Hateful Meme Classification: Model Architecture and Analysis Report

Indian Institute of Information Technology, Allahabad

Bachelor of Technology
(Information Technology)

Prepared by:

1) Disha Soni	IIT2022260
2) Ankit Kumar	IIT2022256
3) Shreya Sinha	IIB2022034
4) Avantika Soni	IIB2022045
5) Taniha Butani	IIT2022207

Under Supervision Of:
Dr. Vrijendra Singh

Contents

1 Introduction

Hateful meme classification is a complex task requiring the integration of advanced Natural Language Processing (NLP) and computer vision techniques. This is because the meaning of a meme often comes from the way the text and image work together. On their own, the text or image might seem harmless, but when combined, they can create harmful or offensive messages. That's why it is very important to analyze both text and images at the same time. Ignoring either one can lead to missing key details and mistakes in identifying hateful content. By understanding both parts together, we can better detect and prevent harmful memes. This report details the architecture, training workflow, hyperparameters, and results obtained using various models, including RoBERTa, Qwen 2.5-5B-Instruct, BERT (PyTorch and TensorFlow), and Qwen 2 VL-7B-Instruct.

2 Our Approach

To classify hateful memes, we developed and tested multiple pipelines that combined text and image analysis using advanced models and techniques. Below is a detailed description of our pipelines:

Pipeline 1: Multi-Modal Feature Extraction and Analysis

1. **Text Extraction and Sentiment Analysis:**
   - We used **Qwen-2-VL-7B Instruct** to extract text from memes.
   - The extracted text was then passed through an **LSTM-based sentiment analysis model**, which was trained on the **ChatGPT sentiment analysis dataset**. This model assessed the hatefulness of the text.
2. **Object and Face Detection:**
   - Objects in the meme were detected using **YOLOv8**.
   - Faces in the memes were extracted using the **Haar Cascade Classifier**.
3. **Feature Extraction from Faces:**
   - Using **DeepFace**, we extracted additional features such as *gender, age, age group, emotion*, and *race* to provide more information about the meme's content.
4. **Combining Features:**
   - All the features (text sentiment, detected objects, facial attributes) were combined into a unified representation for further analysis.

Pipeline 2: Fine-Tuned Models for Hateful Meme Classification

1. **Dataset Preparation:**
   - The features and labels generated from Pipeline 1 were used to create a custom dataset for training advanced models.
2. **Fine-Tuned Models:**
   - We fine-tuned **BERT-based models** implemented in PyTorch and TensorFlow.
   - **RoBERTa** and **Qwen-2.5-5B Instruct** were also fine-tuned on this dataset for text and image analysis.
3. **Integrated Multimodal Analysis:**
   - These models combined text and image inputs, leveraging the custom dataset to classify hateful memes with enhanced accuracy.

Pipeline 3: Pre-Trained Model with Multimodal Inputs

- We utilized the **Qwen-2-VL-7B Instruct model** to process both the extracted text and image tags simultaneously.
- This pre-trained model directly provided predictions and explanations for the classification task.
- Due to computational limitations, we could not fine-tune this model further.

The combination of these pipelines allowed us to explore different approaches for hateful meme classification. Each pipeline had unique strengths, and together they provided valuable insights into the problem. Below, we present a detailed analysis of the models used and their contributions to the classification process.

3 Pre-trained Models used

3.1 YOLOv8 for Object Detection

YOLOv8 (You Only Look Once) is a state-of-the-art, real-time object detection model capable of identifying multiple objects within an image. In the context of meme analysis, YOLOv8 is used to detect various objects or components within the meme image, such as people, text, or any visual cues that are relevant for identifying hateful content.

- **How it Works**: YOLOv8 analyzes the input meme image by passing it through a convolutional neural network (CNN) that predicts bounding boxes around detected objects, along with class labels and confidence scores.
- **Pre-training**: YOLOv8 has been pre-trained on a large dataset containing various objects and is highly effective in real-time detection across diverse categories. This pre-trained knowledge allows the model to quickly identify and extract relevant objects from the meme image.
- **Application in Meme Analysis**: YOLOv8 identifies key objects in memes, such as people, text, or props, which may play an important role in detecting hateful or harmful content.

3.2 Haar Cascade for Face Detection

Haar Cascade Classifier is a machine learning object detection method used to identify faces within images. It works by training a classifier using positive and negative images to detect face-like patterns in a given image.

- **How it Works**: The Haar Cascade algorithm scans an image in multiple stages using a set of predefined Haar features to identify faces. Once the face is detected, it is marked with a bounding box.
- **Application in Meme Analysis**: In memes, detecting faces is crucial for analyzing emotions, gender, and age, as many memes rely on facial expressions and human subjects to convey messages.

3.3 DeepFace for Gender, Age, Emotion, and Race Classification

DeepFace is a deep learning framework for facial recognition and analysis, which uses pre-trained models to classify various aspects of human faces, including gender, age, emotion, and race.

- **How it Works**: DeepFace leverages a variety of pre-trained models (e.g., VGG-Face, Google FaceNet, OpenFace) that are trained to recognize facial features and predict attributes such as gender, age, emotion, and race based on facial landmarks.
- **Pre-training**: These models are pre-trained on large facial datasets, which allows them to generalize well to unseen faces and classify attributes accurately.
- **Application in Meme Analysis**: In the context of meme detection, DeepFace can classify facial attributes, which help determine the sentiment and potential bias in memes. For example, identifying the gender, age, emotion, and race of individuals in the meme can aid in detecting stereotypes or hateful content.

3.4 Overall Workflow for Meme Analysis

In meme analysis, a combination of these pre-trained models is used to extract relevant information from the meme:

- **YOLOv8** detects objects such as text, people, or props.
- **Haar Cascade** detects faces, which are critical for emotion and sentiment analysis.
- **DeepFace** analyzes facial attributes, such as gender, age, emotion, and race, to further assess the sentiment conveyed by the meme.

This multimodal approach provides a comprehensive analysis of memes by extracting both visual and emotional context, making it an effective method for detecting harmful or hateful content.

4 Sentiment Analysis Model

4.1 Text Preprocessing

The first step in preparing the text data involves cleaning and transforming the raw text. This includes the following steps:

- **Removing Punctuation**: A function is defined to remove all punctuation from the text. This is achieved using the `translate` method with `str.maketrans` to replace punctuation characters with an empty string.
- **Lowercasing**: All text is converted to lowercase to maintain consistency and avoid distinguishing between words due to case sensitivity.
- **Tokenization**: The text is tokenized into sequences of integers using a pre-defined tokenizer. This tokenizer converts each word in the text into a unique integer based on the vocabulary.
- **Calculating Word Statistics**: The total number of words across all sentences is computed, along with the number of unique words (vocabulary size).
- **Padding Sequences**: Sequences are padded to ensure uniform input size for the LSTM model. Padding is applied either to a fixed length or to the longest sequence in the dataset.

4.2 Neural Network Architecture

The model is a sequential neural network consisting of the following layers:

- **LSTM Layer 1**: The first LSTM layer has 256 units and returns sequences. The input shape is defined as (61, 1), indicating 61 time steps and 1 feature per step.
- **LSTM Layer 2**: The second LSTM layer has 128 units. It processes the output from the first LSTM layer.
- **Normalization Layer**: A normalization layer is added to standardize the output, helping to stabilize and speed up the training.
- **Dense Layer 1**: A dense layer with 64 units and ReLU activation is used for further processing of the features.
- **Dense Layer 2**: The final dense layer has 3 units and a softmax activation function to predict three output classes (e.g., sentiment labels).

4.3 Model Compilation

The model is compiled with the following settings:

- **Optimizer**: Adam optimizer is used, which adjusts the learning rate dynamically during training to improve convergence.
- **Loss Function**: Categorical cross-entropy loss is used for multi-class classification tasks, where the model predicts one of three classes.
- **Metrics**: Accuracy is used as the evaluation metric during training to track the performance of the model.

5 Model Architectures

5.1 RoBERTa Model

5.1.1 RoBERTa Model Overview

- **Description**: RoBERTa (*Robustly Optimized BERT*) is a transformer-based model pre-trained on large-scale text data for masked language modeling.
- **Architecture**: The model includes 24 transformer layers, each with 1024 hidden units and 16 attention heads, enabling it to capture complex contextual relationships in text.

- **Pre-training Dataset**: RoBERTa was pre-trained on 160 GB of uncompressed text, ensuring its capability to handle diverse language tasks.
- **Adaptation**: Fine-tuned for meme analysis, focusing on sentiment and hateful intent detection from combined textual and visual features.

5.1.2 Fine-Tuning Process

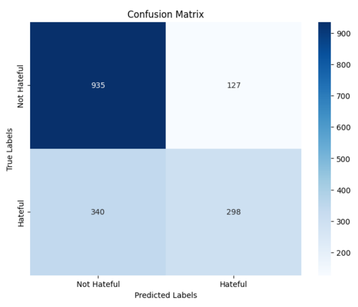
- **Dataset**:
  - Combined textual features such as text, objects, emotions, race, gender, age group, and sentiment analysis into a single input string.
  - Target labels (*hateful* = 1, *non-hateful* = 0) were used for binary classification.
- **Tokenizer**:
  - The `RobertaTokenizer` tokenized input text with truncation and padding up to a maximum of 512 tokens.
- **Hyperparameters**:
  - **Learning Rate**: 1×10^{-5}, optimized for fine-tuning without destabilizing pre-trained weights.
  - **Batch Size**:
    - Training: 32 samples per device.
    - Evaluation: 32 samples per device.
  - **Weight Decay**: 0.01, applied to reduce overfitting.
  - **Epochs**: 25 epochs were chosen to ensure model convergence while avoiding overtraining.
  - **Evaluation Strategy**: Performed evaluation at the end of each epoch.

5.1.3 Implementation and Results

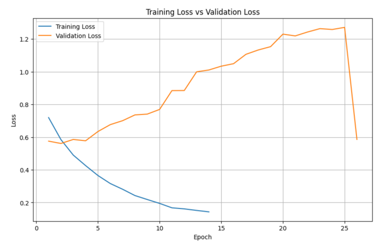
- The model was fine-tuned using the Hugging Face Trainer API, which facilitated efficient training and evaluation.

Table 1: Final and Evaluation Metrics for RoBERTa Fine-Tuning

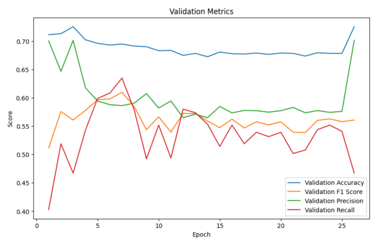
Metric	**Final Metrics**	**Evaluation Metrics**
Accuracy	67.82% (0.678235)	72.53% (0.725294)
F1 Score	0.5578 (0.557801)	0.5607 (0.560677)
Precision	0.5759 (0.575960)	0.7012 (0.701176)
Recall	0.5408 (0.540752)	0.4671 (0.467085)
Validation Loss	1.2714 (1.271412)	-
Evaluation Loss	-	0.5858 (0.585753)



(a) Confusion Matrix



(b) Validation Loss vs Accuracy Loss



(c) Validation Metrics

Figure 1: Visualization of model performance metrics.

5.2 Qwen 2.5-5B Instruct

5.2.1 Model Overview

Qwen (*Qwen2.5-0.5B-Instruct*) is a sequence classification model fine-tuned for binary classification tasks. It uses the Qwen architecture, which has been pre-trained for instruction-based tasks. The model is designed to work efficiently with transformer-based architectures and can be adapted to a variety of text classification tasks.

- **Architecture**: The Qwen2.5 model is based on the transformer architecture and is specifically fine-tuned for instruction-based sequence classification tasks.
- **Pre-training Dataset**: Qwen2.5 has been pre-trained on a large corpus of text data for instruction-following tasks, which helps it to understand a wide range of natural language inputs.

5.2.2 Fine-Tuning Process

Qwen2.5 was fine-tuned for meme classification to classify hateful and non-hateful content. The fine-tuning process involved using the Hugging Face library to load the model and tokenizer, along with custom tokenization, training, and evaluation strategies.

Dataset:

- The dataset consisted of text data from memes, labeled as either hateful or non-hateful:
  - Extracted text from memes.
  - Text labels (*hateful* $= 1$, *non-hateful* $= 0$) for binary classification.
- The dataset was split into training and validation sets.

Tokenizer:

- The `AutoTokenizer` was used to load the pre-trained tokenizer.
- If the tokenizer did not have a padding token, it was set to the model's EOS token to ensure proper padding.
- Tokenization was performed with truncation and padding applied to a maximum sequence length of 128 tokens.

Hyperparameters:

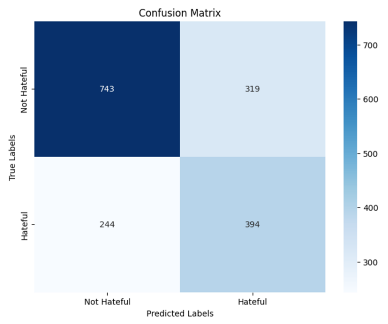
- **Learning Rate**: 5×10^{-5} to ensure stable optimization during fine-tuning.
- **Batch Size**:
  - Training: 16 samples per batch.
  - Evaluation: 16 samples per batch.
- **Weight Decay**: 0.01 to prevent overfitting.
- **Epochs**: 5 epochs to achieve convergence without overtraining.
- **Evaluation Strategy**: Evaluation was performed at the end of each epoch.

5.2.3 Implementation and Results

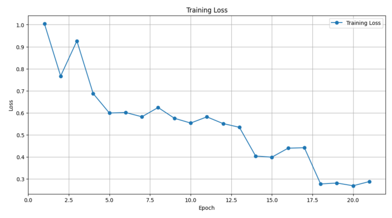
The fine-tuning process was implemented using the Hugging Face Trainer API, which simplifies training, evaluation, and logging.

Table 2: Evaluation Metrics for Qwen2.5 Fine-Tuning

Metric	**Evaluation Result**
Accuracy	66.88% (0.668824)
F1 Score	0.5833 (0.583272)
Precision	0.5526 (0.552595)
Recall	0.6176 (0.617555)
Validation Loss	0.8865 (0.886505)



(a) Confusion Matrix



(b) Training Loss

Figure 2: Visualization of model performance metrics.

5.3 BERT (PyTorch Implementation)

5.3.1 BERT Model Overview

- **Description**: BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based model pre-trained on large text corpora for tasks like masked language modeling (MLM) and next sentence prediction (NSP).
- **Framework**: The model was implemented using PyTorch, providing flexibility and customizability for training and fine-tuning.
- **Pre-training**: BERT was pre-trained on a large corpus of text, enabling it to understand contextual relationships within the language and perform a variety of NLP tasks.
- **Fine-tuning**: BERT was fine-tuned for binary classification tasks, where the model is trained to predict one of two classes (e.g., hateful or non-hateful memes).

5.3.2 Fine-Tuning Process

- **Dataset**:
  - The dataset includes textual features such as meme captions and associated labels (*hateful* $= 1$, *non-hateful* $= 0$) for binary classification.
  - Textual data is tokenized using the BERT tokenizer to convert the input text into numerical tokens compatible with the model.
- **Tokenizer**:
  - The `BertTokenizer` is used to tokenize input text, with truncation and padding applied to ensure each input is of a fixed length (max length of 128 tokens).
- **Hyperparameters**:
  - **Learning Rate**: The learning rate is set at 2×10^{-5}, optimized for fine-tuning BERT without destabilizing the pre-trained weights.

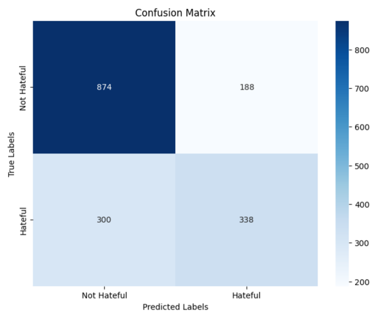
- **Batch Size**:
  - Training: 16 samples per device.
  - Evaluation: 16 samples per device.
- **Weight Decay**: 0.01, applied to avoid overfitting by penalizing large weights.
- **Epochs**: 10 epochs were chosen to ensure model convergence while preventing overfitting.

5.3.3 Implementation and Results

- **Tokenization and Dataset Creation**:
  - Text data is tokenized using the `BertTokenizer`, and a custom `TextDataset` class is defined to handle tokenization and label formatting for training and validation datasets.
- **Model Setup**:
  - The pre-trained `BertForSequenceClassification` model is loaded from the Hugging Face library, adapted for binary classification with two output labels.
- **Trainer API**:
  - The Hugging Face `Trainer` API is used to train and evaluate the model, with custom evaluation metrics and logging strategies.
  - The model is trained for 10 epochs, and the evaluation results, including accuracy, precision, recall, and F1 score, are logged after each epoch.

Table 3: Model Evaluation Metrics

Metric	**Final Metrics**	**Evaluation Metrics**
Accuracy	67.65% (0.676471)	71.29% (0.712941)
F1 Score	0.5557 (0.555735)	0.5808 (0.580756)
Precision	0.5733 (0.573333)	0.6426 (0.642586)
Recall	0.5392 (0.539185)	0.5298 (0.529781)
Validation Loss	1.6969 (1.696857)	-
Evaluation Loss	-	0.5685 (0.568508)



(a) Confusion Matrix

5.4 BERT (TensorFlow Implementation)

5.4.1 Preprocessor Layer

- **Preprocessing Model**: The KerasLayer from TensorFlow Hub is used to load a pre-trained BERT preprocessing model. It tokenizes the input text and generates the input word IDs, attention masks, and token type IDs required for BERT.
- **Preprocessor URL**: `https://kaggle.com/models/tensorflow/bert/frameworks/TensorFlow2/variations/en-uncased-preprocess/versions/3`.
- **Tokenization**: The input text is tokenized into a format that BERT can understand. Example input text: `["Hello, how are you?"]`.

5.4.2 Building the Model

- **Text Input Layer**: The model takes text input as a string using the `tf.keras.layers.Input` layer.
- **Preprocessing**: The text input is passed through the BERT preprocessing layer to generate the necessary input format for the BERT model.
- **BERT Encoder Layer**: The pre-trained BERT model is loaded from TensorFlow Hub and used as a KerasLayer. It outputs the sequence and pooled representations of the input text.
- **Dropout and Dense Layers**: Dropout layers are added after the encoder output for regularization. Two dense layers are added to further process the encoded output before passing it to the output layer.
- **Output Layer**: The final layer uses a sigmoid activation function for binary classification (e.g., hateful or non-hateful).

5.4.3 Model Compilation

- **Optimizer**: The model uses the Adam optimizer with a learning rate of 2×10^{-5}, ensuring stability during fine-tuning.
- **Loss Function**: Binary cross-entropy is used as the loss function for binary classification.
- **Metrics**: Binary accuracy is tracked during training.

5.4.4 Callbacks

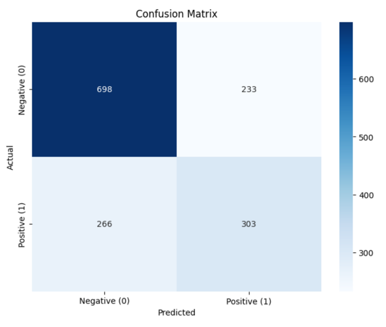
- **ModelCheckpoint**: Saves the model weights whenever validation accuracy improves.

- **EarlyStopping**: Stops training early if validation accuracy doesn't improve for 10 epochs, restoring the best weights.
- **ReduceLROnPlateau**: Reduces the learning rate by a factor of 0.5 if validation accuracy doesn't improve for 3 epochs.

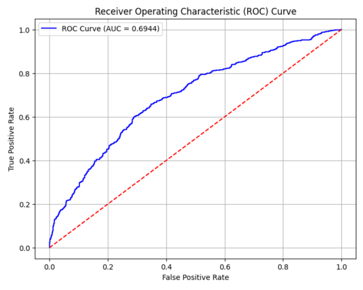
5.4.5 Training the Model

- The model is trained for 10 epochs with the training data `X_train` and validation data `X_valid`.
- The `fit` method is used with callbacks to monitor training and improve the model.

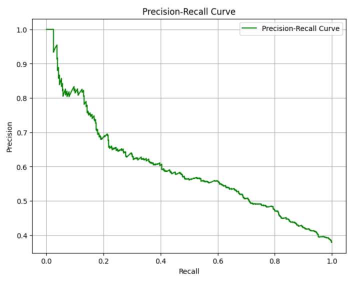
5.4.6 Implementation and Results



(a) Confusion Matrix

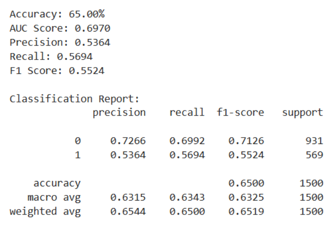


(b) ROC Curve



(c) Precision Recall curve

Figure 4: Visualization of model performance metrics.



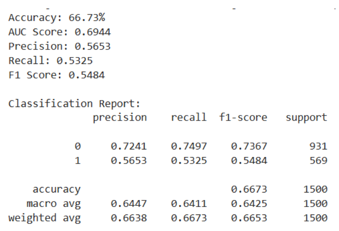
```
Accuracy: 65.00%
AUC Score: 0.6970
Precision: 0.5364
Recall: 0.5694
F1 Score: 0.5524

Classification Report:
              precision    recall  f1-score   support

           0     0.7266    0.6992    0.7126       931
           1     0.5364    0.5694    0.5524       569

    accuracy                         0.6500      1500
   macro avg     0.6315    0.6343    0.6325      1500
weighted avg     0.6544    0.6500    0.6519      1500
```

(a) Without Tags



```
Accuracy: 66.73%
AUC Score: 0.6944
Precision: 0.5653
Recall: 0.5325
F1 Score: 0.5484

Classification Report:
              precision    recall  f1-score   support

           0     0.7241    0.7497    0.7367       931
           1     0.5653    0.5325    0.5484       569

    accuracy                         0.6673      1500
   macro avg     0.6447    0.6411    0.6425      1500
weighted avg     0.6638    0.6673    0.6653      1500
```

(b) With Tags

Figure 5: Classification Report

Table 4: Model Evaluation Metrics

Metric	Without Tags	With Tags
Accuracy	65.00%	66.73%
AUC Score	0.6970	0.6944
Precision	0.5364	0.5653
Recall	0.5694	0.5325
F1 Score	0.5524	0.5484

Table 5: Classification Report for Model without Tags

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.7266	0.6992	0.7126	931
1	0.5364	0.5694	0.5524	569
Accuracy	0.6500			1500
Macro avg	0.6315	0.6343	0.6325	1500
Weighted avg	0.6544	0.6500	0.6519	1500

Table 6: Classification Report for Model with Tags

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.7241	0.7497	0.7367	931
1	0.5653	0.5325	0.5484	569
Accuracy	0.6673			1500
Macro avg	0.6447	0.6411	0.6425	1500
Weighted avg	0.6638	0.6673	0.6653	1500

5.5 Qwen 2 VL-7B Instruct

5.5.1 Model Overview

The Qwen2-VL-7B model is a pre-trained large multimodal model designed to handle both images and text. It is used here for the task of hateful meme detection. The model is loaded from Hugging Face's repository using the `from_pretrained` method, where both the model and the processor are initialized to handle the input format and output generation.

5.5.2 Model Loading

- **Model Initialization**: The model is loaded using the `Qwen2VLForConditionalGeneration` method, which loads the Qwen2-VL-7B model trained on various multimodal tasks.
- **Processor**: The processor is used to apply the required preprocessing and tokenization to the input data. It is initialized via `AutoProcessor.from_pretrained`.
- **Device Allocation**: The model is automatically placed on the GPU using the `device_map="auto"` argument, ensuring that the model performs inference on the available GPU.

5.5.3 Hateful Meme Detection Pipeline

The pipeline involves processing an image and a context prompt to predict whether a meme is hateful or non-hateful. The process is as follows:

- **Step 1: Image Loading**: The image corresponding to the meme is loaded using the `Image.open` method.
- **Step 2: Preparing the Prompt**: A textual prompt is created to provide context to the model, which includes the text related to the image.
- **Step 3: Preparing Model Inputs**: The textual and image inputs are formatted into a structure compatible with the model using the `processor.apply_chat_template` and `processor` methods.
- **Step 4: Model Inference**: The model is used to generate predictions based on the image and prompt. The output is processed and decoded to yield a textual prediction of whether the meme is hateful or non-hateful.
- **Step 5: Interpretation of Results**: The output text is analyzed to classify the meme as either **hateful** or **non-hateful**, along with reasons, depending on the presence of relevant keywords in the generated response.

5.5.4 Non-Fine-Tuned Model

Note: Due to computational limitations, the Qwen2-VL-7B model is used without fine-tuning in this application. Fine-tuning this model would require a much higher computational power, particularly more GPU resources.

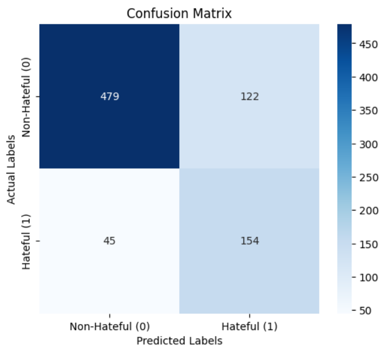
5.5.5 Why Fine-Tuning is Not Applied

- **High GPU Requirements**: Fine-tuning large models such as Qwen2-VL-7B typically requires significant GPU resources due to the large size of the model and the amount of training data required.
- **Computation Constraints**: The available computational infrastructure does not meet the needs for efficiently fine-tuning such large models, especially considering the time and memory demands.
- **Pre-trained Model Sufficiency**: Despite the lack of fine-tuning, the pre-trained Qwen2-VL-7B model is highly capable for this task due to its vast training on multimodal datasets, making it suitable for the classification task without the need for fine-tuning.

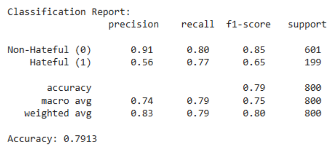
5.5.6 How It Works Without Fine-Tuning

The model works effectively for meme detection without fine-tuning because it has been pre-trained on diverse multimodal data that includes images and textual content. The pre-trained weights allow the model to generate relevant contextual information based on the input text and image, providing an accurate prediction of whether the meme is hateful or not. The key advantage here is that the model's training on a wide variety of data gives it general knowledge about handling different kinds of inputs, making it robust even without task-specific fine-tuning.

5.5.7 Implementation and Results



(a) Confusion Matrix



```
Classification Report:
                 precision    recall  f1-score   support

Non-Hateful (0)       0.91      0.80      0.85       601
    Hateful (1)       0.56      0.77      0.65       199

       accuracy                           0.79       800
      macro avg       0.74      0.79      0.75       800
   weighted avg       0.83      0.79      0.80       800

Accuracy: 0.7913
```

(b) Classification Report

Figure 6: Visualization of model performance metrics.

6 Results and Analysis

Model	**Accuracy**	**F1 Score**	**Precision**
RoBERTa	67.82%	0.5578	0.5758
Qwen 2.5-5B Instruct	66.88%	0.5833	0.5526
BERT (PyTorch)	67.65%	0.5557	0.5733
BERT (TensorFlow) (without tags)	65%	0.5524	0.5364
BERT (TensorFlow) (with tags)	66.73%	0.5484	0.5653
Qwen 2 VL-7B Instruct	*79.13%	*0.80	*0.83

Table 7: Model Performance Metrics

7 Conclusion

This report demonstrates the effectiveness of leveraging multi-modal instruction-tuned models for hateful meme classification. Future work will focus on optimizing these models further and exploring ensemble techniques.