Twitter-XLM-RoBERTa-Base for Multilingual Sentiment Analysis (Model Review and Demo Track) (Group #7)

Brandon Coutinho, Deepthi Sachidanand, and Lauren Mercer
New Jersey Institute of Technology, Newark, NJ 07102

(Dated: May 11, 2025)

Abstract

This paper presents a detailed analysis of XLM-T, a domain-adapted multilingual language model specifically designed for sentiment analysis on Twitter. By examining its architecture, technical innovations, and performance metrics, we demonstrate how XLM-T bridges the gap between general multilingual models and the noisy, emoji-rich environment of social media. Through continued masked language model pretraining on 198 million multilingual tweets, XLM-T adapts to the unique linguistic challenges of Twitter while maintaining cross-lingual transfer capabilities. We discuss key innovations including emoji-aware tokenization and adapter-based fine-tuning, providing implementation examples and comparative performance analysis across multiple languages.

GitHub Repository Link: GitHub Repository: Group_7_DL, Presentation Link: Video Presentation

Keywords: Natural Language Processing, Multilingual Models, Sentiment Analysis, Social Media Analysis, Transfer Learning.

I. INTRODUCTION

The proliferation of social media has created vast amounts of multilingual user-generated content, presenting unique challenges for sentiment analysis systems. Traditional multilingual models like XLM-R (Conneau et al., 2020) and mBERT (Devlin et al., 2019) struggle with informal Twitter content, which contains slang, emojis, hashtags, and code-switching between languages. XLM-T addresses these challenges through domain-specific continued pretraining on Twitter data (Barbieri et al., 2022). By combining the strong multilingual representations from XLM-R with robustness to Twitter's informal patterns, XLM-T significantly improves cross-lingual sentiment analysis in real-world social media settings. This paper examines the architecture, technical innovations, and empirical performance of XLM-T, providing insights for data science practitioners working with multilingual social media data.

II. BACKGROUND AND RELATED WORK

Multilingual Language Models

Recent advances in multilingual language models have enabled cross-lingual transfer learning, where a model trained on resources from high-resource languages can be applied to low-resource languages. Notable examples include:

- mBERT (Multilingual BERT): Trained on Wikipedia data from 104 languages using masked language modeling objectives (Devlin et al., 2019).
- XLM-R (XLM-RoBERTa): Trained on 2.5TB of general-domain web data spanning over 100 languages, achieving state-of-the-art results on cross-lingual benchmarks (Conneau et al., 2020).

Domain-Specific Language Models for Social Media

Several models have been developed specifically for social media text:

- BERTweet: A RoBERTa-based model pretrained on 850 million English tweets, achieving state-of-the-art results on English Twitter NLP tasks (Nguyen et al., 2020).
- AraBERT: A BERT-based model pre trained on Arabic text including social media content (Antoun et al., 2020).

However, most domain-specific models focus on a single language, limiting their applicability for multilingual social media analysis.

III. XLM-T MODEL ARCHITECTURE

XLM-T builds upon the XLM-RoBERTa (XLM-R) architecture, a large multilingual masked language model trained on web data in more than 100 languages. The model retains XLM-R's transformer-based architecture while incorporating several domain-specific adaptations for Twitter content.
XLM-T uses XLM-R as its foundation, leveraging its 550 million parameters organized in a transformer architecture with:

- 24 transformer layers
- 16 attention heads
- 1024-dimensional hidden states
- 250,000 vocabulary tokens

Technical Innovations of XLM-T

Domain-Specific Training on Twitter Data

XLM-R was trained on general-domain web data (e.g., news, Wikipedia). XLM-T continues training XLM-R on 198 million real tweets in 30+ languages. The training uses a masked language modeling (MLM) objective, where 15% of tokens are randomly masked, and the model is trained to predict these masked tokens. This helps the model learn Twitter-specific lexical patterns, slang expressions, and emoji usage across languages.

Multilingual Tweet Understanding

XLM-T was explicitly fine-tuned using the Unified Multilingual Sentiment Analysis Benchmark (UMSAB). The benchmark includes balanced, labeled sentiment tweets in 8 languages, improving cross-lingual performance.Result: XLM-T excels in zero-shot and cross-lingual sentiment classification.

Emoji and Slang Representation

Emojis are a powerful signal in social media sentiment. XLM-T treats emojis as semantically meaningful tokens, improving classification accuracy.

Adapter-Based Fine-Tuning

XLM-T uses adapter modules (lightweight layers) for task-specific tuning. These adapters are trained on sentiment tasks while keeping the base model frozen, increasing the efficiency of fine-tuning, faster convergence, better domain focus.

IV. XLM-T FINE-TUNING APPROACH

Fine tuning is the process of adapting a pre-trained language model (like XLM-R or BERT) to a specific downstream task, such as sentiment classification, by continuing training on labeled data relevant to that task. XLM-T fine-tunes using adapters for efficient multilingual sentiment analysis on Twitter. Only the adapter layers and the final classification head are trained. This setup is modular, faster, and yields better performance on social media data than general-domain models.

Pre-trained Model Initialization

This pretraining step makes XLM-T better suited for informal, emoji-filled, and multilingual Twitter content.

- Base model: XLM-RoBERTa
- Checkpoints from Hugging Face: xlm-roberta-base
- Continued pretraining was done on 198 million tweets across 30+ languages to create XLM-T

Fine Tuning strategy - Add Adapter Layers

Instead of tuning the entire XLM-R model:

- Insert lightweight adapter modules inside each transformer block.
- The main LM weights are frozen (not updated during training).
- Only the small adapter layers and final classification layer are trained.

Using adapters drastically reduces memory and compute requirements, allows fast task switching (just swap adapters) and encourages modular, reusable components.

Add a Classification Head

A dense layer is added on top of the adapter output to classify tweets into: Positive, Neutral and Negative.

A custom benchmark (Unified Multilingual Sentiment Analysis Benchmark (UMSAB) was built from standardized sentiment datasets across 8 languages. Preprocessing ensured equal distribution of labels and balanced datasets.

Training Configuration

- Optimizer: AdamW (weight decay for regularization)
- Learning rate: 1e-4 or 1e-3
- Batch size: 16–32
- Epochs: 3–5
- Loss function: Cross-Entropy Loss
- Evaluation metric: Macro-F1 Score (due to class imbalance)

Cultural Aspects Handled in the Model

While XLM-T doesn't explicitly encode cultural knowledge, it implicitly captures cultural and linguistic nuances by:

- Using Twitter data directly, which naturally contains slang, emoji, and idioms

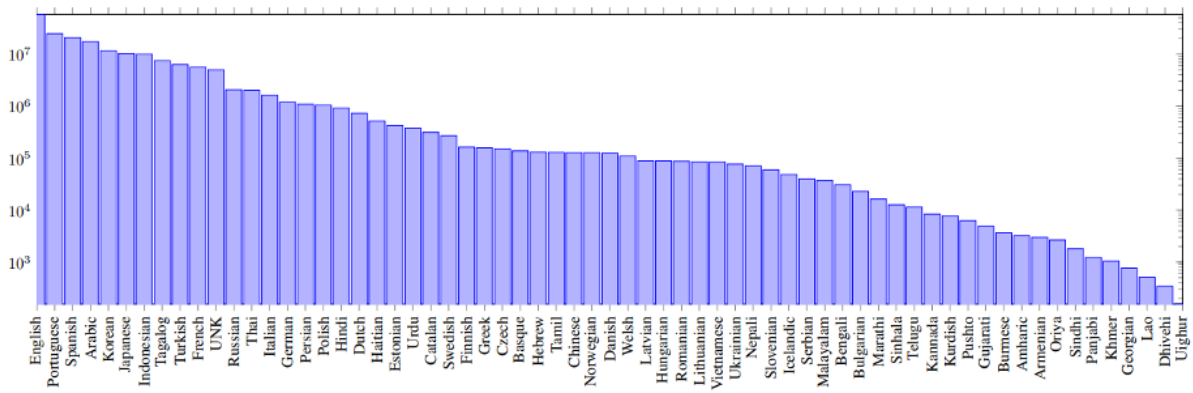


Figure 1: Distribution of languages of the 198M tweets used to finetune the Twitter-based language model (log scale). UNK corresponds to unidentified tweets according to the Twitter API.

FIG. 1. Distribution of languages in dataset used for finetuning XLM-T model from original paper.

Lang.	Dataset	Time-Train	Time-Test
Arabic	SemEval-17 (Rosenthal et al., 2017)	09/16-11/16	12/16-1/17
English	SemEval-17 (Rosenthal et al., 2017)	01/12-12/15	12/16-1/17
French	Deft-17 (Benamara et al., 2017)	2014-2016	Same
German	SB-10K (Cieliebak et al., 2017)	8/13-10/13	Same
Hindi	SAIL 2015 (Patra et al., 2015)	NA,3-month	Same
Italian	Sentipolc-16 (Barbieri et al., 2016)	2013-2016	2016
Portug.	SentiBR (Brum and Nunes, 2017)	1/17-7/17	Same
Spanish	Intertass (Díaz-Galiano et al., 2018)	7/16-01/17	Same

FIG. 2. Languages Dataset Train and Test Times

- Balanced label distributions across languages to prevent bias
- Diverse language families: Indo-European (English, Spanish, German), Semitic (Arabic), Dravidian (Hindi group), etc.
- Arabic and English tweets had high cross-lingual embedding similarity, due to shared hashtags and trending topics (e.g., iPhone, vegetarianism).
- Hindi embeddings were the most dissimilar to English, likely due to script and structural differences.
- Emojis and platform-specific tokens play a significant role in encoding tweet semantics.

Zero-Shot Cross-Lingual Transfer

This is the ability of a model to perform a task in a target language without having seen any labeled data in that language during training. Subword tokenization (SentencePiece) ensures that semantically similar words in different languages map to similar vector representations. Even if a word or phrase wasn't seen in the target language, the model can still understand its meaning based on context and its shared semantic space.

Evaluation

- Monolingual tasks using TweetEval (emoji, irony, hate speech, etc.). Trained and tested in the same language.
- Multilingual sentiment tasks from UMSAB. Trained on all 8 languages together
- Bilingual tasks - trained in one language, fine-tuned on another language.

Fine Tuning Code in XLM-T

- Import Required Libraries to load HuggingFace modules for tokenizing input text, loading a pretrained model, training and evaluation setup, loading datasets (like CSV files or Hugging Face-hosted datasets)
- Load the XLM-T Tokenizer and Model - This is the pretrained base model fine-tuned on 198M multilingual tweets. Using a domain-specific model gives you better performance than general-purpose XLM-R.
- Load and Tokenize the Dataset
- Set Training Hyperparameters
- Set Evaluation Metric
- Train the Model - Trainer handles training loops, validation, logging, model saving, etc.It fine-tunes the XLM-T model specifically on the data, improving its accuracy for sentiment classification on Twitter.

	XLM-R									XLM-Twitter								
	Ar	**En**	**Fr**	**De**	**Hi**	**It**	**Pt**	**Es**	***All-1***	**Ar**	**En**	**Fr**	**De**	**Hi**	**It**	**Pt**	**Es**	***All-1***
Ar	63.6	**64.1**	54.4	53.9	22.9	57.4	62.4	62.2	*59.2*	67.7	**66.6**	62.1	59.3	46.3	63.0	60.1	65.3	*64.3*
En	64.2	68.2	61.6	63.5	23.7	**68.1**	65.9	67.8	*68.2*	64.0	66.9	60.6	67.8	35.2	67.7	61.6	**68.7**	*70.3*
Fr	45.4	52.1	72.0	36.5	16.7	43.3	40.8	**56.7**	*53.6*	47.7	**59.2**	68.2	38.7	20.9	45.1	38.6	52.5	*50.0*
De	43.5	**64.4**	55.2	73.6	21.5	60.8	60.1	62.0	*63.6*	46.5	65.0	56.4	76.1	36.9	**66.3**	65.1	65.8	*65.9*
Hi	48.2	52.7	43.6	47.6	36.6	**54.4**	51.6	51.7	*49.9*	50.0	55.5	51.5	44.4	40.3	**56.1**	51.2	49.5	*57.8*
It	48.8	65.7	63.9	**66.9**	22.1	71.5	63.1	58.9	*65.7*	41.9	59.6	60.8	64.5	24.6	70.9	**64.7**	55.1	*65.2*
Pt	41.5	63.2	57.9	59.7	26.5	59.6	67.1	**65.0**	*65.0*	56.4	**67.7**	62.8	64.4	26.0	67.1	76.0	64.0	*71.4*
Es	47.1	63.1	56.8	57.2	26.2	57.6	**63.1**	65.9	*63.0*	52.9	66.0	64.5	58.7	30.7	62.4	**67.9**	68.5	*66.2*

FIG. 3. Zero-shot cross-lingual sentiment analysis results from the original paper.

	Monolingual			Bilingual		Multilingual	
	FT	**XLM-R**	**XLM-T**	**XLM-R**	**XLM-T**	**XLM-R**	**XLM-T**
Ar	45.98	63.56	**67.67**	63.63 (En)	67.65 (En)	64.31	66.89
En	50.85	68.18	66.89	65.07 (It)	67.47 (Es)	68.52	**70.63**
Fr	54.82	71.98	68.19	**73.55** (Sp)	68.24 (En)	70.52	71.18
De	59.56	73.61	76.13	72.48 (En)	75.49 (It)	72.84	**77.35**
Hi	37.08	36.60	40.29	33.57 (It)	55.35 (It)	53.39	**56.39**
It	54.65	71.47	70.91	70.43 (Ge)	**73.50** (Pt)	68.62	69.06
Pt	55.05	67.11	75.98	71.87 (Sp)	**76.08** (En)	69.79	75.42
Sp	50.06	65.87	68.52	67.68 (Po)	**68.68** (Pt)	66.03	67.91
All	51.01	64.80	66.82	64.78	69.06	66.75	**69.35**

FIG. 4. Crosslingual sentiment analysis F1 results on target languages from the paper

```
MODEL =  "cardiffnlp/twitter-xlm-roberta-base"
tokenizer = AutoTokenizer.from_pretrained(MODEL)
model = AutoModel.from_pretrained(MODEL)
```

FIG. 5. Initializing tokenizer and model from the pretrained cardiffnlp/twitter-xlm-roberta-base model from HuggingFace.

```
training_args = TrainingArguments(
    output_dir='./results',                  # output directory
    num_train_epochs=EPOCHS,                 # total number of training epochs
    per_device_train_batch_size=BATCH_SIZE,  # batch size per device during training
    per_device_eval_batch_size=BATCH_SIZE,   # batch size for evaluation
    warmup_steps=100,                        # number of warmup steps for lr scheduler
    weight_decay=0.01,                       # strength of weight decay
    logging_dir='./logs',                    # directory for storing logs
    logging_steps=10,                        # when to print log
    load_best_model_at_end=True,             # load or not best model at the end
)
```

FIG. 6. Training arguments for finetuning XLM-T model

```
trainer = Trainer(
    model=model,                         # the instantiated 🤗 Transformers model
    args=training_args,                  # training arguments, defined above
    train_dataset=train_dataset,         # training dataset
    eval_dataset=val_dataset             # evaluation dataset
)

trainer.train()
trainer.save_model("./results/best_model") # save best model
```

FIG. 7. Trainer for finetuning XLM-T model using the Training arguments

V. MODEL EVALUATION AND PERFORMANCE ANALYSIS

This section discusses in detail the evaluation and performance of the XLM-T model. We explore 5 notebooks here in the context of the XLM-T model:

- Running a classifier on a text file
- Extracting embeddings from a text file
- Experimenting with some use cases we have focused on in our project
- Application to TweetEval dataset with the pre-fine-tuned model
- Application to TweetEval dataset with full fine-tuning and evaluation pipeline.

In each of these notebooks, we apply a custom preprocessing function to normalize the tweets by replacing the noise with placeholders. Any mentions (i.e., tokens beginning with "@") are replaced with @user, and any URLs (i.e., tokens beginning with "http") are replaced with http (Figure 8). By stripping off the extraneous characters, the model is able to focus on the actual linguistic content of the tweet rather than the noisy metadata such as the handles and links, and thus generalizes better.

```
def preprocess(corpus):
  outcorpus = []
  for text in corpus:
    new_text = []
    for t in text.split(" "):
        t = '@user' if t.startswith('@') and len(t) > 1 else t
        t = 'http' if t.startswith('http') else t
        new_text.append(t)
    new_text = " ".join(new_text)
    outcorpus.append(new_text)
  return outcorpus
```

FIG. 8. Custom preprocessing function for normalizing tweets by removing mentions and links.

In all notebooks, we load the multilingual XLM-T model and its tokenizer for feature extraction as shown in the Figure 9.

```
CUDA = True # set to true if using GPU (Runtime -> Change runtime Type -> GPU)
BATCH_SIZE = 32
MODEL = "cardiffnlp/twitter-xlm-roberta-base-sentiment"
tokenizer = AutoTokenizer.from_pretrained(MODEL, use_fast=True)
config = AutoConfig.from_pretrained(MODEL) # used for id to label name
model = AutoModelForSequenceClassification.from_pretrained(MODEL)
if CUDA:
  model = model.to('cuda')
_ = model.eval()
```

FIG. 9. Loading model, tokenizer, config for feature extraction using the pretrained XLM-T model from HuggingFace

Running a classifier on a text file

We begin by importing the core classes from the HuggingFace Transformers library for AutoTokenizer, AutoModelForSequenceClassification, and AutoConfig; the PyTorch data loader, NumPy, and the SciPy softmax function. The model, its configuration, and tokenizer are loaded from the pretrained "cardiffnlp/twitter-xlm-roberta-base-sentiment" model from HuggingFace. Since we used the T4 GPU on Google Colab, a CUDA-enabled GPU is available for use, so the model is sent to the GPU.

For this test, we downloaded a sample eight-language collection from the XLM-T GitHub repository (cardiffnlp/xlm-t) and stored it in our local environment. The file is read into a list, and loaded using the DataLoader in batches of 32 tweets to allow iteration over minibatches. A forward function is defined, which wraps tokenization (with padding and truncation) and model inference. For model inference, the inputs are converted to PyTorch tensors, moved to GPU, and passed through the sequence-classification head to produce raw logits, which are then converted into probabilities via softmax.

Finally, each batch is preprocessed with the custom preprocessor shown in Figure 8. The preprocessed text is passed to the forward function and the class probabilities are computed. Note that there are 3 sentiment classes present in the dataset — negative, neutral and positive. The argmax over the computed sentiment class probabilities is taken, and these predictions are appended to the main predictions list for all batches. After completing iterations over all batches, example tweets are selected from each language and printed out alongside their predicted labels (retrieved from the pretrained model config via config.id2label), demonstrating end-to-end multilingual sentiment inference.

Extracting embeddings from a text file

We load the model, config, and tokenizer from the pretrained "cardiffnlp/twitter-xlm-roberta-base" model from HuggingFace, and send the model to the GPU, similar to the notebook above. However, note the slight difference in the model name from the first notebook: "cardiffnlp/twitter-xlm-roberta-base" is used here, which is the XLM-R language model trained on 200M tweets for 30+ languages, while the "cardiffnlp/twitter-xlm-roberta-base-sentiment" is the same XLM model, but is fine-tuned on the UMSAB multilingual sentiment analysis dataset.

The core "encode" function (Figure 10) accepts a batch of texts, applies preprocessing, tokenizes them (with padding and truncation), and feeds them through the transformer to obtain hidden-state embeddings. Subsequently, these per-token embeddings are reduced to a single fixed-size vector per example by taking the maximum over the sequence length (i.e., max-pooling). Finally, the

```
def encode(text, cuda=True):
  text = preprocess(text)
  encoded_input = tokenizer(text, return_tensors='pt', padding=True,
                            truncation=True)
  if cuda:
    encoded_input.to('cuda')
    output = model(**encoded_input)
    embeddings = output[0].detach().cpu().numpy()
  else:
    output = model(**encoded_input)
    embeddings = output[0].detach().numpy()

  embeddings = np.max(embeddings, axis=1)
  #embeddings = np.mean(embeddings, axis=1)
  return embeddings
```

FIG. 10. Encode function used for extracting embeddings.

script iterates over the entire dataset in batches, computing and storing all 768-dimensional embeddings. It then normalizes these embeddings to unit length and computes a full cosine similarity matrix. A helper function retrieves the most and least similar tweets to a given query tweet by sorting similarity scores. The query tweet is printed, and its top-10 most and least similar counterparts (prefixed with the similarity scores) from a specified slice of the dataset are also printed, in order to illustrate how semantic similarity can be explored in the multilingual embedding space.

Experimenting with some use cases

Similar to the first notebook, we load the model, config, and tokenizer from HuggingFace, and send the model to the GPU. For the first experiment, the pretrained "cardiffnlp/twitter-xlm-roberta-base" model is used, and for the second and third experiments, we utilize the "cardiffnlp/twitter-xlm-roberta-base-sentiment" model.

For semantic similarity, each tweet is tokenized and passed through the XLM-T encoder. The token-wise hidden states are mean-pooled to produce a fixed 768-dimensional embedding. The 768-dimensional size of the embeddings comes directly from the underlying XLM-RoBERTa "base" architecture on which XLM-T is built. In the Hugging Face Transformers implementation, the XLMRobertaConfig class, which defines the model's hyperparameters, specifies hidden_size = 768, meaning that each token is represented by a 768-element vector in the model's hidden layers. Because XLM-T is simply XLM-RoBERTa further pre-trained on Twitter data, it inherits this same hidden dimensionality.

Three main use cases are demonstrated here:

- **Computing tweet similarity**: given a Spanish query tweet, we embed it along with a tweet in a small multilingual list (English, Dutch, French, Italian). Then we compute the cosine similarity between them and rank the closest tweets. This allows the model's cross-lingual embedding space to be evaluated qualitatively, and illustrates the cross-language semantic alignment.
- **Simple sentiment inference via the pipeline API**: we use the fine-tuned "cardiffnlp/twitter-xlm-roberta-base-sentiment" model to classify a sample tweet. This highlights how easy it is to utilize the Transformers pipeline essentially making it a zero-coding inference.
- **Full-scale experiments on the UMSAB (Universal Multilingual Sentiment Analysis Benchmark) Spanish subset**: we fetch the raw train/validation/test data splits directly from the XLM-T GitHub repository (cardiffnlp/xlm-t) via HTTP. We apply the preprocessing function in batches of 32 tweets and pass them through a sequence-classification head (AutoModelForSequenceClassification). We then apply a softmax to logits, and finally evaluate with scikit-learn's classification_report (Figure 11).

```
print(classification_report(dataset['test_labels'], all_preds))
```

```
              precision    recall  f1-score   support

           0       0.70      0.80      0.75       290
           1       0.62      0.55      0.58       290
           2       0.75      0.74      0.75       290

    accuracy                           0.70       870
   macro avg       0.69      0.70      0.69       870
weighted avg       0.69      0.70      0.69       870
```

FIG. 11. Classification Report from full scale experiments on the UMSAB Spanish subset

Application to TweetEval dataset with the pre-fine-tuned model

Similar to the above notebooks, we load the pre-trained "cardiffnlp/twitter-xlm-roberta-base-sentiment" model and tokenizer from HuggingFace, and send the model to the GPU. The model is then to evaluation mode to disable dropout.

The TweetEval dataset is fetched through the Hugging Face Datasets API, selecting the "sentiment" split from it. This yields the train/validation/test texts and their corresponding labels. As we are not retraining/fine-tuning here, we do not require the train and validation subsets, so we focus on the test text and test labels (Figure 12).

```
# 2. Load TweetEval sentiment dataset
ds = load_dataset("cardiffnlp/tweet_eval", "sentiment")
texts = ds["test"]["text"]
labels = ds["test"]["label"]
```

FIG. 12. Loading TweetEval dataset sentiment partition and only using test subset (texts/labels)

```
# 6. Report results
print(f"Accuracy: {accuracy:.4f}")
print(f"Macro-F1: {macro_f1:.4f}\n")
print("Precision, Recall, F1-score, Support per class:")
for cls, p, r, f, s in zip(
    ["negative", "neutral", "positive"], precision, recall, f1, support
):
    print(f"{cls.capitalize():<9}  Precision: {p:.4f}, Recall: {r:.4f},"+
          f" F1: {f:.4f}, Support: {s}")

print("\nFull classification report:")
print(classification_report(labels, preds, target_names=["negative", "neutral",
                                                         "positive"]))
```

```
Accuracy: 0.6814
Macro-F1: 0.6839

Precision, Recall, F1-score, Support per class:
Negative   Precision: 0.6137, Recall: 0.8703, F1: 0.7198, Support: 3972
Neutral    Precision: 0.7714, Recall: 0.5467, F1: 0.6399, Support: 5937
Positive   Precision: 0.6824, Recall: 0.7019, F1: 0.6920, Support: 2375

Full classification report:
              precision    recall  f1-score   support

    negative       0.61      0.87      0.72      3972
     neutral       0.77      0.55      0.64      5937
    positive       0.68      0.70      0.69      2375

    accuracy                           0.68     12284
   macro avg       0.69      0.71      0.68     12284
weighted avg       0.70      0.68      0.68     12284
```

FIG. 13. Classification Report for pre-fine-tuned XLM-T applied on TweetEval dataset.

Each tweet is preprocessed with the simple normalization routine shown in Figure 8. Preprocessed texts in batches of 32 tweets are tokenized with padding and truncation. Under PyTorch's no_grad() context, these batches are forwarded through the model to obtain raw logits, which are then converted into class predictions by taking the index of the maximal logit (argmax).

Finally, the standard classification metrics such as accuracy, per-class precision, recall and F1-score are computed with scikit-learn and printed along with the classification_report (Figure 13). This end-to-end pipeline encapsulates dataset loading, preprocessing, batched inference, and evaluation, demonstrating how the XLM-T model performs on a multilingual sentiment task with minimal custom code.

Application to TweetEval dataset with full fine-tuning and evaluation pipeline

As in the previous notebooks, we load the pre-trained "cardiffnlp/twitter-xlm-roberta-base-sentiment" model and tokenizer from HuggingFace, and send the model to the GPU. Here, we perform a full fine-tuning of the model on the TweetEval sentiment subset. We also configure the model hyperparameters — batch size, learning rate, epochs.

The TweetEval dataset is fetched through the Hugging

Face Datasets API, selecting the "sentiment" split from it. This yields the train/validation/test texts and their corresponding labels. As we are fine-tuning in this case, we retain and utilize all three subsets (texts/labels).

Tweets are preprocessed with a map transformation that normalizes them in a similar manner as all the notebooks above. The entire dataset is then tokenized in batches with truncation. A DataCollatorWithPadding ensures uniform tensor shapes for each batch. Custom metrics (accuracy, precision, recall, macro-F1) are loaded via the evaluate library in the compute_metrics callback function, thereby tracking the evaluation at the end of each epoch.

The training is managed by the Transformers Trainer function, with early stopping (patience=2) based on macro-F1, mixed-precision (fp16) enabled for efficiency on T4 GPUs, and best-model checkpointing. After completing the training, the fine-tuned model is evaluated on the test split, printing overall metrics and detailed per-class scores via scikit-learn's classification_report (Figure 14), thus closing the loop on this end-to-end multilingual sentiment fine-tuning experiment. As we can see

```
# 12. Detailed Classification Report
print("\nDetailed classification report:")
print(classification_report(labels, preds, target_names=["negative","neutral",
                                                         "positive"]))
```

```
Detailed classification report:
              precision    recall  f1-score   support

    negative       0.70      0.77      0.73      3972
     neutral       0.73      0.64      0.68      5937
    positive       0.65      0.75      0.70      2375

    accuracy                           0.70     12284
   macro avg       0.69      0.72      0.70     12284
weighted avg       0.71      0.70      0.70     12284
```

FIG. 14. Classification report after full fine-tuning of XLM-T applied on TweetEval dataset.

here, comparing the pre- and post-fine-tuned results in Figures 13 and 14 respectively, we see there is an improvement in the overall performance of the model post-fine-tuning. These results also show an improvement over the performance of some of the other models listed in the paper for the sentiment category (Figure 15).

VI. MODEL DEMO

This section describes how a Google Colab notebook was used to build a working demo for multilingual sentiment analysis using a pre-trained transformer model. The notebook combines machine learning and a web-based interface, letting users enter tweets and get real-time predictions on their sentiment. It's a hands-on example of how natural language processing tools can be applied interactively using Python.The notebook has two main goals. First, it lets users classify the sentiment of tweets—positive, negative, or neutral—using a model that works across languages. Second, it goes deeper by computing sentence embeddings and comparing tweet similarity across different languages. These two features highlight how the same model can be used for both classification and more advanced text analysis.

Sentiment Analysis with Pre-trained Model

The notebook starts by installing the three main Python libraries needed for the notebook: transformers, torch, and gradio. (Figure 16) Since Google Colab sessions reset frequently, it's necessary to reinstall these packages each time the notebook is reopened. The transformers library is used to load the pre-trained XLM-RoBERTa model from Hugging Face for sentiment analysis. torch is the deep learning framework that powers the model and handles the computations. Gradio is used to create the interactive web interface that lets users input tweets and get instant sentiment predictions. Running this command ensures that all the required tools are available before the rest of the code is executed.

We install Hugging Face's transformers library which is the go-to library for working with state-of-the-art NLP models. By importing its pipeline function, we get a one-line interface that handles tokenization, model loading, and inference. (Figure 17) In this case, calling pipeline, automatically downloads the XLM-RoBERTa checkpoint and tokenizer from the cardiffnlp/twitter-xlm-roberta-base-sentiment hub, prepares incoming text for the model, runs the text through the network, and returns both labels and confidence scores. Without transformers, you'd have to write dozens of lines to load the checkpoint, convert your text into input IDs, create attention masks, run the model, and interpret its raw outputs yourself.

A few sample tweets are passed into the model, and it returns sentiment labels along with confidence scores. For example, a tweet praising a new app might be labeled "positive" with 93% confidence, while one describing a poor experience could be labeled “negative” with 95 % confidence. These examples show how quickly the model can identify tone and intent with minimal setup. (Figure 18)

Interactive Sentiment Analysis with Gradio

To make the demo more accessible, Gradio is used to create an interactive web interface. A simple function processes user input and returns the sentiment label and confidence score. This function is linked to a Gradio app with a text box for input and an output area for results.

When the app runs, users can enter any tweet and get an instant prediction. In Colab, demo.launch() creates a temporary public link that can be shared, though it expires after 72 hours unless hosted through another platform like Hugging Face Spaces. This interactive section

	Emoji	Emotion	Hate	Irony	Offensive	Sentiment	Stance	ALL
SVM	29.3	64.7	36.7	61.7	52.3	62.9	67.3	53.5
FastText	25.8	65.2	50.6	63.1	73.4	62.9	65.4	58.1
BLSTM	24.7	66.0	52.6	62.8	71.7	58.3	59.4	56.5
RoB-Bs	30.9±0.2 (30.8)	76.1±0.5 (76.6)	46.6±2.5 (44.9)	59.7±5.0 (55.2)	79.5±0.7 (78.7)	71.3±1.1 (72.0)	68±0.8 (70.9)	61.3
RoB-RT	31.4±0.4 (**31.6**)	78.5±1.2 (**79.8**)	52.3±0.2 (**55.5**)	61.7±0.6 (62.5)	80.5±1.4 (**81.6**)	72.6±0.4 (**72.9**)	69.3±1.1 (**72.6**)	**65.2**
RoB-Tw	29.3±0.4 (29.5)	72.0±0.9 (71.7)	46.9±2.9 (45.1)	65.4±3.1 (65.1)	77.1±1.3 (78.6)	69.1±1.2 (69.3)	66.7±1.0 (67.9)	61.0
XLM-R	28.6±0.7 (27.7)	72.3±3.6 (68.5)	44.4±0.7 (43.9)	57.4±4.7 (54.2)	75.7±1.9 (73.6)	68.6±1.2 (69.6)	65.4±0.8 (66.0)	57.6
XLM-Tw	30.9±0.5 (30.8)	77.0±1.5 (78.3)	50.8±0.6 (51.5)	69.9±1.0 (**70.0**)	79.9±0.8 (79.3)	72.3±0.2 (72.3)	67.1±1.4 (68.7)	64.4
SotA	*33.4*	*79.3*	*56.4*	*82.1*	*79.5*	*73.4*	*71.2*	*67.9*
Metric	M-F1	M-F1	M-F1	$F^{(i)}$	M-F1	M-Rec	AVG ($F^{(a)}$,$F^{(f)}$)	TE

Table 1: TweetEval test results. For neural models we report both the average result from three runs and its standard deviation, and the best result according to the validation set (parentheses). *SotA* results correspond to the best TweetEval reported system, i.e., BERTweet.

FIG. 15. TweetEval dataset results. Specifically sentiment partition results are used for comparison with pre- and post-fine-tuning classification reports.

```
[ ] pip install transformers torch gradio
```

FIG. 16. Install Hugging Face Transformers, PyTorch, and Gradio to run the multilingual sentiment analysis app locally or in Colab

```
[ ] from transformers import pipeline

    # Load the pre-trained sentiment analysis pipeline
    model_path = "cardiffnlp/twitter-xlm-roberta-base-sentiment"
    sentiment_task = pipeline("sentiment-analysis", model=model_path, tokenizer=model_path)
```

FIG. 17. Initializing the Hugging Face sentiment analysis pipeline using the XLM-RoBERTa model trained on Twitter data

is useful for demos and presentations because users can try it out without needing to read or run code. (Figure 19) (Figure 20)

Multilingual Dataset Collection

The notebook also pulls in multilingual tweet data to support broader analysis. Using a `%%bash` cell, it downloads tweet text and sentiment labels for several languages—Arabic, Danish, Dutch, English, French, German, Italian, Norwegian, Portuguese, and Spanish. (Figure 21)The data is stored in separate folders by language, creating a structured dataset that's ready for similarity comparisons.

Since the model was trained on similar data, this gives us a way to test how well it handles different languages and how consistent its predictions are across them.

Embedding Extraction and Cross-Language Comparison

This section of the code powers the cross-language similarity part of the demo. It compares a tweet written in Spanish to tweets in other languages by converting each one into a vector using a multilingual transformer model. It starts by loading the XLM-RoBERTa model and tokenizer from Hugging Face, which is capable of handling a wide range of languages. Before the text is passed into the model, it's cleaned up with a preprocess function that replaces usernames and links with placeholder tokens, which helps standardize the input. The get_embedding function then takes that cleaned text, runs it through the model, and averages the hidden states to get a fixed-size sentence embedding. These embeddings capture the meaning of the sentence in a way that allows for comparison across languages. (Figure 22)

To do the comparison, the compare_spanish_tweet function pulls in a list of Spanish tweets, randomly selects one, and uses it as the reference. It then loops through folders of tweet data in other languages, samples a few from each, and calculates cosine similarity between those and the Spanish tweet's embedding. (Figure 23) The top five most similar tweets are printed out with their language and similarity score. (Figure 24) This gives a quick way to see how well the model captures the meaning of a sentence, even across different languages. It's a good example of how multilingual embeddings can be used to connect ideas and sentiments beyond just classification.

The output shows a randomly selected Spanish tweet:

"Hola estoy que reviento por haberme comido un box de kebab ayudadme a morir gracias"

This tweet expresses physical discomfort in a kind of humorous way after overeating. The model then com-

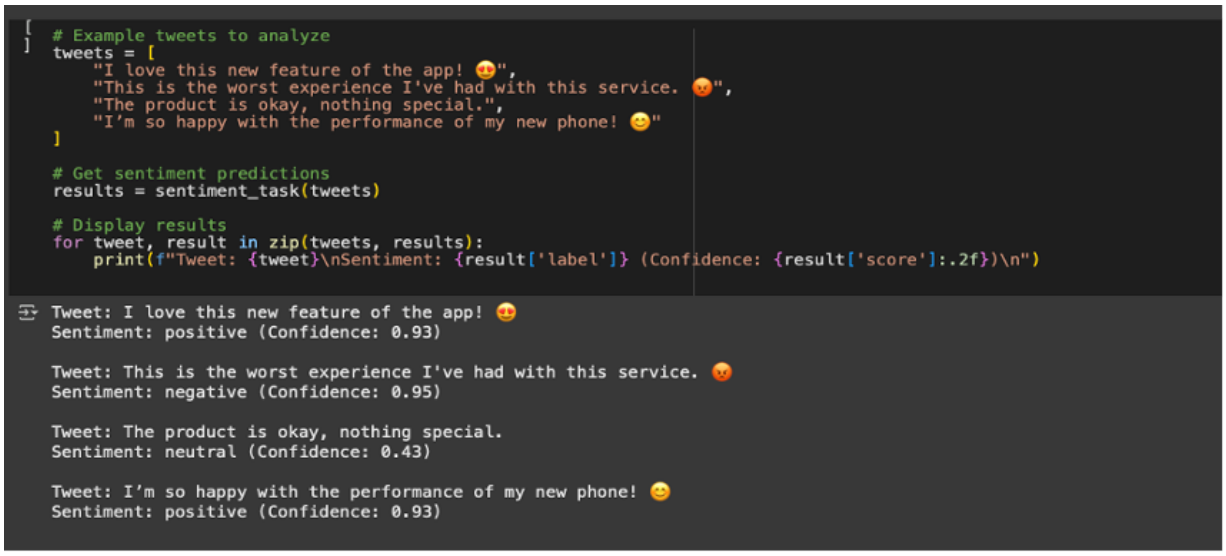


FIG. 18. Running sentiment analysis on a list of example tweets with confidence scores.

FIG. 19. Setting up the Gradio interface to analyze sentiment from user-input tweets.

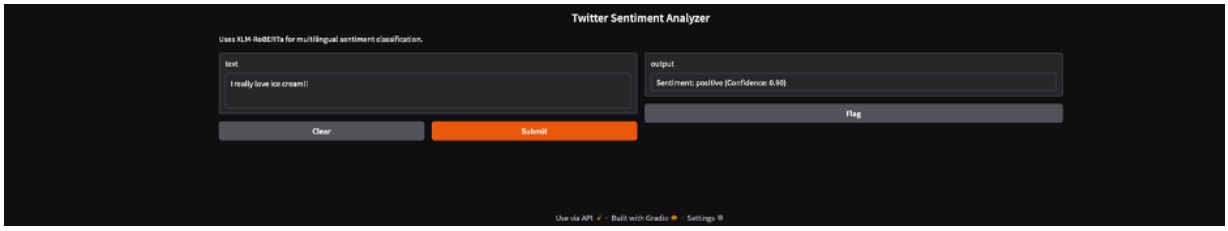
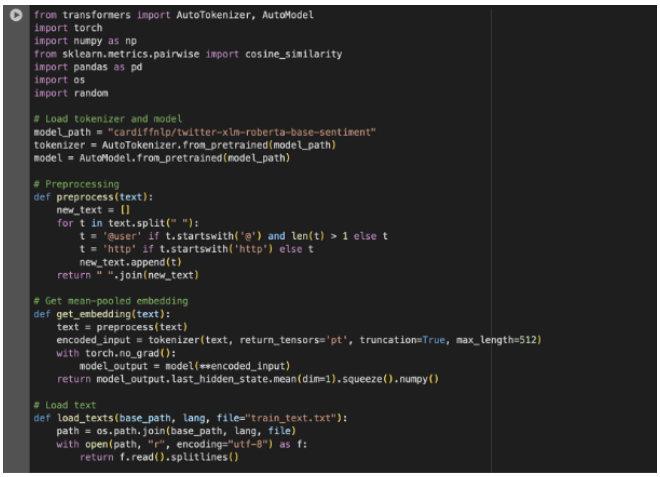


FIG. 20. Interactive web app for real-time tweet sentiment prediction

FIG. 21. Bash script to download multilingual TweetEval sentiment datasets

pares it to tweets in other languages and returns the top five most similar ones. Interestingly, most of the highest-scoring matches are in Portuguese, and they all share a casual, emotional, or food-related tone. One is even about overeating and wanting to end the conversation by eating everything, which is a close match in tone and context. This output highlights how well the model captures not just language patterns but also sentiment and



```python
from transformers import AutoTokenizer, AutoModel
import torch
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd
import os
import random

# Load tokenizer and model
model_path = "cardiffnlp/twitter-xlm-roberta-base-sentiment"
tokenizer = AutoTokenizer.from_pretrained(model_path)
model = AutoModel.from_pretrained(model_path)

# Preprocessing
def preprocess(text):
    new_text = []
    for t in text.split(" "):
        t = '@user' if t.startswith('@') and len(t) > 1 else t
        t = 'http' if t.startswith('http') else t
        new_text.append(t)
    return " ".join(new_text)

# Get mean-pooled embedding
def get_embedding(text):
    text = preprocess(text)
    encoded_input = tokenizer(text, return_tensors='pt', truncation=True, max_length=512)
    with torch.no_grad():
        model_output = model(**encoded_input)
    return model_output.last_hidden_state.mean(dim=1).squeeze().numpy()

# Load text
def load_texts(base_path, lang, file="train_text.txt"):
    path = os.path.join(base_path, lang, file)
    with open(path, "r", encoding="utf-8") as f:
        return f.read().splitlines()
```

FIG. 22.

Tokenization, preprocessing, and mean-pooling to create sentence embeddings

```python
# New demo function
def compare_spanish_tweet(spanish_tweet_index=None, base_path="sentiment", sample_size=100):
    spanish_tweets = load_texts(base_path, "spanish", "train_text.txt")

    if spanish_tweet_index is None:
        spanish_tweet_index = random.randint(0, len(spanish_tweets) - 1)

    spanish_tweet = spanish_tweets[spanish_tweet_index]
    spanish_embedding = get_embedding(spanish_tweet)

    print("📌 Spanish tweet:")
    print(f"   {spanish_tweet}\n")

    results = []
    for lang in os.listdir(base_path):
        if lang != "spanish" and os.path.isdir(os.path.join(base_path, lang)):
            tweets = load_texts(base_path, lang, "train_text.txt")
            sampled_tweets = random.sample(tweets, min(sample_size, len(tweets)))
            for tweet in sampled_tweets:
                emb = get_embedding(tweet)
                score = cosine_similarity([spanish_embedding], [emb])[0][0]
                results.append((lang, tweet, round(score, 4)))

    top_matches = sorted(results, key=lambda x: x[2], reverse=True)[:5]
    df = pd.DataFrame(top_matches, columns=["Language", "Tweet", "Similarity Score"])
    print("🔍 Top 5 Most Similar Tweets:\n")
    print(df.to_string(index=False))

# Run
compare_spanish_tweet()
```

FIG. 23.

Function to compare a Spanish tweet to tweets from other languages using cosine similarity

intent across different languages.

Technical Considerations

The notebook depends on several Python packages, including transformers, torch, sklearn, pandas, and gradio. Since Google Colab sessions are temporary, these packages need to be reinstalled each time the session is restarted. In addition, the transformer model used is approximately 1 GB in size, which means initial loading may take time.

While Gradio provides a convenient way to serve interactive apps, the public URLs it generates are only valid for 72 hours unless the app is deployed permanently using a service like Hugging Face Spaces.

The notebook pulls together several modern NLP tools to show how multilingual sentiment classification and semantic search can work together. It combines Hugging Face pipelines for fast classification, embedding extraction for deeper analysis, and an interactive UI with Gradio to make it user-friendly. Whether for presentations or experiments, it shows how transformer models can be applied to real social media data in practical and engaging ways.

VII. FUTURE APPLICATIONS OF XLM-T

Multimodal Sentiment Analysis

As tweets increasingly include images, videos, and GIFs, future versions of XLM-T could integrate multimodal learning by combining text embeddings with image features. This would improve the detection of emotional tone and sarcasm when visual context complements or contradicts the text.

Real-Time Social Media Monitoring Systems

XLM-T could serve as the foundation for real-time, multilingual dashboards that track:

- Global public opinion on political events
- Consumer feedback on product launches
- Misinformation or sentiment shifts during crises.

Personalized Mental Health Monitoring

With responsible fine-tuning and privacy protections, XLM-T could be used in wellness applications to track emotional tone over time — detecting signs of stress or depression based on multilingual social media posts, especially when used in conjunction with private journaling or chatbot platforms.

VIII. CONCLUSION

This project presented a comprehensive exploration of XLM-T, a multilingual language model specifically adapted for sentiment analysis on Twitter. By continuing pretraining from XLM-R on 198 million tweets across over 30 languages, XLM-T effectively bridges the gap between general-purpose multilingual models and domain-specific, noisy social media text.

Key innovations such as emoji- and slang-aware tokenization, adapter-based fine-tuning, and balanced multilingual benchmarks like UMSAB enabled XLM-T to

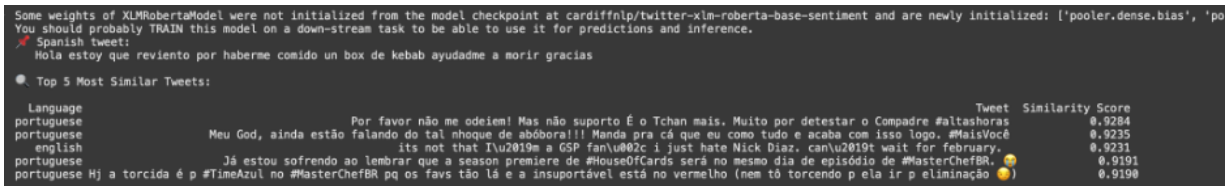


FIG. 24. Output showing top 5 most similar tweets across languages based on embedding similarity

outperform strong baselines in both monolingual and zero-shot multilingual settings. The model demonstrates robust cross-lingual transfer capabilities, particularly in low-resource scenarios where training data is limited.

From a broader perspective, this work validates the importance of domain adaptation, parameter-efficient transfer learning, and culturally aware evaluation in building real-world NLP systems. XLM-T serves as a blueprint for adapting foundational language models to dynamic, multilingual, and informal digital communication platforms like Twitter. Future work can extend XLM-T to include multimodal inputs, support a broader range of low-resource languages, and investigate ethical concerns around cultural bias and real-time deployment. Overall, XLM-T represents a significant step toward building socially and linguistically inclusive NLP systems.

IX. REFERENCES

Barbieri, Francesco, et al. "XLM-T: Multilingual Language Models in Twitter for Sentiment Analysis and Beyond." arXiv.Org, 11 May 2022, arxiv.org/abs/2104.12250.