Configuration Manual

MSc Research Project
Data Analytics

Apoorva Vishwas Rasal
Student ID: 22225277

School of Computing
National College of Ireland

Supervisor: Mr. Hicham Rifai

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Apoorva Vishwas Rasal
Student ID:	22225277
Programme:	Data Analytics
Year:	2024
Module:	MSc Research Project
Supervisor:	Mr. Hicham Rifai
Submission Due Date:	12/08/2024
Project Title:	Configuration Manual
Word Count:	1520
Page Count:	10

Signature:	
Date:	14th September 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Apoorva Vishwas Rasal
22225277

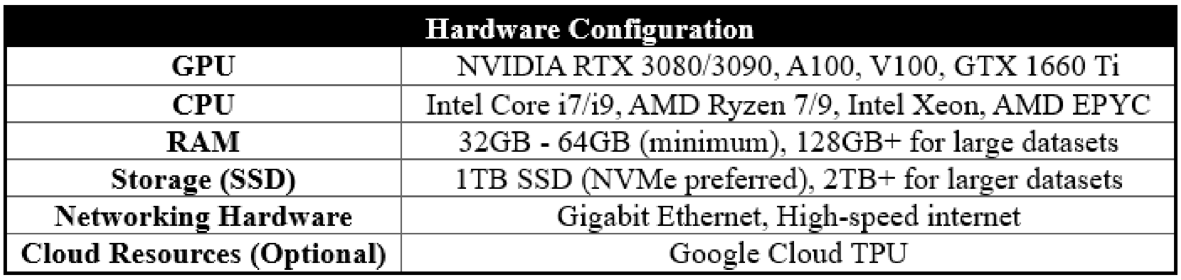
1 Introduction

The presented configuration manual provides the description of the hardware and software that is used in the present study along with the step-by-step explanation of the procedure that has been adopted in the paper titled "**Cognitive Captions: Empowering Images with AI-Generated Descriptions**".

2 System Configuration

2.1 Hardware Configuration

In an image caption generation project, high GPU is useful for training models and sufficient RAM and fast SSD storage is essential for handling large datasets for fast data retrieval. Figure 1 illustrates the table of the recommended hardware components necessary for the development of this project.



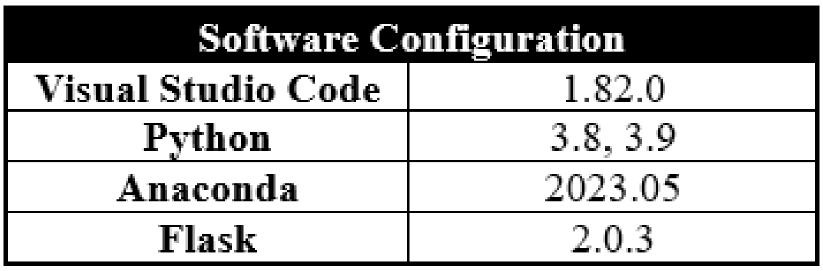
Hardware Configuration	
GPU	NVIDIA RTX 3080/3090, A100, V100, GTX 1660 Ti
CPU	Intel Core i7/i9, AMD Ryzen 7/9, Intel Xeon, AMD EPYC
RAM	32GB - 64GB (minimum), 128GB+ for large datasets
Storage (SSD)	1TB SSD (NVMe preferred), 2TB+ for larger datasets
Networking Hardware	Gigabit Ethernet, High-speed internet
Cloud Resources (Optional)	Google Cloud TPU

Figure 1: Hardware Configurations

2.2 Software Configuration

The following Figure 2 presents all the softwares utilized in this study identifying their respective version:

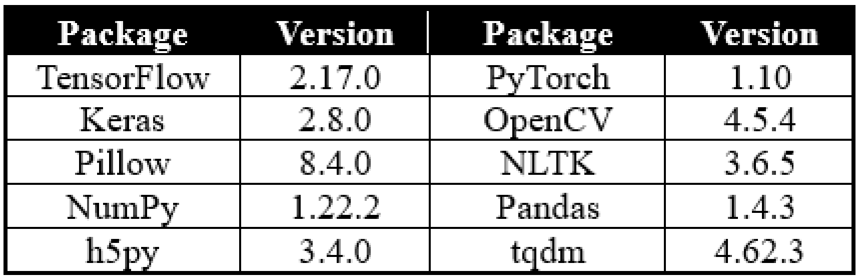
Visual Studio Code:
VS Code on the other hand is an IDE that is used for coding and debugging and managing projects smartly. They support extension for Python and other languages, which facilitates productivity of the image caption generation.



Software Configuration	
Visual Studio Code	1.82.0
Python	3.8, 3.9
Anaconda	2023.05
Flask	2.0.3

Figure 2: Software Configurations

Python Libraries:
Python[1] is the main working language for creating and executing the models and for most of the data manipulation and library interfacing. It is also supporting several libraries which are included in the presented study. The description and version of those libraries are described in the following Figure 3:



Package	Version	Package	Version
TensorFlow	2.17.0	PyTorch	1.10
Keras	2.8.0	OpenCV	4.5.4
Pillow	8.4.0	NLTK	3.6.5
NumPy	1.22.2	Pandas	1.4.3
h5py	3.4.0	tqdm	4.62.3

Figure 3: Python Libraries

Anaconda and Jupyter Notebook:
Anaconda[2] offers solutions related to package management and deployment, which means it takes care of all dependencies for Python projects, and the second tool is Jupyter Notebook, which is an interactive tool for Data Science used for prototyping and Data Visualization.

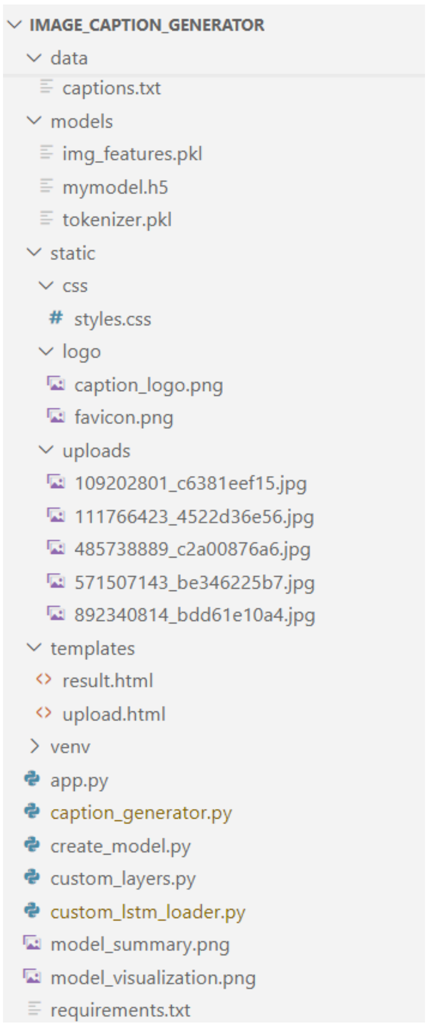
Flask:
Flask[3] is used to implement the web interface for deploying the image captioning models where users can interact with the deep learning application through a browser.

[1]`https://www.python.org/`
[2]`https://www.anaconda.com/download`
[3]`https://flask.palletsprojects.com/en/3.0.x/`

3 Environment Setup

3.1 Installing Required Packages and creating file structure

This section explains the installation of required packages for the project, which are prerequisite for the backend and model operations as part of the subsequent Section 4 and Section 6, respectively. The Image Caption Generator project is organized as shown in Figure 4 into several key directories: data for sample captions, models for the trained models along with the other necessary files and documents, static for HTML CSS, logos and images uploaded by the users, and templates/ for HTML files which in charge of the handling of interactions from the users. Within it, the necessary scripts in Python are stored among which it is possible to mention app.py file for the Flask application, caption_generator.py for caption processing and generation, and the create_model.py along with custom_layers.py for model creation. A virtual environment also known as a venv/ and requirements.txt ensure that dependencies are dealt with in the most efficient manner hence making the project easy to maintain as well as extend.

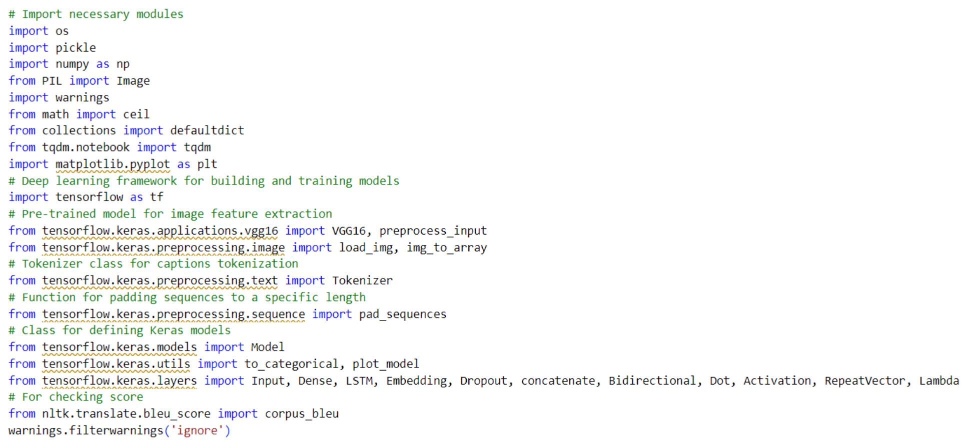


```
IMAGE_CAPTION_GENERATOR
  data
    captions.txt
  models
    img_features.pkl
    mymodel.h5
    tokenizer.pkl
  static
    css
      styles.css
    logo
      caption_logo.png
      favicon.png
    uploads
      109202801_c6381eef15.jpg
      111766423_4522d36e56.jpg
      485738889_c2a00876a6.jpg
      571507143_be346225b7.jpg
      892340814_bdd61e10a4.jpg
  templates
    result.html
    upload.html
  venv
  app.py
  caption_generator.py
  create_model.py
  custom_layers.py
  custom_lstm_loader.py
  model_summary.png
  model_visualization.png
  requirements.txt
```

Figure 4: File Structure

3.2 Importing Necessary Libraries

The libraries shown in Figure 5 are needed for various part of the project including image processing which is explained in Section 4.2, model creation which is explained in Section 6.2 and evaluation which is explained in Section 6.4.



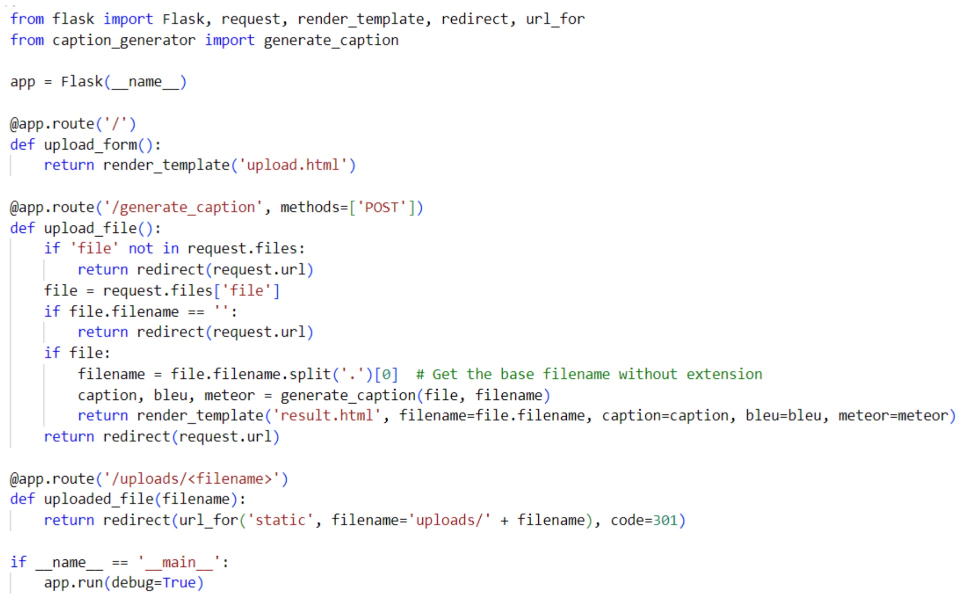
```python
# Import necessary modules
import os
import pickle
import numpy as np
from PIL import Image
import warnings
from math import ceil
from collections import defaultdict
from tqdm.notebook import tqdm
import matplotlib.pyplot as plt
# Deep learning framework for building and training models
import tensorflow as tf
# Pre-trained model for image feature extraction
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
# Tokenizer class for captions tokenization
from tensorflow.keras.preprocessing.text import Tokenizer
# Function for padding sequences to a specific length
from tensorflow.keras.preprocessing.sequence import pad_sequences
# Class for defining Keras models
from tensorflow.keras.models import Model
from tensorflow.keras.utils import to_categorical, plot_model
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout, concatenate, Bidirectional, Dot, Activation, RepeatVector, Lambda
# For checking score
from nltk.translate.bleu_score import corpus_bleu
warnings.filterwarnings('ignore')
```

Figure 5: Importing Necessary Libraries

4 Backend Implementation

4.1 Flask Application Setup

In this part, the creation of the web interface using Flask application as shown in Figure 6 and the routes that connect the image uploading and captioning services are described. The real process of caption generation described in Section 4.2 is based on the image preprocessing and model predicting stages described further.

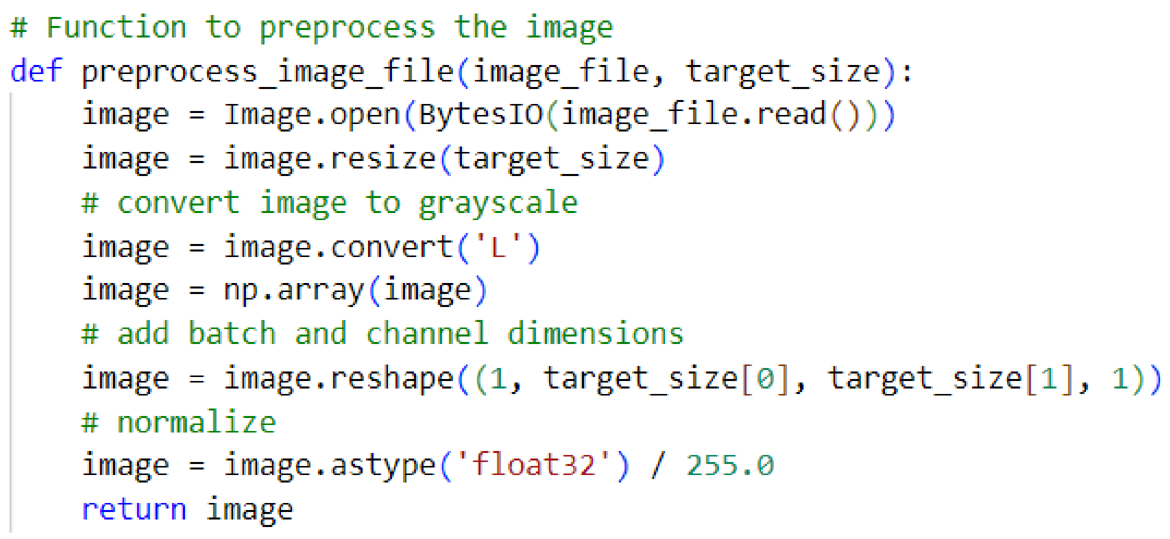


```python
from flask import Flask, request, render_template, redirect, url_for
from caption_generator import generate_caption

app = Flask(__name__)

@app.route('/')
def upload_form():
    return render_template('upload.html')

@app.route('/generate_caption', methods=['POST'])
def upload_file():
    if 'file' not in request.files:
        return redirect(request.url)
    file = request.files['file']
    if file.filename == '':
        return redirect(request.url)
    if file:
        filename = file.filename.split('.')[0]  # Get the base filename without extension
        caption, bleu, meteor = generate_caption(file, filename)
        return render_template('result.html', filename=file.filename, caption=caption, bleu=bleu, meteor=meteor)
    return redirect(request.url)

@app.route('/uploads/<filename>')
def uploaded_file(filename):
    return redirect(url_for('static', filename='uploads/' + filename), code=301)

if __name__ == '__main__':
    app.run(debug=True)
```

Figure 6: Flask Application Setup

4.2 Image Preprocessing and Caption Generation

The steps explained in Figure 7 about image preprocessing and caption generation are the fundamental steps of the application's functioning. This section lays down the data which is used in section 6.3 for building the model and caption generation.



```python
# Function to preprocess the image
def preprocess_image_file(image_file, target_size):
    image = Image.open(BytesIO(image_file.read()))
    image = image.resize(target_size)
    # convert image to grayscale
    image = image.convert('L')
    image = np.array(image)
    # add batch and channel dimensions
    image = image.reshape((1, target_size[0], target_size[1], 1))
    # normalize
    image = image.astype('float32') / 255.0
    return image
```

Figure 7: Image Preprocessing Caption Generation

5 Frontend Implementation

5.1 HTML Templates

The HTML[4] templates in this project constitute the front end for uploading images and showing generated captions and associated accuracy measures. These templates are important for the users to navigate and interact within the application.

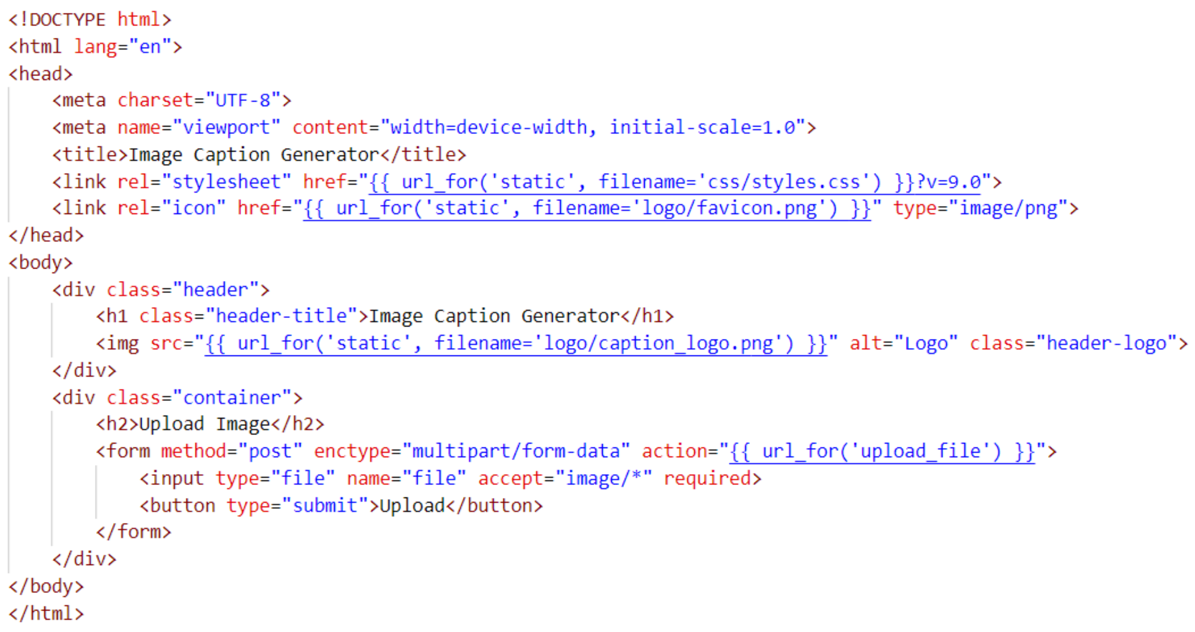
5.1.1 Upload Form (upload.html)

This section provides information on the HTML code of the image upload form as shown in Figure 8, which communicates with the backend routes explained in Section 4.1.
The upload.html template concerns the interface where a user can upload an image file in the application to generate captions. The key elements include:

- Header Section: Has a title with the text "Image Caption Generator" and a logo to provide a coherent and captivating appearance, which are written in HTML and then styled with CSS[5].

- Upload Form: Enables a user to choose and upload a picture file. The form sends data to the backend route named generate_caption through a Post method. Just like in the previous form, the file input is mandatory to prevent the user from submitting the form without choosing the image.

[4]https://html.com/
[5]https://www.w3schools.com/css/



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Image Caption Generator</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/styles.css') }}?v=9.0">
    <link rel="icon" href="{{ url_for('static', filename='logo/favicon.png') }}" type="image/png">
</head>
<body>
    <div class="header">
        <h1 class="header-title">Image Caption Generator</h1>
        <img src="{{ url_for('static', filename='logo/caption_logo.png') }}" alt="Logo" class="header-logo">
    </div>
    <div class="container">
        <h2>Upload Image</h2>
        <form method="post" enctype="multipart/form-data" action="{{ url_for('upload_file') }}">
            <input type="file" name="file" accept="image/*" required>
            <button type="submit">Upload</button>
        </form>
    </div>
</body>
</html>
```

Figure 8: Image Upload Form

Upon submitting the form with an image the file is sent to the server to begin the caption generation process described in Section 4.1.

5.1.2 Result Display (result.html)

This section describes the HTML template to render the caption generated along with the uploaded image and other performance metric in terms of BLEU[6] and METEOR[7] scores.
The result.html (Figure 9) is used to display the results when an image has been processed and a caption produced out of it. The key elements include:

- Result Display: The uploaded image is shown alongside the caption generated by the model. The image is sourced from the server, and the caption is dynamically inserted into the template.

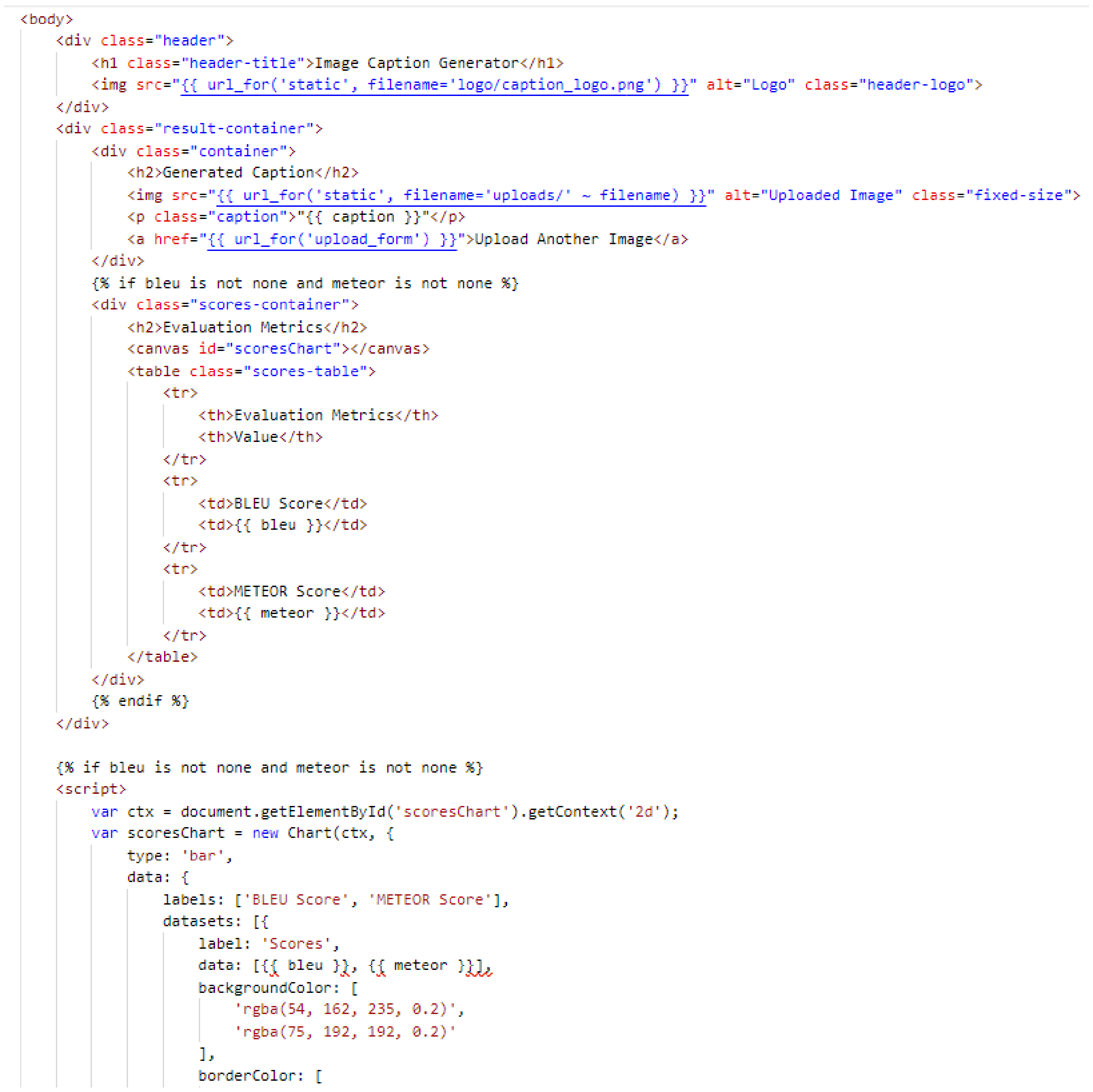
- Evaluation Metrics: If the BLEU and METEOR scores are available, the scores are given in tabular form and graphically with the help of Chart.

By using this template, the users have an easy way of making sure that the generated caption is correct, and also comparing it with the reference captions using the defined evaluation criteria.

5.2 CSS Styling (styles.css)

This section contains CSS that was used for the styling of the application and to enhance the user experience as they complete the form in Section 5.1.1 and while viewing the results using the result display template.

[6]https://thepythoncode.com/article/bleu-score-in-python
[7]https://www.nltk.org/api/nltk.translate.meteor_score.html



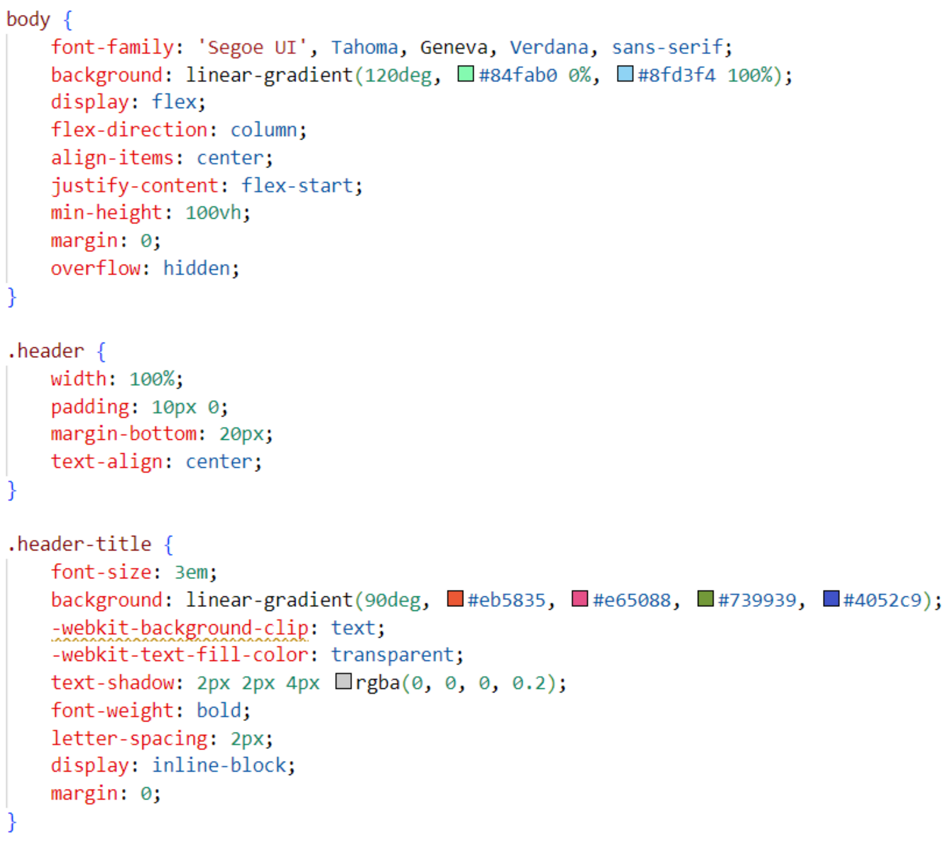
```
<body>
    <div class="header">
        <h1 class="header-title">Image Caption Generator</h1>
        <img src="{{ url_for('static', filename='logo/caption_logo.png') }}" alt="Logo" class="header-logo">
    </div>
    <div class="result-container">
        <div class="container">
            <h2>Generated Caption</h2>
            <img src="{{ url_for('static', filename='uploads/' ~ filename) }}" alt="Uploaded Image" class="fixed-size">
            <p class="caption">"{{ caption }}"</p>
            <a href="{{ url_for('upload_form') }}">Upload Another Image</a>
        </div>
        {% if bleu is not none and meteor is not none %}
        <div class="scores-container">
            <h2>Evaluation Metrics</h2>
            <canvas id="scoresChart"></canvas>
            <table class="scores-table">
                <tr>
                    <th>Evaluation Metrics</th>
                    <th>Value</th>
                </tr>
                <tr>
                    <td>BLEU Score</td>
                    <td>{{ bleu }}</td>
                </tr>
                <tr>
                    <td>METEOR Score</td>
                    <td>{{ meteor }}</td>
                </tr>
            </table>
        </div>
        {% endif %}
    </div>

    {% if bleu is not none and meteor is not none %}
    <script>
        var ctx = document.getElementById('scoresChart').getContext('2d');
        var scoresChart = new Chart(ctx, {
            type: 'bar',
            data: {
                labels: ['BLEU Score', 'METEOR Score'],
                datasets: [{
                    label: 'Scores',
                    data: [{{ bleu }}, {{ meteor }}],
                    backgroundColor: [
                        'rgba(54, 162, 235, 0.2)',
                        'rgba(75, 192, 192, 0.2)'
                    ],
                    borderColor: [
```

Figure 9: Caption Generation Display

The CSS styles the whole UI as shown in Figure 10, so the upload form and the results display are made visually engaging and therefore, more user-oriented. Key elements styled include:

- Body and Header: The application of gradient background and the layout with objects centralized minimize design's clutter look.
- Container Elements: The upload form and the displayed results are wrapped into containers with padding, border-radius, and a boxes shadow for a clean look.
- Buttons and Links: Aligned for conformity, hover effects to increase usability for the end user.

It should also be noted that this styling helps make the application not only functional but also beautiful and therefore more enjoyable to use.



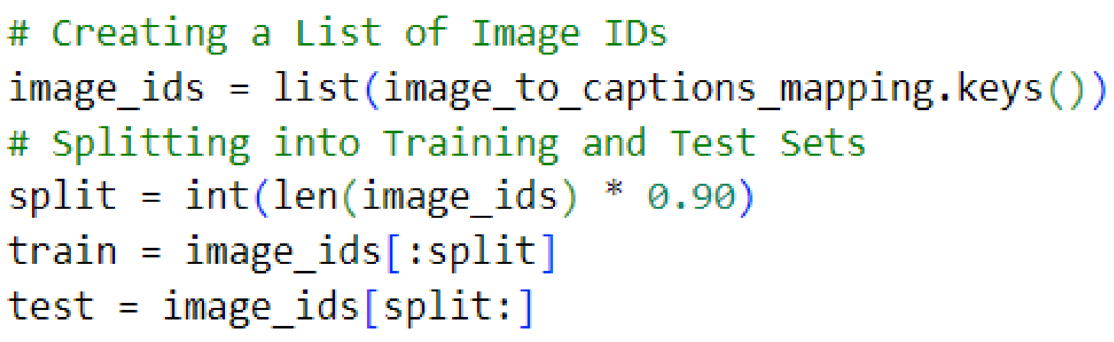
```css
body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    background: linear-gradient(120deg, #84fab0 0%, #8fd3f4 100%);
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: flex-start;
    min-height: 100vh;
    margin: 0;
    overflow: hidden;
}

.header {
    width: 100%;
    padding: 10px 0;
    margin-bottom: 20px;
    text-align: center;
}

.header-title {
    font-size: 3em;
    background: linear-gradient(90deg, #eb5835, #e65088, #739939, #4052c9);
    -webkit-background-clip: text;
    -webkit-text-fill-color: transparent;
    text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.2);
    font-weight: bold;
    letter-spacing: 2px;
    display: inline-block;
    margin: 0;
}
```

Figure 10: CSS Styling

6 Model Training and Evaluation

6.1 Data Preparation

This section covers data pre-processing for model training which is applied in section 6.3 for training the model. The data is split into 90% train and 10% test as shown in Figure 11.

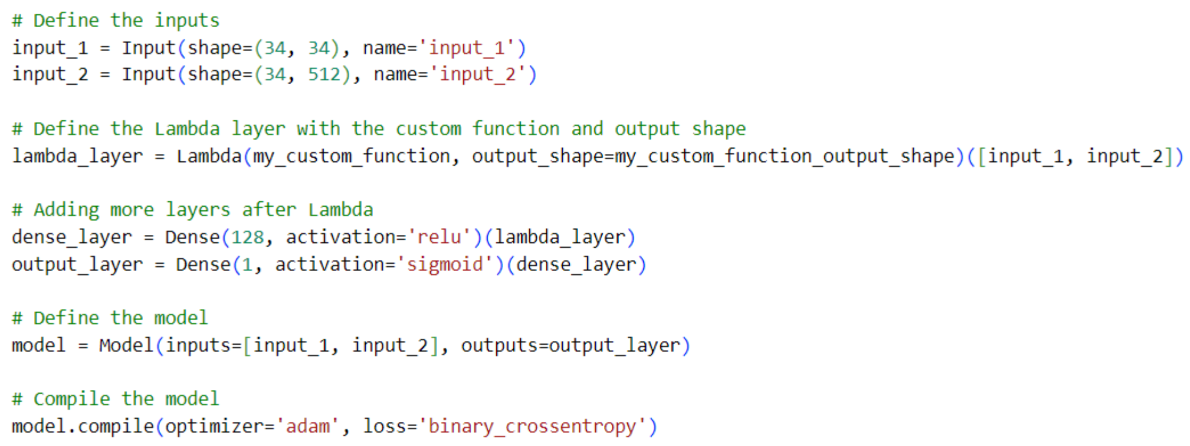


```python
# Creating a List of Image IDs
image_ids = list(image_to_captions_mapping.keys())
# Splitting into Training and Test Sets
split = int(len(image_ids) * 0.90)
train = image_ids[:split]
test = image_ids[split:]
```

Figure 11: Data Preparation

6.2 Model Definition

In this section as shown in Figure 12, the model is configured with the above said architecture which is to be trained on the data set prepared in section 6.1.

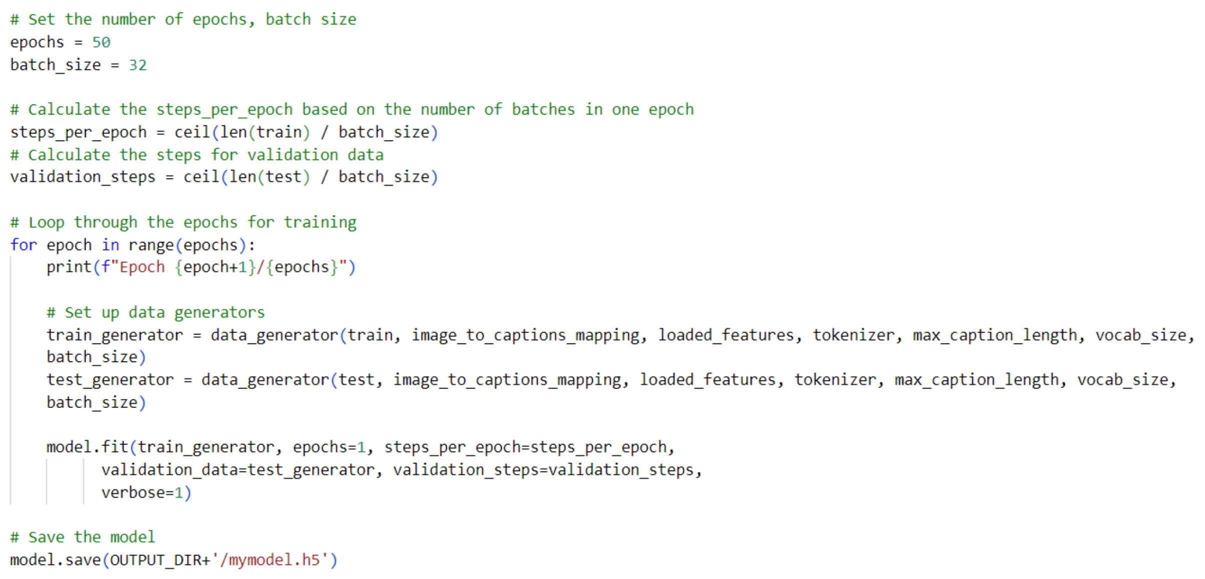


```
# Define the inputs
input_1 = Input(shape=(34, 34), name='input_1')
input_2 = Input(shape=(34, 512), name='input_2')

# Define the Lambda layer with the custom function and output shape
lambda_layer = Lambda(my_custom_function, output_shape=my_custom_function_output_shape)([input_1, input_2])

# Adding more layers after Lambda
dense_layer = Dense(128, activation='relu')(lambda_layer)
output_layer = Dense(1, activation='sigmoid')(dense_layer)

# Define the model
model = Model(inputs=[input_1, input_2], outputs=output_layer)

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy')
```

Figure 12: Model Definition

6.3 Model Training

This section develops from the findings made in section 6.1 and Section 6.2. In this case, the data prepared earlier are used to train the model and separate training and validational sets are defined as shown in Figure 13.



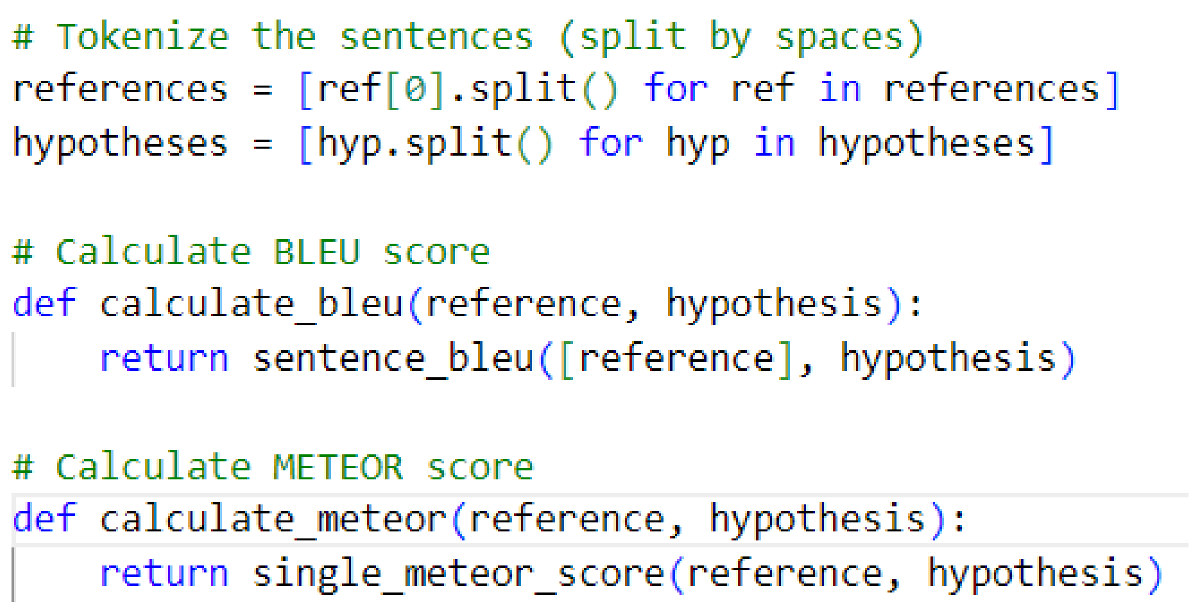
```
# Set the number of epochs, batch size
epochs = 50
batch_size = 32

# Calculate the steps_per_epoch based on the number of batches in one epoch
steps_per_epoch = ceil(len(train) / batch_size)
# Calculate the steps for validation data
validation_steps = ceil(len(test) / batch_size)

# Loop through the epochs for training
for epoch in range(epochs):
    print(f"Epoch {epoch+1}/{epochs}")

    # Set up data generators
    train_generator = data_generator(train, image_to_captions_mapping, loaded_features, tokenizer, max_caption_length, vocab_size,
    batch_size)
    test_generator = data_generator(test, image_to_captions_mapping, loaded_features, tokenizer, max_caption_length, vocab_size,
    batch_size)

    model.fit(train_generator, epochs=1, steps_per_epoch=steps_per_epoch,
          validation_data=test_generator, validation_steps=validation_steps,
          verbose=1)

# Save the model
model.save(OUTPUT_DIR+'/mymodel.h5')
```

Figure 13: Model Training

To train the model, over multiple epochs, the training process uses the training set defined in Section 6.2 and the data split defined in Section 6.1, whilst testing the validation of the model on an independent test set.

6.4 Evaluation Metrics

This section (Figure 14) explains how to compute evaluation metrics like BLEU and METEOR from the model of Section 6. These metrics are widely employed in the assessment of the quality of the generated captions with the help of the reference captions.



```python
# Tokenize the sentences (split by spaces)
references = [ref[0].split() for ref in references]
hypotheses = [hyp.split() for hyp in hypotheses]

# Calculate BLEU score
def calculate_bleu(reference, hypothesis):
    return sentence_bleu([reference], hypothesis)

# Calculate METEOR score
def calculate_meteor(reference, hypothesis):
    return single_meteor_score(reference, hypothesis)
```

Figure 14: Evaluation Metrics

- BLEU Score: The calculate_bleu function performs BLEU, which is the Bilingual Evaluation Understudy, to determine the precision of the n-gram in generated caption up to the standard captions. As a result, the higher BLEU is closer to the reference captions, which makes it possible to use the score to evaluate the performance of generated captions.

- METEOR Score: The calculate_meteor function EST (METS, Metric for Evaluation of Translation with Explicit ORdering) score is calculated for assessing the caption quality based on synonymy, stemming and word order. Therefore, METEOR can be considered as an extension of BLEU which gives a finer analysis and measures specific aspects such as the extent of reference's coverage by the generated text, its comprehensibility.

When combined, these scores provide a holistic assessment of the model's ability to create technically correct and semantically relevant captions as discussed in Section 6.3.