EmojiCloud: a Tool for Emoji Cloud Visualization

Yunhe Feng
University of Washington
yunhe@uw.edu

Cheng Guo
University of Washington
cguo3@uw.edu

Bingbing Wen
University of Washington
bingbw@uw.edu

Peng Sun
CUHK, Shenzhen
sunpengzju@gmail.com

Yufei Yue[*]
Amazon.com
heurekayue@gmail.com

Dingwen Tao
Washington State University
dingwen.tao@wsu.edu

Abstract

This paper proposes EmojiCloud, an open-source Python-based emoji cloud visualization tool, to generate a quick and straightforward understanding of emojis from the perspective of frequency and importance. EmojiCloud is flexible enough to support diverse drawing shapes, such as rectangles, ellipses, and image masked canvases. We also follow inclusive and personalized design principles to cover the unique emoji designs from seven emoji vendors (e.g., Twitter, Apple, and Windows) and allow users to customize plotted emojis and background colors. We hope EmojiCloud can benefit the whole emoji community due to its flexibility, inclusiveness, and customizability.

1 Introduction and Background

Emojis play a significant role in social business listening, sentiment analysis, cross-language communication, and politics. People from different language and cultural backgrounds love emojis and use them very frequently. According to a recent survey (Team, 2015), almost everyone online had experience in using emojis. It is important to generate a fast and straightforward understanding of emojis in many research and applications.

Inspired by the word cloud (Bielenberg and Zacher, 2005; Dubinko et al., 2007), which has be adopted as an effective way to visualize the frequency and importance of words in text mining, we thought the word cloud of emojis seemed to be a good solution. However, as shown in Figure 1, the word cloud will change and modify emojis' original and important features, such as colors (❤️⚽), directionalities (😂🌺), and textures (🙌💚). These inaccurate emoji representations may lead to misunderstanding. For example, when emojis 🙂👍 are upside down, they turn into 🙃👎 that conveys different sentiments and meanings. Also, miscolored emojis such as 🧑🧑🧑 may cause the problem of personal identity representations.

In addition, the word cloud of emojis fails to capture the diversity of emoji appearances customized by emoji vendors. As different emoji renderings across viewing platforms may cause communication errors and diverse interpretations (Miller Hillberg et al., 2018; Miller et al., 2016), it is very important to support vendor-wise emoji visualization. Although several online platforms, such as Talkwalker[1], Slido[2], and Poll Everywhere[3], offer emoji cloud services for various needs, they are not open-source and fail to provide APIs or functions for flexible usages in text mining. Moreover, these services are just targeting one emoji cloud canvas shape and one emoji vendor respectively.

Figure 1: Word cloud of emojis

In this paper, we design and implement EmojiCloud, a counterpart of the word cloud for emoji visualization. Instead of plotting words, EmojiCloud draws emoji images in a clear and cloud-like fashion and keeps all detailed emojis features. EmojiCloud is flexible to support diverse canvases, such as rectangles, ellipses, and image-masked shapes. It also enables users to customize emoji clouds by specifying emoji vendors (e.g., Twitter, Google, and Apple) and individual self-made emoji images when creating emoji clouds. As the first open-source Python-based emoji cloud visualization tool (to our best knowledge), EmojiCloud facilitates the understanding of emojis and will bring broader impacts of emojis in many domains. We believe it is

---

[*]The work does not relate to his position at Amazon.

[1]https://www.talkwalker.com/blog/emoji-analysis-crack-consumer-code

[2]https://whatsnew.slido.com/en/say-it-with-an-emoji-

[3]https://blog.polleverywhere.com/emoji-quiz-online/

a valuable and important tool to the emoji community and even the text mining community.

2 EmojiCloud Design & Implementation

This section presents emoji image preparation, EmojiCloud layout designs, and implementation.

2.1 Emoji Image Retrieval and Preprocessing

As emoji vendors, such as Twitter, Apple, and Google, can implement emoji designs into their products, emojis encoded with the same Unicode characters may demonstrate distinct appearances across platforms. To make EmojiCloud accurate and inclusive, we take emoji appearance variances across diverse vendors into consideration. Specifically, we propose an emoji image retrieval framework that collects cross-vendor emoji data from the official website of Unicode Full Emoji List[4]. The framework crawls and stores the latest emoji images provided by seven vendors (i.e., Apple, Google, Meta, Windows, Twitter, JoyPixels, and Samsung) automatically. Considering new emojis are always requested by users (Feng et al., 2019) and the Unicode Consortium releases new emojis every year accordingly, the proposed framework is able to check and download newly added emoji images when an emoji image cannot be found in local storage.

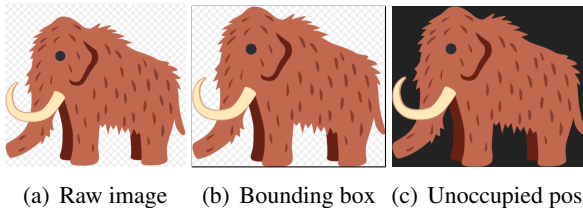(a) Raw image   (b) Bounding box   (c) Unoccupied pos.

Figure 2: Preprocessing original emoji images by determining bounding boxes and marking unoccupied pixel positions (colored as black in Figure 2(c))

The retrieved emoji images need to be preprocessed to remove white-colored surrounding pixels that may cause a sparse emoji layout in EmojiCloud. As all retrieved emoji images are formatted in PNG, all these white-colored surrounding pixels have an alpha value of zero, representing full transparency. We propose a two-step transparency-based white space removal approach to reserve meaningful emoji pixels. First, as shown in Figure 2(b), we calculate the bounding box of an original emoji image by removing surrounding white-

[4]https://unicode.org/emoji/charts/full-emoji-list.html

colored rows and columns. Second, within the bounding box, we mark the positions of pixels that are not part of the emoji representation (see the black pixels in Figure 2(c)) as unoccupied. We use E and U to represent the emoji image pixel values and the pixel unoccupied statuses, where $E_{x,y}$ represents the emoji pixel value at the coordinate (x, y) and $U_{x,y} \in \{0, 1\}$ indicates whether the pixel at (x, y) is unoccupied ($U_{x,y} = 1$) or not ($U_{x,y} = 0$). The above two image operations help create a compact emoji cloud layout. Note that white-colored pixels composing emoji representations, such as the white pixels inside the cooked rice emoji 🍚, are not touched because they have a positive alpha value.

2.2 EmojiCloud Layout Design

This section presents how to determine emoji sizes by frequency weights and design emoji layouts.

2.2.1 Emoji Size Calculation

We use a quintuple $e = (a, b, w, E, U)$ to represent an emoji, where a, b, w are the width, height, and edge-level frequency weight of emoji e. Recall that E and U represent pixel values and pixel unoccupied statuses. Suppose we have a list of emojis $\mathbf{e}$ to create an emoji cloud, where the ith emoji in $\mathbf{e}$ is expressed as $e_i = (a_i, b_i, w_i, E^i, U^i)$. The image sizes of all emojis have been standardized, i.e., $a_i * b_i = c; \forall i \in [1, |\mathbf{e}|]$, where c is a constant and $|\mathbf{e}|$ is the total count of emojis.

To ensure all emojis $\mathbf{e}$ can be drawn on the canvas without overlapping, we must adjust the weighted emoji plotting sizes with an edge rescale ratio r. Let's say the drawable emoji cloud canvas size is s. The rule of thumb is that r satisfies the following inequality.

$$s \geq \sum_{i=1}^{|\mathbf{e}|} w_i^2 * a_i * b_i * r^2 = \sum_{i=1}^{|\mathbf{e}|} w_i^2 * c * r^2 \quad (1)$$

where a *possible* maximum edge rescale ratio r can be $\sqrt{s/(c * \sum_{i=1}^{|\mathbf{e}|} w_i^2)}$. Thus, the rescaled width and height of emoji e_i are expressed as $a'_i = a_i * w_i * r$ and $b'_i = b_i * w_i * r$. The edge rescale ratio r decays at a rate of 0.9 if there is not enough room to plot all emojis on the canvas (see Line 36 in Algorithm 1).

2.2.2 Emoji Layout

Suppose we have a canvas with a $m * n$ rectangle bounding box in pixel. We use C and V to repre-

Algorithm 1: EmojiCloud Layout

1 Input: **e**: a list of emojis; (m, n, s, C, V): a canvas with width m, height n, drawable size s, pixel values C, and pixel painting eligibility V on the canvas; c: the standardized size of emoji images;

2 Output: C: an emoji cloud image;

3 $\mathbf{e} \leftarrow$ sort the emoji list **e** by emoji weights $\mathbf{w} = [w_1, w_2, .., w_{|\mathbf{e}|}]$ in reverse order;

4 $r \leftarrow \sqrt{s/(c * \sum_{i=1}^{|\mathbf{e}|} w_i^2)}$;   // rescale ratio in Equation 1

5 for $x = 1 \rightarrow m$ **do**   // x coordinate of canvas

6   **for** $y = 1 \rightarrow n$ **do**   // y coordinate of canvas

7   **if** $V_{x,y} = 1$ **then**   // canvas pixel is eligible for painting

8   append (x, y) into the canvas pixel coordinate list $\mathbf{p_c}$;   // build $\mathbf{p_c}$

9 $\mathbf{p_c} \leftarrow$ sort $\mathbf{p_c}$ by the Euclidean distance between $(x, y) \in \mathbf{p_c}$ and the canvas center $(m/2, n/2)$;

10 $count \leftarrow 0$;   // count of plotted emojis

11 while $count < |\mathbf{e}|$ **do**   // not all emoji have been plotted

12 $count \leftarrow 0$;   // no emoji has been plotted

13   **for** $i = 1 \rightarrow |\mathbf{e}|$ **do**   // iterate the emojis sorted by weights in reverse order

14 $(a_i, b_i, w_i, E^i, U^i) \leftarrow e_i$;   // parse e_i into its quintuple representation

15 $a_i' \leftarrow a_i * w_i * r$; $b_i' \leftarrow b_i * w_i * r$;   // rescale width and height of e_i

16 $E^{i'}, U^{i'} \leftarrow$ update E^i, U^i based on r ;   // rescale E^i, U^i based on r

17   **for** $(x, y) \in \mathbf{p_c}$ **do**   // (x, y) is where the center of e_i to be located

18 $flag \leftarrow True$;   // indicate the possibility of plotting e_i

19 $\mathbf{p_t} \leftarrow [\ ]$;   // a list of canvas temporal coordinates to plot e_i

20   **for** $x' = 1 \rightarrow a$ **do**   // x coordinate of emoji image

21   **for** $y' = 1 \rightarrow b$ **do**   // y coordinate of emoji image

22   **if** $U_{x',y'}^{i'} = 0$ **then**   // emoji pixel (x', y') is not unoccupied

23 $x_o \leftarrow x' - a_i'/2$; $y_o \leftarrow y' - b_i'/2$;   // the offsets to e_i center

24 $x_t \leftarrow x + x_o$; $y_t \leftarrow y + y_o$;   // canvas temporal coordinate for e_i

25   append (x_t, y_t) to $\mathbf{p_t}$;

26   **if** $V_{x_t, y_t} = 0$ **then**   // canvas pixel is not eligible for painting

27 $flag \leftarrow False$;   // no room to plot e_i at (x, y)

28   break;   // iterate the next $(x, y) \in \mathbf{p_c}$

29   **if** $flag = True$ **then**   // the emoji e_i can be plot at (x, y)

30   **for** $(x_t, y_t) \in \mathbf{p_t}$ **do**   // iterate temporal pixel coordinates

31 $C_{x_t, x_t} \leftarrow E_{x_t - x + a_i'/2, y_t - y + b_i'/2}^{i'}$;   // plot emoji e_i

32 $V_{x_t, y_t} \rightarrow 0$;   // set painting eligibility as negative

33   remove (x_t, y_t) from $\mathbf{p_c}$;   // delete (x_t, y_t) for computing efficiency

34 $count \leftarrow count + 1$;   // increase the number of plotted emoji by 1

35   break;

36 $r \leftarrow r * 0.9$;   // decay the edge rescale ratio by 0.9

37 return C

sent pixel values and the pixel painting eligibility on the canvas. To be more specific, $C_{x,y}$ represents canvas pixel values at the coordinate (x, y) and $V_{x,y} \in \{0, 1\}$ indicates the painting eligibility of (x, y), where $x \in [1, m]$ and $y \in [1, n]$. The design of V controls the drawable shape (e.g., a circle or an ellipse) on the canvas (see section 2.3 for details). As it is possible that not all pixel coordinates are eligible for painting, the drawable canvas size s in Equation 1 does not always equal to $m * n$. Thus, a canvas can be expressed as (m, n, s, C, V).

For aesthetic purposes, we arrange an emoji with a larger weight (indicting more importance) closer to the canvas center, where more attention is usually given. First, we sort the emoji

list e based on their corresponding weights in reverse order (see Line 3 in Algorithm 1). Then, we sort $\mathbf{p_c}$, a list of canvas pixel coordinates (x, y) that are eligible for painting emojis (i.e., $V_{x,y} = 1$), by their Euclidean distances to the canvas center $(m/2, n/2)$. For each sorted emoji $e_i \in \mathbf{e}$, we rescale its original representation $(a_i, b_i, w_i, E^i, U^i)$ into $(a_i', b_i', w_i', E^{i'}, U^{i'})$ using the edge rescale ratio r.

Next, we attempt to draw emoji e starting from the canvas pixel coordinate $(x, y) \in \mathbf{p_c}$ that is closest to the center of the canvas. As the center $(a_i'/2, b_i'/2)$ of emoji e will be mapped at (x, y) on the canvas, the rest emoji pixel coordinate (x', y') will be mapped at $(x_t = x + x_o, y_t = y + y_o)$, where

x_o and y_o are offsets of $x' - a'_i/2$ and $y' - b'_i/2$. If any occupied emoji pixel coordinate (x', y') (i.e., $U^{i'}_{x',y'} = 0$) fails to be mapped to the canvas coordinate (x_t, y_t) (i.e., $V_{x_t,y_t} = 0$), we continue to check the next canvas pixel coordinate that is the second most closest to the center of canvas (see Line 17-28 in Algorithm 1).

When emoji e_i can be plotted successfully on the canvas ($flag = True$), we copy and paste each pixel value in $E^{i'}$ into the canvas C. In addition, the painting eligibilities of the involved pixel coordinates on the canvas are set as 0. For computing efficiency, we delete corresponding pixel coordinates from the sorted $\mathbf{p_c}$ and increase the count of plotted emojis (see Line 29-35 in Algorithm 1).

2.3 Canvas Design

The proposed EmojiCloud is flexible to support diverse drawable canvas shapes, including rectangle, ellipse, image-masked, and arbitrary canvases.

2.3.1 Default Canvas

We set the default canvas shape as an $m * n$ rectangle, and all pixel coordinates within the rectangle are eligible to draw emojis. The painting eligibility $V_{x,y}$ is set as 1 for all $x \in [1, m]$ and $y \in [1, n]$.

2.3.2 Ellipse Canvas

Suppose we have a drawable ellipse area within an $m * n$ rectangle bounding box for plotting emojis. The semi-major and semi-minor axes' lengths are expressed as $m/2$ and $n/2$. The center pixel coordinate is expressed as $(m/2, n/2)$. If pixel coordinate (x, y) on canvas satisfies the following inequality, $V_{x,y}$ is set as 1.

$$(\frac{x - \frac{m}{2}}{\frac{m}{2}})^2 + (\frac{y - \frac{n}{2}}{\frac{n}{2}})^2 \le 1 \tag{2}$$

Otherwise, the coordinate (x, y) is outside of the ellipse, and the corresponding $V_{x,y}$ is set as 0. When m equals n, a circle canvas is defined.

2.3.3 Masked Canvas

EmojiCloud also allows users to specify a masked canvas based on a PNG background image. Similar to the emoji image preprocessing in Section 2.1, we first determine a $m * n$ bounding box of the image by removing the surrounding transparent pixels. Then we detect the image contour and draw a boundary accordingly (e.g., converting 🐦 into 🐦). To be more specific, we scan the alpha values of pixels in the preprocessed image by row and by

column respectively. Recall that the alpha channel in PNG controls pixel transparency. During the scanning, we identify pixels that cause a alpha value change greater than a threshold θ (by default $\theta = 10$) as boundary pixels. After all boundary pixels are determined, they will be colored by specified colors. If one pixel coordinate (x, y) is inside the boundary, the corresponding $V_{x,y}$ is set as 1.

2.3.4 Arbitrary Canvas

Users are allowed to specify arbitrary canvas drawable shapes by configuring the painting eligibility $V_{x,y}$ for pixel coordinate (x, y) on the canvas.

2.4 EmojiCloud Inclusive Design

EmojiCloud is flexible and inclusive to handle emoji images designed by seven vendors (i.e., Apple, Google, Meta, Windows, Twitter, JoyPixels, and Samsung.) We provide an option for users to specify the vendor of interest. In addition, users can customize and combine emojis based on their requirements. For example, a red apple emoji (U+1F34E) 🍎 can be replaced by 🍎 for marketing campaigns. The sauropod (U+1F995) 🦕 and T-Rex emoji (U+1F996) 🦖 can be combined as 🦖 if it is not necessary to distinguish dinosaur species.

2.5 Implementation and Open Source

We develop open-source EmojiCloud in Python, one of the most popular programming languages for natural language processing and text mining. EmojiCloud has been packaged as a Python library and published through Python Package Index (PyPI). Users can run `pip install EmojiCloud` to install the EmojiCloud package and use `from EmojiCloud import EmojiCloud` to call for EmojiCloud functions in Python scripts. An EmojiCloud tutorial is available at **https://pypi.org/project/EmojiCloud/**.

3 EmojiCloud Evaluation

In this section, we demonstrate that EmojiCloud is able to support diverse canvas shapes, different emoji vendors, and customized emoji images.

3.1 Visualization on Different Canvases

EmojiCloud allows users to select the drawable canvas shapes to generate emoji cloud images. As shown in Figure 3, we plot an identical list of emojis (with the same weights) on rectangle, ellipse, and image masked canvases. Emojis with large weights are placed as close to the canvas center as

possible. To make the emoji cloud image compact, emojis with small weights can also be placed near the canvas center if there is enough room.

(a) Rectangle   (b) Ellipse   (c) Mask

Figure 3: EmojiCloud on different canvases

3.2 Visualization for Different Emoji Vendors

EmojiCloud is flexible to cover seven different emoji vendors. Users can specify the vendor when creating EmojiCloud images. Figure 4 shows the generated EmojiCloud images of the same emoji list for Twitter, Apple, Google, Windows, JoyPixels, Meta, and Samsung. Although vendors demonstrate different layout patterns, heavy-weight emojis are always placed in the center zone, where people usually pay more attention.
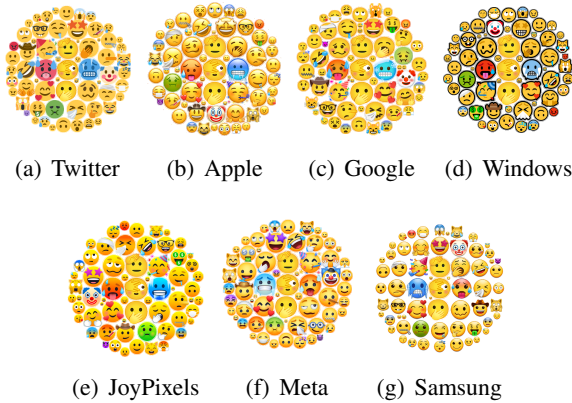

(a) Twitter   (b) Apple   (c) Google   (d) Windows

(e) JoyPixels   (f) Meta   (g) Samsung

Figure 4: EmojiCloud for different vendors

3.3 Visualization of Customized Emojis

Besides the above seven vendors, EmojiCloud supports arbitrary emoji representations designed and specified by users. Figure 5 demonstrates an EmojiCloud case of FIFA World Cup, where the default trophy (U+1F3C6) 🏆 is customized by 🥇. Also, EmojiCloud allows users to customize the canvas background (see the color of grass on soccer fields in Figure 5). Customized emojis make EmojiCloud more appropriate and accurate to depict the investigated case studies.

3.4 Running Time Evaluation

The running time of EmojiCloud depends on how many emojis are plotted and the emoji cloud canvas
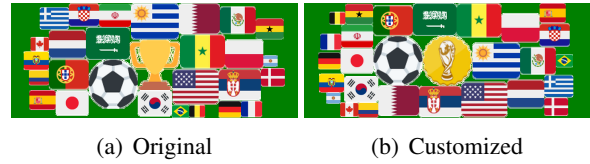

(a) Original   (b) Customized

Figure 5: EmojiCloud for FIFA World Cup Trophy

size. We measured the running time of EmojiCloud on a laptop with an AMD Ryzen 7 4800HS processor and 16 GB RAM. We evaluated the emoji count from 25 to 60 with an increasing step of 5 on canvases of 300*300, 400*400, and 500*500 pixels. EmojiCloud with the same setting ran 10 times to ensure the running time was accurate. As shown in Figure 6, the running time generally increases as emoji counts and canvas sizes increase. An emoji cloud (containing up to 60 emojis) on a 300*300 pixels canvas can be generated within 5 seconds.
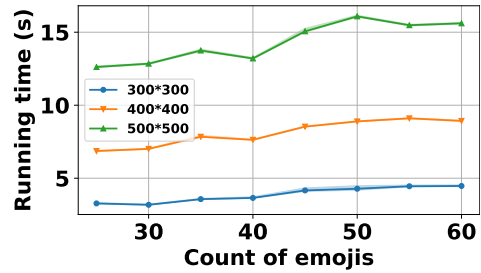

Figure 6: Running time of EmojiCloud

4 Conclusion and Future Work

We propose and develop open-source EmojiCloud to visualize a cluster of emojis according to associated frequency weights. EmojiCloud enables a novel and informative way to investigate emojis in natural language processing and text mining. We think EmojiCloud will benefit the whole emoji community due to its flexibility, inclusiveness, and customizability.

In the future, we will keep updating the open-source EmojiCloud based on the users' feedback, such as adding new functions and covering more emoji vendors. To further improve the flexibility and convenience of EmojiCloud, we will provide an online EmojiCloud service via **www. emojicloud.org**. In addition, we will explore the possibility of merging words and emojis in a unified word-emoji cloud.

References

Kai Bielenberg and Marc Zacher. 2005. Groups in social software: Utilizing tagging to integrate individual contexts for social navigation. *Digital Media. Bremen, Germeny, University Bremen. Master of Science in Digital Media*, 120.

Micah Dubinko, Ravi Kumar, Joseph Magnani, Jasmine Novak, Prabhakar Raghavan, and Andrew Tomkins. 2007. Visualizing tags over time. *ACM Transactions on the Web (TWEB)*, 1(2):7–es.

Yunhe Feng, Wenjun Zhou, Zheng Lu, Zhibo Wang, and Qing Cao. 2019. The world wants mangoes and kangaroos: A study of new emoji requests based on thirty million tweets. In *The World Wide Web Conference*, pages 2722–2728.

Hannah Jean Miller, Jacob Thebault-Spieker, Shuo Chang, Isaac Johnson, Loren Terveen, and Brent Hecht. 2016. "blissfully happy" or "ready tofight": Varying interpretations of emoji. In *Tenth international AAAI conference on web and social media*.

Hannah Miller Hillberg, Zachary Levonian, Daniel Kluver, Loren Terveen, and Brent Hecht. 2018. What i see is what you don't get: The effects of (not) seeing emoji rendering differences across platforms. *Proceedings of the ACM on Human-Computer Interaction*, 2(CSCW):1–24.

Emogi Research Team. 2015. *Emoji Report*.