EmojiGen

Eck, Elizabeth
University of Washington
eckliz@uw.edu

Hunt, Patrick William II
University of Washington
phunt22@uw.edu

Abstract

Have you ever needed to use an emoji, but the one you are looking for simply does not exist? Custom image generation is becoming more popular with tools like Chat GPT's 4o model and Apple's Genmoji which offers the ability to create an emoji of your own. However, these current applications have limitations to them. They are expensive, time consuming, or close-sourced making it difficult for accessibility and creativity. Our approach includes fine-tuning and retrieval-augmented generation (RAG) to create high-quality emoji generation. We evaluate our methods using CLIP prompt alignment, inference speed benchmarks, and visual tests, aiming to create a fast, accessible, and high-quality emoji generation tool for everyone.

1. Introduction

1.1. Background & Motivation

The use of emojis has become an essential part of communication these days and is extremely useful to add more meaning and creativity to a text. These single characters add emotion, humor, and context to a message that could otherwise be useless without that personality. The issue is, there are only a select amount of emojis currently available to use. There could be a situation where you find yourself asking the question "Why is there no emoji for this - I wonder if there was a way to create it on my own". While new emojis are being released every year, it's not the same as being able to produce an emoji exactly as you want it on the spot. This is where our project comes in. Through EmojiGen, we want to bridge the gap between emoji and generative AI. AI has been seen to be able to generate images based on a prompt, so why not make it more specific to the emoticon style? Our goal while keeping that in mind, is to create emoji images based on a prompt. For example, you could type the description of your wanted emoji as "a cat riding a skateboard" and the output would be exactly that. We are not the first to come up with this idea, as Apple already has made progress with this feature. However, we want our model to create any emoji you want - with no limitations in an open source project.

1.2. Related Work

There have been several projects that also tackled AI generated emojis. Apple's Genmoji lets users create stickers from text descriptions but is closed-source, tied to the Apple ecosystem, and somewhat slow (3–10 seconds). Google's Emoji Kitchen allows emoji mashups but only from existing designs, not new creations. In research, EmotiGAN [3] generating emojis from text using GANs, proving early on that text-to-emoji was possible, though results were basic by today's standards. More recently, projects like emojis.com (YC-backed) seem to have utilized a RAG system to facilitate the creation of personalized emojis based on user-provided text descriptions but at a high cost to users. While each of these efforts shows strong interest and some success, none fully combine high quality, speed, and accessibility—this gap is what our project aims to address by building a fast, open-source emoji generator using retrieval-augmented generation and distillation techniques.

1.3. Plan

Our plan is to fine-tune an open source model with an Apple emoji dataset to learn realistic emoji styles. We will then perform baseline tests to make sure that this model produces coherent outputs. Inspired by previous related work, we will add a retrieval-augmented (RAG) system using IP-Adapter [2] to incorporate similar emojis as visual references, and experiment with prompt augmentation using a lightweight LLM that makes text prompts more descriptive. We will evaluate all versions on our benchmarks across all architectures.

1.4. Expected Challenges

One of our biggest challenges is data. Any emoji dataset is going to be limited in size in one specific style, and furthermore comes with a lot of challenges in preprocessing—such as things involving skin tones, hair colors, backgrounds, etc. Luckily, Evan Zhou has done a lot of this

work in a similar open-source project Open Genmoji [8]. Another concern is balancing RAG, as we want to improve generation quality but not allow prompts to drift too far or overfit to the example image.

1.5. Expected Outcome

We expect to have a working open-source AI emoji generator that produces quality emojis with a comparable latency time of the existing solutions (3-10 seconds on GPU). We anticipate that tactics like each of these tactics will provide an improvement to quality, and hope that some combination of these tactics will lead to an architecture that generates quality emojis with minimal inference time.

2. Methods

2.1. Data

For fine-tuning data, we collected a dataset of apple emojis paired with their label captions. This was done with a Phython script that was written by Evan Zhou (open source on GitHub). The original set contains 3790 emojis, but to keep things simpler, we pared this down to 1901 emojis by excluding skin tone variants and using the default yellow skin tone (also with a script written by Evan Zhou). An example of a label would be "grinning face emoji" or "crying laughing emoji." We then make an 80/10/10 split of train/validate/test split of our data. Since our stable diffusion [6] models can't generate with a transparent background, we left the background on all emojis white for training.

2.2. Fine-Tuning

We fine-tuned our models (Stable Diffusion XL Base 1.0 [6] and Stable Diffusion v1-5 [4]) and experimented with different approaches to find what works best for emoji generation.

Hyperparameters: We experimented with different learning rates and batch sizes to get configurations that maximized validation accuracy. However, batch sizes were constrained by our GPU's memory, so we also used gradient accumulation to effectively increase that batch size.

LoRA vs. DreamBooth: We experimented with both approaches with the goal of seeing what would work better for our use case. LoRA [1] offers faster training and less memory requirements, while DreamBooth [7] can be a good fit for teaching a model to generate images in a specific style, which might be a good fit for adapting to the style of emojis.

2.3. Evaluation

We designed our evaluation to answer these key questions: Is it fast? Does it make sense? Does it actually work for real use cases?

Time per inference: We measured how long it takes to generate each emoji across different configurations. This is crucial since existing solutions can take 3-10 seconds.

Similarity to the test prompt: We used CLIP [5] score to measure how well our generated models match the input prompts. This gives us a very objective measure of how well our model is sticking to the input prompts.

"Vibe" benchmark: This is our practical benchmark. Very similar to how people have created benchmarks for modern LLMs such as the Pokemon or Tetris Benchmarks, we created a set of truly unseen prompts to see how well each model can truly generalize to unseen data that . This benchmark is meant to be more subjective, in contrast from the more objective score above, and be something that is more visual and interpreted by humans.

2.4. CLI Tools & Design

We created a CLI so that users can interface with the model in a simple way. Using the `emoji-get` CLI, users can just type in a prompt like

```
$ emoji-gen cat riding a skateboard
```

to generate an emoji. Additionally, users can configure and fine-tune models with the `emoji-dev` CLI, which includes commands like `$ emoji-dev start-server` (to start the inference server to avoid loading the model every time) or `$ emoji-dev fine-tune --model [MODELNAME]` (to fine-tune a model). To sync files to your local computer, all you have to do is `emoji-dev sync` from your local computer (after filling out the `.env` file).

More information is available in the `README.md` file of our GitHub repository, which walks through the whole workflow. This GitHub repository is public, so anyone is able to clone it and recreate our work, as long as they have a Google Cloud VM with a GPU (we used L4, but T4 works as well) and fill in the `.env` file.

3. Experiments

3.1. Sanity Checks:

Initially, we checked our base models with a 10 prompt benchmark set of unseen prompts to experiment. This did not relate to the test data, but the goal was to see how well the models perform at generating emojis out of the box and to see the effects of changing inference variables like guidance and steps. We used this as a preliminary step to build an intuition of how good the models are and what the inference parameters actually do. This also confirmed that we needed to find ways to improve these base models, mainly fine-tuning.

3.2. Benchmarking Fine-Tuned Models

We tested all of the evaluations that were mentioned above in 2.3.

Speed Analysis: We measured inference time across all of our model variants for all trials. The goal is to use this metric to evaluate speed vs. quality tradeoffs.

Quality Metrics: We ran CLIP scores on the test set generations to see how well the generated emojis matched their prompts, which helps us measure how closely the outputs are to emojis in the test set.

Unseen Prompts: Previously described as our "vibe" benchmark in 2.3, this is a more subjective test meant to see how the quality of outputs is on truly unseen prompts with no ground truth label. This is a way to see how the model generalizes and accounts for more subjective features that might be harder to measure with an objective metric. We simply will visually evaluate the model on these prompts.

3.3. Prompt Augmentation Layer

In our data, generally the emoji prompt-image pairs have very concise prompts, that are sometimes not very descriptive of the actual image, such as "winking face emoji." Without seeing this emoji, it is hard to tell exactly what is happening—is the left or right eye winking? Is there a smile? Raised eyebrows? We thought that this could lead to model confusion, and so adding more descriptive prompts could lead to improved generation, and a small LLM to add more information to the prompt before it goes into the diffusion model would be a way to do this automatically. We used a small, open-source LLM to do this, and manually experimented with system prompts and evaluate on the above benchmarks to see what works best.

3.4. Retrieval Augmented Generation

Similar to the above section, we believed that giving the model more information would help improve final outputs, especially when generating in a specific style. The owners of emojis.com wouldn't respond to our emails about access to their search API (perhaps they are scared of us), so we determined that the best option was to choose the "most similar emoji" from our dataset (regardless of split) and feed that into the model at inference using an IP-Adapter. The "most similar emoji" is determined by the CLIP similarity of the user inputted prompt to the emoji captions from our dataset.

An obvious issue with this approach is that some user prompts will be more similar to the dataset or further from it in general. One fix that we want to try is using the CLIP score as an indicator of how much to weight the "most similar emoji." The scale parameter of our Stable Diffusion models are [0,1], where 0 doesn't use the image and 1 means the output is conditioned only on the image prompt, with 0.5 being recommended. Since CLIP scores are [-1,1], we propose that using a ReLU or slightly modified ReLU function would move the range of CLIP to be directly passable as a scale parameter, where more similar prompts use more of the sample image, and less similar prompts use less of it.

Another solution to this from the HuggingFace IP-Adapter docs is to use InstantStyle—a way to insert IP-Adapter's at specific layers. We think that targeting specific layers that have to do with layout and style could lead to some great gains.

4. Results

4.1. Finetuning

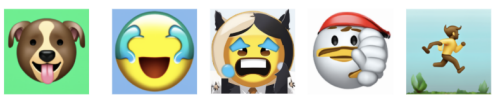
We fine-tuned Stable Diffusion XL, Stable Diffusion 3, and Stable Diffusion 3.5 using Dreambooth LoRA on our emoji set. Here were emojis on our base models when asked to generate "[X] emoji."

As you can see, our fine-tuned models demonstrated that they had learned the general style of emojis. Here you can see SD-XL's fine tune:

And here is SD-3.5, which gave us our best results:

This was not without faults in training, as some of our training attempts failed, yet interestingly still would learn the general patterns of our dataset:

Furthermore, many times our generated images seemed to be slightly "sloppy," as seen below:



While this is a good start, the following methods look to bring promising improvements.

4.2. Retrieval Augmented Generation

Unfortunately, this section was slightly more difficult and complicated than expected. We encountered issues implementing the IP-Adapter packages for the models, and decided that we did not have the time to finish this section, and may revisit this later. However, we were able to retrieve the "most similar" image to each prompt in our emoji set (1901 emojis), which looks to yield promising results. We do this by pre-computing the CLIP score of each emoji caption, and storing those vectors in a file. At inference, we compute the CLIP for the entered prompt, and use cosine similarity to find the prompt that is most alike, then grab the image for that caption. In practice, we found that the most similar CLIP score generally ranges from 0.6 to 0.9, and while generally fairly accurate to the prompt, there were some outliers. In the IP-Adapter package that we looked at, the default scale factor value (how much to weight the reference image versus the prompt) was 0.5 (50%), which means that they are equally weighted. It is likely that a function to translate the highest CLIP score to a more appropraite range (or simply use 0.5) is still needed.

While this approach is not particularly efficient or scalable, we still believe that RAG could make a large impact, particularly in the case of emojis. One specific area that we were excited to see was the results for flags, which we found to be very rarely correct in our models. By using a reference emoji (since all flags are in our dataset), we would likely see much better results.

As seen in our flags test set, most of the flags are completely nonsensical, as it is hard for the model to generate things that are factually correct. Interestingly, here we see the model adds a knife to the American Flag, picks up on some concepts from the Texas, Australia, and Japan Flags, and generates a great Finnish Flag—for the prompt "Swedish flag sks emoji." The other two flags are "Ivory Coast Flag sks emoji" and "Washington State Flag sks emoji."

4.3. Prompt Augmentation

Previously mentioned in Section 4.1, even our best fine-tuned models seem to be confused with certain prompts. We recognized two such cases: 1. The prompt has an indirect meaning 2. The prompt is not descriptive enough

On our test set, we could see that "smiling cat with heart eyes emoji" and "locked emoji" did not match the expected output, but technically were correct.

Here, we realized that training got us far, but we needed better inputs to our models. So, we turned to an LLM which seemed to be an exciting way to ensure that we consistently had improved prompts.

Our first experiment was to run single image prompts through different system prompts, and see which ones captured style the best as seen in the picture below.

From here, we ran side by side comparisons of prompts with and without the LLM augmenting the prompt. We found that this yielded incredible results. Notably, we achieved on the 2 classes outlined above, but also achieved a more consistent sense of style, including a more consistent white background.

The left is without augmentation, and the right is with augmentation, showing a clear improvement. This proved to scale beyond those two examples, yielding consistently improved results.

4.4. Conclusion

Overall, we had success creating a project that could generate emojis. This project lets users generate custom emojis using generative AI techniques like fine-tuning, retrieval-augmented generation, and prompt engineering. We wanted to build something open-source that people could try out themselves while also experimenting with new ways to guide image generation effectively. From fine-tuning multiple models, we found that SD-3.5 gave the best overall results, followed by SD-XL. Some models

learned style, but were semantically off. Next, we tackled prompt quality, finding that the best results came from running our prompts through a small LLM Flan T5 to make the prompts more descriptive. These made a huge difference in output quality. We also implemented RAG which basically used CLIP embeddings to find the most similar emoji in our dataset. That image gets fed as a reference during generation, which helped with visual grounding. We built a command-line interface to make it easy to generate your own emojis. With just a few steps, you can run emoji-gen [your prompt] to generate results. We included an emoji-dev CLI to help with training, inference, and data preparation.

Although this project is not perfect, we still managed to produce fun emojis based on a prompt in a reasonable amount of time. We learned a lot about finetuning, prompt engineering, model selection, and tradeoffs of different techniques.

References

[1] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. 2

[2] Ziqing Li, Yuwei Zhang, Ziyu Zhang, Songyang Liu, Yuning Jiang, Chen Qian, and Chen Change Loy. Ip-adapter: Text-to-image diffusion models with image prompting. *arXiv preprint arXiv:2308.06721*, 2023. 1

[3] Han Liu, Mengdi Zhan, and Xiaoyuan Hu. Emoti-gan: Emoji art generation using generative adversarial networks. CS229 Final Project, Stanford University, 2017. https://cs229.stanford.edu/proj2017/final-reports/5244346.pdf. 1

[4] David Podell, Zach English, Kunal Lacey, Andreas Blattmann, Theodor Dockhorn, Jan Müller, Joseph Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023. 2

[5] Alec Radford, Jong Wook Kim, Luke Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pam Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. *arXiv preprint arXiv:2103.00020*, 2021. 2

[6] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. *arXiv preprint arXiv:2112.10752*, 2022. 2

[7] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. *arXiv preprint arXiv:2208.12242*, 2022. 2

[8] Evan Zhou. Open-genmoji: An open-source emoji generation project. https://github.com/EvanZhouDev/open-genmoji, 2023. Accessed: 2025-05-26. 2