Design and Implementation of a Text Editor with Emoji Search Engine

Ivan Iazykov
Course: Web Information Retrieval
Institution: Tsinghua University

May 23, 2025

Abstract

This paper presents the design and implementation of a text editor application that natively supports an unlimited number of custom and animated emojis, rendered as normal inline text characters. It integrates key information retrieval techniques, including crawling, results ranking, and the use of Large Language Models. The system is engineered to provide seamless emoji search and prediction, ensuring high relevance and responsiveness. By utilizing Word2Vec for smart emoji search and fine-tuned and base BERT models for end-of-sentence emoji prediction, the architecture unifies query processing and prediction under a consistent similarity framework. The system is deployed using Docker, which enables platform-independent, containerized deployment, orchestrating independent services: MongoDB database for emoji data storage, Python as the backend for LLM model inference, and a Node.js web application as the single user entry point and a central hub. A key contribution of this work is the development of a standalone emoji toolkit featuring a multimodal processing pipeline for emoji retrieval and classification, unifying text-based embedding techniques with custom emoji support. The architecture is designed for seamless integration into any text-input-based application and can be deployed effortlessly across diverse environments, making advanced emoji search and prediction capabilities universally accessible.

1 Introduction

I designed and implemented the project alone, though everything below is a work and contribution of one person. It took me quite a while to settle on an idea that was both novel and personally interesting – building a useful search engine nowadays is a complex task, when nearly every conceivable use case already has a solution. Inspired by the success of multimodal AI at the intersections of existing technologies, I decided to shift the paradigm by integrating a search engine with a text editor.

While some people rarely use emojis and others rely on a single one for every scenario, I find them invaluable for adding emotional nuance, especially when a conversation doesn't have a serious tone. Intriguingly, research published last year [**1**] found a modest positive correlation between emoji usage and emotional intelligence. According to their work, people with higher emotional intelligence tend to send more emojis to friends. However,

despite there being around 4,000 standard Unicode emojis [**2**], the real options are limited; for instance, users expressing a laughter emotion would typically use either *"face with tears of joy"* or *"rolling on the floor laughing"* (these are Unicode emoji names). It raises a question of having a vast database alongside a text-based interface with expanded search that surfaces dozens of alternatives – the backbone idea for the design of my project.

2 Existing Solutions

Emojis are primarily used on mobile devices, as for today this is the most natural environment for chatting. When we look at existing mobile solutions (see Fig. 1), several interfaces stand out. Microsoft's SwiftKey Keyboard, which I personally use, offers a solid built-in emoji search that quickly finds relevant icons. WeChat doesn't support emoji search, but has a pack of built-in custom emojis. Apple's native keyboard also provides emoji search, and with its latest iOS update has introduced "Genmoji (Beta)" for text-prompted emoji generation. The output quality is generally poor, but it shows user demand for custom emoji creation. Beyond these, the only major app combining both emoji search and custom emojis is **Discord**, originally a gamers' chat platform that has since expanded into broader communities. Discord demonstrates the value of custom emoji ecosystems, but its requirement of a $5-per-month subscription for full custom-emoji access highlights a significant barrier.

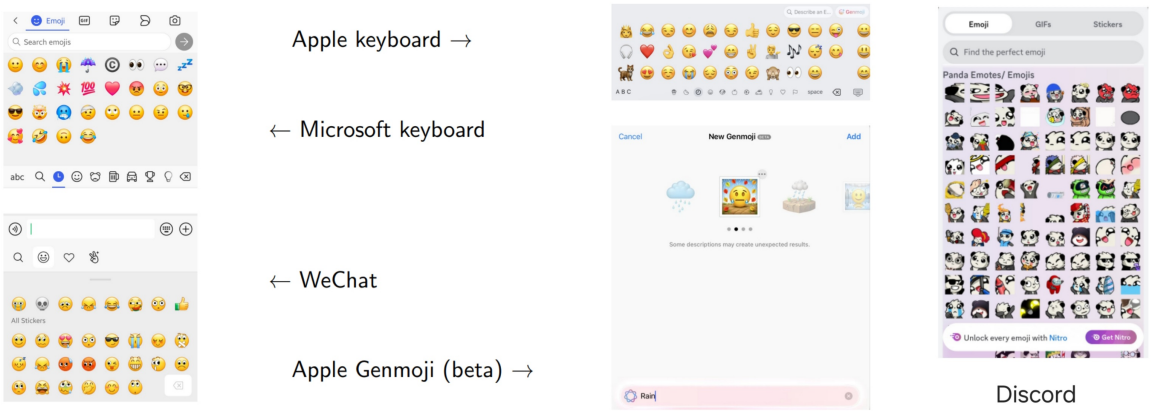

Figure 1: Mobile emoji interfaces

3 Proposed Solution

I propose **Search2Emoji**, a web-based text editor that is built from scratch to seamlessly integrate emoji search and prediction into the writing experience. It natively supports an unlimited number of custom and animated emojis, rendered as normal inline text characters. Unlike Discord's premium-locked custom emoji system, Search2Emoji is completely free, presents a full text-editor UI rather than a chat window, and predicts Discord emojis, which Discord itself cannot.

Available functionality:

- **Emoji search** – search for custom emojis using Word2Vec model or for default Unicode emojis with normal search
- **Emoji prediction** – predict emoji at the end of the sentence via fine-tuned or base BERT models
- **Text editor** – default features of a text editor, e.g., font style, font size, save, dark mode models

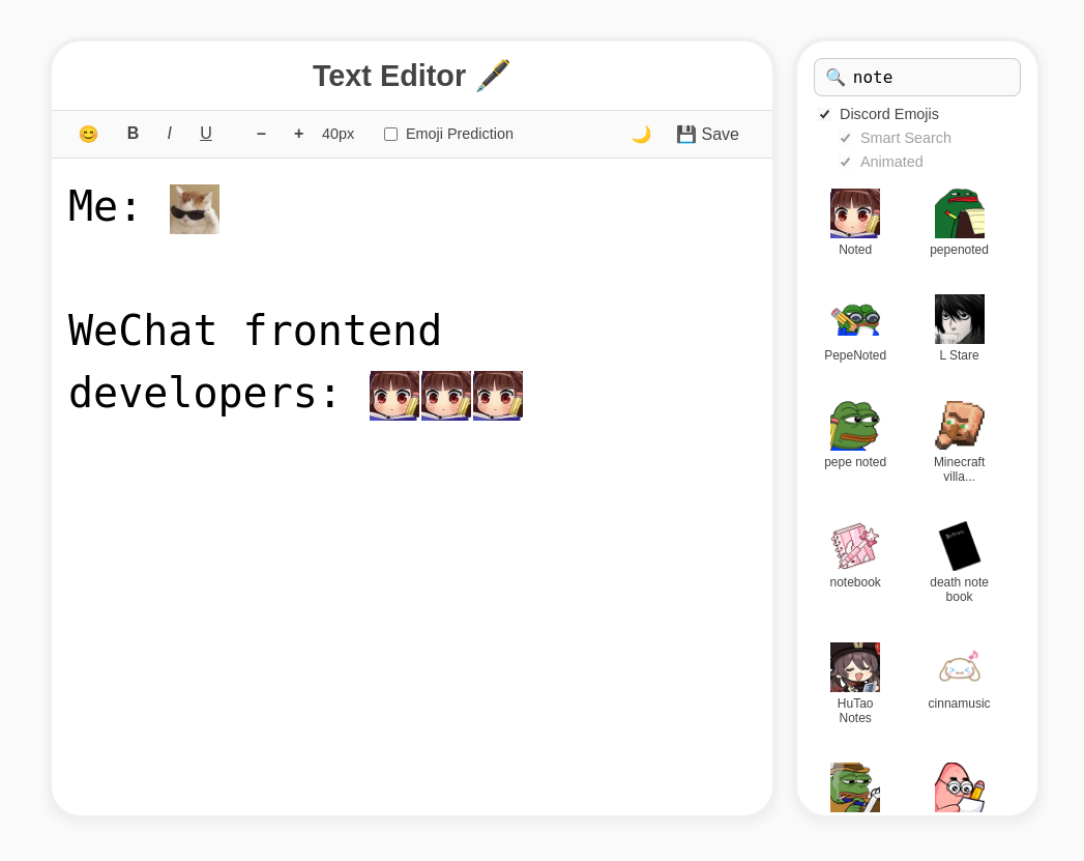

Figure 2: Search2Emoji. Web text editor with emoji search

3.1 Architecture

The text editor application is deployed within Docker, which functions as an isolated virtualized environment, enabling the creation of application components as separate *services*. All services are containerized, ensuring consistent deployment and scalability, whether on a local laptop, a server, or in the cloud. Containerization is orchestrated by the `docker-compose.yml` file, which simplifies the definition and management of services. An overview of the application architecture is illustrated in Fig. 3.

There are 3 services (independent containers) that operate autonomously:

1. **MongoDB Database**

   A non-relational database that stores data as flexible, JSON-like documents instead of traditional tables with rows and columns.

2. **Python LLM Backend**

   Hosts two pre-trained BERT models behind an application setup with `FastAPI`. FastAPI provides asynchronous request handling. The API is hosted by `Uvicorn`, a high-performance ASGI server written in Python, which runs the backend application and handles incoming HTTP methods like `GET` and `POST`.

3. **Node.js Web Application**

   Acts as the single entry point for end users, hosting the text editor interface in the browser. Built on the Node.js runtime, it centralizes all API endpoints (e.g., emoji search, prediction requests) and routes user queries to both the MongoDB database and the Python backend.

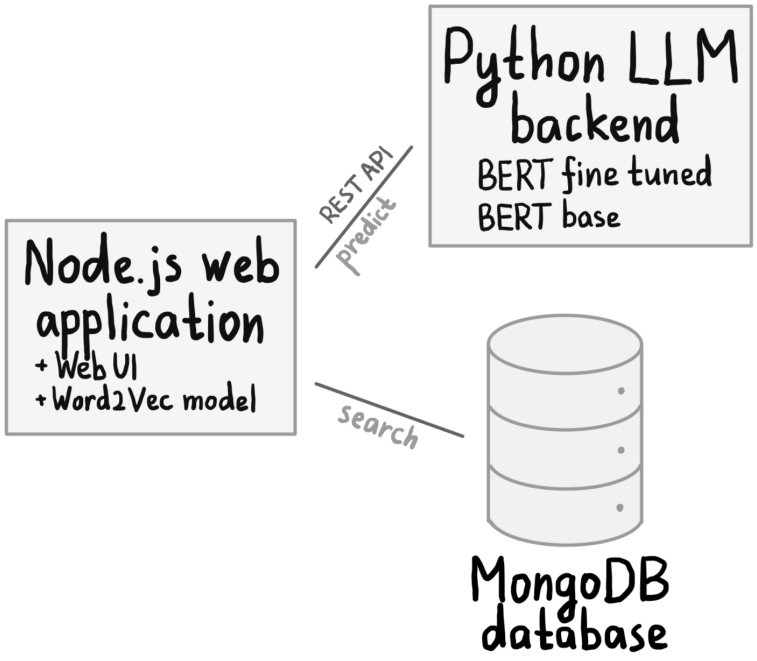

Figure 3: Architecture diagram

3.2 Project Structure

Project's folder organization:

`./backend` – web application server

- `node_modules` – standard folder for Node.js dependencies
- `index.js` – main backend file (Node.js server startup)

– `package.json`, `package-lock.json` – supporting files for managing Node.js dependencies

– `emoji.json`, `emoji_twitter.json` – Unicode emojis database

– `glove.6B.50d.word2vec.bin` – word embedding model in Word2Vec format

`./frontend` – web application UI (HTML and CSS source code)

`./models` – LLM models server

– `Dockerfile` and `requirements.txt` – expansion to build a custom container with pre-installed requirements for running LLMs

– `main.py` – main backend file (Unicorn server startup)

– `stopwords.txt` – list of words that cannot represent a sentence

`./scripts` – files for initializing tables

– `convert-model.py` – utility script to convert from GloVe to Word2Vec model binary format

– `crawl-emojis.py` (followed up by `fill-in-tags.py`) – crawler script

– `init-tables.js` – table creation script

`init.sh` – automated setup

`docker-compose.yml` – configuration file for Docker Compose

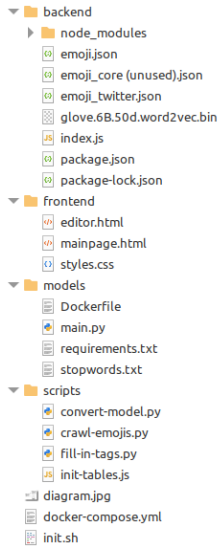

Figure 4: Structure

3.3 Implementation Details

This part provides an overview of its implementation.

1. **Step 1. Crawler and Database**

   In the first step, an emoji crawler script adapted to `https://emoji.gg` systematically collects image links, names, and tags and stores them into MongoDB. Starting from a seed URL, a `requests.Session` with custom headers fetches listing pages sorted by download count; `BeautifulSoup` then locates the relevant `<div>` elements to extract each emoji's detail-page URL. For every emoji found, the crawler delays between requests to respect server limits, visits its detail page to retrieve the tag list alongside the image link and name, and accumulates these records in batches. Each batch is bulk-inserted into a MongoDB collection. Duplicates are skipped automatically thanks to a unique index on the link field, and the process terminates early if a listing page yields no entries. This design achieves the goal of saving structured emoji data (`link`, `name`, `tags`) without downloading the image files themselves.

2. **Step 2. Emoji Search. Word2Vec**

   With the emoji database populated, the text editor's emoji button would show up a search field that sends a `GET` request to the Node.js endpoint **`/api/emoji-search`** whenever the user types. Query parameters define its behavior: `q` (the search query

input), `mode` (direct or smart), `database` (crawled or Unicode), and `type` (animated filter). In crawled database mode, the backend connects to MongoDB and applies a regex filter on names and tags; in smart mode, if there's no direct tag match but the tag exists in the Word2Vec vocabulary, it computes **cosine similarity** to find the nearest known tag (otherwise it falls back to the original query or returns null). Up to 100 results are retrieved, de-duplicated by name, and optionally filtered by animation type. In Unicode mode, the search filters a local `emoji.json` list by name, group, or subgroup—prioritizing exact matches then partials, and returns up to 100 standard emoji characters. This design leverages a pre-trained Word2Vec model to enrich search relevancy via semantic similarity.

3. **Step 3. Emoji Prediction. BERT**

   When the user types in the text editor and predictions are activated, the `POST` request is sent to **/api/emoji-predict**, carrying the last sentence before the caret in the body and two query parameters: `model` (choosing between the fine-tuned on Twitter data RoBERTa classifier [**3**] or the base DistilBERT embedding matcher) and `database` (crawled or Unicode). Upon receiving this request, the Node.js backend forwards it to the local LLM service at **/infer/model**. In crawled database mode, the predicted emoji is first mapped to a search tag – either one of 20 predefined labels (labels are taken from the Emoji Prediction task [**4**]) for the fine-tuned classifier or the raw name from the base model – and then an emoji search is performed as in Step 2; the first matching image link is returned. In Unicode mode, the system bypasses database lookup and immediately returns the standard emoji character. The LLM backend responds with emoji predictions in real time, because everything we initialize and calculate on the server startup: load emoji list and LLMs, precompute all emoji name embeddings (*embeddings matrix* represents all emoji names as dense vectors in the same semantic space, enabling similarity comparisons with future input sentences; emoji name is initially a sentence, like *"smiling face with open hands"*).

		Emoji	Emotion	Hate	Irony	Offensive	Sentiment	Stance	ALL
Val	SVM	25.0	63.8	73.1	63.4	72.7	68.4	67.9	62.0
	FastText	23.2	62.9	71.7	62.7	70.0	62.2	67.3	60.0
	BLSTM	19.4	62.6	72.1	60.6	72.1	61.9	63.4	58.9
	RoB-Bs	24.7±0.3 (24.3)	73.1±1.7 (74.9)	76.5±0.3 (76.6)	73.7±0.6 (73.7)	77.1±0.6 (77.6)	71.4±1.9 (72.7)	71.4±1.9 (73.9)	67.7
	RoB-RT	24.4±1.5 (**26.2**)	75.4±1.5 (**77.0**)	77.8±1.1 (**79.6**)	74.7±1.5 (**75.6**)	77.2±0.6 (**77.7**)	73.0±1.2 (**74.2**)	72.9±1.0 (**75.2**)	**69.4**
	RoB-Tw	23.4±1.1 (24.6)	67.6±0.9 (68.6)	74.3±2.0 (76.6)	70.0±0.3 (70.7)	76.1±0.6 (76.2)	70.5±1.0 (69.4)	68.3±2.4 (71.4)	65.4
Test	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
	SotA	36.0*	-	65.1	70.5	82.9	68.5	71.0	-
Metric		M-F1	M-F1	M-F1	$F^{(i)}$	M-F1	M-Rec	AVG ($F^{(a)}$,$F^{(f)}$)	TE

Figure 5: TweetEval validation and test results

For reference, TweetEval introduces a RoBERTa-based model fine-tuned for emoji prediction on 50,000 English tweets, as part of a unified benchmark (7 classification tasks for tweets, see Fig. 5). This is the model that was used for my project.

4 Conclusion and Future Work

This paper demonstrates the successful design and implementation of a web-based text editor with native integration, search and prediction of custom and animated emojis. Evaluation under normal typing speed shows average emoji-search latencies of approximately `150 ms` and prediction response times around `1000 ms`, highlighting responsiveness in real-time use.

Future work will focus on enhancing and extending the core capabilities of Search2Emoji. Planned improvements include:

- **Advanced embedding models**: find and test models that might be more efficient for sentence embeddings (e.g. `sentence-transformers/all-MiniLM-L6-v2`, used by other project team).
- **Reusable library**: package the editor as an includable web component to bring Search2Emoji into other applications with minimal effort.
- **Editable emojis**: find and test generative AI models to edit emojis with text prompts. Possibly involves fine-tuning.

The project successfully met all requirements (creating a working search engine), providing valuable experience in using key Information Retrieval techniques. Future updates will be available at `Project Repository`.

5 Project Manual

The project can be cloned from my GitHub repository:
`git clone git@github.com:fata1error404/tsinghua-web-ir.git`

You must have Docker and Docker Compose installed; refer to documentation [**5**] for more details. Docker runs individual containers, while Docker Compose manages multi-container applications using the `docker-compose.yml` configuration. Execute the commands in the terminal and open the Web UI in a browser:

First time initialization
`chmod +x init.sh` (to make init.sh script executable)
`./init.sh` – automated setup
`python scripts/crawl-emojis.py` – populate Emoji table (manually terminate the script once you have the desirable number of emojis)
`python scripts/fill-in-tags.py` – populate Tags table

To start the text editor for subsequent launches
`docker compose up`

Accessing the Web UI
`http://localhost:3000` – web application

The emoji database can be connected to and managed using MongoDB Compass. To connect, create a new connection and use the following URI: `mongodb://localhost:27017/`.

References

[1] Dubé S., Gesselman A., Kaufman E., Bennett-Brown M., Ta-Johnson V., Garcia J. *Beyond words: Relationships between emoji use, attachment style, and emotional intelligence*, 2024
https://doi.org/10.1371/journal.pone.0308880

[2] *Unicode Emoji Database*
https://unicode.org/emoji/charts/full-emoji-list.html

[3] F. Barbieri, J. Camacho-Collados, L. Espinosa Anke, and L. Neves *TweetEval: Unified Benchmark and Comparative Evaluation for Tweet Classification*, 2020
https://arxiv.org/abs/2010.12421

[4] *SemEval-2018 Task 2, Multilingual Emoji Prediction*
https://competitions.codalab.org/competitions/17344

[5] *Docker Documentation*
https://docs.docker.com/

[6] *MongoDB Documentation*
https://www.mongodb.com/docs/