# Programming with Python: Writen Assignment

## EXPLORING PROMPT STRATEGIES FOR HATE SPEECH DETECTION USING LARGE LANGUAGE MODELS

STUDENT: HUGO ALBUQUERQUE COSME DA SILVA

ID:92126125

## TABLE OF CONTENTS

## 1. Introduction

Hate speech has become a widespread problem on social media, influencing both online conversations and interactions in everyday life. As the amount of content shared on these platforms continues to grow, it becomes impossible for human moderators to monitor every piece of activity for harmful speech. This challenge has led researchers and technologists to investigate automated methods for detecting hate speech, with generative AI emerging as a promising solution[1][2].

Traditional approaches to detecting hate speech, which often rely on rule-based systems or machine learning algorithms, struggle with the complexities and contextual nuances of language. These systems frequently misclassify hate speech, offensive language, and normal speech, leading to high rates of false positives or missed threats[3]. Recent advancements in generative AI, particularly in large language models (LLMs), offer new ways to address these shortcomings[5][5]. These generative models, with their capabilities in natural language understanding and reasoning, can potentially outperform traditional systems by identifying subtle patterns and offering explanations for their decisions.

This research examines various generative AI techniques, including zero-shot, few-shot, and chain-of-thought prompting, in the context of hate speech detection. Using datasets such as HateXplain and Stormfront, the study compares how these methods perform in classifying social media messages while also addressing challenges like bias, false positives, and the need for models that provide clear explanations[2][6]. Through this evaluation, the research aims to contribute to the development of more accurate and equitable systems for detecting hate speech, ultimately creating safer online spaces[7].

## 2. Main body

To conduct experiments on hate speech detection, the HateXplain dataset was selected. It consists of 20,148 posts from Twitter and Gab, with each post annotated by multiple people to minimize individual biases. Each post is labeled from three distinct perspectives, providing a well-rounded analysis of hate speech. Key features of this dataset include a 3-class classification system that labels posts as hate speech, offensive, or normal. This system, commonly used in the field, adds depth by distinguishing between hate speech and offensive content[8].

Another feature is the identification of the target community in each post, showing which group was subjected to hate or offensive language. This helps researchers identify patterns and potentially reduce bias in detection models. Additionally, the dataset includes rationales, specific text portions used by annotators to explain their decisions. These rationales improve interpretability and transparency. The specific version used during the experiments were obtained from HuggingFace[9].

HateXplain's inclusion of rationales makes it especially suitable for explainable AI models, enabling researchers to train systems that not only detect hate speech but also explain their reasoning. The dataset supports evaluation on both performance metrics (like accuracy and F1-score) and explainability metrics (such as plausibility and faithfulness), offering a comprehensive assessment of models. By including target community annotations and rationales, HateXplain helps develop models that are accurate and less biased against particular communities, a critical factor in hate speech detection[10].

As highlighted in the previous paragraph, HateXplain can be used for a plethora of research experiments. To reduce the project's scope, some preprocessing and experiment design decisions were made:

1. For every post it was calculated the majority vote. The function that performed this calculation can be found at appendix 2.
2. The dataset was filtered to include only posts whose majority vote was either "normal" or "hate speech". In theory this should make the classification task easier since, the discrepancy between hate-speech messages and normal messages is bigger than hate speech messages and offensive messages
3. Xml Tags were removed from the data. The function used to perform this operation can be checked at appendix 2.
4. For each particular message, the tokens that were deemed relevant by all three annotators were extracted to the *relevant_pieces* column
5. Only the test dataset was used to perform the experiments. The rationale here is related to costs savings on LLM API calls, and a shorter time to iterate on the classification experiments.

Other minor preprocessing steps were applied to the data. The interested reader can check the full data processing notebook at the written assignment repository in the path *written_assignment/dataset_loading.ipynb* .

## 2.1 Zero shot prompt plus single LLM call

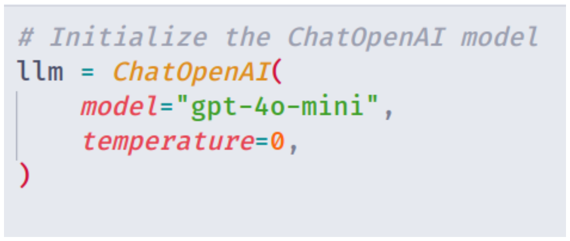
### 2.1.1 - Method description

The initial approach involved making a single call to the Large Language Model (LLM) with a zero-shot prompt. In this case, "zero-shot" means that no examples were provided in the prompt to help the model with classification. The experiments used OpenAI's gpt-4o-mini-2024-07-18 model, with LangChain helping to manage and streamline the process. Zero-shot learning in NLP dates back to 2008, originally called "dataless classification," though it wasn't linked to prompting LLMs at that time[12]. Zero-shot prompting for LLMs became more prominent with the rise of large pretrained models like GPT-3. The 2020 GPT-3 paper by Brown et al. highlighted the model's ability to perform tasks without the need for task-specific fine-tuning, paving the way for zero-shot prompting[13].

### 2.1.2 - Implementation details and results

The first step performed was to create a Pydantic class called HateSpeechAnalysis. This was done because it is necessary to enforce a structured output. LLM calls are normally not idempotent i.e. the outputs tend to have some variability, even if an output is specified in the system prompt. Sentences like "Here is you output...", "Sure, I will generate the json for you...." Are normally present in the output of the LLM call, what makes the logic for parsing the json object more complex. In order to address this issue, Langchain developed a method called "with_structured_output"[11]. This method leverages tool usage under the hood and "forces" the LLM to comply with the data schema that is specified in the Pydantic Class. The reader can check the details of the implemented Pydantic class in appendix 3.

An essential factor in working with LLMs is the "temperature" parameter, which controls how random the model's responses are. A lower temperature (near 0) makes the model more predictable and focused, while a higher temperature increases variability, allowing for more creative and diverse answers. In this experiment, the temperature was set to 0 in order to maximize the experiment's reproducibility:



```python
# Initialize the ChatOpenAI model
llm = ChatOpenAI(
    model="gpt-4o-mini",
    temperature=0,
)
```

Figure 1 - Defining the LLM Object

The following text is the system prompt:

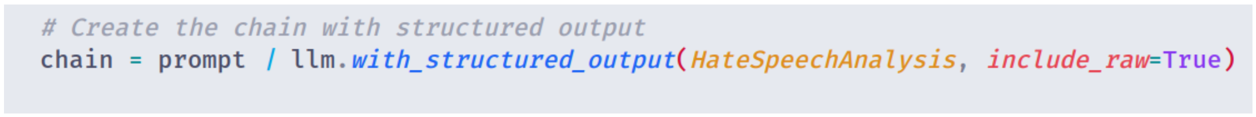
"*You are an AI trained to detect hate speech. Analyze the given message and determine if it's normal or hate speech. Provide a classification, confidence score, and brief reasoning.*"

The following is the Human message:

*"Analyze the following message for hate speech: {message}"*

Where message is a placeholder for the hateXplain dataset messages. The complete prompt can be checked on appendix 3.

Finally, to create the chain, the prompt is passed to the LLM using the with_structured_output_method. The include_raw parameter adds both the raw json as well as the parsed pydantic object to the output:



```python
# Create the chain with structured output
chain = prompt | llm.with_structured_output(HateSpeechAnalysis, include_raw=True)
```

Figure 2 - Creating the chain for experiment 1

To run the experiments, the parsed test set from HateXplain was loaded and the messages were formatted in a way that the chain would recognize them. The inferences were then performed sequentially using a looping logic. The inference code can be checked in appendix 3.

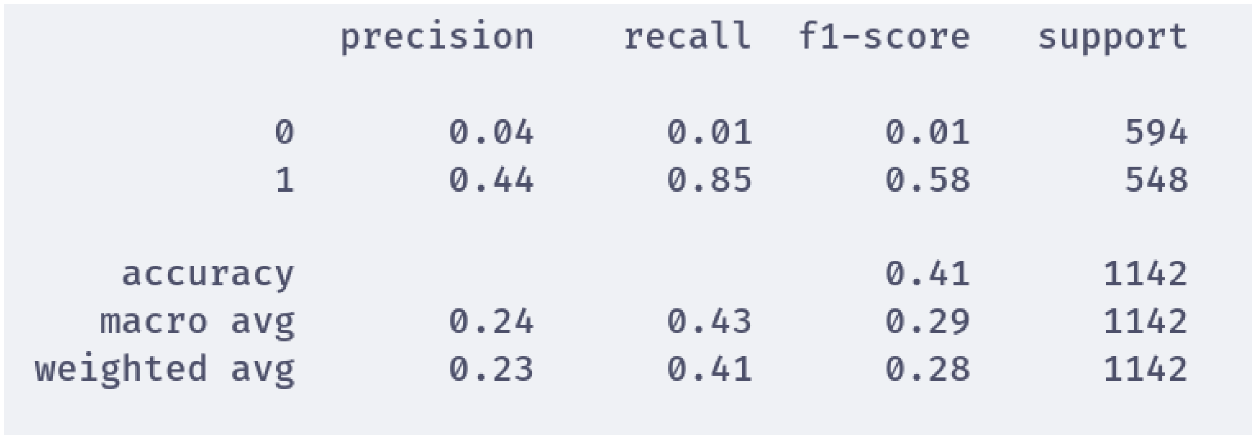
The classification results were parsed and three columns were added to the dataset:

1. classification: "normal" or "hate speech" (the result of the classification)

2. Confidence: the confidence score output by the LLM

3. Reasoning: a simple explanation of why the LLM judged the call normal or hate speech

From this results, the classification report was created with the help of Scikit-learn library:



```
              precision    recall  f1-score   support

           0       0.04      0.01      0.01       594
           1       0.44      0.85      0.58       548

    accuracy                           0.41      1142
   macro avg       0.24      0.43      0.29      1142
weighted avg       0.23      0.41      0.28      1142
```

Figure 3 - Classification report experiment 1.Obs: In the above figure, 0 means "normal" and 1 means "hate speech".

A confusion matrix was built:

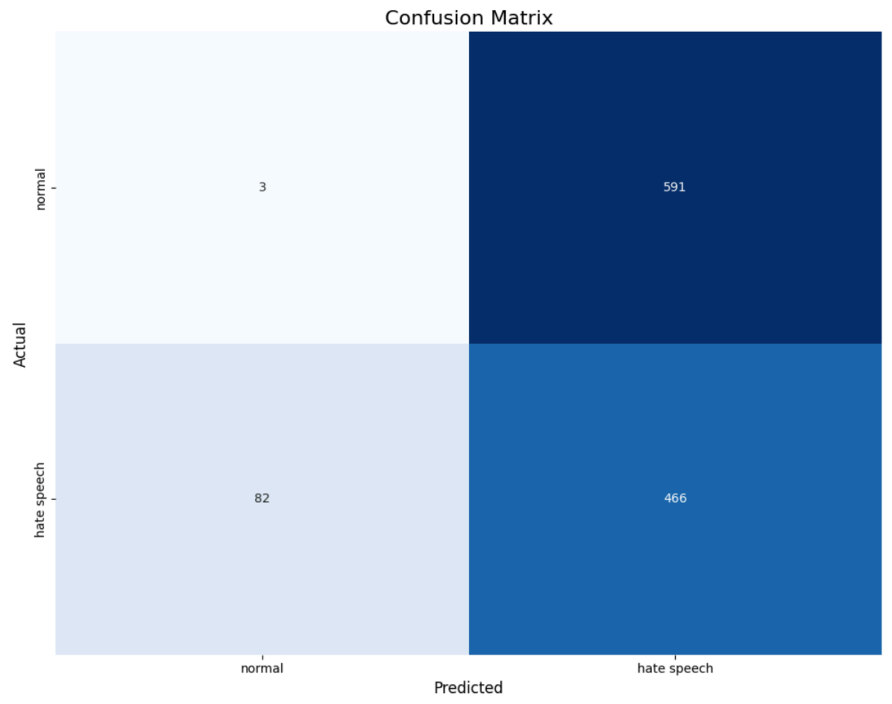


Figure 4 - Confusion matrix experiment 1

A ROC (Receiver Operating Characteristic) Curve was plotted with the AUC (Area Under the Curve) score.

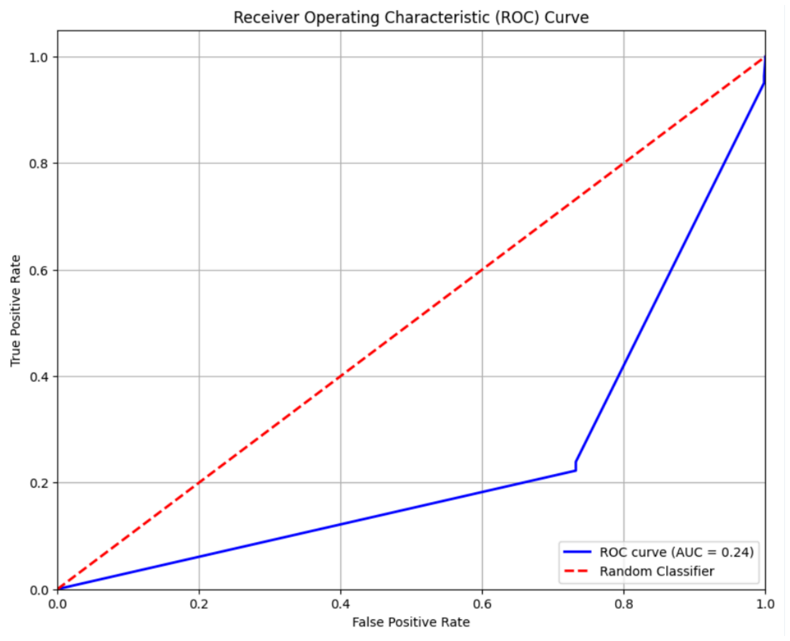


Figure 5 - ROC Curve experiment 1

The full experiment code can be found in a jupyter notebook in the assignment's repository in the path *written_assignment/experiment1.ipynb*.

### 2.1.3 - Results discussion

The results reveal several important issues with the current classifier approach. Most notably, the model shows a very high recall rate for hate speech messages, with a recall of 0.85, meaning it effectively identifies most hate speech in the dataset. This suggests the model is strongly biased toward detecting hate speech, which may be useful in situations where it's crucial to avoid missing any hate speech. However, this high recall comes at the expense of precision.

Both hate speech and normal messages suffer from low precision, especially in the normal class, where precision is alarmingly low at just 0.04, indicating that almost all predictions for normal messages are incorrect. Hate speech predictions are somewhat better, with a precision of 0.44, but this is still far from ideal. The low precision suggests the classifier is over-predicting hate speech, leading to a significant number of false positives.

The confusion matrix further highlights the issue: out of 594 normal messages, the model correctly classified only 3, misclassifying 591 as hate speech. This massive imbalance points to a strong bias toward labeling messages as hate speech. While the model correctly identified 466 out of 548 hate speech messages, the cost is an overwhelming number of false positives for normal messages. This indicates the model’s decision boundary is heavily skewed toward hate speech, compromising its accuracy in distinguishing normal content.

The ROC curve and AUC score reinforce this poor performance. With an AUC of 0.24, the classifier is performing worse than random guessing (which would yield an AUC of 0.5), signaling its inability to effectively differentiate between hate speech and normal messages.

One major shortcoming is the system's reliance on a zero-shot approach, where the classifier struggles to distinguish hate speech from normal messages because it is only given a broad definition of hate speech, without specific examples. As a result, the model defaults to a less nuanced and less accurate interpretation.

To improve the model, adopting a few-shot learning approach could make a big difference. By providing a handful of annotated examples of both hate speech and normal messages, the model could better align with human judgment, reducing bias and improving precision. These examples would help the model develop a clearer understanding of the differences between hate speech and normal speech, allowing it to make more balanced, contextually informed decisions and ultimately produce more reliable results. This approach will be evaluated in the next section.

## 2.2 - Few shot learning

### 2.2.1 - Method description

Few-shot prompting has proven to be a highly effective technique for utilizing large language models (LLMs) to perform a variety of tasks with only a small number of examples. This approach has drawn significant attention due to its ability to deliver competitive results across a wide range of natural language processing (NLP) tasks. [14].

In essence, few-shot prompting involves presenting a limited set of examples to guide the LLM in carrying out a specific task. By leveraging the extensive knowledge embedded in pre-trained LLMs, this method allows the model to generalize from a handful of examples to previously unseen situations. One of its primary strengths is its flexibility, enabling researchers to deploy LLMs on new tasks or in unfamiliar domains without requiring fine-tuning or large amounts of labeled data.

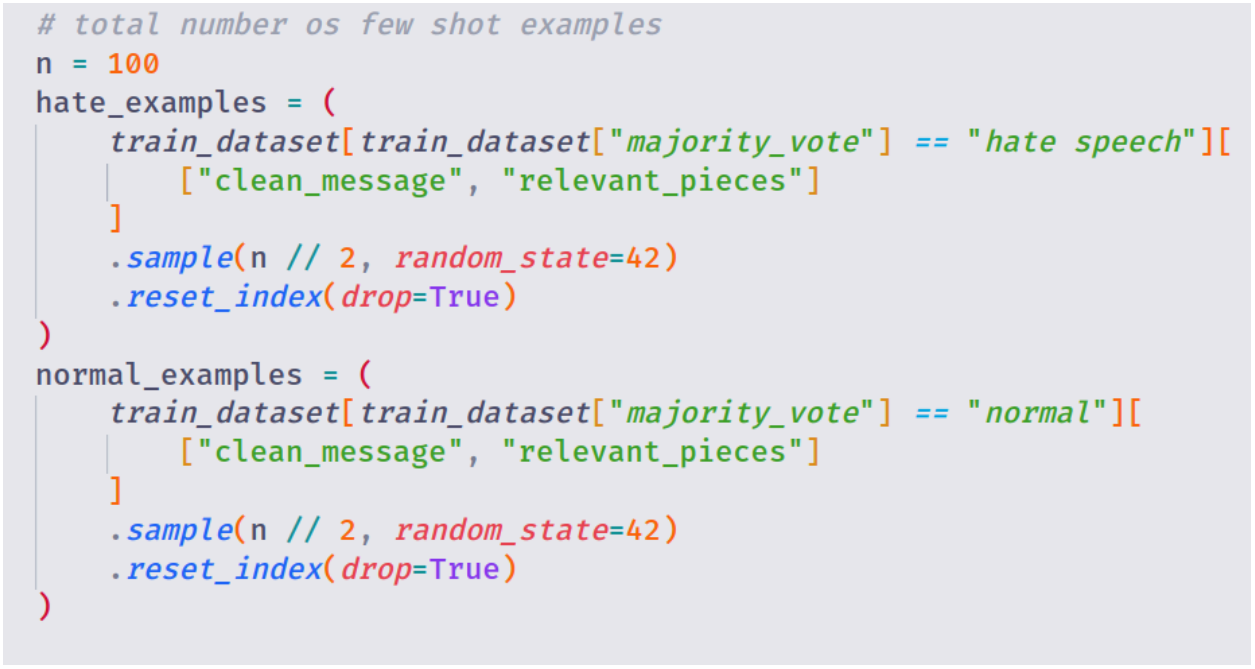
Determining the optimal number of examples in the prompt remains an active area of research. Recent studies reveal that this number is often task-specific, with simpler tasks benefiting from fewer examples and more complex tasks requiring additional examples. Notably, there appears to be an asymptotic trend in many studies, where the performance gains diminish after a certain number of examples are included [14][15][16].

### 2.2.2 - Implementation details and results

This experiment's details are covered in experiment2.ipynb notebook. The experiment started by loading the train and test set from the cleaned hateXplain dataset that was showed in previous stages. The same set of 1142 messages will be evaluated. The pydantic Class HateSpeechAnalysis and the LLM are defined the same way as in the previous experiment. The main change lies in the prompt as can be seen in appendix 4. The human message now contains the placeholders for the normal few shot examples and hate few shot examples.

After the chain was created the next step was to sample the few shot examples thatwill be embedded in the human message during inference. The data was sampled from the training dataset, to avoid any data any data leakage. If the samples also come from the test set, while the 1142 messages are being evaluated, one of the few shot examples could be the message being evaluated, what would be an unfair advantage for the LLM.

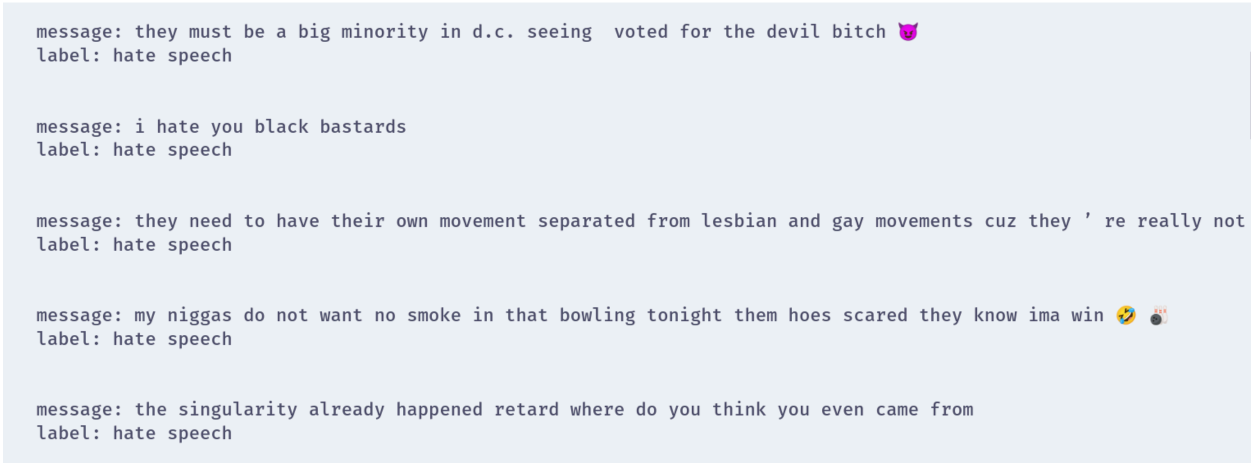
I have chosen 100 examples, 50 normal messages and 50 hate messages. As mentioned in the previous section, there is no right number of examples, but the rationale followed was that since the performance was poor in the first experiment, the more context it has about these messages and how they are being labeled the better. Besides, adding too much examples would not hurt performance. As highlighted by the papers mentioned in the previous section, there are diminishing improvements in performance after a certain number of examples is added to the prompt context.



```python
# total number os few shot examples
n = 100
hate_examples = (
    train_dataset[train_dataset["majority_vote"] == "hate speech"][
        ["clean_message", "relevant_pieces"]
    ]
    .sample(n // 2, random_state=42)
    .reset_index(drop=True)
)
normal_examples = (
    train_dataset[train_dataset["majority_vote"] == "normal"][
        ["clean_message", "relevant_pieces"]
    ]
    .sample(n // 2, random_state=42)
    .reset_index(drop=True)
)
```

Figure 6 - Sampling examples from the training dataset

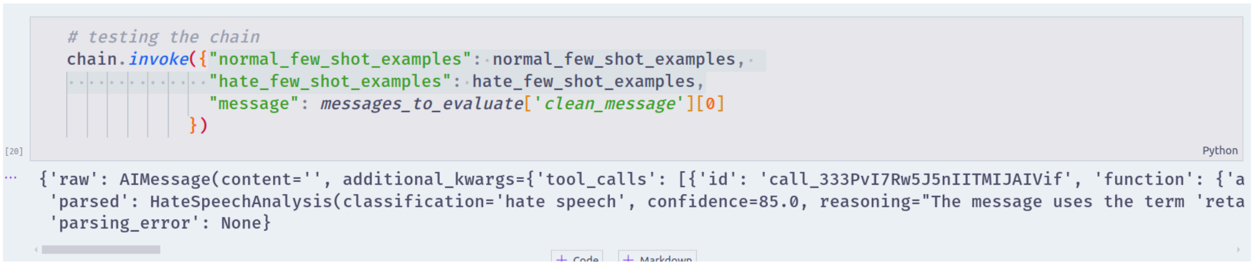
Finally, the messages were put in the format that they will be embedded in the prompt. Here is some examples of "hate_few_shot_examples":



```
message: they must be a big minority in d.c. seeing  voted for the devil bitch 😈
label: hate speech


message: i hate you black bastards
label: hate speech


message: they need to have their own movement separated from lesbian and gay movements cuz they ’ re really not
label: hate speech


message: my niggas do not want no smoke in that bowling tonight them hoes scared they know ima win 🤣 🎳
label: hate speech


message: the singularity already happened retard where do you think you even came from
label: hate speech
```

Figure 7 - Examples of hate messages that will be used as few-shot-examples

Another important distinction is that now the chain will have three inputs: The normal few-shot-examples; the hate few-shot-examples and the message to be evaluated. Below we see a test invocation of the chain:



```
# testing the chain
chain.invoke({"normal_few_shot_examples": normal_few_shot_examples,
              "hate_few_shot_examples": hate_few_shot_examples,
              "message": messages_to_evaluate['clean_message'][0]
             })
```

```
{'raw': AIMessage(content='', additional_kwargs={'tool_calls': [{'id': 'call_333PvI7Rw5J5nIITMIJAIVif', 'function': {'a
 'parsed': HateSpeechAnalysis(classification='hate speech', confidence=85.0, reasoning="The message uses the term 'reta
 'parsing_error': None}
```

Figure 8 - Testing the chain invocation

Therefore, each message needed to be converted into a dictionary with the appropriate keys, before it is sent to the LLM. After this process was completed, the classification was performed sequentially as in the previous experiment. The details of the procedure can be found at appendix 4.

The same figures and metrics computed during experiment 1 were also computed for experiment 2. This will allow a direct comparison between each setting's performance.

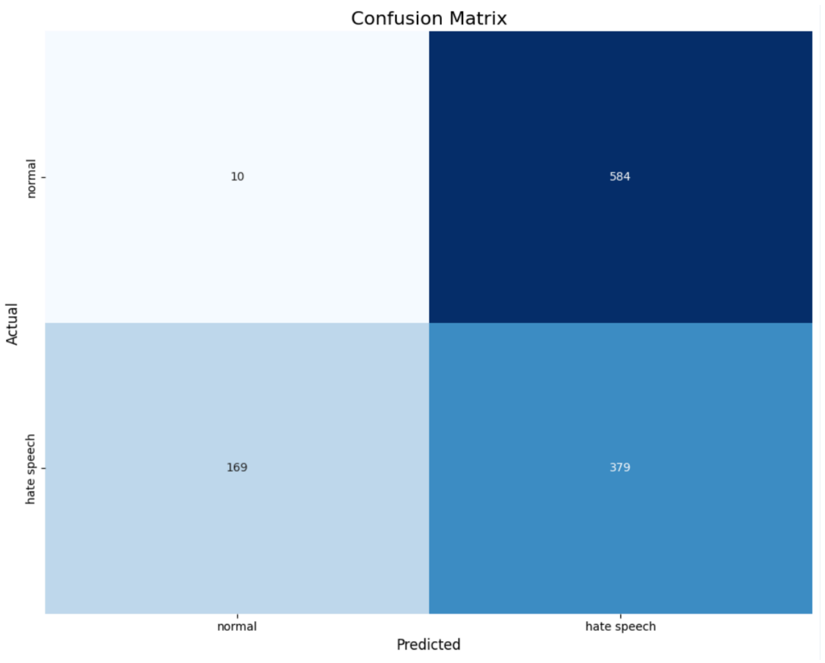
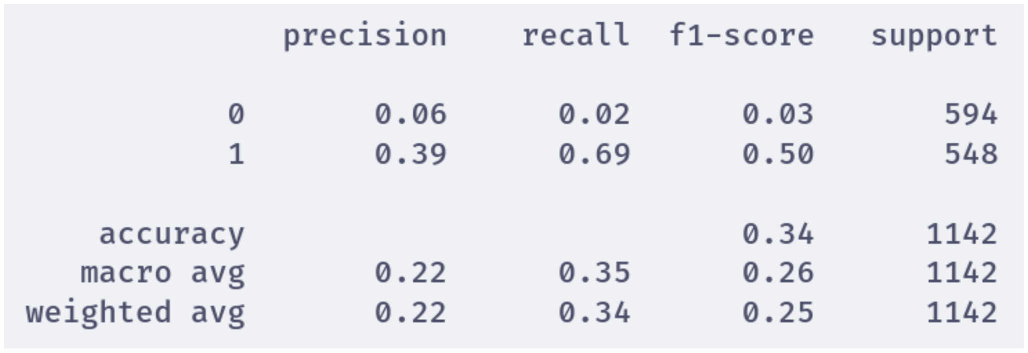


Figure 9 - Confusion matrix – Experiment 2



```
              precision    recall  f1-score   support

           0       0.06      0.02      0.03       594
           1       0.39      0.69      0.50       548

    accuracy                           0.34      1142
   macro avg       0.22      0.35      0.26      1142
weighted avg       0.22      0.34      0.25      1142
```

Figure 10 - Classification report – Experiment 2

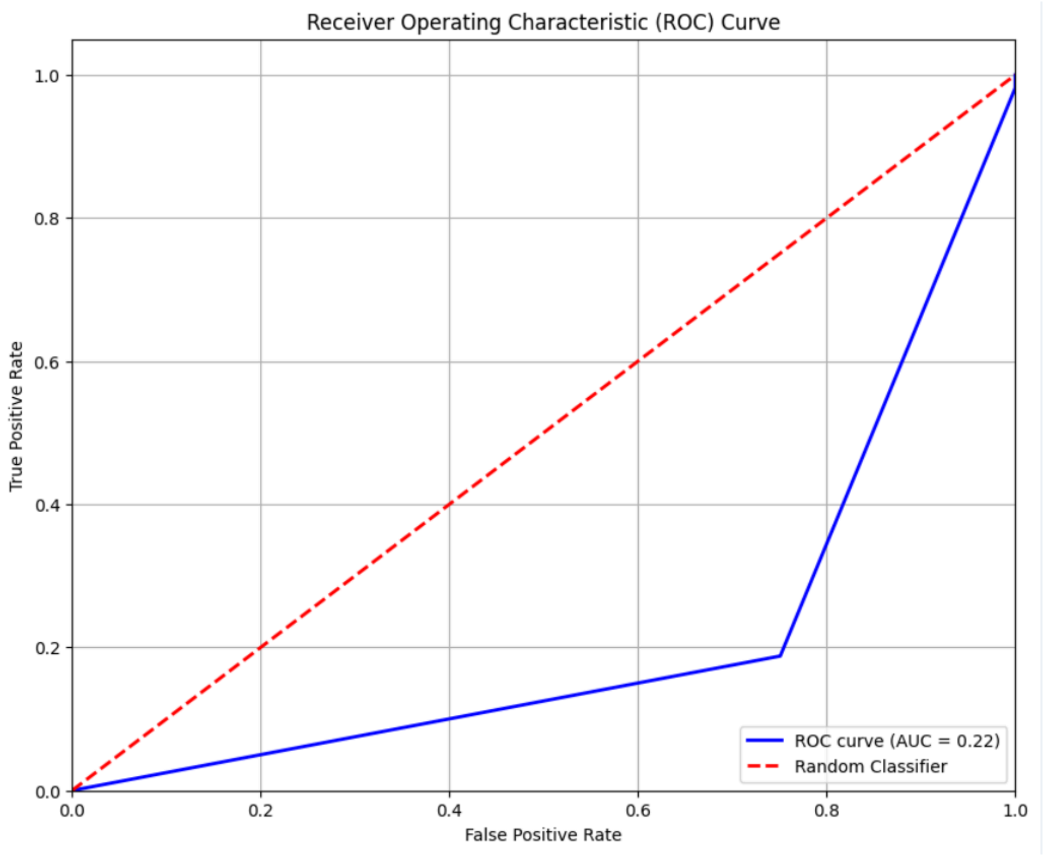


Figure 11 - ROC Curve – Experiment 2

### 2.2.3 - Results discussion

Unexpectedly, the results showed a decline in performance. While the number of correctly classified normal messages increased from 3 to 10, the number of correctly classified hate speech messages dropped from 466 to 379. This led to a significant rise in False Negatives (hate speech predicted as normal), increasing from 80 to 169, and a slight increase in False Positives (normal messages predicted as hate speech), which went from 584 to 591.

Looking at the classification report, the overall accuracy decreased by three percentage points, from 44% to 41%. Although the F1-score for normal messages improved slightly, from 0.01 to 0.03, the F1-score for hate speech dropped substantially, from 0.58 to 0.50. The AUC score, already poor, worsened further, falling from 0.24 to 0.21.

These results were disappointing and raised concerns about possible issues in the experimental process. A thorough review of the experiment's code revealed no errors. The next step was to examine the dataset used in the experiment, and this is where the poor performance started to make sense.

The HateXplain dataset is annotated by three reviewers, and the ground-truth label was determined using a majority-vote strategy. The assumption was that this would create a clear distinction between

normal and hate messages. However, this was not the case. Below are some examples of messages labeled as "normal" under this majority-vote approach:

"what i am willing to recognize is that people willingly follow a raghead rapist and think they are doing something moral"

"white boys and ghetto white girls make me sick i can not stand you all"

"imagine being a lying kike shill like..."

"continues to brag about muting continues to prove my point about him being a kike continues to do the kike thing of projecting mental deficiencies"

"this is fucking bullshit the sand niggers got told they were being sent back so they rioted they were not protesting conditions daily mail is horse shit"

"listen lady kill yourself"

"following on from the other socialist tory pm the equally useless europhile traitor camoron theresa sharia may decides to try the hug a moslem suicide bomber approach"

"i tolled this nigger to tears"

"he a midget kike yeah been known for a while on the right and everyone who agreed he was a midget kike got banned from all the alt right forums"

"jews are fucking liars architects of diversity and division"

These examples, which were part of the few-shot examples fed into the prompt, are clearly not normal by any reasonable standard. Upon further inspection, the same issue persisted across the training, validation, and test sets. This led to the hypothesis that including mislabeled messages in the prompt may have confused the language model, degrading performance.

These findings prompted me to find another suitable dataset and repeat experiments 1 and 2 to determine whether the low performance is reproducible or specific to the HateXplain dataset. The details of this new experimental setup will be covered in the next section.

## 2.3 - Revised experiments with the Stormfront dataset

### 2.3.1 - Method description

The stormfront dataset was introduced in the paper titled "Hate Speech Dataset from a White Supremacy Forum" by de Gibert et al., published in the Proceedings of the 2nd Workshop on Abusive Language Online (ALW2) in 2018[17]. It consists of 10568 sentences extracted from Stormfront, a white supremacist online forum. The sentences are manually annotated at the sentence level as either containing hate speech or not. The content was extracted from Stormfront using web-scraping techniques, covering posts from 2002 to 2017.

The core dataset consists of:

- 1,119 sentences labeled as HATE
- 8,537 sentences labeled as NOHATE

Each sentence is provided with a post identifier, a user identifier, a sub-forum identifier and the number of previous posts the annotator had to read before making a decision The specific version of the dataset used in the experiments were obtained from hugging-face[18]. The preprocessing steps performed for this dataset can be found in appendix 5.

Once the datasets were preprocessed the experiment consisted of basically repeating experiments 1 and 2 with this other dataset. A few extra details will be discussed in the following section.

### 2.3.2 - Implementation details and results

The first part of the experiment involved re-running experiment 1 with a preprocessed version of the training dataset, stormfront_train.csv, which was generated using the dataset_loading2.ipynb script (See appendix 5 for further details). This dataset comprises 450 normal messages and 450 hate messages. All experimental parameters and settings were maintained identical to those in Experiment 1. For a detailed review of the original experiment setup, please refer to Section 3.1.2.

The same classification metrics were computed and are shown below:

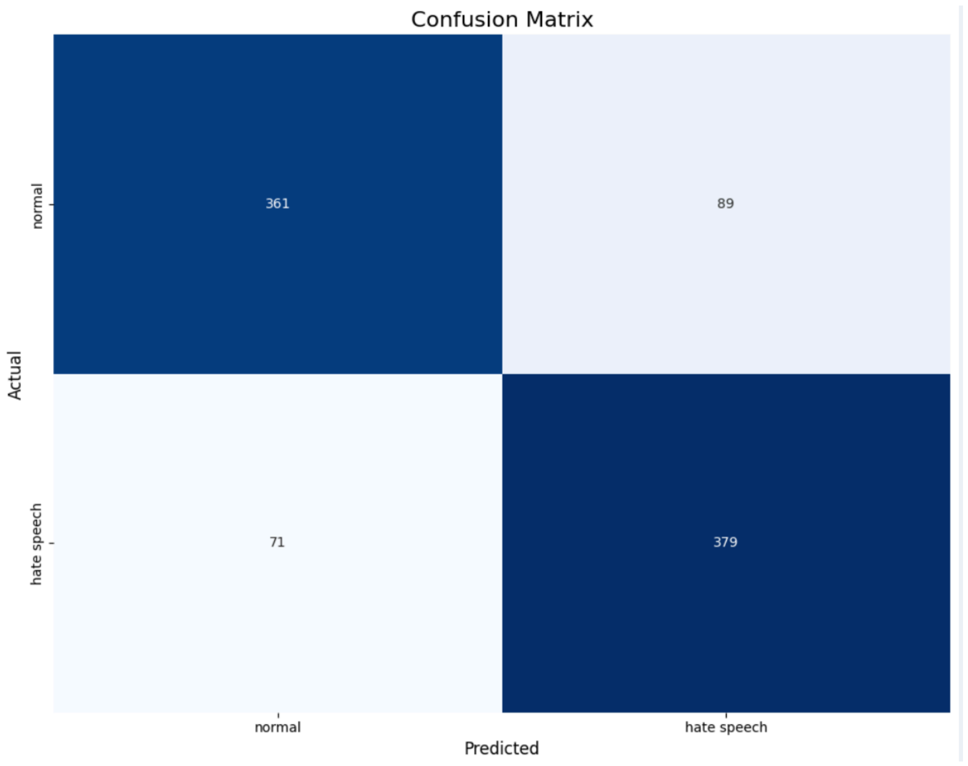
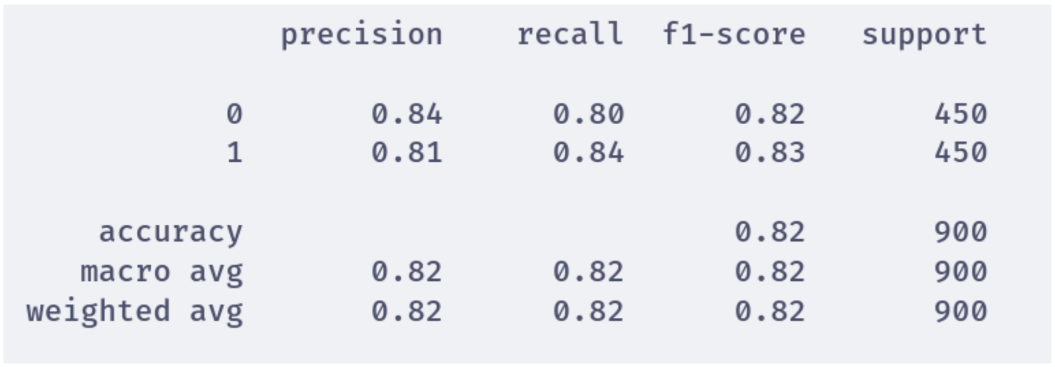


Figure 12 - Confusion matrix – zero shot prompt – Stormfront dataset



```
              precision    recall  f1-score   support

           0       0.84      0.80      0.82       450
           1       0.81      0.84      0.83       450

    accuracy                           0.82       900
   macro avg       0.82      0.82      0.82       900
weighted avg       0.82      0.82      0.82       900
```

Figure 13 - Classification report – zero shot prompt – Stormfront dataset

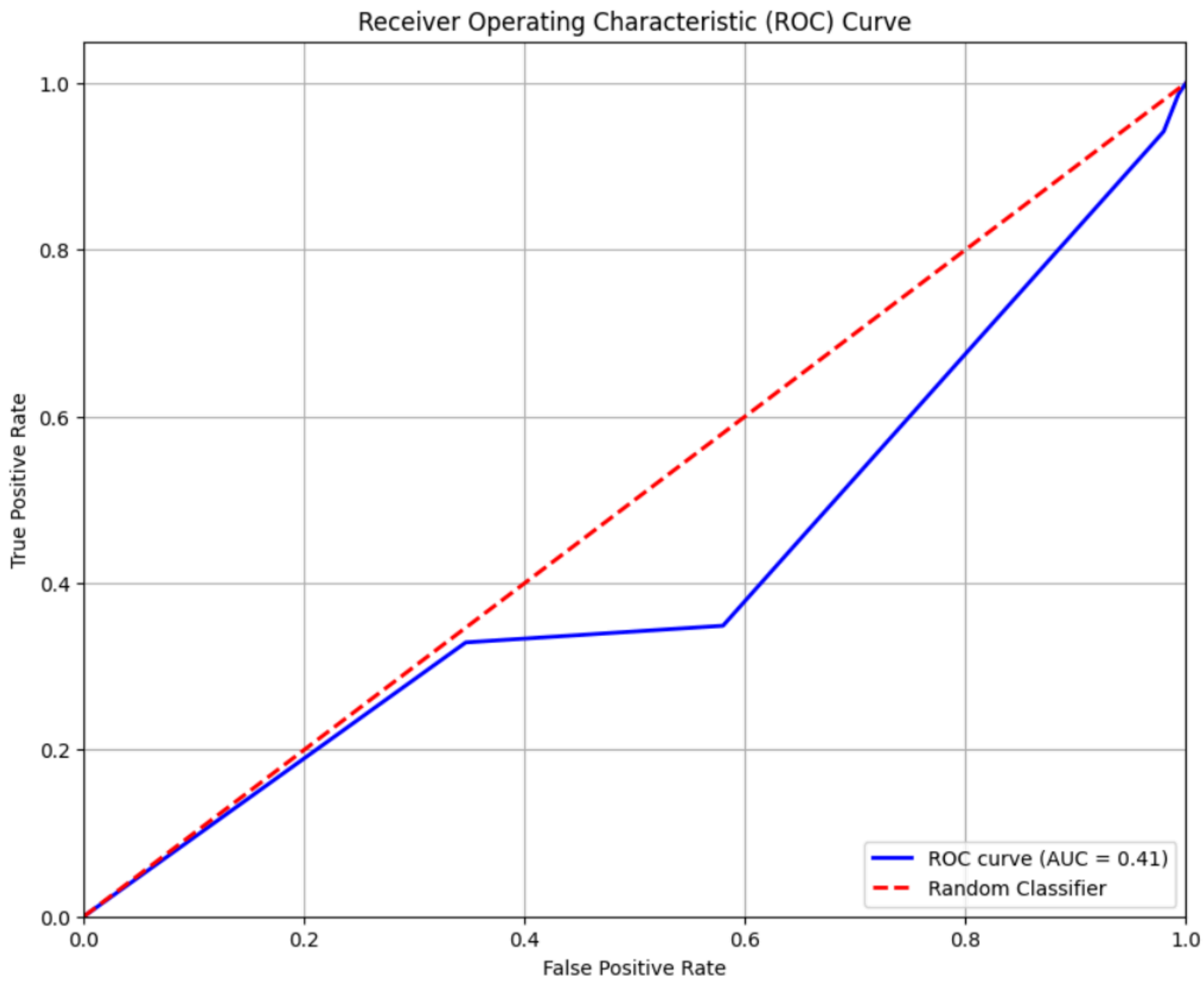


Figure 14 - ROC curve – zero shot prompt – Stormfront dataset

The second half of the experiment consisted of reproducing experiment 2 with the new dataset. Here some details deserve attention: The few shot examples were sampled from the "stormfront_train.csv".

50 hate messages and 50 normal messages were sampled from it. Therefore, the prompt will have 100 few-shot examples in its context. The classifications were performed on the "stormfront_test.csv" dataset (also generated in dataset_loading2.ipynb). This dataset contains 100 test messages and is balanced .

The classification results for this dataset are the following:

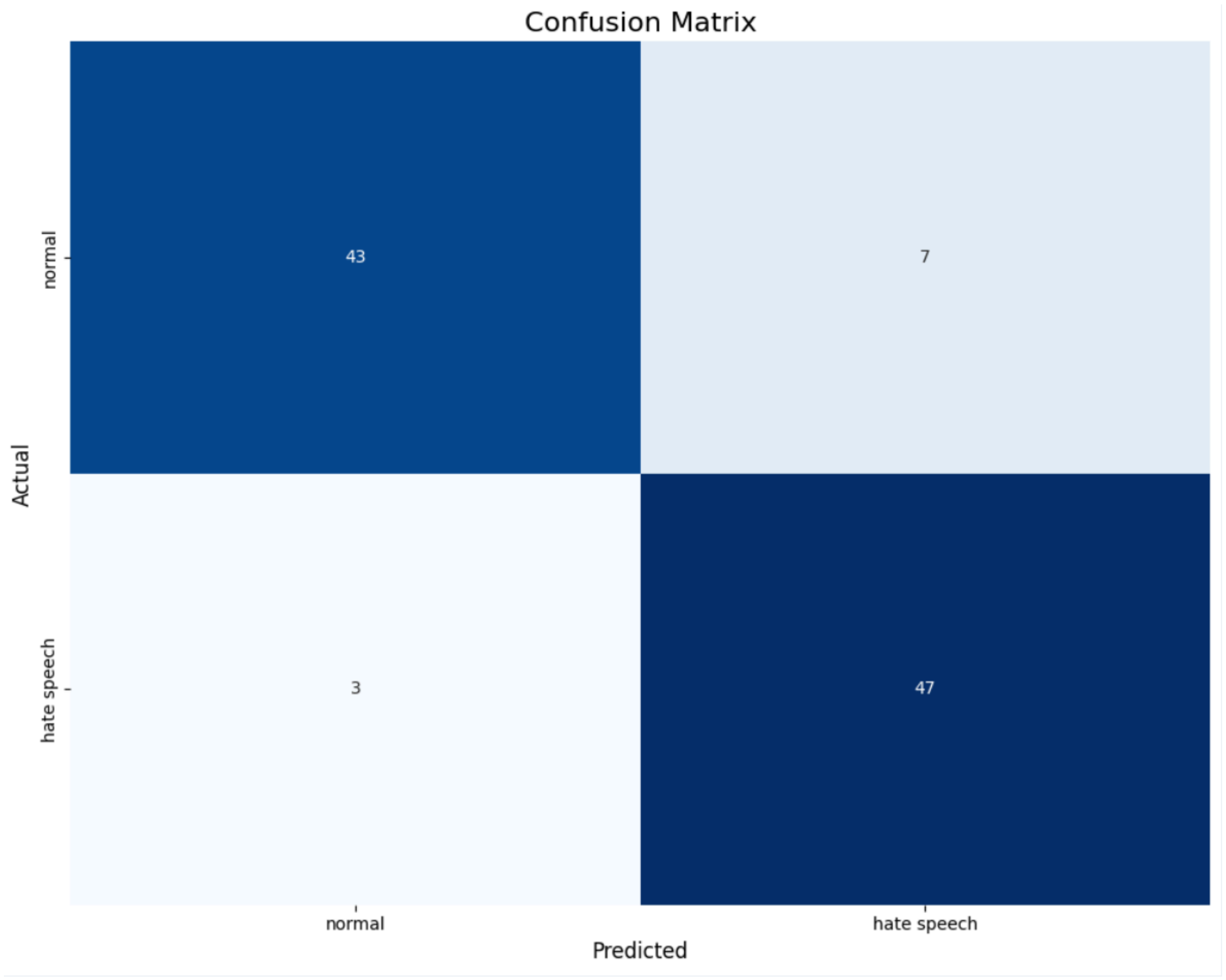
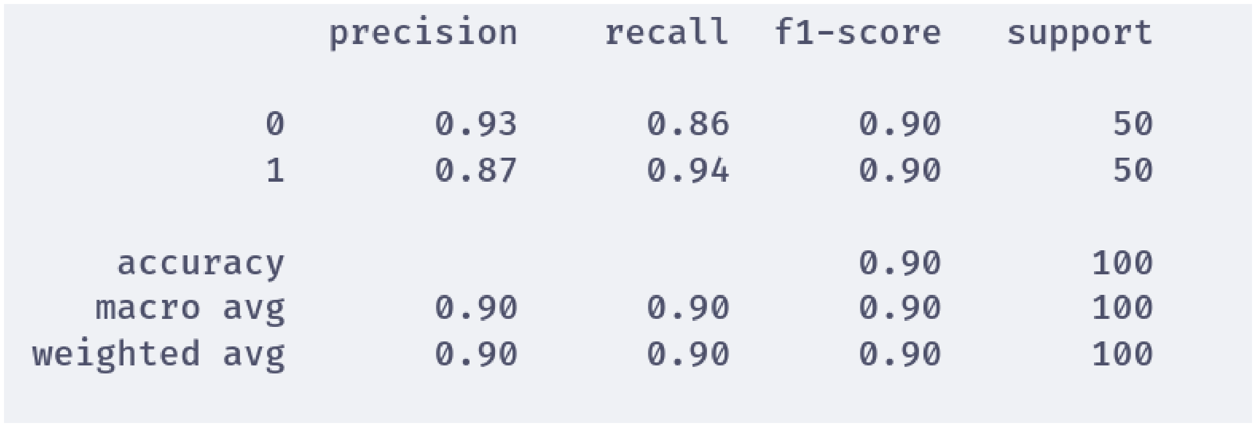


Figure 15 - Confusion matrix – few shot examples – Stormfront dataset



```
              precision    recall  f1-score   support

           0       0.93      0.86      0.90        50
           1       0.87      0.94      0.90        50

    accuracy                           0.90       100
   macro avg       0.90      0.90      0.90       100
weighted avg       0.90      0.90      0.90       100
```

Figure 16 - Classification report – few shot examples – Stormfront dataset

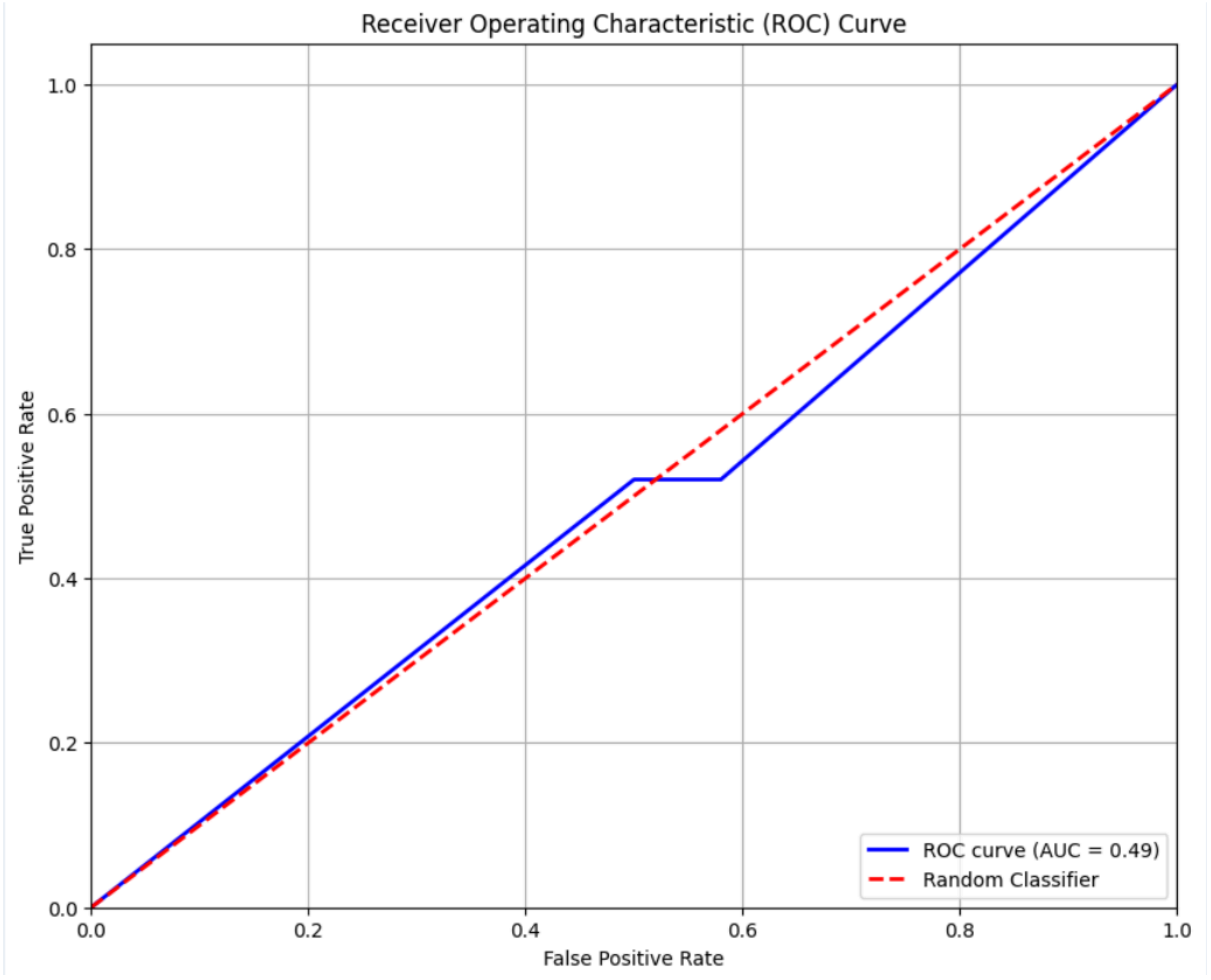


Figure 17 - ROC Curve – few shot examples – stormfront dataset

### 2.3.3 - Results discussion

The results were very interesting and indeed confirmed that the problem was in the data annotation of the hateXplain dataset and the form it was preprocessed. Overall the classification results improved substantially when compared to experiments 1 and 2.

Looking at the zero-shot prompt results, it is possible to see that the number of True Positives is now much higher than the False Positives (379 vs 89) and the same logic applies to True Negatives VS False Negatives (361 vs 71). The precision for the hate-speech predictions is now 81%, which means that anytime the system predicts a message is hate speech, it gets it correctly 81% of the time. The recall is also higher at 84%, which means that out of the true hate speech messages present in the test set, the system is detecting 84% of the cases.

This is equivalent to an f1-score of 0.83. This is 25 percentage points higher than the results that were obtained for experiment 1, where the f1 score was 0.58. Previously there was a high-recall low-precision scenario, and the system was flagging almost every message as hate speech, as can be seen in the confusion matrix from section 3.1.2. Now the LLM does a much better job at distinguishing the type of message, with and overall accuracy of 82%. That's double of that obtained in experiment 1 (41%).

When comparing the new zero-shot prompting experiment results with the new few-shot prompting experiment results, it was possible to observe some improvements. The overall accuracy jumped from 82% to 90%, the precision in the detecting hate speech went from 81% to 87% and the recall increased 10 percentage points from 84% to 94%. These results support the argument that adding few shot examples to the human message in the prompt aids the LLM to better distinguish between the two types of messages.

Some remarks must be made about the ROC curves and ROC-AUC scores. These metrics depend on the probabilities the LLM generate for the confidence score. The classification metrics reviewed in the previous paragraphs and these scores are telling a different story. The ROC-AUC scores improved when compared to the original experiments 1 (0.24 to 0.41) and 2 (0.22 to 0.49), but these numbers don't make sense with the precision and recall metrics that were computed. One hypothesis is that the confidence numbers the LLM is generating are somewhat random and the tool calling that langchain uses to make the output comply with the Pydantic class is simply generating a number in the desired range, but it is not actually drawing from a real probability distribution. To confirm that, more rigorous experiments would need to be conducted, but this is out of scope for this project. Nonetheless, it is definitely an interesting finding.

## 2.4 - Chain of thought prompting

### 2.4.1 - Method description

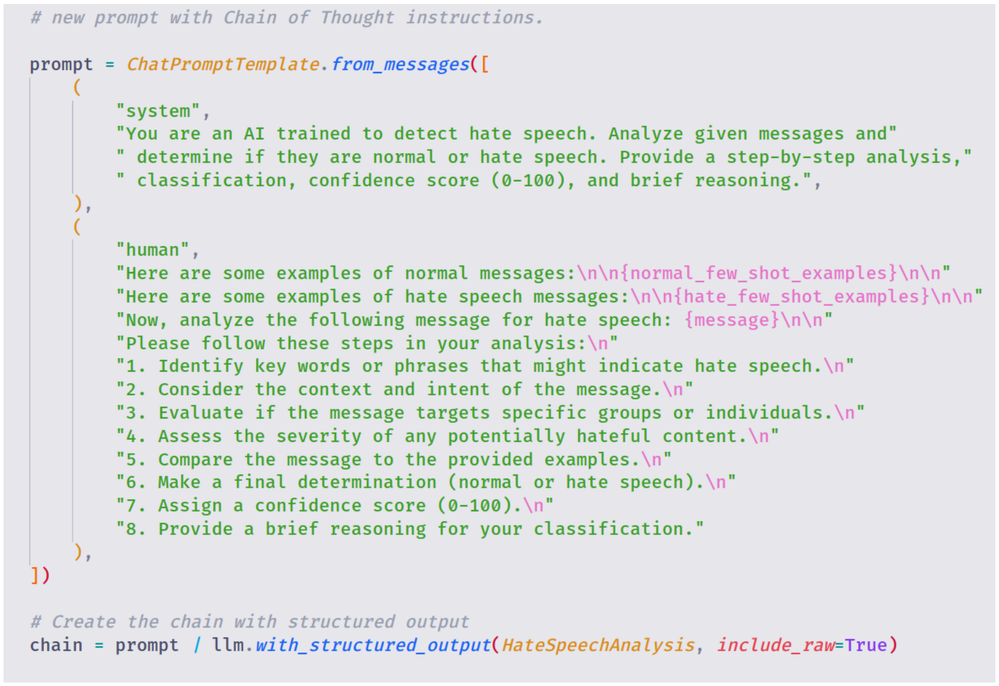
Chain of Thought (CoT) prompting is a technique used to enhance the reasoning capabilities of large language models (LLMs) by encouraging them to generate intermediate steps or rationales before providing a final answer [19]. Among a variety of studies, a seminal paper by Wang et. al showed that Cot prompting resulted in significant improvements in the performance of LLM-based inference systems[20].

Examples of enhancements include improving the multi-step reasoning capabilities and the overall accuracy for complex tasks. In another study, the accuracy of GPT-3 in arithmetic reasoning tasks increased from 17.7% to 78.7% simply by adding the CoT instruction "let's think step by step"[21].

Therefore the hypothesis for the hate-speech detection is that by adding CoT instructions, the LLM will be able to analyze the message more thoroughly, and ultimately classify them better. The experiment will consist of adding the chain of thought instructions in the prompt, specifically the human message. All the other settings, datasets and parameters remain identical to the previous example.

## 2.4.2 - Implementation details and results

The following CoT instructions were added to the prompt:



```
# new prompt with Chain of Thought instructions.

prompt = ChatPromptTemplate.from_messages([
    (
        "system",
        "You are an AI trained to detect hate speech. Analyze given messages and"
        " determine if they are normal or hate speech. Provide a step-by-step analysis,"
        " classification, confidence score (0-100), and brief reasoning.",
    ),
    (
        "human",
        "Here are some examples of normal messages:\n\n{normal_few_shot_examples}\n\n"
        "Here are some examples of hate speech messages:\n\n{hate_few_shot_examples}\n\n"
        "Now, analyze the following message for hate speech: {message}\n\n"
        "Please follow these steps in your analysis:\n"
        "1. Identify key words or phrases that might indicate hate speech.\n"
        "2. Consider the context and intent of the message.\n"
        "3. Evaluate if the message targets specific groups or individuals.\n"
        "4. Assess the severity of any potentially hateful content.\n"
        "5. Compare the message to the provided examples.\n"
        "6. Make a final determination (normal or hate speech).\n"
        "7. Assign a confidence score (0-100).\n"
        "8. Provide a brief reasoning for your classification."
    ),
])

# Create the chain with structured output
chain = prompt | llm.with_structured_output(HateSpeechAnalysis, include_raw=True)
```

Figure 18 - Prompt with Chain of Thought (CoT)

The classification process was broken down into eight steps, that are similar to what a human annotator would go through in order to label the data. As can be seen, the few shot examples are still there. The amount of few shot examples remained unchanged from the previous experiment (50 normal messages 50 hate messages).

The classification metrics computed after inference was performed were the following:

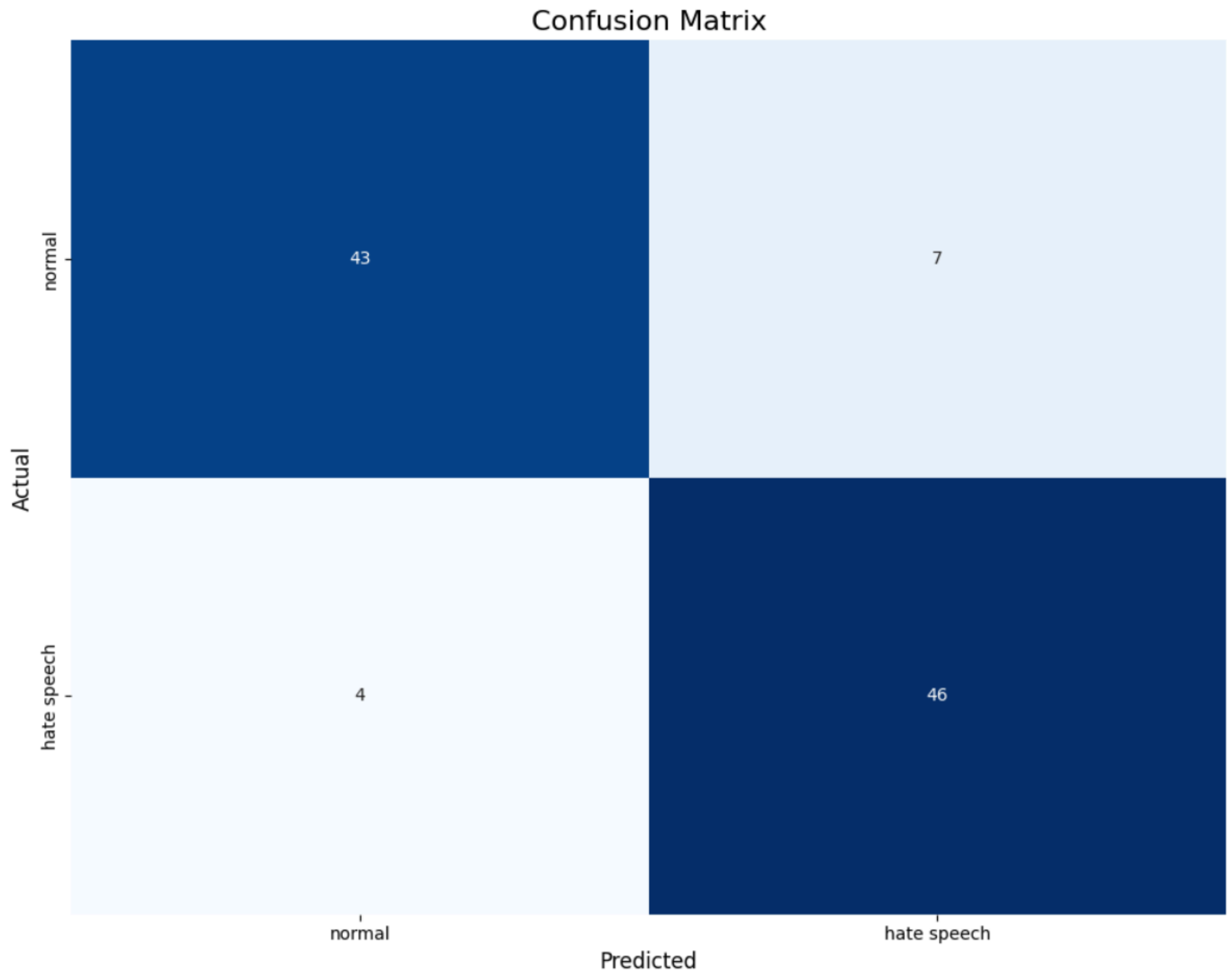
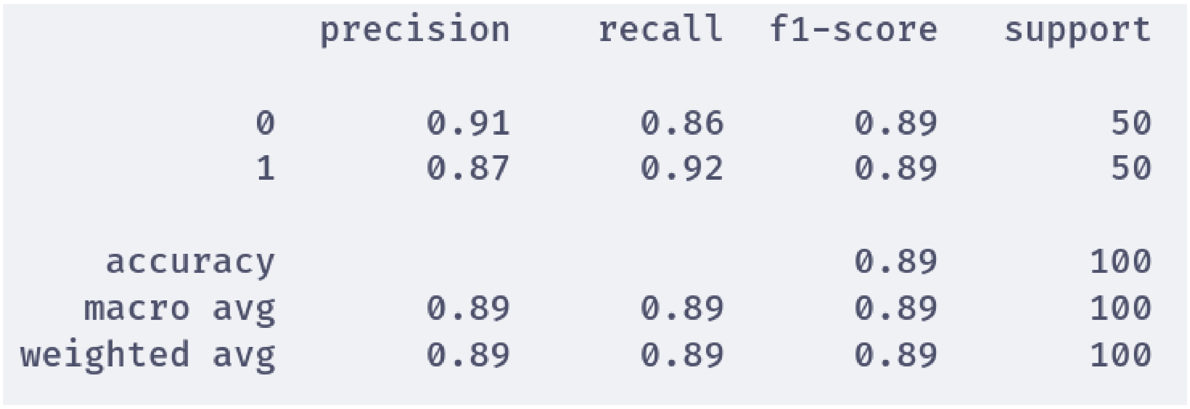


Figure 19 - Confusion Matrix – CoT experiment



```
              precision    recall  f1-score   support

           0       0.91      0.86      0.89        50
           1       0.87      0.92      0.89        50

    accuracy                           0.89       100
   macro avg       0.89      0.89      0.89       100
weighted avg       0.89      0.89      0.89       100
```

Figure 20 - Classification report – CoT experiment

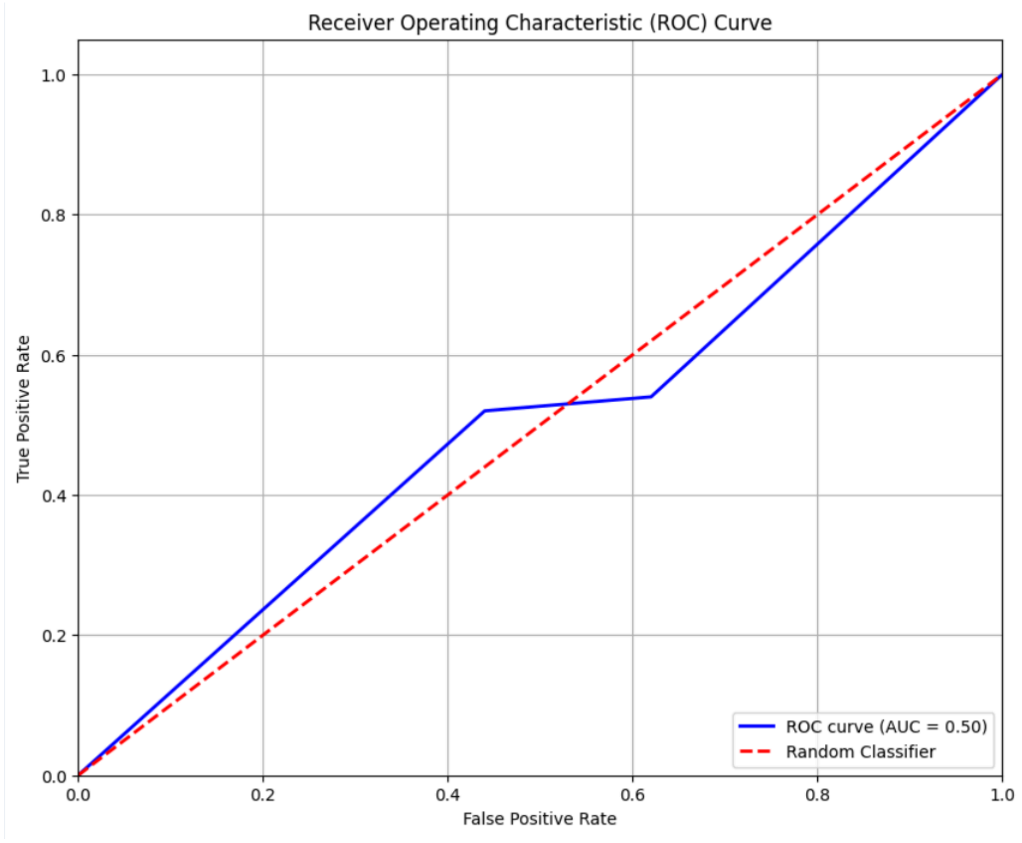


Figure 21 - ROC Curve – CoT Prompt

### 2.4.3 - Results discussion

The results were almost identical to those of the previous experiment. In fact, the overall accuracy decreased 1 percentage point. Comparing the confusion matrix from this and the last experiment, it is possible to note only 1 example that was previously correctly classified as hate speech now is classified as a normal message. But aside from that the results are identical.

The reason why the performance did not improve could be explained by the fact that the few shot examples in the LLMs prompt already give it enough context about what the task definitions and subtleties. Another hypothesis is that hate-speech detection is not a very difficult and long multi-step process. Therefore, given that the LLM already has clear instructions in the system prompt and good and diverse examples, adding the chain of Thought offer diminishing. It is a "Nice to have" not a "must have". Nonetheless as an overall remark, given the task sensitive performance of LLM systems, the best heuristic seems to be always experiment with both settings and move on with what performs best.

# 3. Conclusion

This work brings attention to both the opportunities and challenges of using generative AI architectures for detecting hate speech. We experimented with zero-shot, few-shot, and chain-of-thought approaches, and found that their effectiveness varied depending on the dataset. While zero-shot learning was straightforward, it struggled with precision, particularly in recognizing normal messages. Few-shot learning showed some improvements by providing examples, but its success heavily relied on the dataset quality. The chain-of-thought method highlighted the role of task complexity in requiring intermediate reasoning steps, though its improvements were minimal in this case.

Further testing with the Stormfront dataset revealed that better data quality and labeling significantly boosted performance compared to the HateXplain dataset. These findings emphasize the importance of careful dataset curation and thoughtful prompt design for large language models in hate speech detection.

While generative AI offers promising approaches to improve hate speech detection, several challenges remain, such as managing bias, reducing false positives, and providing clear contextual examples. Due to the limited scope of this study, very few was discussed about the explanations (present "reasoning" key) output by the LLM. Therefore future research could analyze whether the explanations given are coherent with the labels. This could have reveal interesting insights and potentially unlock more explainable solutions. Another finding was that the confidence scores generated seemed a little bit random, what hindered the roc-auc scores. Another study could explore the score generation in a more systematic way and evaluate whether it is possible to extract robust confidence scores from the LLM responses.

Finally, an interesting investigation would be integrating multi-agent systems in an attempt to create a more reliable and interpretable solutions. In conclusion, with social media platforms facing increasing volumes of harmful content, advancements in generative AI have the potential to play a crucial role in fostering safer online interactions.

## 4. References

[1] Srivastava, A., Gupta, S., Sharma, S., Sharma, S., Sharma, S., & Sharma, S. (2023). SS-GAN-mBERT: A semi-supervised approach for hate speech detection in Indo-European languages using generative adversarial networks and multilingual BERT. PLoS ONE, 18(9), e0290960. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC11049309/

[2] Keller, Y., Dror, G., & Dalal, O. (2024). Detecting anti-Semitic hate speech using large language models. arXiv. https://arxiv.org/abs/2405.03794

[3] Aljameel, S. S., Alabbad, D. A., Alshehri, N. A., Alqahtani, S. S., Alqahtani, D. A., & Alsuhaibani, R. S. (2023). A deep learning approach to hate speech detection on Twitter. Expert Systems, 40(5), e13562. https://onlinelibrary.wiley.com/doi/10.1111/exsy.13562

[4] - Kaya, A., Ozcelik, O., & Toraman, C. (2024). ARC-NLP at ClimateActivism 2024: Stance and Hate Speech Detection by Generative and Encoder Models Optimized with Tweet-Specific Elements. CASE. https://aclanthology.org/2024.case-1.15.pdf

[5] - Assis, G., Amorim, A., Carvalho, J., de Oliveira, D., Vianna, D.Q., & Paes, A. (2024). Exploring Portuguese Hate Speech Detection with Transformers. LatinX in AI at North American Chapter of the Association for Computational Linguistics Conference 2024. https://research.latinxinai.org/papers/naacl/2024/pdf/Gabriel_Assis.pdf

[6] - Chhikara, M., Malik, S.K., & Jain, V. (2024). Identification of social network automated hate speech using GLTR with BERT and GPT-2 : A novel approach. Journal of Information and Optimization Sciences. https://doi.org/10.47974/JIOS-1549

[7] - Hashmi, E., YAYILGAN YILDIRIM, S., Hameed, I.A., Yamin, M.M., Ullah, M., & Abomhara, M. (2024). Enhancing Multilingual Hate Speech Detection: From Language-Specific Insights to Cross-Linguistic Integration. IEEE Access, 12, 121507-121537. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10662891

[8] Mathew, B., Saha, P., Yimam, S. M., Biemann, C., Goyal, P., & Mukherjee, A. (2021). HateXplain: A benchmark dataset for explainable hate speech detection. Proceedings of the AAAI Conference on Artificial Intelligence, 35(17), 14867-14875.

[9] Hate-speech-CNERG. (n.d.). Hugging Face. https://huggingface.co/datasets/Hate-speech-CNERG/hatexplain

[10] Mamun, M. B., Tsunakawa, T., Nishida, M., & Nishimura, M. (2024). Hate speech detection by using rationales for judging sarcasm. Applied Sciences, 14(11), 4898.

[11] LangChain. (n.d.). How to return structured data from a model. https://python.langchain.com/docs/how_to/structured_output/

[12] Wikipedia. (n.d.). Zero-shot learning. https://en.wikipedia.org/wiki/Zero-shot_learning

[13] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). Language models are few-shot learners. arXiv. https://arxiv.org/abs/2005.14165

[14] Perez, E., Kiela, D., & Cho, K. (2021). True few-shot learning with language models. Advances in Neural Information Processing Systems, 34.

[15] Gao, T., Fisch, A., & Chen, D. (2020). Making pre-trained language models better few-shot learners. arXiv. https://arxiv.org/abs/2012.15723

[16] Schick, T., & Schütze, H. (2020). It's not just size that matters: Small language models are also few-shot learners. arXiv. https://arxiv.org/abs/2009.07118

[17] de Gibert, O., Perez, N., García-Pablos, A., & Cuadros, M. (2018). Hate speech dataset from a white supremacy forum. arXiv. https://arxiv.org/abs/1809.04444

[18] Odegiber. (n.d.). Hate_speech18. Hugging Face. https://huggingface.co/datasets/odegiber/hate_speech18

[19] Zhang, A., Xu, Y., Lin, Z., Zheng, C., Deng, Z., Lai, K., Yao, Y., Zhou, C., Zhang, B., Wang, Y., Shen, S., Liang, P., Zhu, J., Zhuang, B., Wang, W., & Zhu, X. (2022). Automatic chain of thought prompting in large language models. arXiv. https://arxiv.org/abs/2210.03493

[20] Wang, B., Deng, C., Sun, S., Chen, Z., Cheng, Y., Xu, Y., Gui, T., Shen, Y., Huang, X., & Wu, F. (2022). Towards understanding chain-of-thought prompting: An empirical study of what matters. arXiv. https://arxiv.org/abs/2212.10001

[21] Chen, J., Xu, J., Zhu, J., Xu, Y., Luo, J., Wang, L., Yang, Z., & Zheng, C. (2023). When do you need chain-of-thought prompting for ChatGPT?. arXiv. https://arxiv.org/abs/2304.03262

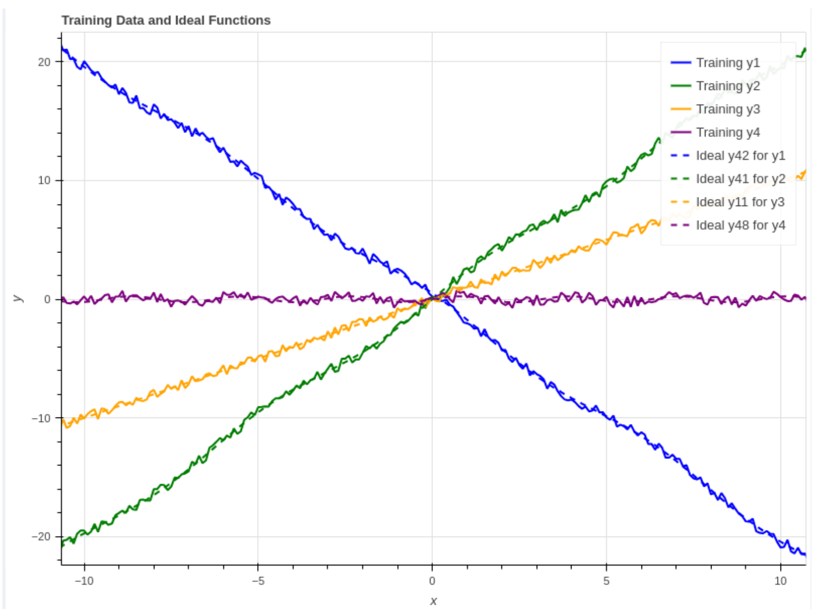
# 5. Appendices

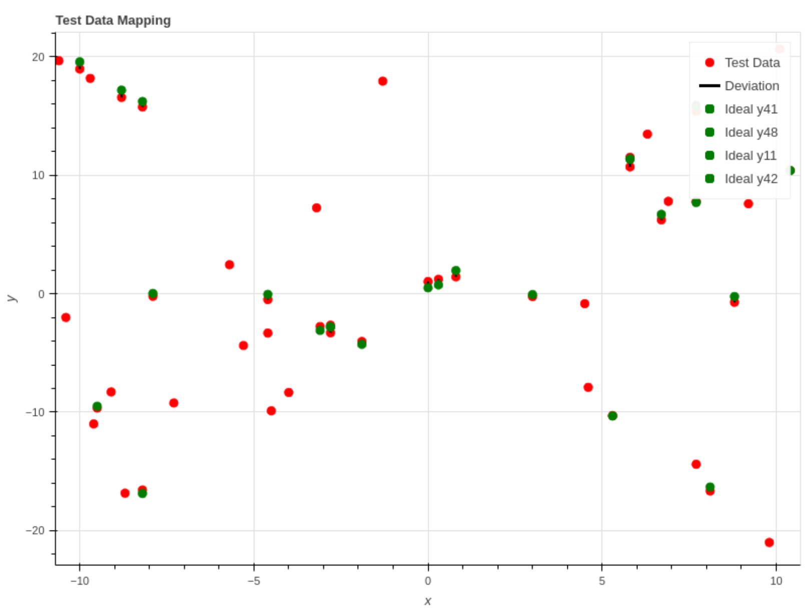
## 1 Base task and Project repository

All the code base from both the base task and the written assignment can be found on my github at:

https://github.com/hualcosa/programming_with_python_DLMDSPWP01

The repository is public an contain a very detailed README.md file detailing how to execute the solution, run the unit tests and generate the visualizations. Here is a glimpse of the visualizations generated:

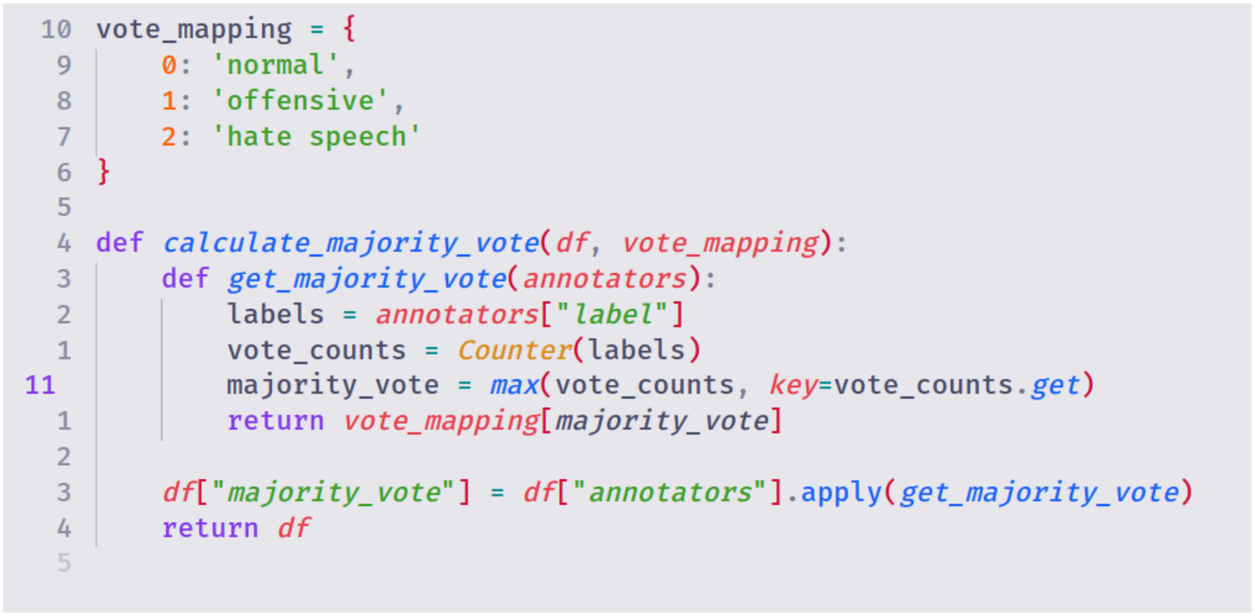




The code for this written assignment is in the same repository in the folder **writen_assignment.**

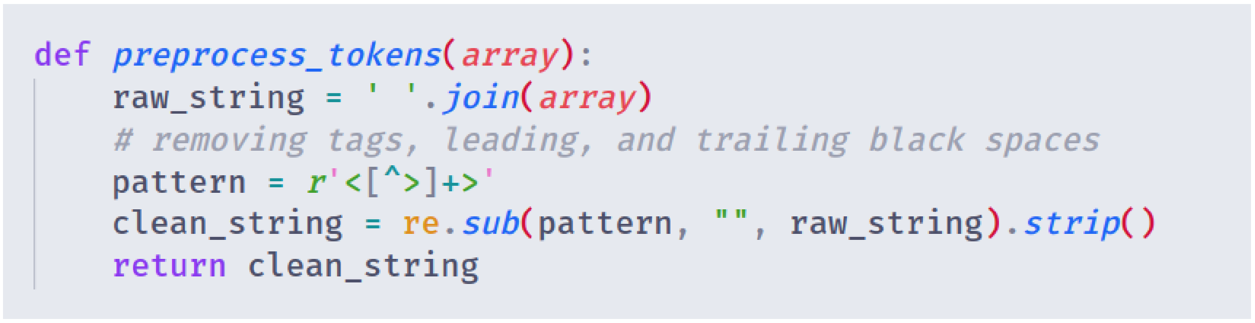
## 2 Dataset preprocessing functions

Function used to extract the majority-vote from the annotators for a specific message.



```python
vote_mapping = {
    0: 'normal',
    1: 'offensive',
    2: 'hate speech'
}

def calculate_majority_vote(df, vote_mapping):
    def get_majority_vote(annotators):
        labels = annotators["label"]
        vote_counts = Counter(labels)
        majority_vote = max(vote_counts, key=vote_counts.get)
        return vote_mapping[majority_vote]

    df["majority_vote"] = df["annotators"].apply(get_majority_vote)
    return df
```

Function used to remove xml tags:



```python
def preprocess_tokens(array):
    raw_string = ' '.join(array)
    # removing tags, leading, and trailing black spaces
    pattern = r'<[^>]+>'
    clean_string = re.sub(pattern, "", raw_string).strip()
    return clean_string
```

Function used to extract the consensus relevant tokens:

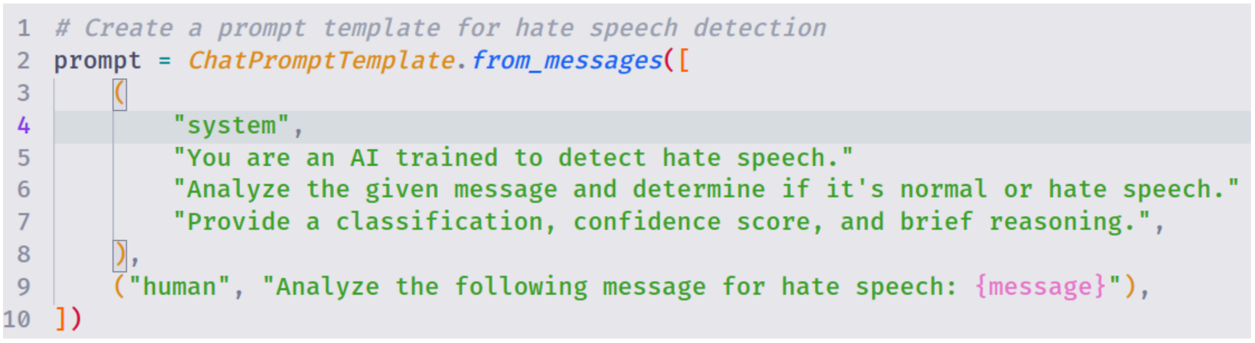
```
def process_relevant_tokens(row):
    # if the message contains relevant tokens for the review, extract and process it
    if row['rationales'].size > 0:
        try:
            # 1. Compute the intersection of relevant tokens
            rationales = np.vstack(row["rationales"])
            intersection_mask = np.all(rationales == 1, axis=0)
            # 2. Apply the mask to post_tokens
            filtered_tokens = row["post_tokens"][intersection_mask]
            # 3. Preprocess the resulting token array
            preprocessed_text = preprocess_tokens(filtered_tokens)

            return preprocessed_text
        except:
            # return the original rationales in case it is not possible to process them
            print("Error processing rationales for message: ", row['id'])
            return rationales
    # if the message is normal, return None
    return None
```

## 3 - Experiment 1 Implementation details

Pydantic class that defines the output format.

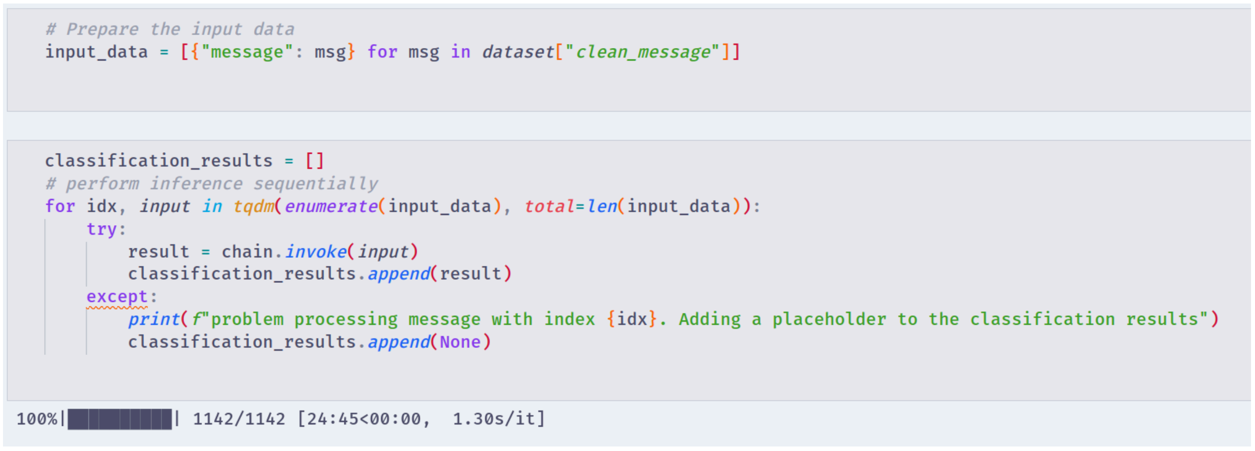
```
# Define the pydantic class with JSON schema for hate speech detection
class HateSpeechAnalysis(BaseModel):
    '''
    A pydantic class that specifies the json schema of the hate speech detection task
    '''
    classification: str = Field(
        ..., description="Classification of the message: 'normal' or 'hate speech'"
    )
    confidence: float = Field(
        ..., ge=0, le=1, description="Confidence score of the classification (0 to 1)"
    )
    reasoning: str = Field(..., description="Brief explanation for the classification")

    class Config:
        json_schema_extra = {
            "example": {
                "classification": "normal",
                "confidence": 0.95,
                "reasoning": "The message contains no offensive language or discriminatory content.",
            }
        }
```

Prompt template used:



```python
# Create a prompt template for hate speech detection
prompt = ChatPromptTemplate.from_messages([
    (
        "system",
        "You are an AI trained to detect hate speech."
        "Analyze the given message and determine if it's normal or hate speech."
        "Provide a classification, confidence score, and brief reasoning.",
    ),
    ("human", "Analyze the following message for hate speech: {message}"),
])
```

Inference logic:



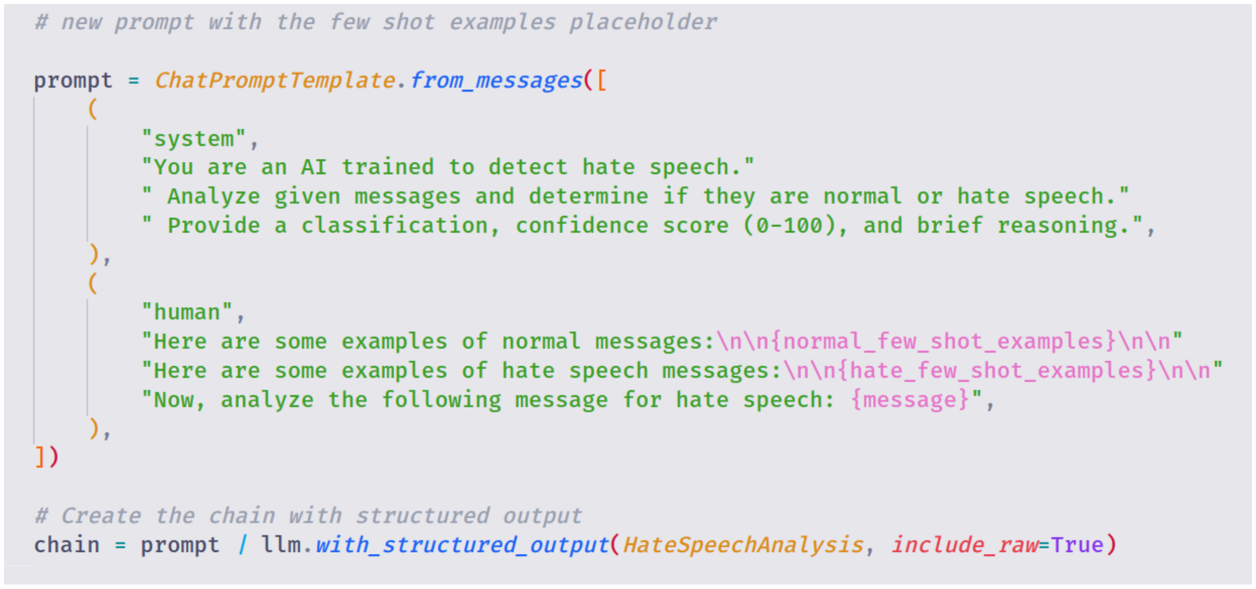
```python
# Prepare the input data
input_data = [{"message": msg} for msg in dataset["clean_message"]]
```

```python
classification_results = []
# perform inference sequentially
for idx, input in tqdm(enumerate(input_data), total=len(input_data)):
    try:
        result = chain.invoke(input)
        classification_results.append(result)
    except:
        print(f"problem processing message with index {idx}. Adding a placeholder to the classification results")
        classification_results.append(None)
```

```
100%|██████████| 1142/1142 [24:45<00:00,  1.30s/it]
```

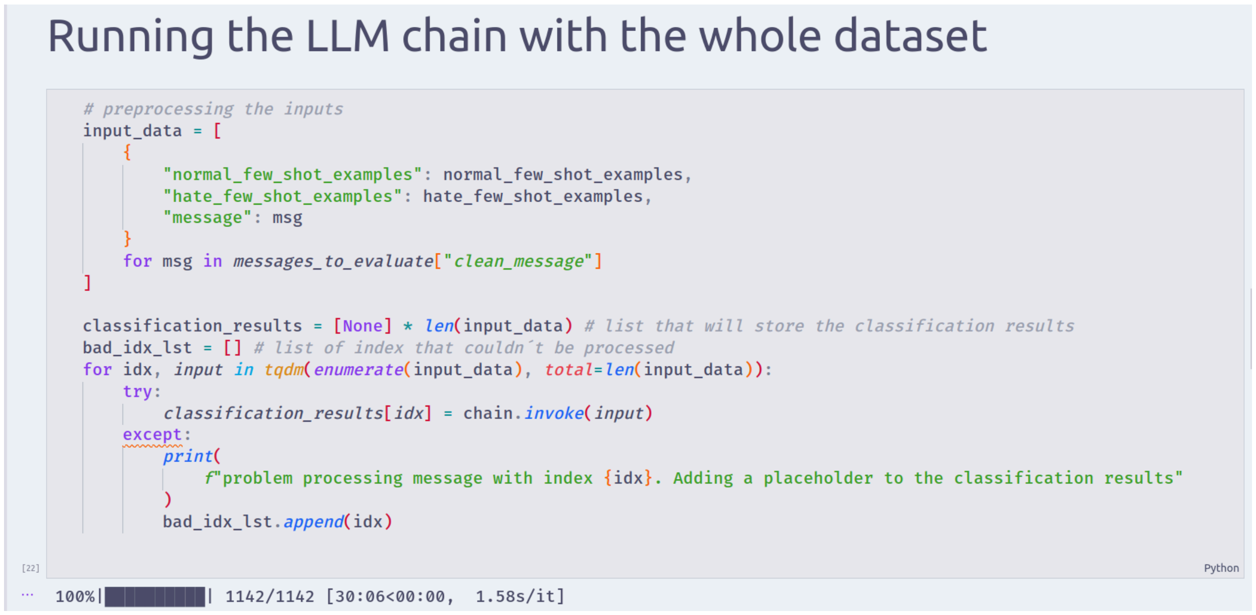
# 4 - Experiment 2 Implementation details

New prompt with few-shot examples placeholders:



```python
# new prompt with the few shot examples placeholder

prompt = ChatPromptTemplate.from_messages([
    (
        "system",
        "You are an AI trained to detect hate speech."
        " Analyze given messages and determine if they are normal or hate speech."
        " Provide a classification, confidence score (0-100), and brief reasoning.",
    ),
    (
        "human",
        "Here are some examples of normal messages:\n\n{normal_few_shot_examples}\n\n"
        "Here are some examples of hate speech messages:\n\n{hate_few_shot_examples}\n\n"
        "Now, analyze the following message for hate speech: {message}",
    ),
])

# Create the chain with structured output
chain = prompt | llm.with_structured_output(HateSpeechAnalysis, include_raw=True)
```

Classification inference logic:



## Running the LLM chain with the whole dataset

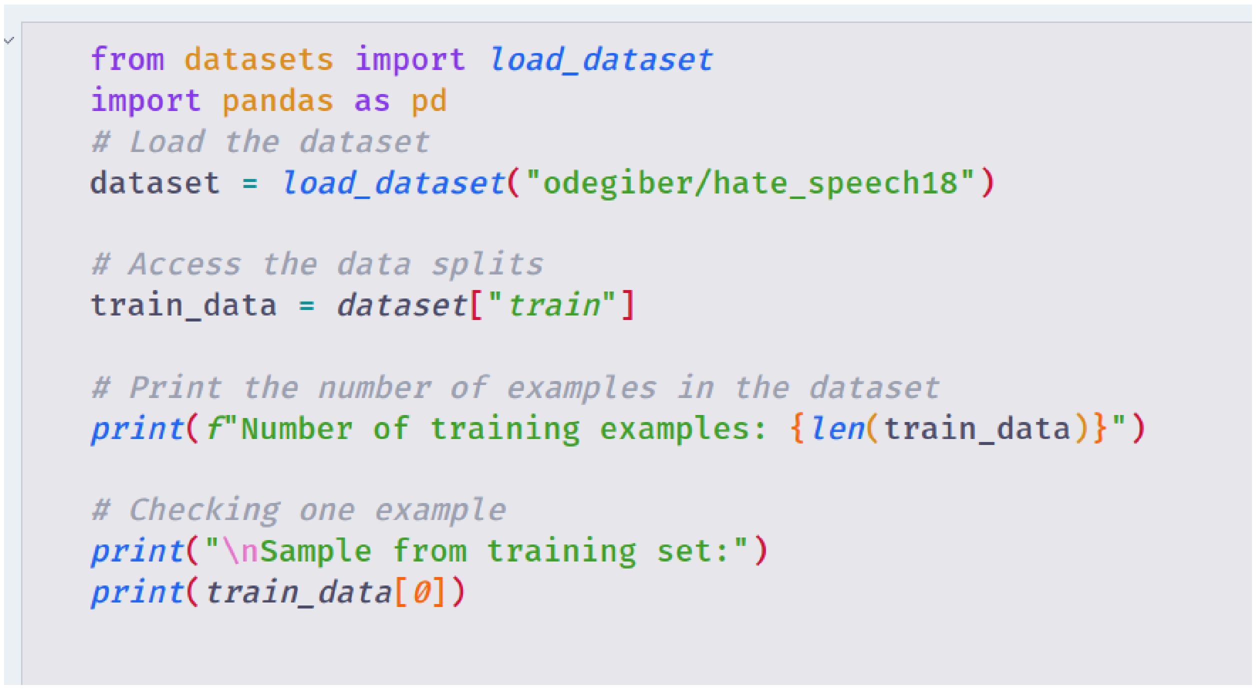
```python
# preprocessing the inputs
input_data = [
    {
        "normal_few_shot_examples": normal_few_shot_examples,
        "hate_few_shot_examples": hate_few_shot_examples,
        "message": msg
    }
    for msg in messages_to_evaluate["clean_message"]
]

classification_results = [None] * len(input_data) # list that will store the classification results
bad_idx_lst = [] # list of index that couldn´t be processed
for idx, input in tqdm(enumerate(input_data), total=len(input_data)):
    try:
        classification_results[idx] = chain.invoke(input)
    except:
        print(
            f"problem processing message with index {idx}. Adding a placeholder to the classification results"
        )
        bad_idx_lst.append(idx)
```

[22] Python

```
100%|██████████| 1142/1142 [30:06<00:00,  1.58s/it]
```

## 5 - Stormfront dataset preprocessing details

The dataset version present on hugging face contains 10944 training examples



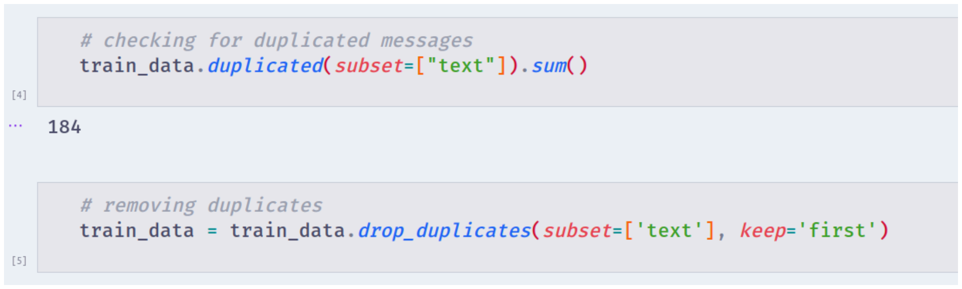
```python
from datasets import load_dataset
import pandas as pd
# Load the dataset
dataset = load_dataset("odegiber/hate_speech18")

# Access the data splits
train_data = dataset["train"]

# Print the number of examples in the dataset
print(f"Number of training examples: {len(train_data)}")

# Checking one example
print("\nSample from training set:")
print(train_data[0])
```

Result:
Number of training examples: 10944

Sample from training set:

{'text': 'As of March 13th , 2014 , the booklet had been downloaded over 18,300 times and counting .', 'user_id': 572066, 'subforum_id': 1346, 'num_contexts': 0, 'label': 0}

The dataset had 184 duplicated messages. Therefore I have removed them.

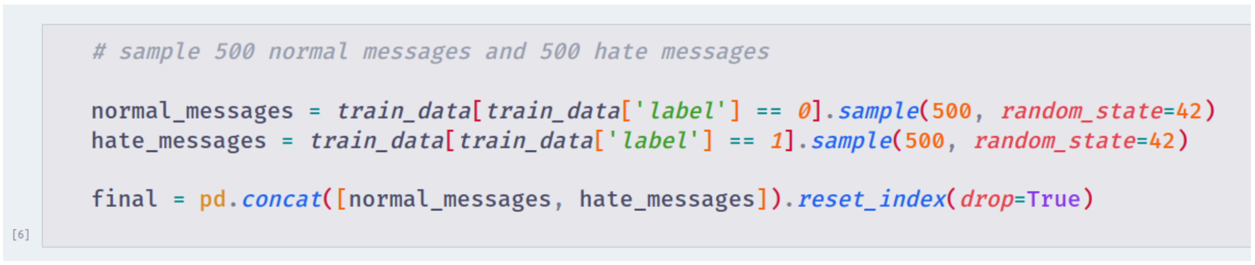


```python
[4]
# checking for duplicated messages
train_data.duplicated(subset=["text"]).sum()
```

```
184
```

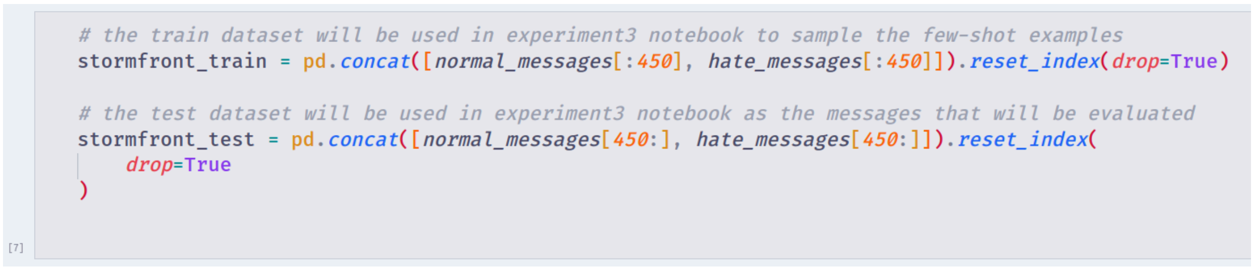
```python
[5]
# removing duplicates
train_data = train_data.drop_duplicates(subset=['text'], keep='first')
```

In order to perform the experiments and iterate faster, I have sampled 1000 messages, 500 normal messages and 500 hate messages:



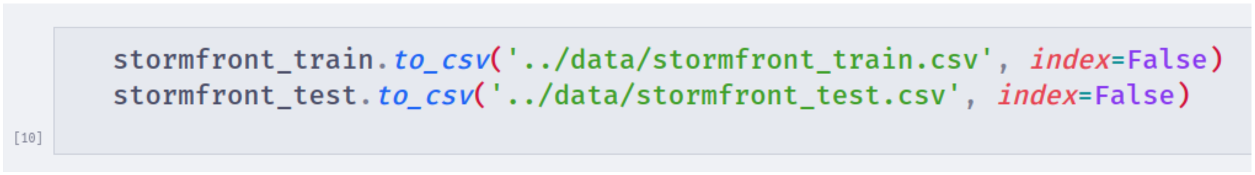
```
# sample 500 normal messages and 500 hate messages

normal_messages = train_data[train_data['label'] == 0].sample(500, random_state=42)
hate_messages = train_data[train_data['label'] == 1].sample(500, random_state=42)

final = pd.concat([normal_messages, hate_messages]).reset_index(drop=True)
[6]
```

Then out of these 1000 messages, 900 messages formed the training set and 100 formed the test set. The balance between normal and hate messages remained the same:



```
# the train dataset will be used in experiment3 notebook to sample the few-shot examples
stormfront_train = pd.concat([normal_messages[:450], hate_messages[:450]]).reset_index(drop=True)

# the test dataset will be used in experiment3 notebook as the messages that will be evaluated
stormfront_test = pd.concat([normal_messages[450:], hate_messages[450:]]).reset_index(
    drop=True
)
[7]
```

Finally, the resulting datasets were saved in the /data folder in the assignment's repository.



```
stormfront_train.to_csv('../data/stormfront_train.csv', index=False)
stormfront_test.to_csv('../data/stormfront_test.csv', index=False)
[10]
```

The full code for the stormfront dataset preprocessing steps can be found at notebooks/dataset_loading2.ipynb in the assignment's repository.