HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Emoji Prediction on Social Media

Supervisor:

Dr. Nguyễn Bá Ngọc

Students:

Lương Thái Khang - 20224866

Hanoi, January 2026

ABSTRACT

Emoji usage is an important part of modern online communication, but selecting an appropriate emoji can interrupt writing flow. This report presents an end-to-end pipeline for **emoji prediction from text**: (i) constructing a cleaned tweet-style dataset with 43 emoji classes, (ii) benchmarking classical and neural text classifiers, and (iii) deploying the best model for client-side inference in the browser. We compare TF–IDF baselines (linear SVM, logistic regression, and Naive Bayes), FastText-style subword models, TextCNN, bidirectional RNNs with attention, and a fine-tuned DistilBERT Transformer [3, 4, 7]. On the provided test split (83,543 samples), the Transformer achieves the best performance with **top-1 accuracy 26.3%** and **top-5 accuracy 56.6%**. Finally, we export the Transformer to quantized ONNX and integrate it into a static web demo using Transformers.js, enabling fast, privacy-preserving emoji suggestions directly in the browser [6, 10].

Table of Contents

1 INTRODUCTION

1.1 Motivation

Emojis enrich text with emotional nuance, emphasis, and cultural context. On social media and messaging platforms, users often add one or more emojis to convey sentiment (e.g., excitement, sarcasm), summarize a message, or increase engagement. However, manually choosing an emoji requires extra attention and can interrupt writing flow. A lightweight emoji suggestion system can improve user experience by predicting emojis from the text being written.

1.2 Scope of This Repository

This repository focuses on **single-label emoji prediction**: given a short English text (tweet-like sentence), predict exactly one emoji from a fixed set of 43 emojis. The repo includes:

- A cleaned dataset split into train/test CSV files (`data/`).
- A model benchmark suite covering classical baselines and neural architectures (`tune.py`, `train_eval.py`, `ml.py`).
- A Python inference helper for the deployed Transformer model (`infer.py`).
- A browser deployment path based on ONNX + Transformers.js (`scripts/export_onnx.py`, `docs/`).

1.3 Contributions

The main contributions of this work are:

- A practical preprocessing pipeline for tweet-style text with emoji removal and normalization.
- A consistent evaluation framework for top-k metrics over multiple model families.
- A deployment workflow that exports the best Transformer model to quantized ONNX and runs it client-side in a static web application.

1.4 Report Organization

Section 2 reviews related work in text classification and emoji prediction. Section 3 formalizes the task and evaluation metrics. Section 4 describes the dataset and preprocessing. Section 5 presents the models. Section 6 reports experiments and results. Section 7 describes deployment. Finally, Section 8 concludes and discusses limitations and future work.

2 RELATED WORK

2.1 Emoji Prediction and Text Classification

Emoji prediction can be viewed as standard multi-class text classification, closely related to sentiment analysis and emotion recognition in social media text. Classical pipelines typically rely on sparse lexical features (e.g., TF–IDF over word/character n-grams) paired with linear classifiers. These methods are strong baselines for short-text tasks and remain competitive when data is limited [5].

2.2 Neural Models for Text

Neural approaches learn task-specific representations and can better capture compositional patterns beyond surface n-grams:

- **FastText-style models** use word embeddings with subword n-grams to handle misspellings and morphological variations efficiently [3].
- **Convolutional architectures** (TextCNN) extract local n-gram features using 1D convolutions and max pooling, often performing strongly on sentence classification [4].
- **Recurrent encoders** (LSTM/GRU) capture sequential context; attention mechanisms improve pooling by weighting informative tokens more heavily [1].

2.3 Transformers

Transformers replace recurrence with self-attention and dominate many NLP benchmarks [9]. Large pretrained models such as BERT [2] and its compressed variants such as DistilBERT [7] can be fine-tuned for downstream classification tasks with relatively small changes to the model head.

2.4 Client-side Deployment

Running NLP models in the browser is increasingly feasible via ONNX export and WebAssembly backends. ONNX provides an interoperable model format [6], and Transformers.js enables local model loading and inference in JavaScript applications [10]. This repo leverages both to deliver privacy-preserving emoji suggestions without server-side inference.

3 PROBLEM FORMULATION

3.1 Task Definition

Let $\mathcal{Y} = \{1, 2, \ldots, C\}$ be a finite emoji label set with $C = 43$ classes. We are given a dataset

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}, \tag{1}$$

where x_i is an input text string and $y_i \in \mathcal{Y}$ is the associated emoji label. The goal is to learn a function f_θ parameterized by θ that maps text to a distribution over emojis.

3.2 Learning Objective

Most models in this repo produce unnormalized scores (logits) $z = f_\theta(x) \in \mathbb{R}^C$, converted to class probabilities via the softmax:

$$p_\theta(y = c \mid x) = \frac{\exp(z_c)}{\sum_{j=1}^{C} \exp(z_j)}. \tag{2}$$

The standard training objective for neural models is cross-entropy:

$$\mathcal{L}(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \log p_\theta(y_i \mid x_i). \tag{3}$$

3.3 Evaluation Metrics (Top-k)

Because multiple emojis can be plausible for a text, we evaluate models using **top-k** metrics. Let $\mathrm{TopK}(x)$ denote the set of k highest-scoring predicted labels for input x.

Top-k Accuracy.

$$\mathrm{Acc@}k = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}\left[y_i \in \mathrm{TopK}(x_i)\right]. \tag{4}$$

Macro Recall@k and Macro F1@k. For each class c, we treat the event "c is included in top-k" as a binary prediction and compute per-class precision/recall/F1, then macro-average across classes. This is implemented in `ml.topk_metrics` and reported for $k \in \{1, 3, 5\}$.

4 DATASET

4.1 Source and Collection

The dataset is derived from tweet-style text collected using `snscrape`. Data collection was performed by issuing queries per emoji and retaining text samples that contain the queried emoji. English filtering is performed using `pycld3`, inspired by the WiLI language identification benchmark [8]. Because scraped social media text is noisy, some non-English samples and duplicates may remain.

4.2 Cleaning and Normalization

The preprocessing notebook (`data-preprocess.ipynb`) applies a simple cleaning function that:

- Normalizes Unicode using NFKC.
- Removes URLs, mentions, hashtags, and emojis.
- Lowercases and collapses repeated whitespace.

4.3 Train/Test Split Statistics

The final dataset is stored as two CSV files (`data/train_data.csv`, `data/test_data.csv`). Table 1 summarizes the split.

Table 1: Dataset statistics.

Split	Samples	#Classes	Class count range
Train	751,885	43	16,718–17,846
Test	83,543	43	1,858–1,983

5 MODELS

5.1 Common Preliminaries

Tokenization for non-Transformer neural models uses a simple regex-based tokenizer that splits on word characters and punctuation. A vocabulary is constructed from the training set with a configurable maximum size and minimum frequency, then sequences are padded per batch.

5.2 TF–IDF Baselines

We use a feature union of:

- word n-grams (e.g., $(1, 2)$),
- character n-grams (e.g., $(3, 5)$),

converted to TF–IDF features and trained with one of:

- Linear SVM (`LinearSVC`),

- Logistic Regression (`LogisticRegression`),
- Multinomial Naive Bayes (`MultinomialNB`).

5.3 FastText-Style Subword Model

The FastText-style classifier [3] represents each token by a word embedding and augments it with hashed subword n-grams to improve robustness to rare words and spelling variants. Subword features are generated for each token and mapped into a fixed number of hash buckets; embeddings are aggregated and fed to a linear classification head.

5.4 TextCNN

TextCNN applies multiple 1D convolution filters with different kernel widths over the embedding sequence, followed by max-over-time pooling and a dropout-regularized linear layer [4]. This captures local n-gram patterns efficiently.

5.5 BiRNN with Attention

The BiLSTM/BiGRU model encodes the sequence bidirectionally, then applies an attention mechanism to compute a weighted sum of hidden states [1]. Compared to average pooling, attention can emphasize informative tokens (e.g., sentiment-bearing words).

5.6 Transformer (DistilBERT Fine-tuning)

We fine-tune DistilBERT [7] for sequence classification by adding a classification head and optimizing cross-entropy. Fine-tuning leverages pretrained language knowledge and yields the strongest performance in our experiments.

6 EXPERIMENTS

6.1 Experimental Protocol

Hyperparameters are tuned with `tune.py` using a train/validation split and a small, fixed grid per model family. The best configuration is selected by macro F1@1 and stored in `outputs/best_params.json`. Final evaluation is performed with `train_eval.py` on the held-out test set, producing `outputs/test_results.csv`. All experiments use $k \in \{1, 3, 5\}$ for top-k metrics and a fixed random seed (default 42). Neural models are trained with Adam/AdamW optimizers; Transformer fine-tuning uses a linear warmup/decay schedule [2].

6.2 Results

Table 2 reports test performance for each model. The Transformer achieves the best overall results, improving both top-1 and top-5 accuracy over classical baselines.

Table 2: Test-set results (`outputs/test_results.csv`).

Model	Top-1 Acc	Top-3 Acc	Top-5 Acc
TF–IDF + SVM	0.2109	0.3602	0.4460
TF–IDF + LogReg	0.2363	0.4170	0.5172
TF–IDF + NB	0.2234	0.4064	0.5112
FastText	0.2153	0.3996	0.5042
TextCNN	0.2094	0.3821	0.4864
BiLSTM + Attention	0.2254	0.4064	0.5101
BiGRU + Attention	0.2213	0.3983	0.5009
Transformer	**0.2632**	**0.4596**	**0.5659**

6.3 Discussion

Two trends are notable. First, strong TF–IDF baselines (especially logistic regression) remain competitive on short, noisy text. Second, the fine-tuned Transformer benefits from pretrained representations and achieves the best top-k performance, which is important for emoji suggestion interfaces where multiple emojis can be shown to the user.

7 DEPLOYMENT

7.1 Python Inference

The script `infer.py` provides a minimal CLI for running inference with a saved Hugging Face-style model directory (default `outputs/transformer_deploy`). It loads the tokenizer and model, computes

probabilities, and prints the top-k predictions for each input text.

7.2 Browser Demo (Client-side ONNX)

The `docs/` directory contains a static web application that loads a local model using Transformers.js [10]. Remote model downloads are disabled; all assets are served from `docs/models/emoji-model/`. The main UI (`docs/index.html`) provides real-time emoji suggestions while typing; `docs/debug.html` offers a debugging interface for batch predictions and latency measurements.

7.3 ONNX Export and Quantization

To enable browser inference, `scripts/export_onnx.py` exports the Transformer model to ONNX [6] and optionally applies dynamic quantization to int8 for smaller downloads and faster CPU execution. The exported model is stored under `docs/models/emoji-model/onnx/` and is consumed by the web app.

8 CONCLUSION

This report described a complete emoji prediction system from dataset construction to browser deployment. We benchmarked classical TF–IDF baselines, lightweight neural architectures, and a fine-tuned DistilBERT Transformer. On the provided test set, the Transformer achieved the best top-k accuracy, making it well-suited for suggestion-based user interfaces. We also presented a practical deployment workflow based on ONNX export, quantization, and client-side inference via Transformers.js.

8.1 Limitations and Future Work

The current formulation predicts a *single* emoji per text, while real messages may contain multiple emojis. Future work could explore multi-label prediction, calibration of confidence scores for better ranking, and domain adaptation to non-twitter text. Additionally, stronger pretrained models or multilingual training could improve robustness across topics and languages.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *International Conference on Learning Representations (ICLR)*. 2015.

[2] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of NAACL-HLT*. 2019.

[3] Armand Joulin et al. "Bag of Tricks for Efficient Text Classification". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*. 2017.

[4] Yoon Kim. "Convolutional Neural Networks for Sentence Classification". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014.

[5] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[6] *ONNX: Open Neural Network Exchange*. https://onnx.ai/. Accessed 2026-02-01. 2026.

[7] Victor Sanh et al. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: *arXiv preprint arXiv:1910.01108* (2019).

[8] Martin Thoma. *The WiLI benchmark dataset for written language identification*. arXiv:1801.07779. 2018.

[9] Ashish Vaswani et al. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017.

[10] Xenova. *Transformers.js*. https://github.com/xenova/transformers.js. Accessed 2026-02-01. 2026.