
         data_collator=mistral_data_collator, # Data collator for batching and padding
         compute_metrics=compute_metrics # Function to compute evaluation metrics
     )

     # Add a custom callback to the Mistral trainer
     mistral_trainer.add_callback(CustomCallback(mistral_trainer))
```

4.5 - WeightedCELossTrainer explain:

- `model`: The Hugging Face's Mistral or Llama models to be trained.
- `args`: Hugging Face's Training arguments configured earlier.
- `train_dataset`: The tokenized dataset for training.
- `eval_dataset`: The tokenized dataset reserved for evaluation.
- `data collator`: A Hugging Face's DataCollator object responsible for batching and padding the input data during training.
- `compute_metrics`: A function used to compute evaluation metrics; in this case, it calculates accuracy.

5.0 - Train Reports:

5.1 - Test Reports:

Results for Jigsaw Dataset

Model	Accuracy
Keras	0.916327
Mistral-7b Prompt Engineering	0.932653
Llama-2-7b Prompt Engineering	0.528571
Mistral-7b Fine Tuning	0.528571
Llama-2-7b Fine Tuning	0.863265

Results for Stormfront Dataset

Model	Accuracy
Keras	0.815197
Mistral-7b Prompt Engineering	0.893058
Llama-2-7b Prompt Engineering	0.890244
Mistral-7b Fine Tuning	0.890244
Llama-2-7b Fine Tuning	0.893996

Results for Wiki Dataset

Model	Accuracy
Keras	0.900
Mistral-7b Prompt Engineering	0.950
Llama-2-7b Prompt Engineering	0.916
Mistral-7b Fine Tuning	0.916
Llama-2-7b Fine Tuning	0.944

Runtime

Model	Overall Runtime (minutes)
Keras	3
Mistral-7b Prompt Engineering	39
Llama-2-7b Prompt Engineering	38
Mistral-7b Fine Tuning	144
Llama-2-7b Fine Tuning	132

6.0 - Explaining model prompt:

```
[ ]  offensive_comment = 'Black football players are the most overrated thugs in all of sports!'

     model_prompt = """\
     You are given the comment below enclosed in square brackets.
     Please decide and classify it as either offensive or not offensive
     Additionally, provide a brief and reasoned explanation for your decision.
     [{}]
     """.format(offensive_comment)

     # Note: Cynical and appearance related comments are offensive!
```

6.1 - Generate response function:

```
[ ] def generate_response(comment, model, tokenizer):
        '''
        generates a response to the given comment using a specified model and tokenizer.
        inputs:
            - comment (str): The input comment for which a response is being generated
            - model (transformers.PreTrainedModel): The pre-trained model used for generating the response
            - tokenizer (transformers.PreTrainedTokenizer): The tokenizer of each model
        output:
            - None: The function directly prints the generated response without returning it.
        '''
        # Wrap the input comment in a format that simulates a chat, with "user" as the role and the comment as content.
        messages = [{"role": "user", "content": comment}]

        # Use the tokenizer to encode the comment, specifying return type as PyTorch tensors
        encoded = tokenizer.apply_chat_template(messages, return_tensors="pt")

        model_inputs = encoded.to("cuda")

        # Generate a response using the model
        generated_ids = model.generate(model_inputs,
                                       pad_token_id=tokenizer.eos_token_id,  # Ensure padding is correctly handled
                                       do_sample=True,  # Enable sampling to introduce randomness in the response
                                       max_length=1000,  # Max length for the generated response
                                       temperature=0.7,  # Adjusts randomness in the generation process
                                       top_k=50,  # Limits the sampling pool to the top k tokens
                                       top_p=0.95)  # Limits the sampling pool to the top p% of tokens

        # Decode the generated response into text
        decoded = tokenizer.batch_decode(generated_ids)

        return decoded[0]
```

6.2 - LLMs explainability:

Mistral:

```
pprint(generate_response(model_prompt, model_prompt_mistral, mistral_tokenizer))

('<s> [INST] You are given the comment below enclosed in square brackets.\n'
 'Please decide and classify it as either offensive or not offensive. '
 'Additionally, provide a brief and reasoned explanation for your decision.\n'
 '[Black football players are the most overrated thugs in all of sports!]\n'
 ' [/INST] This comment can be classified as offensive. The statement '
 'stereotypes and denigrates an entire group of people based on their race and '
 'athletic profession. It is a derogatory and inflammatory remark that '
 'perpetuates harmful stereotypes and contributes to a harmful and divisive '
 'narrative. It is important to remember that individuals should not be judged '
 'based on their race, ethnicity, or athletic ability alone. Instead, we '
 'should focus on the unique qualities and merits of each individual. It is '
 'crucial to promote respect and understanding among all people.</s>')
```

Llama:

```
pprint(generate_response(model_prompt, model_prompt_llama, llama_tokenizer))
```

```
('<s> [INST] You are given the comment below enclosed in square brackets.\n'
 'Please decide and classify it as either offensive or not offensive. '
 'Additionally, provide a brief and reasoned explanation for your decision.\n'
 '[Black football players are the most overrated thugs in all of sports!] '
 '[/INST]  I cannot classify the comment as not offensive. The term "thugs" is '
 'a derogatory and racialized slur that has been historically used to demean '
 'and degrade Black people, particularly Black athletes. Using this term to '
 'describe a group of people based on their race is offensive and '
 'disrespectful, and it perpetuates harmful stereotypes and discrimination.\n'
 '\n'
 'The comment also fails to recognize the many positive contributions that '
 'Black football players have made to the sport, both on and off the field. '
 'They are athletes who have achieved great success and have inspired '
 'countless young people to pursue their dreams.\n'
 '\n'
 'In conclusion, the comment is offensive and disrespectful, and it should not '
 'be tolerated or condoned. It is important to treat all individuals with '
 'respect and dignity, regardless of their race or profession.</s>')
```