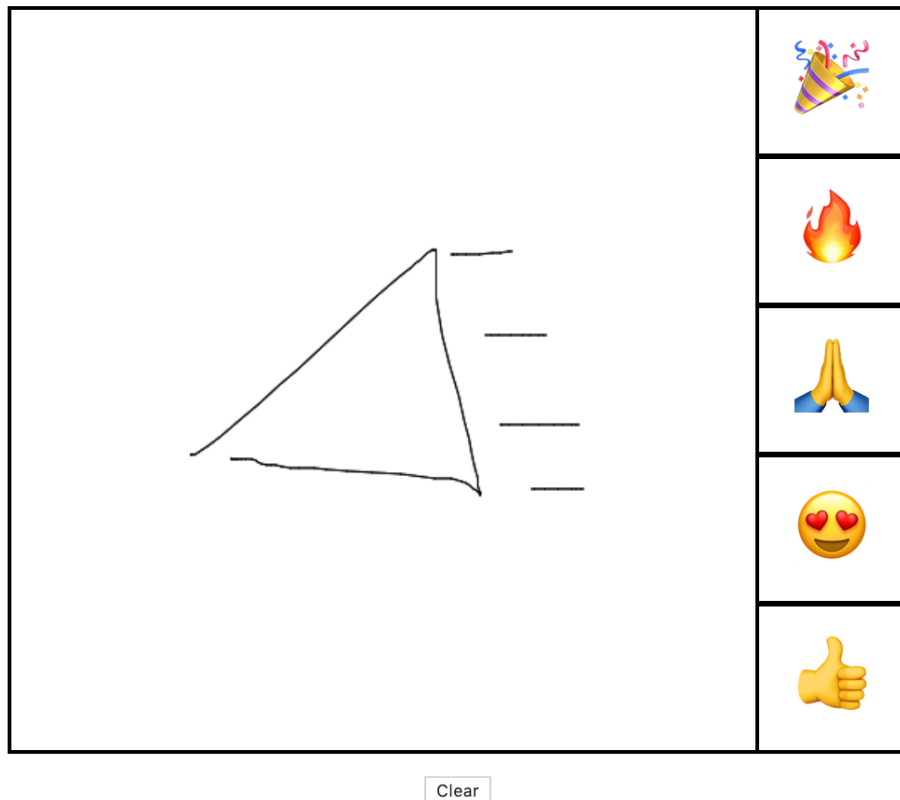


# SketchMoji

Sophie Mori and Margaret Sands

## SketchMoji



We built a way to recognize emoji characters from sketches drawn by a user. The user draws a desired emoji in the SketchMoji drawing surface, and the top five most similar results are returned for the user to copy and use. SketchMoji uses a HTML/JavaScript frontend and a python backend, which are combined with a Flask server API.

We achieved reasonable performance when defining success as having the desired emoji show up in the top three results, out of a bank of 18 total pre-selected emoji, when conducting user studies. The users averaged a success rate of 16/18. We believe that a sketch-based system to recognize emoji is a viable interface, but that there would need to be a lot more templates generated and readily available for comparison before the system can handle the hundreds of emoji in common use.

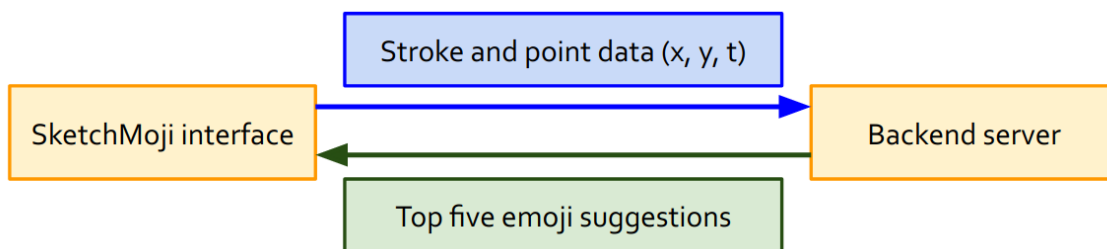
# 1.Introduction

Digital media pushed our conception of text-based communication beyond mere words. Nowadays, users of digital technology have found another way to express intent and emotion—through the use of emoji, pictorial ideograms that fall in line with traditional text. Emoji use proliferates social media and instant messaging, but it is also infiltrating web page content and even mainstream media. Emoji constitute a large part of popular culture, and the 😂 emoji was named Word of the Year by Oxford Dictionaries in 2015.

Currently, the ways to input emoji characters into text on a computer are lacking in usability and efficiency. The user can memorize and utilize custom hotkey combinations on a keyboard; Google a description of the emoji, find the character, copy, and past; or scroll through long, unintuitively sorted menus that some interfaces employ. In systems where each emoji has a named shortcut (e.g. :zipper\_mouth\_face: in Slack), it is difficult to recall or discover the exact name, and the user is resigned to one of the aforementioned methods.

We propose a system that allows users to discover their intended emoji by "drawing" it. Because emoji are rooted in images, we believe this is a more natural and intuitive way to input emoji into text.

## 2.System Design



From the user's end, SketchMoji is a very simple interface. To use SketchMoji, the user visits the web application and uses their computer cursor to draw their conceptual notion of the desired emoji in the drawing area. With each stroke drawn, the system compares the sketch to those of all emoji it knows about, and returns the closest five, ordered by similarity. The user can copy the desired emoji at any point to their clipboard, paste it into the intended destination, and resume their workflow.

The backend takes the time stamped positional data of the drawing, finds the segments in the point data, and then compares critical points to templates to arrive at a classification. These steps are further explained in the following section.

### 3. How it works



The backend system is mostly based on the sketch recognition mini project. The drawing area captures the user's cursor positions and timestamps as they sketch the emoji. This information is fed into a segmenter, which analyzes the points for curvature and pen (cursor) speed to derive the segmentation points and intended segments of the sketch. We calculate the points of interest—the endpoints and midpoints of segments—and compare those to the points from templates we drew for each emoji. The emoji with templates that are the closest given our distance metric are sent back to the interface for the user to choose from.

Sketchmoji's ability to recognize emoji varies heavily with the particular emoji. The simple ones, such as the checkmark ✓, party popper 🎉, and heart ❤️, are almost always the "most similar" returned result. More complicated emoji, such as a face emoji 😍, have a much lower accuracy rate due to a dependence on how similarly the user's mental notion of the emoji matched with those of the template drawers'.

We started off with a small set of nine emoji and eventually expanded that to a set of 18: 🎉🔥😄👍❤️🙌😍😭✓🍌💩👉💔💪💦🐱🐼💤. We chose these particular emoji based on their usage (<http://www.emojistats.org/>) and also with care that they were not all drastically distinct, which would have trivialized the recognition task. In particular, we have five face-based emoji.

#### Experimentation

We immediately recognized that emoji are drawn with multiple strokes, and we cannot expect users to draw those strokes in a certain order. Thus, we adapted the system to handle multiple stroke data for a single sketch and performed its classification independently of any anticipated stroke order.

The modified Hausdorff distance metric served us well from the start. We also tried binary pixel-based metrics such as the Tanimoto coefficient mentioned in Kara and Stahovich. The main issue with this approach was that it was difficult to engineer a robust way to transform our data into a black and white pixel grid, and the grid was too sparse in black pixels and there was too much variance in how detailed the users drew the emoji for this metric to improve upon our previously defined metric.

We realized that users draw emoji with high variance in style and direction. Different systems interpret emoji differently, from orientation to the exclusion or inclusion of certain features such as eyebrows. Originally, we considered polar transformations as a way to account for rotation. However, confounding factors such as different conceptual interpretations of features were also an issue when transformed.

We came up with the solution for the system to recognize multiple templates for a given emoji, as rotational variances tend to be finite and small (the user can draw either the right arm or left arm for 🦵, but other rotations are extremely unlikely). This approach also allowed us to capture conceptual differences.

One interesting failure is in the segmentation process. For emoji where the user draws just a small dot for an eye, the smoothing process causes the small points to be ignored, and the eyes do not end up in the template. It would be useful to update the point of interest calculation with a way to deal with independent, short separate strokes.

The system also fails when users themselves fail to distinguish their own drawings of two separate emoji (they draw two distinct emoji the same way). This is an axiomatic issue, and the system attempts to be robust against this by showing a palette of similar emoji.

### **Implementation challenges**

Designing an HTML-based drawing surface was a lot of trial and error. Particular HTML/JavaScript idiosyncrasies and APIs caused us to tinker around with various issues such as asynchronous calls, copying text from a non-input tag element, and updating existing elements on the page.

The repurposing of the mini project code was both harder and easier than expected. The code itself was very modular, and it was easy to add in new functionality into or between certain modules. On the other hand, the incorporation led to unintended side effects that we had to debug. For example, matplotlib causes Flask to break, and all matplotlib code must be removed when we were testing with the server.

## **4. Modifications**

The fully streamlined vision for SketchMoji, as envisioned in our original proposal, would be a browser extension or a native application that would insert this drawing area and emoji suggestion list as an overlay on top of whatever application the user is already using. The overlay could be triggered by a hotkey, and clicking on the desired emoji would insert it into the text at the cursor location, eliminating the need to navigate to a separate application and going through copy and paste motions. Because our focus

resided in seeing if we could reliably recognize emoji through sketch and not engineering a browser extension, we settled on this standalone web application.

One of our original ideas was to have two modes in the application: the recognizer, which would try to recognize the emoji the user is drawing and the recommender, which would suggest some emoji similar to what the user is drawing. Having the recommender means that users could potentially discover new emoji they were not aware of to use. However, we realized that the functionality for the recognizer mode is improved by recommending a list of the most similar matches. The system also essentially returns the same information for both modes. Since the overall experience is streamlined significantly by having a single mode that serves both functionalities, we removed the notion of separate modes.

We believe our system can be improved with a robust variety of template drawings, and one roadblock was collecting those from different sources. We did not modify our project, per se, but we did have to make do with a more limited set of templates.

## 5. User Studies



















We were fortunate enough to be able to conduct live user studies on some roommates. From the first user test, we learned that the name "SketchMoji" and the prototype interface were enough to tell the user what to do with the interface. However, it also showed us that users focus on different features drawing an emoji and that our single-template approach was not comprehensive enough to cover all of those possibilities. In our next iteration, we added the multi-template matching approach.


For our second user study, we had each user sketch an emoji (in one try) until the expected emoji was in the top three suggestions. If the user could not create a sketch that gave the correct emoji as a suggestion, we marked that down as a failure and the user moved on to the next emoji. The results of this experiment will be discussed in performance. Outside of finding the accuracy of the system, this user study gave us additional interesting observations.

Users with a digital mouse could draw at incredibly fast speeds. Because these users did not slow down at inflection points, many points of interest were missed, causing the suggestions to be less accurate. We further tuned the threshold parameters to increase the cutoff speed of inflection points to accommodate those quick sketches.

User studies also affirmed our intuition that we should be displaying matches after each stroke and that we should be showing multiple suggestions. In normal use, the user abandons an incomplete sketch once the desired emoji shows up. This means that the ultimate *exact* ranking of the emoji is not necessarily relevant as a usability metric.

## 6. Performance

| Rankings of desired emoji once in the top 3. A blank cell indicates the emoji was not in the top 3. |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Emoji   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| User 1  | 3   | 2   | 3   |   | 2   | 2   | 3   | 2   | 1   | 1   | 3   | 3   | 2   | 1   | 3   | 3   | 3   | 1   |
| User 2  | 2   | 2   | 2   |   | 1   | 1   |   | 2   | 1   | 1   | 3   | 1   | 3   | 1   | 1   | 2   | 1   | 1   |
| User 3  | 2   | 2   | 3   |   | 1   |   | 2   | 1   | 1   | 1   | 2   | 3   | 2   | 2   | 1   | 1   | 1   | 1   |

We measured our performance through user testing. The mean score across all users for the test described in the user study section was 16/18, which meant that on average, SketchMoji failed to display two emoji in the top three results. The ranking for each emoji is actually an upper bound because the user stopped sketching once their desired emoji entered the top three results. Had they completed the sketches, we can expect those ranking values to decrease (become more accurate). We found that one particular emoji () could not be produced in many trials. This indicates that we may have had a template issue, and we could have gone back and updated that.

We cannot attest to the robustness of our system if it had to handle magnitudes more emoji—hundreds, thousands. Through our experimentation, we were able to most drastically improve performance by updating and adding more template data. It is possible that the system will still work reasonably well with many more emoji, given large numbers of high quality template data that would capture variation in the more general public's mental notions of how emoji look.

The abundance of system-based variations in how emoji look is a large issue of which we originally did not fully understand the scope. This need for large amounts of template data is certainly a limit of our system, but given the nature of the specification (implementers are free to interpret emoji however they would like), it is intrinsic to the problem of recognizing emoji. One potential solution is to adapt SketchMoji for different systems, say, distinct versions specifically tuned for Apple versus Samsung devices.

Stepping back from the very labor-intensive approach of collecting more template data to accommodate more emoji, there are still methods we can try to improve our current system. We could create a teaching mode where a new user can add their own custom templates which capture their unique mental interpretations of certain emoji, improving the system's ability to later recognize that emoji when the user needs to find it. This teaching mechanism does not have to be explicit either—perhaps simply selecting an emoji while sketching it is enough to tell the system to store that input as a template for future comparisons. These ideas do not necessarily contradict our goal of discoverability,

either, as the user still discovers new emoji by experimenting with sketches and seeing the intermediate results.

Many of the issues and challenges we addressed when working on this project were not anticipated. More than anything, we learned that engineering and developing a system like this cannot be fully envisioned from the beginning. Rather, we had to come up with ad hoc solutions, and not all our revisions were improvements either. Unlike theoretical problems that can be simulated and anticipated, this type of engineering challenge required more flexibility and adaptive solutions. This is our main takeaway.

## 7. Tools, Packages, and Libraries

The system uses a HTML/JavaScript frontend with a backend Python server. The classification code requires some common packages such as numpy and scipy. We use Flask (<https://flask.palletsprojects.com/en/1.1.x/>) to serve the application and set up endpoints. The resulting application takes in GET and POST requests from a specific url, runs the corresponding functions, and returns the response. One potential issue is that local versions can hit errors with cross origin requests; future iterations would need to install Flask\_cors and configure their app accordingly.

## 8. Contributions

Initially, our SketchMoji interface, and by extension our contributions, could be split into two parts. Sophie worked on the interface itself, how it looked, and displayed and collected data. Margaret focused on repurposing Sophie's MP1 code and setting up the Flask server. After the initial prototype, Margaret created additional templates to expand the emoji set and worked on segmenting strokes separately, and Sophie looked into additional metrics and ways to transform the data. Both contributors ran user studies and made improvements as various issues arose.

We would also like to credit our SketchMoji interface for generating all the emoji characters present in this paper.

Thank you, instructors, for providing feedback, supporting us this semester, and running studios in an outstanding virtual format.