Emotion Draw: Bridging Emotions and Art through AI
Spring 2024

Vyacheslav Stepanyan
BS in Engineering Sciences
American University of Armenia

Anahit Baghdasaryan
BS in Data Science
American University of Armenia

Instructor: Davit Abgaryan
American University of Armenia

***Abstract*—This project's main objective is to bridge the gap between the emotional and visual aspects of human expression. Therefore, "Emotion Draw" presents a solution for predicting emotions through textual inputs and transforming them into captivating visual representations. By utilizing innovative techniques and advanced AI models such as ALBERT (A Lite BERT)[5], a powerful transformer-based language model, and Stable Diffusion v2-1[7], a text-to-image model based on latent diffusion algorithms, "Emotion Draw" offers users a unique platform to explore and express their emotions through abstract visual masterpieces. With a focus on simplicity and accessibility, our user-friendly interface invites individuals of all backgrounds to engage in the creative process and discover their artistic potential.**

I. Introduction

"Emotion Draw" represents an innovative usage of state-of-the-art technologies, integrating a BERT-based model ALBERT (A Lite BERT)[5] for natural language processing (NLP) and a powerful deep learning text-to-image model, the Stable Diffusion v2-1[7]. This advanced integration allows the platform to predict emotions from user sentence inputs and create images that reflect these emotions. Therefore, by leveraging advanced NLP and image generation techniques, "Emotion Draw" tries to surpass the traditional boundaries between textual and visual expression, offering users a unique and immersive creative experience by empowering them to explore and communicate their emotions in engaging and innovative ways.

The methodology employed by "Emotion Draw" encompasses several key components. Firstly, in the scope of this project, we have utilized a dataset of emotions designed by Saravia and the Hugging Face team (Saravia et al., EMNLP 2018)[9], created explicitly for NLP tasks, as a foundation for fine-tuning the ALBERT[5] model to predict emotions from sentences, treating it as a classification task. Moreover, to make the process of fine-tuning end-to-end and flexible, we have designed a Python class (a code template for creating objects) `MulticlassClassificationTrainer()` from scratch, which includes methods for automatically preprocessing data (building dataloaders, etc.), training the model, evaluating the model on a test set, plotting the confusion matrices, etc. Subsequently, the fine-tuned model predicts emotions from user-provided sentences, effectively transforming text into raw emotions. These emotions serve as a part of the prompt for the stable diffusion model[7], guiding the creation of abstract visual masterpieces inspired by spiritualist art. Next, we have utilized FastAPI[6] to ensure seamless communication between the frontend and backend. At the same time, we have crafted a user-friendly interface with JavaScript React[3], which invites users to showcase their creativity and craft their own emotional artworks.

II. Data Acquisition

A. Overview

In this project's scope, in order to fine-tune the ALBERT[5] model, we have utilized a dataset of emotions designed by Saravia and the Hugging Face team (Saravia et al., EMNLP 2018)[9], created explicitly for NLP tasks, which provides a rich collection of documents annotated with their corresponding emotions. The dataset preparation and annotation processes are explored in the publication "CARER: Contextualized Affect Representations for Emotion Recognition.[9]" This dataset is organized into training, validation, and test sets to support developing and evaluating machine learning models. Specifically, the training set comprises 16,000 entries, while both the validation and test sets contain 2,000 entries each. An example entry in the dataset is formatted as follows: "I feel like I am still looking at a blank canvas blank pieces of paper; sadness". Similarly, most of the sentences in the dataset follow the same specific format, starting with "I am ..." or "I...". While this format simplifies the annotation process, it may limit the effectiveness of fine-tuned models in extracting sentiment from inputs of different formats. Moreover, the dataset includes six distinct emotion labels: Anger, Joy, Love, Fear, Sadness, and Surprise, each assigned to a corresponding document. You can find this dataset in the GitHub repository[10] of this project; it is available in the `/Emotion_Draw/bert_part/data/raw/` directory.

B. Preprocessing

- Firstly, we read the raw text files to process the data for model training and split their contents into sentences and labels.

```
#Read the text file
with open('../data/raw/train.txt', 'r'
    ) as file:
    lines = file.readlines()

#Split each line by semicolon
data = [line.strip().split(';') for
    line in lines]
```

- The data is then organized into a DataFrame for exploratory data analysis (EDA), which includes descriptive statistics, checking for missing values, examining data types, and identifying unique labels.

```
#Create DataFrame
train_df = pd.DataFrame(data, columns
    =['Sentence', 'Labels'])

#Display the DataFrame
train_df

# Descriptive statistics
train_df.describe()

# Check for missing values
train_df.isnull().sum()

# Data types of columns
train_df.dtypes

# Unique labels
train_df['Labels'].unique()
```

- We perform label encoding to convert the categorical feature of emotions into numerical values, preparing the dataset for model training.

```
# Encode labels
train_df['Labels_Encoded'] =
    label_encoder.fit_transform(
    train_df['Labels'])
train_df
```

- The processed dataset is saved into CSV formatted files for subsequent use, ensuring consistency and reproducibility in the training process.

```
# Specify the file path where you want
    to save the CSV file
file_path = '../data/processed/
    train_data.csv'

# Save the DataFrame to a CSV file
train_df.to_csv(file_path, index=False
    )
```

We apply similar procedures to the validation and test sets, with necessary adjustments to file paths and configurations per the specific setup.

You can find the detailed preprocessing steps in the associated notebook in the project's GitHub repository[10] at the following location: `/Emotion_Draw/bert_part/notebooks/data_creation.ipynb`

III. Emotion Extraction

A. The Model Choice

*1) **BERT**:* BERT, short for Bidirectional Encoder Representations from Transformers, is a pre-trained natural language processing (NLP) model based on transformer architecture[11], specifically on the encoder component, introduced by Google in 2018 and initially presented in the seminal paper "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding"[2] by Devlin et al. Its bidirectional architecture affords it the capability to capture contextual information from both preceding and subsequent tokens within a given text sequence. BERT's advent revolutionized NLP through the provision of a pre-trained model, trained on vast corpora of textual data encompassing the Toronto Book Corpus[12] and Wikipedia, partaking in two unsupervised tasks: masked language modeling (MLM) and next sentence prediction (NSP). This pre-trained model is very useful in Transfer Learning scenarios across a spectrum of downstream tasks, thereby minimizing the necessity for designing task-specific architectures from scratch.

*2) **ALBERT**:* ALBERT, short for A Lite BERT, is a variant of the BERT (Bidirectional Encoder Representations from Transformers) model developed by the adept minds at Google AI and proposed in the paper "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations"[5] by Lan et al., is designed to address some of the limitations of the original BERT model, such as its large size and computational cost, while maintaining or even improving performance on an array of NLP tasks.

ALBERT introduces several parameter reduction techniques, exemplified by the factorization of the embedding matrix, decomposing it into two smaller matrices, and deploying recurrent (repeating) layers distributed among groups. These strategies culminate in a reduction in the total number of model parameters while preserving performance metrics.

ALBERT embarks on its pre-training journey via the execution of the following unsupervised learning tasks:

- **Masked Language Modeling (MLM):** Similar to BERT, MLM is one of ALBERT's pre-training tasks, where a subset of input tokens are randomly masked, and the model is tasked with predicting these masked tokens, fostering robust representations of words and phrases.
- **Sentence Order Prediction (SOP):** Besides MLM, ALBERT integrates the SOP task to predict the sequential arrangement of input sentences. This auxiliary task helps the model capture relationships between sentences and improve its understanding of document-level context.

*3) **Our Choice: ALBERT**:* ALBERT offers several advantages vis-à-vis classification tasks:

- **Parameter Efficiency:** ALBERT achieves a significant reduction in the number of model parameters compared to BERT. This reduction in parameters makes ALBERT more suitable for deployment in resource-constrained environments.
- **Accelerated Training:** Due to its reduced parameter count, ALBERT achieves faster training times relative to BERT, thereby reducing computational costs.
- **Performance Preservation:** Despite parameter reduction endeavors, ALBERT aims to retain or even improve performance compared to BERT on various NLP tasks. This makes it an attractive choice for classification tasks where both performance and efficiency are crucial considerations.

B. MulticlassClassificationTrainer()

This section discusses the script that defines a Python class `MulticlassClassificationTrainer()` for fine-tuning, evaluating, and making inferences with a BERT-based sequence classification model using PyTorch and Hugging Face's Transformers library. **Note:** You can find this script in the project's GitHub Repository[10], located at `/Emotion_Draw/bert_part/model/`.

1) Initialization:

```
__init__(self, model_name, num_labels,
    batch_size, num_epochs, learning_rate,
     max_length, device, suffix="")
```

Arguments:

- `model_name (str)`: Name of the pretrained model from Hugging Face's model hub.
- `num_labels (int)`: Number of classes in the classification task.
- `batch_size (int)`: Batch size for training.
- `num_epochs (int)`: Number of epochs for training.
- `learning_rate (float)`: Learning rate for optimization.
- `max_length (int)`: Maximum length of input sequences.
- `device (str)`: Device to use for training ('cpu' or 'cuda').
- `suffix (str, optional)`: Suffix to add to model checkpoint and confusion matrix filenames (default: ' ', empty string).

2) Functions:

- **_initialize_model():** Initializes the model, tokenizer, optimizer, loss function, and moves the model to the specified device.

```
_initialize_model(self)
```

- **_process_data():** Initializes the model, tokenizer, optimizer, loss function, and moves the model to the specified device.

```
_process_data(self, df)
```

Arguments:

  - `df (DataFrame)`: Input DataFrame containing 'Sentence' and 'Labels_Encoded' columns.

Returns:

  - `TensorDataset`: Tensor dataset containing processed data.

- **save_state():**
Saves the model and optimizer states to a checkpoint file.

```
save_state(self, epoch,
    model_state, optimizer_state,
    path)
```

Arguments:

  - `epoch (int)`: Epoch number.
  - `model_state (dict)`: State dictionary of the model.
  - `optimizer_state (dict)`: State dictionary of the optimizer.
  - `path (str)`: Path to save the checkpoint file.

- **load_state():**
Loads model and optimizer states from a checkpoint file.

```
load_state(self, path)
```

Arguments:

  - `path (str)`: Path to the checkpoint file.

Returns:

  - `int`: Epoch number.
  - `dict`: Model state dictionary.
  - `dict`: Optimizer state dictionary.

- **train():**
Trains the model using the provided training and validation datasets.

```
train(self, train_df, val_df,
    log_dir, checkpoint_path=None)
```

Arguments:

  - `train_df (DataFrame)`: Training dataset DataFrame.
  - `val_df (DataFrame)`: Validation dataset DataFrame.
  - `log_dir (str)`: Directory path to save TensorBoard logs.
  - `checkpoint_path (str, optional)`: Path to a checkpoint file to resume training (default: None).

- **save_confusion_matrix():**
Saves the confusion matrix to a file.

```
save_confusion_matrix(self, cm,
    epoch, mode)
```

Arguments:

  - `cm (array)`: Confusion matrix array.
  - `epoch (int)`: Epoch number.
  - `mode (str)`: Mode of confusion matrix ('train', 'val', 'test').

- **load_model():**
  Loads the model from a checkpoint file.

```
load_model(self, checkpoint_path)
```

  Arguments:
  - `checkpoint_path (str)`: Path to the checkpoint file.

- **evaluate():**
  Evaluates the model on the test dataset.

```
evaluate(self, test_df, mode='
    train')
```

  Arguments:
  - `test_df (DataFrame)`: Test dataset DataFrame.
  - `mode (str, optional)`: Evaluation mode ('train' or 'test') (default: 'train').

  Returns:
  - `float`: Average loss.
  - `float`: Accuracy.
  - `float`: F1 score.
  - `array`: Confusion matrix.

- **single_inference():**
  Performs single inference on a given sentence.

```
single_inference(self,
    checkpoint_path, train_df,
    sentence)
```

  Arguments:
  - `checkpoint_path (str)`: Path to the checkpoint file.
  - `train_df (DataFrame)`: DataFrame containing label mappings.
  - `sentence (str)`: Input sentence for inference.

  Returns:
  - `dict`: Dictionary containing predicted labels and probabilities.

- **print_cm():**
  Prints the confusion matrix.

```
print_cm(self, mode, df)
```

  Arguments:
  - `mode (str)`: Mode of confusion matrix ('train', 'val', 'test').
  - `df (DataFrame)`: DataFrame containing label mappings.

C. Step-by-Step Fine-Tuning

This section discusses the complete end-to-end process for fine-tuning BERT-based sequence classification models on the dataset of emotions addressed in Data Acquisition. The goal is to train models capable of accurately classifying emotions expressed in the input sequence.

Note: You can find an associated notebook in the project's GitHub repository, located at `/Emotion_Draw/bert_part/notebooks/BERT-based_Sequence_Classification.ipynb`[10]

1) Import Packages: We begin by importing necessary packages for data manipulation, model training, and evaluation.

```
import torch
import pandas as pd
import os
import logging

import sys
sys.path.append(os.path.dirname(os.getcwd
    ()))

from model.Multiclass_BERT import
    MulticlassClassificationTrainer

logging.getLogger("matplotlib.colorbar").
    setLevel(logging.ERROR)
```

2) Choose a Model: Choose a BERT-based model for training in the sequence classification task (consider including other similar models if applicable). Assign a distinctive suffix to differentiate between various experiments.

```
model_names = ['bert-base-uncased', '
    roberta-base', 'albert-base-v2']
MODEL_NAME = 'albert-base-v2'
SUFFIX = 'experiment-tech-test'
```

3) Specify the Hyperparameters: Define the hyperparameters, including batch size, number of epochs, learning rate, and maximum sequence length, which are crucial for training the model effectively. Additionally, specify the number of labels and the device for training, ensuring compatibility with either GPU or CPU resources.

```
num_labels = 6
batch_size = 32
num_epochs = 3
learning_rate = 1e-5
max_length = 128
device = torch.device('cuda' if torch.cuda
    .is_available() else 'cpu')
```

4) Instantiate the Class: Instantiate the `MulticlassClassificationTrainer()` class with the chosen model and specified hyperparameters.

```
trainer = MulticlassClassificationTrainer(
    MODEL_NAME, num_labels, batch_size,
    num_epochs, learning_rate, max_length,
     device, suffix=SUFFIX)
```

5) Load Data: Load the training and validation datasets from CSV files II.

```
train_df = pd.read_csv('../data/processed/
    train_data.csv')
val_df = pd.read_csv('../data/processed/
    val_data.csv')

train_df = train_df.head(200)  # Limiting
    data for a quick test
val_df = val_df.head(200)  # Limiting data
     for a quick test
```

6) Training: Train the model using the training dataset and validate it using the validation dataset. TensorBoard logs are written to the specified directory for visualization.

```
log_dir = f'../runs/{MODEL_NAME}_{SUFFIX}'
trainer.train(train_df, val_df, log_dir)
```

7) Evaluate on the Test Set: Load the best-performing model checkpoint and evaluate it on the test dataset.

```
model_path = "../models_trained/
    multiclass_experiment-tech-test_albert
    -base-v2_best_checkpoint.pth"

trainer.load_model(model_path)

test_df = pd.read_csv('../data/processed/
    test_data.csv')
test_df = test_df.head(100)  # Limiting
    data for a quick test

trainer.evaluate(test_df, mode='test')
```

8) Inference for a Single Example: Perform inference for individual sentences using the model checkpoint.

```
checkpoint = "../models_trained/
    multiclass_experiment-tech-test_albert
    -base-v2_best_checkpoint.pth"

sentence = "My boss made me do all the
    frustrating work."

trainer.single_inference(checkpoint,
    train_df, sentence)
```

9) Display Confusion Matrices: Visualize the confusion matrices for the training, validation, and test sets.

```
trainer.print_cm(mode="train", df=train_df
    )
trainer.print_cm(mode="val", df=val_df)
trainer.print_cm(mode="test", df=test_df)
```

10) TensorBoard: Use TensorBoard to visualize the training process. Start TensorBoard on port 6008 (adjust the port as needed) and monitor the training logs.

```
%reload_ext tensorboard
%tensorboard --logdir=../runs --port=6008
    --load_fast=false
```

If the specified port is busy, identify the PID (Process ID) using `!lsof -i :6008` and terminate it using `!kill PID`.

IV. Evaluation Metrics and Results

A. Accuracy

In sequence classification tasks, a fundamental evaluation metric is accuracy, which measures the proportion of correctly classified samples out of the total number of samples. It serves as a key indicator of the model's performance in correctly assigning labels to input sequences. Mathematically, accuracy is calculated as follows:

$$\text{Accuracy} = \frac{\text{Number of correctly classified samples}}{\text{Total number of samples}}$$

Accuracy provides insights into how well the model predicts the correct class labels for the given input sequences. A higher accuracy score indicates better performance, with 100% accuracy representing perfect classification.

B. F1 Score

In sequence classification tasks, another important evaluation metric is the F1 score, which combines precision and recall to provide a balanced measure of model performance. Precision measures the proportion of correctly predicted positive samples out of all samples predicted as positive, while recall measures the proportion of correctly predicted positive samples out of all actual positive samples. F1 score, calculated as the harmonic mean of precision and recall, balances both metrics and is defined as:

$$\text{F1 score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

where:

- Precision = $\frac{TP}{TP+FP}$, where TP is the number of true positives and FP is the number of false positives.
- Recall = $\frac{TP}{TP+FN}$, where FN is the number of false negatives.

The F1 score ranges from 0 to 1, with higher values indicating better model performance. It provides a comprehensive assessment of a model's ability to correctly classify both positive and negative samples, taking into account both false positives and false negatives.

C. Results

In the table Experiment Results, you'll discover the outcome of assessing performance metrics across experiments 1, 2, and 3. It's noteworthy that throughout these experiments, the albert-base-v2 model was employed and underwent training for a span of 20 epochs. It's worth mentioning that the learning rates differed slightly across these experiments; specifically, experiments 1 and 2 employed a learning rate of 1e-5, whereas experiment 3 utilized a slightly higher rate of 2e-5. Additionally, variations in batch sizes were observed: experiments 1 and 2 operated with a batch size of 32, while experiment 3 utilized a larger batch size of 64.

An important distinction arises concerning the data utilization. In experiment 1, only a fraction, precisely 10%, of the available data was utilized. This limitation was due to the

training process being executed on a CPU device. Consequently, experiment 1 yielded less optimal results compared to experiments 2 and 3. Notably, experiments 2 and 3 made full use of the dataset, employing 100% of the available data. Moreover, these experiments benefited from the computational advantages of training on a GPU, which contributed to their superior performance.

V. The Artist: Stable Diffusion v2-1

Stable diffusion models represent a significant advancement in the realm of text-to-image generation. These models operate on the principle of latent diffusion[8], utilizing a fixed, pretrained text encoder.

A. Overview

The Stable Diffusion v2-1[7] is a diffusion-based model again designed for text-to-image generation.

Stable Diffusion v2-1[7] model stands out for its enhanced capabilities and refinements. Developed by Robin Rombach and Patrick Esser, this model is fine-tuned from its predecessor, stable-diffusion-2[7], with additional training steps and adjustments to parameters, resulting in improved performance and versatility.

Leveraging latent diffusion techniques, this model offers a sophisticated approach to image synthesis. Moreover, this model is combined with a fixed, pretrained text encoder (OpenCLIP-ViT/H)[4]. This combination allows for the generation of high-quality images. The model architecture and training methodologies are outlined in detail in the associated GitHub repository[4].

While the model offers impressive capabilities, it is essential to acknowledge its limitations, biases, and ethical considerations:

- **Limitations:** The model may not achieve perfect photorealism, struggle with certain tasks like rendering legible text, and exhibit challenges in generating specific compositions or accurately depicting faces and people.
- **Bias:** Models like Stable Diffusion v2-1 may reflect and reinforce biases present in their training data, particularly concerning language and cultural representations.

Viewer discretion is advised due to potential biases and limitations.

B. Our Approach

In this project's scope, in utilizing the Stable Diffusion v2-1[7] model, our approach lies in prompt engineering. We opted not to fine-tune the model, instead focusing on crafting an optimal prompt to achieve our artistic vision.

Our current prompt is crafted to illustrate the predicted emotions using carefully selected color palettes, watercolor techniques, and smooth color transitions.

In the following script, the `color_emotion` dictionary maps each emotion to a set of corresponding colors. For each emotion in the list of predicted emotion, a prompt is generated that specifies an "ENERGY ART STYLE" representation. The resulting prompts are designed to evoke the intended emotional response through visual art.

```
prompts = []

color_emotion = {
    'anger': "Red, Black, Dark Brown,
        Orange, Dark Gray",
    'fear': "Black, Dark Purple, Dark Blue
        , Dark Green, Gray",
    'joy': "Yellow, Orange, Bright Green,
        Sky Blue, Pink",
    'sadness': "Gray, Dark Blue, Black,
        Dark Green, Pale Purple",
    'love': "Red, Pink, White, Lavender,
        Peach",
    'surprise': "Bright Orange, Neon Green
        , Yellow, Silver, Electric Blue"
}

for emotion in emotions:
    prompts.append(f"ENERGY ART STYLE
        representation of the feeling of {
        emotion}. Use colors {
        color_emotion[emotion]}.
        Waterpaint. Smooth color
        transitions.")
```

Fig. 1: Examples of Generated Images Using the Stable Diffusion v2-1 Model with Our Custom Prompts

We encourage creative freedom in modifying the prompt to explore diverse artistic outcomes, ensuring both flexibility and innovation in the generative process. Our mission is to empower users to shape their unique creative vision and fully leverage the model's potential through thoughtful and personalized prompt customization.

VI. FastAPI Integration

In the project's GitHub repository[10], you can find the module `/Emotion_Draw/api/api`, which defines a FastAPI[6] web application for generating image representations based on emotions predicted from user-provided text prompts. Below is a breakdown of its functionality:

1) Functionality:

- Reads an authentication token from a file for model access.
- Imports necessary libraries and modules.
- Sets up CORS (Cross Origin Resource Sharing) middleware to enable communication between frontend and backend.
- Defines an endpoint ("/") that accepts text prompts and generates image representations based on the predicted emotions.

TABLE I: Experiment Results

Experiment No.	Checkpoint Saved From	Accuracy (%)	F1 Score (%)	Avg Loss
1	Epoch 20	79.80	72.44	0.74
1	The epoch with the lowest validation loss	79.75	71.98	0.70
2	Epoch 20	92.10	88.31	0.27
2	The epoch with the lowest validation loss	91.95	87.63	0.17
3	Epoch 20	92.20	88.36	0.26
3	The epoch with the lowest validation loss	93.25	89.27	0.14

- Utilizes a pre-trained Stable Diffusion model for image generation.
- Returns confidence scores, predicted emotions, and generated images as responses.

2) Usage:

- Run the FastAPI ApplicationVI-3: Start the server to expose the defined endpoints.
- Send Requests: Send requests to the "/generate" endpoint with text prompts.
- Receive Responses: Obtain generated images, predicted emotions, and confidence scores as responses.

3) How to Run the FastAPI Application?:

The following command will simultaneously start the backend and frontend servers, opening the frontend application and the FastAPI docker in the default web browser. You can find the `run.py` file in the project's GitHub repository[10] located at the top main directory.

```
$ python run.py
```

Or, alternatively, you can start only the FastAPI server.

```
$ uvicorn Emotion_Draw.api.api:app --
    reload
```

4) Usage Example:

```
import requests

# Example prompt
prompt = "I feel happy today!"

# Send request to the API
response = requests.get(f"http://localhost
    :8000/?prompt={prompt}")

# Retrieve responses
print(response.text)
```

VII. Front-End Implementation

For the frontend of "Emotion Draw", we leveraged JavaScript and React[3] to create a dynamic and interactive user experience. By combining the power of these technologies with the Chakra UI[1] component library, we crafted a visually appealing and intuitive interface for generating and exploring emotion-based images.

1) Key Features Implemented:

- **Chakra UI Integration:** We utilized Chakra UI, a flexible and accessible component library for React, to streamline the development of our frontend. This allowed us to easily implement various UI elements such as headings, containers, buttons, inputs, modals, tooltips, and progress indicators with consistent styling and functionality.
- **State Management with React Hooks:** React hooks, including `useState`, `useRef`, and `useDisclosure`, were employed for efficient state management within our application. These hooks enabled us to manage dynamic data, facilitating seamless interaction between components.
- **Asynchronous Data Fetching with Axios:** We utilized the Axios library for making asynchronous HTTP requests to our backend server. This allowed us to fetch image data and associated emotions based on user-provided prompts, enabling real-time generation of emotion-based images.
- **Modal and Tooltip Components:** We implemented modal and tooltip components using Chakra UI's Modal and Tooltip components, enhancing user interaction and providing additional context and information. These components offer intuitive ways to display detailed information, instructions, and supplementary content without cluttering the main interface.
- **Event Handling and User Interaction:** React event handling mechanisms were employed to manage user interactions such as input submission, image selection, and modal closure. Additionally, we utilized event listeners to trigger image downloads and handle keyboard events for enhanced accessibility and user convenience.

2) How to Run the JS React Application?:

The following command will simultaneously start the backend and frontend servers, opening the frontend application and the FastAPI docker in the default web browser. You can find the `run.py` file in the project's GitHub repository[10] located at the top main directory.

```
$ python run.py
```

Or, alternatively, you can start only the frontend server.

```
$ cd Emotion_Draw/client
$ npm start
```

Note: Refer to Appendix to see the screenshots of the interface.

VIII. Conclusion and Future Work

In conclusion, our project has successfully implemented an interactive web application, "Emotion Draw," that transforms text inputs into captivating visual representations of emotions. By integrating the advanced NLP model ALBERT, coupled with Stable Diffusion for image generation, we have provided users with an engaging and innovative experience, bridging the gap between textual and visual expression.

While our current system achieves fair results, there are several avenues for future work and improvement. Firstly, we acknowledge that the generation process from our webpage currently takes approximately 30 seconds, which can be optimized to enhance user experience and responsiveness.

Looking ahead, we aspire to incorporate additional datasets for further fine-tuning of BERT-based models for sequence classification. By diversifying our training data, we aim to improve the generalization of our models to handle real-life user-provided prompts better, leading to more accurate and nuanced emotion representations.

Furthermore, we envision enhancing the frontend interface to be more flexible and responsive across various devices. By optimizing the user interface design and layout, we seek to provide users with a seamless and intuitive experience regardless of the device they are using.

Moreover, we plan to integrate a database into the project in our future endeavors. This would enable users to create accounts, sign in, and save their generated pictures in personalized albums. Additionally, users would have the opportunity to follow other members of the community, explore their generated artworks, and save them in their personal albums if desired. This social aspect of the platform aims to foster a sense of community and collaboration among users, enriching their overall experience with "Emotion Draw."

We hope that "Emotion Draw" will inspire and motivate the community to explore further innovations in the field of emotion representation and visualization. We welcome contributions from researchers, developers, and enthusiasts alike to collaborate and enhance the capabilities of our platform. Together, we can continue to push the boundaries of technology and creativity, unlocking new possibilities for understanding and expressing human emotions.

References

[1] Segun Adebayo. Chakra ui - a simple, modular and accessible component library that gives you the building blocks you need to build your react applications. https://v2.chakra-ui.com/.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[3] Facebook. React. https://react.dev/, 2013.

[4] Gabriel Ilharco, Mitchell Wortsman, Ross Wightman, Cade Gordon, Nicholas Carlini, Rohan Taori, Achal Dave, Vaishaal Shankar, Hongseok Namkoong, John Miller, Hannaneh Hajishirzi, Ali Farhadi, and Ludwig Schmidt. OpenCLIP, July 2021.

[5] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2020.

[6] Sebastián Ramírez. Fastapi. https://fastapi.tiangolo.com, 2018. Version 0.95.2.

[7] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10684–10695, June 2022.

[8] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022.

[9] Elvis Saravia, Hsien-Chi Toby Liu, Yen-Hao Huang, Junlin Wu, and Yi-Shin Chen. CARER: Contextualized affect representations for emotion recognition. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3687–3697, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[10] Vyachslav Stepanyan and Anahit Baghdasaryan. Emotion Draw. https://github.com/vyacheslavstepanyan1/Emotion_Draw/, May 2024.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[12] Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books, 2015.

APPENDIX

In this appendix, we provide additional supplementary materials to complement the main content of the document.

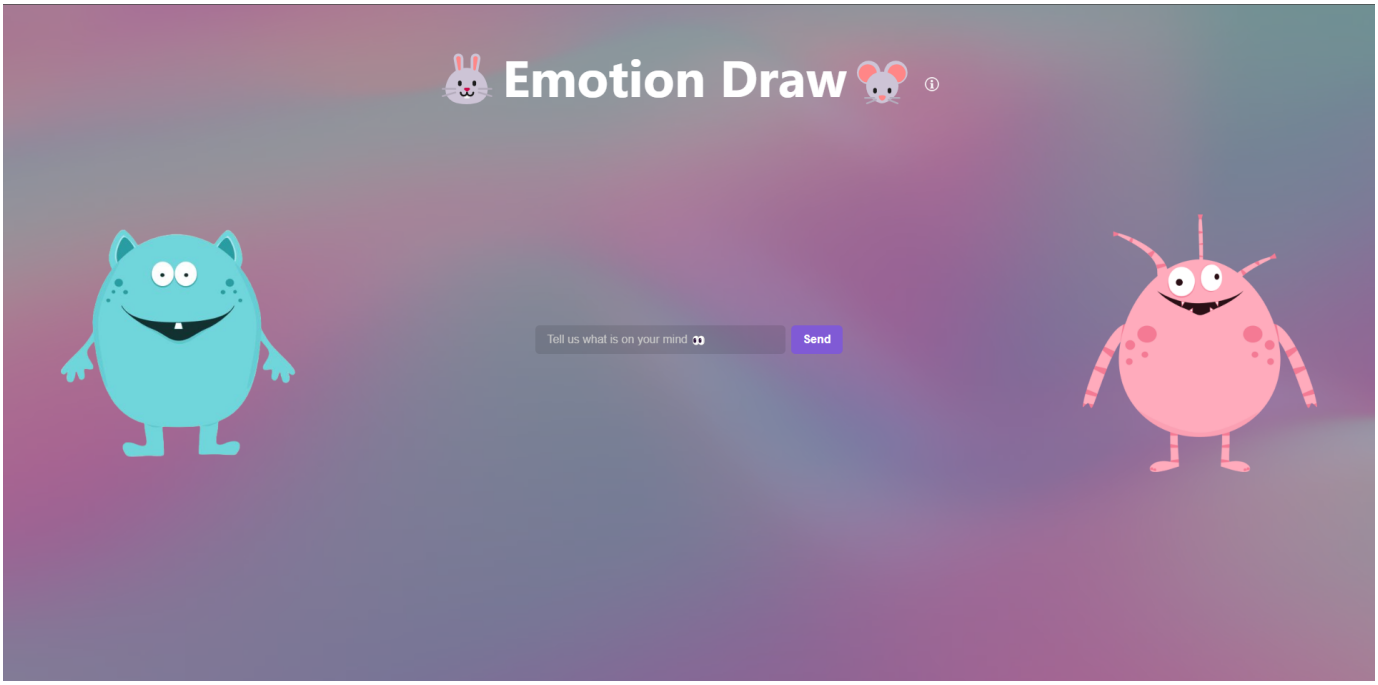

Fig. 2: The Web Page Interface

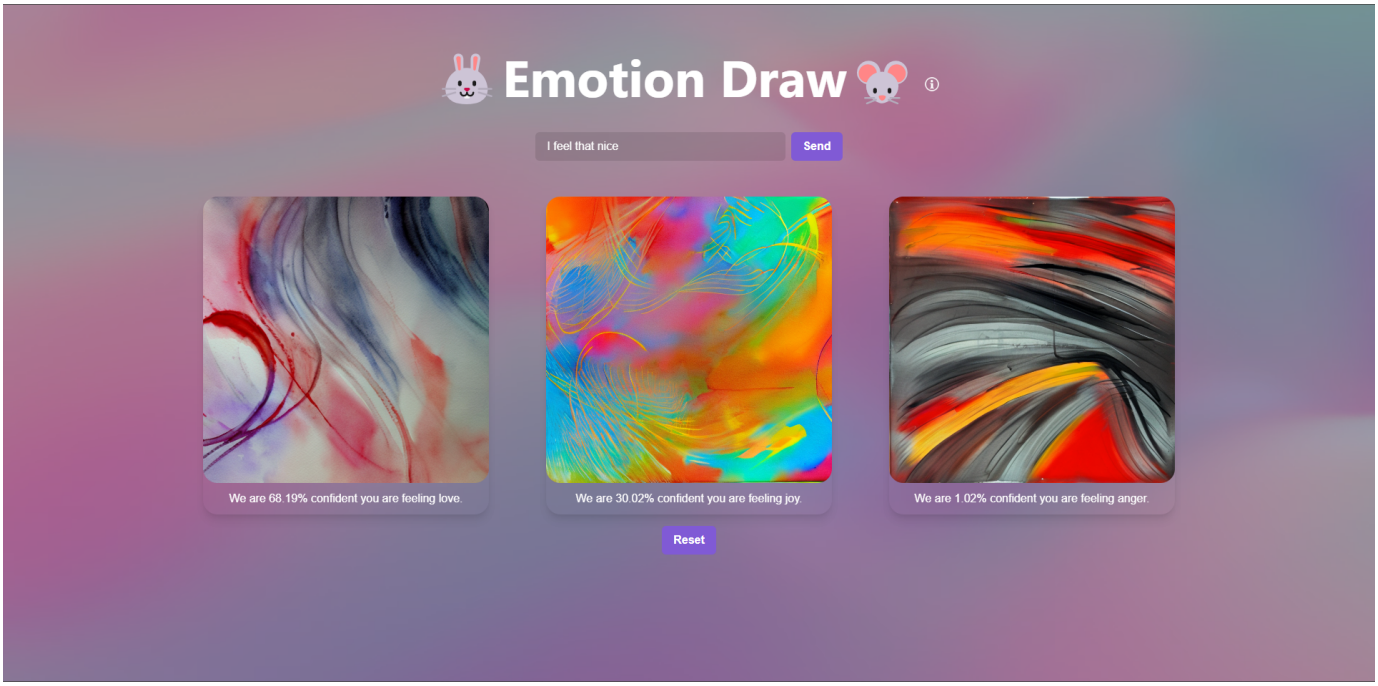

Fig. 3: The Web Page Interface With Generations

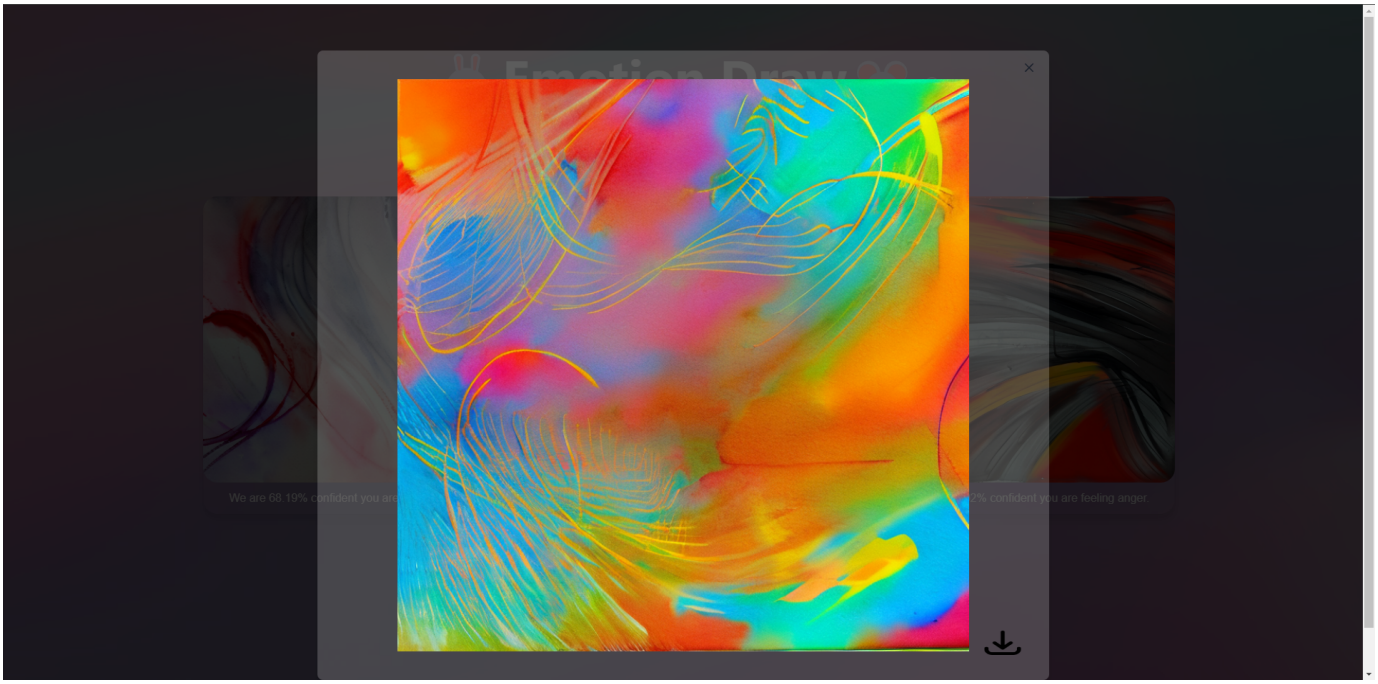

Fig. 4: Downloading the Generated Picture