COMPSCI/SFWRENG 4AL3

Group 8 Final Project Proposal

Part A: Team Members

Developer Name	Email	MacID	GitHub
Zifan(Frank) Si	siz@mcmaster.ca	siz	ZifanSi
Xianzhao(Paul) Duan	duanx15@mcmaster.ca	duanx15	PaulDuanGitHub

Part B: Project Overview

Title: Context-Aware Emoji Prediction for Chat Messages

Emojis have become an essential part of digital communication, serving as visual symbols that help convey emotions, tone, and nuance often lost in text-based interactions. With the rise of remote work, instant messaging, and the popularity of social media platforms such as TikTok, Instagram, and X, emoji use has grown dramatically, shaping how people express themselves online. Current input tools on modern smartphones attempt to predict emojis for users, but these predictions are largely keyword-driven and fail to account for the broader sentiment or context of a conversation. As a result, they may suggest emojis that feel mismatched or too literal, limiting their usefulness.

Plantform	Style	Why context matters ?
TikTok / Instagram comments	Playful, hype, memes, quick reactions	Same phrase can be playful, so "I'm crying" → 😂 more than 😢.
X (Twitter)	Short takes, news, sarcasm, slang	Sarcasm is common; "great." can map to 😒.
Work chats (MS Teams/Slack)	Polite, concise, status updates	Tone skews formal; "nice" is supportive 🙂, not 😂/🤣.

We propose an approach to emoji prediction that incorporates sentiment analysis and context analysis. By analyzing not only the explicit words typed but also the emotional tone and surrounding conversational context, such a system could predict different meanings that are carried by the same phrase — for instance, the message “I’m crying” could imply two different emotions and emojis 😢 and 😂. Leveraging techniques from natural language processing (NLP) and machine learning, models trained on real-world text–emoji pairings from social media could better predict the most contextually appropriate emoji.

Emoji panels are slow to browse, and keyword triggers often miss context. A context-aware suggester reduces friction, improves tone clarity, and speeds up composition in chat.

By solving this problem, users can express themselves more precisely and effectively; accordingly, the readers or information receivers can understand the senders’ hidden emotions and intentions better. We are essentially helping people communicate by providing context-fitted emojis. As a result of applying the solution, the user experience of posting or instant messaging will also be improved by emoji prediction and saving time of looking for proper emojis.

Part C: Task Definition

Our task is to take a short, chat-style message, using whatever the user has already typed, plus any immediately available context, and suggest the top-k emojis from a fixed set of 20 that best match the writer’s tone and intent. Instead of guessing from keywords alone, the system looks at wording, common slang, and nearby context to decide whether a phrase is sad, playful, sarcastic, celebratory, etc., then returns one best emoji along with a ranked list of alternatives so the user can pick quickly. The goal is to make messaging faster and clearer: fewer mismatched suggestions, less scrolling through emoji panels, and better alignment between what the sender means and what the reader perceives, whether the message appears in work chats or casual social apps.

This project will address the following key attributes of this classification task:

Attribute	Description
Type of Data	Short, chat-like text (tweets / messages).
Learning problem	Supervised multiclass classification with ranked recommendations
# Classes	20 (each corresponding to a unique emoji)
Labeling	Single-label (one emoji per message)

Prediction interface	Return top-k suggestions (default k=5; also report top-1)
Input (x)	A tokenized text string (may include hashtags/mentions/URLs/emojis)
Output (ŷ, π□)	One predicted emoji (top-1) plus a ranked list of the top-k suggested emojis
Assumptions	Class imbalance expected; handle via weighting/stratified splits; text standardized in preprocessing
Out-of-scope	Images/video; generating new emojis; multi-label supervision

Part D: Problem Statement & Real-World Impact

Text does not always show tone. A short message can be serious, sarcastic, or playful, and readers may guess wrong. Many emoji suggesters only match keywords and ignore the conversation, so the suggestions often feel off. We want a context-aware emoji helper that reads a short chat message, uses a little nearby context, and suggests one best emoji plus a small ranked list from a fixed set of twenty. It should work on real chat text with slang, typos, hashtags, mentions, and links. It should handle phrases that change meaning with context and avoid favoring only the most common emojis.

This helps people type faster and communicate tone more clearly. In work chats it can reduce confusion and follow-up messages. In social apps it can make expression feel natural and fun. We will judge success by higher Top-1, Top-5, and Macro-F1 scores than simple keyword rules and by steady performance on both frequent and rare emojis. We will respect dataset licenses and platform terms and avoid storing personal content beyond policy. We will report per-class results and confusion patterns to find and reduce systematic errors.

Key challenges:

Challenge	Description
Noisy Text	Slang, hashtags, mentions, and links clutter messages and confuse the model.

	E.g. “that drop was fireee 🔥 fr fr check this **https://t.co/**… @djmax #weekend” — model must ignore link/mention/hashtag noise.
Context dependence	The same sentence can mean different things depending on earlier messages. E.g. A: “We lost by 30.” B: “I’m crying.” → 😢; but after a funny meme, “I’m crying.” → 😂.
Ambiguity & sarcasm	Similar emojis overlap in meaning, and sarcasm flips sentiment. E,g. “Yeah, great meeting” could mean 🙂 or 😒 depending on tone; “nice job…” may be sincere or sarcastic.
Class imbalance	A few emojis dominate the data, so the model over-predicts them E.g. ❤️ and 😂 appear far more than 😔 or 😱, so the model keeps suggesting ❤️/😂 even when a rarer emoji fits better.
Short messages	Very little text to read; tiny cues matter, so errors are easy. E.g. “sure.” vs “sure!!” — small punctuation change shifts tone; model must infer 🙂 vs 😑 with minimal context.

Part E: Data Sources & Collection Plan

First, we’ll use the Twitter Emoji Prediction dataset from Kaggle (about 70k messages across 20 emojis [1]). We’ll download the train/test CSVs and the label-to-emoji mapping file. In our proposal we’ll include two links: the Kaggle page and the original/source page that explains how the data was collected and labeled.

Next, we’ll follow the dataset license and platform rules. If we later add new tweets, we’ll use the official Twitter/X API, respect rate limits with pagination/backoff, and store only allowed fields (often just tweet IDs).

Then, note that each row has the message text and a label_id that maps to one of the 20 emojis. Labels are already provided, so no hand-labeling is needed. If we do a tiny sanity check, it will take about 15–30 seconds per item, and we'll double-check any disagreements.

Also, before training we'll clean the text: remove or normalize URLs, @mentions, and hashtags, lowercase when helpful, keep punctuation that shows tone (like "!!" and "…") and remove near-duplicates. Because some emojis are far more common, we'll handle class imbalance with stratified splits and/or class-weighted loss.

Finally, we'll use the entire dataset (it's small enough and helps capture rare emojis). We'll keep data in a versioned /data/ folder with a short data card (schema, steps, license, and dataset hash) and pin versions/seeds for reproducibility.

Quick summary table:

Item	Our plan (Current)
Dataset & size	Twitter Emoji Prediction, ~70k rows, 20 emoji classes
Links to include	Kaggle URL + original/source URL (collection and labels)
Access & compliance	Kaggle download; API augmentation only via Twitter/X with rate-limit/backoff; store allowed fields/IDs
Fields & labels	text, label_id; mapping file label_id ↔ emoji (single-label)
Preprocessing	Normalize/remove URLs/mentions/hashtags; keep tone punctuation; deduplicate
Imbalance handling	Stratified splits and/or class-weighted loss
Scope	Use the entire dataset to learn rarer emojis
Reproducibility	Versioned /data/, data card, pinned versions, fixed seeds, logged mapping file

Part F: Expected size and three example labeled instances

This dataset [1] from Kaggle will be the major data source for the project. This dataset contains 20 distinct emojis to predict, with 70,000 rows of training data, in the following format:

Text	Label
“Just got the job offer!!”	3
“Miss you a lot today.”	5
“That new trailer was hilarious”	7

There is also a mapping dataset that maps labels to specific emojis with the following format:

Emoji	Label
🎉	3
😢	5
😂	7

All examples come from Twitter (now X) and remain the property of their original authors; we will use the dataset under its license and follow platform terms. Because the dataset is simple—each row has just the message text and a label ID—we will use the **entire** dataset to get enough coverage of rarer emojis and improve model reliability. No hand-labeling is required since labels are already provided, but we will still clean the data before training: remove or normalize URLs, @mentions, and hashtags; lowercase when helpful; keep punctuation that signals tone (like “!!” and “…”) ; and deduplicate near-identical messages. These steps reduce noise without stripping away cues the model needs to infer sentiment and context.

Part G: Proposed Solution

We want to suggest one best emoji and a short top-k list from a fixed set of 20 for each short chat message. The input is the message text; the target label is a single emoji ID that maps to an actual emoji. We’ll lightly clean the text (remove or normalize links, @mentions, and hashtags), keep useful tone signals like “!!” or “…”, and then turn the text into numbers the model can learn from.

Light text cleaning (what we do and why)

Step	Example (before → after)	Why it helps

Remove/normalize links	“check this https://t.co/xyz…” → “check this”	Links add noise, not tone.
Handle mentions/hashtags	“@alex that’s wild #friday” → “that’s wild”	Strips tokens that don’t change sentiment.
Keep tone punctuation	“sure!! … ok” → keep “!!”, “…”	Punctuation carries emotion/intent.

We’ll try two ways to represent the text. First, TF-IDF, which is a simple and strong classic method for short texts. Second, pretrained embeddings (word or sentence level), which often capture meaning and tone better than keywords alone. After we build these features, we’ll train a few models and compare them.

How we represent text:

Method	What it captures	Used for
TF-IDF	Word importance in short texts	Strong classic baseline with Linear SVM
Pretrained embeddings	Semantics and nuance beyond keywords	Neural models (LSTM**[2]**/BiLSTM or small transformer)

For models, we’ll start with two baselines: a tiny keyword-rule system (just to set a floor) and a Linear SVM trained on TF-IDF (a solid non-neural baseline). Our main models will be LSTM and BiLSTM, which read the text in sequence and can pick up context. If time allows, we’ll also try a small transformer (for example, DistilBERT) to see if it improves results.

To check quality, we’ll split the data into train, validation, and test sets with the class balance preserved. We’ll report Top-1 accuracy, Top-5 accuracy, and Macro-F1 so rare emojis are not ignored. We’ll also look at a confusion matrix to see which emojis the model mixes up (for example, 😢 vs 😂), and we’ll run small “what helps?” tests, like TF-IDF vs embeddings and with or without class weighting.

How we judge model quality:

Metric	What	Why
Top-1 accuracy	% where top prediction is correct	Basic correctness.
Top-5 accuracy	% where correct emoji appears in top-5	Matches “suggestion list” UX.
Macro-F1	Average F1 over all 20 classes	Treats rare emojis fairly.

Our plan is inspired by prior work on emoji prediction and sentiment in short text, including the Twitter Emoji Prediction task on Kaggle and papers such as IEEEXplore 10085173, ACL E17-2017**[3],** and arXiv 1708.00524**[4]**. We’ll build with Python, pandas, scikit-learn (TF-IDF, SVM, metrics), PyTorch or Keras (LSTM/BiLSTM), optional Hugging Face tokenizers/embeddings, NLTK/emoji utilities, and matplotlib for plots.

Main risks are short messages, class imbalance (some emojis are very common), and changing slang. We’ll keep tone punctuation, use stratified splits and class-weighted loss, and watch for drift in our error analysis. For reproducibility, we’ll version the data and code, pin library and dataset versions, fix random seeds, and log the exact label-emoji mapping and settings we used.

References

[1] Kaggle, “Twitter Emoji Prediction.” [Online]. Available: https://www.kaggle.com/datasets/hariharasudhanas/twitter-emoji-prediction. Accessed: Oct. 8, 2025. Kaggle

[2] IEEE Xplore, “Document 10085173.” [Online]. Available: https://ieeexplore.ieee.org/document/10085173. Accessed: Oct. 8, 2025.

[3] F. Barbieri, M. Ballesteros, and H. Saggion, “Are Emojis Predictable?,” in *Proc. 15th Conf. European Chapter of the Association for Computational Linguistics (EACL), Vol. 2: Short Papers*, Valencia, Spain, Apr. 2017, pp. 105–111. [Online]. Available: https://aclanthology.org/E17-2017/. ACL Anthology

[4] B. Felbo, A. Mislove, A. Søgaard, I. Rahwan, and S. Lehmann, “Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm,” *arXiv preprint* arXiv:1708.00524, 2017. [Online]. Available: https://arxiv.org/abs/1708.00524. arxiv.org