INTRODUCTION

1.1 Project Overview

The project aims to implement an Image Caption Generation system using deep learning techniques, specifically combining Convolutional Neural Networks (CNNs) for image feature extraction and Recurrent Neural Networks (RNNs), specifically Long Short-Term Memory (LSTM), for generating human-readable captions. The dataset used is Flickr8k, containing images and corresponding captions. The ResNet50 pre-trained model is employed to extract image features. The captions are pre-processed, tokenized, and converted into numerical sequences. A custom generator function is designed to feed the image features, previously generated text, and the next word to predict into the model.

The neural network model consists of an image feature extraction sub-model, a captions generation sub-model, and connected LSTM layers. The model is trained on this architecture using a subset of the dataset. The trained model, along with the vocabulary dictionary, is saved for future use. Finally, sample predictions are generated by randomly selecting images and predicting captions using the trained model.

1.2 Purpose

This project focuses on building an advanced Image Caption Generation system through the integration of cutting-edge deep learning techniques. By combining Convolutional Neural Networks (CNNs) with Recurrent Neural Networks (RNNs) using Long Short-Term Memory (LSTM) cells, the model leverages the powerful ResNet50 pre-trained architecture for feature extraction. The training process involves utilizing the well-established Flickr8k dataset, aiming to develop a sophisticated system capable of generating precise and contextually relevant captions for a variety of images.

The overarching purpose is to empower machines with a human-like ability to comprehend and describe visual content. Through this, the project seeks to advance the field of computer vision, enabling applications in accessibility tools, content indexing, and multimedia data understanding. Ultimately, the project's objective is to contribute to the progress of artificial intelligence by fostering a nuanced understanding of the intricate relationship between images and language. This, in turn, holds the potential to drive significant advancements in both computer vision and AI as a whole

LITERATURE SURVEY

2.1 Existing problem

Existing problems in the literature pertaining to image caption generation predominantly revolve around the limitations of traditional methods, which often struggle to capture the nuanced relationships between visual content and descriptive language. Conventional approaches lack the capacity to discern intricate details within images, resulting in generic and sometimes inaccurate captions. Additionally, these methods may falter in handling complex scenes, dynamic contexts, or images with multiple objects. Moreover, issues related to scalability and adaptability arise, hindering their performance on diverse datasets. The need for more sophisticated models that integrate deep learning techniques becomes evident, as such models have shown promise in capturing intricate visual features and context, offering a potential solution to the shortcomings of earlier methods.

2.2 Problem Statement Definition

The problem at hand revolves around the shortcomings of existing image caption generation methods, prompting the need for an advanced and more effective solution. Key issues include:

- **Semantic Gap:** Traditional methods struggle to bridge the semantic gap between visual content and language, resulting in captions that lack detail and precision.
- **Limited Context Understanding:** Conventional approaches often fail to comprehend the broader context of images, leading to inadequate descriptions, especially in complex scenes or scenarios involving multiple objects.
- **Inaccuracies and Ambiguities:** The inherent limitations of non-deep learning methods contribute to inaccuracies and ambiguities in generated captions, impacting the overall quality and informativeness.
- **Scalability Challenges:** Traditional models may face scalability challenges, particularly when dealing with diverse datasets or evolving visual recognition tasks, restricting their adaptability and generalization.
- **Dependency on Handcrafted Features:** Many existing methods rely heavily on handcrafted features, making them less adept at automatically extracting intricate visual details and adapting to the wide variability of real-world images.
- **Lack of Dynamic Adaptability:** The rigid nature of conventional models hampers their ability to dynamically adapt to different image types, making them less robust in handling variations in content, style, and context.

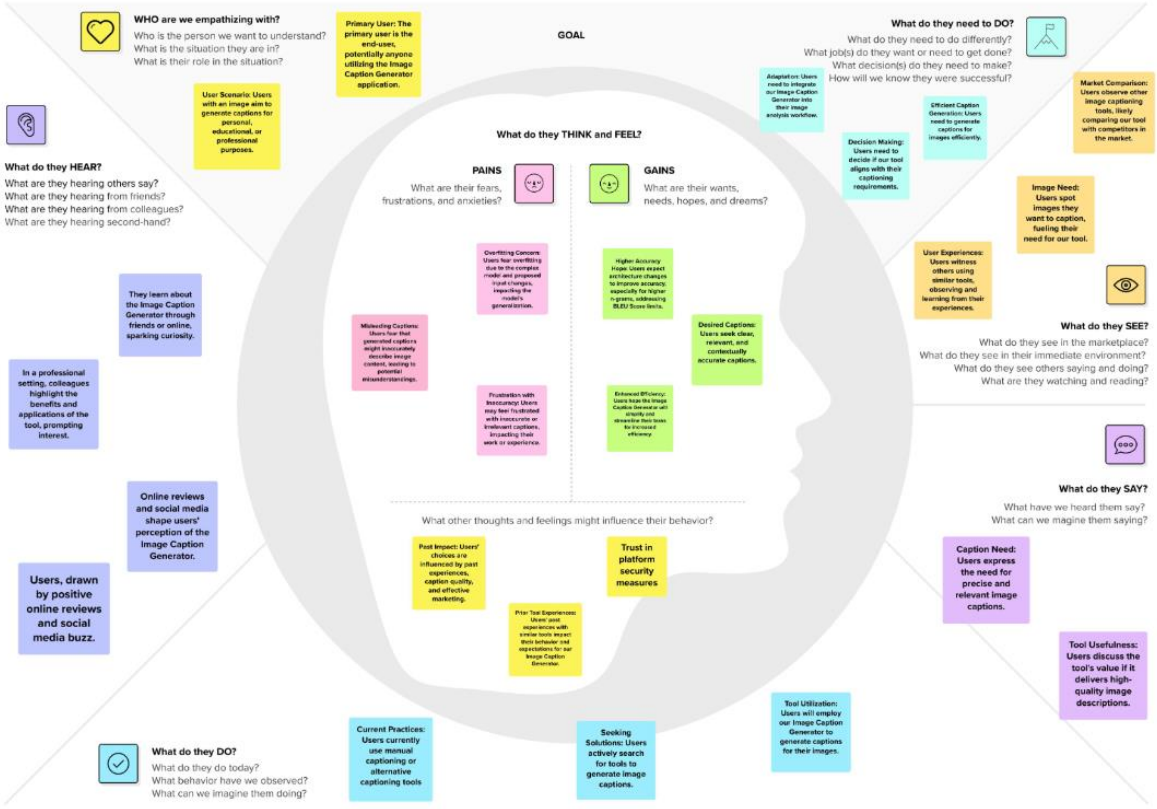
Addressing these issues requires the development of a novel image caption generation system that leverages the power of deep learning, specifically combining Convolutional Neural Networks (CNNs) for robust image feature extraction and Recurrent Neural Networks (RNNs) for context-aware caption generation.

IDEATION AND PROPOSED SOLUTION

3.1 Empathy Map Canvas

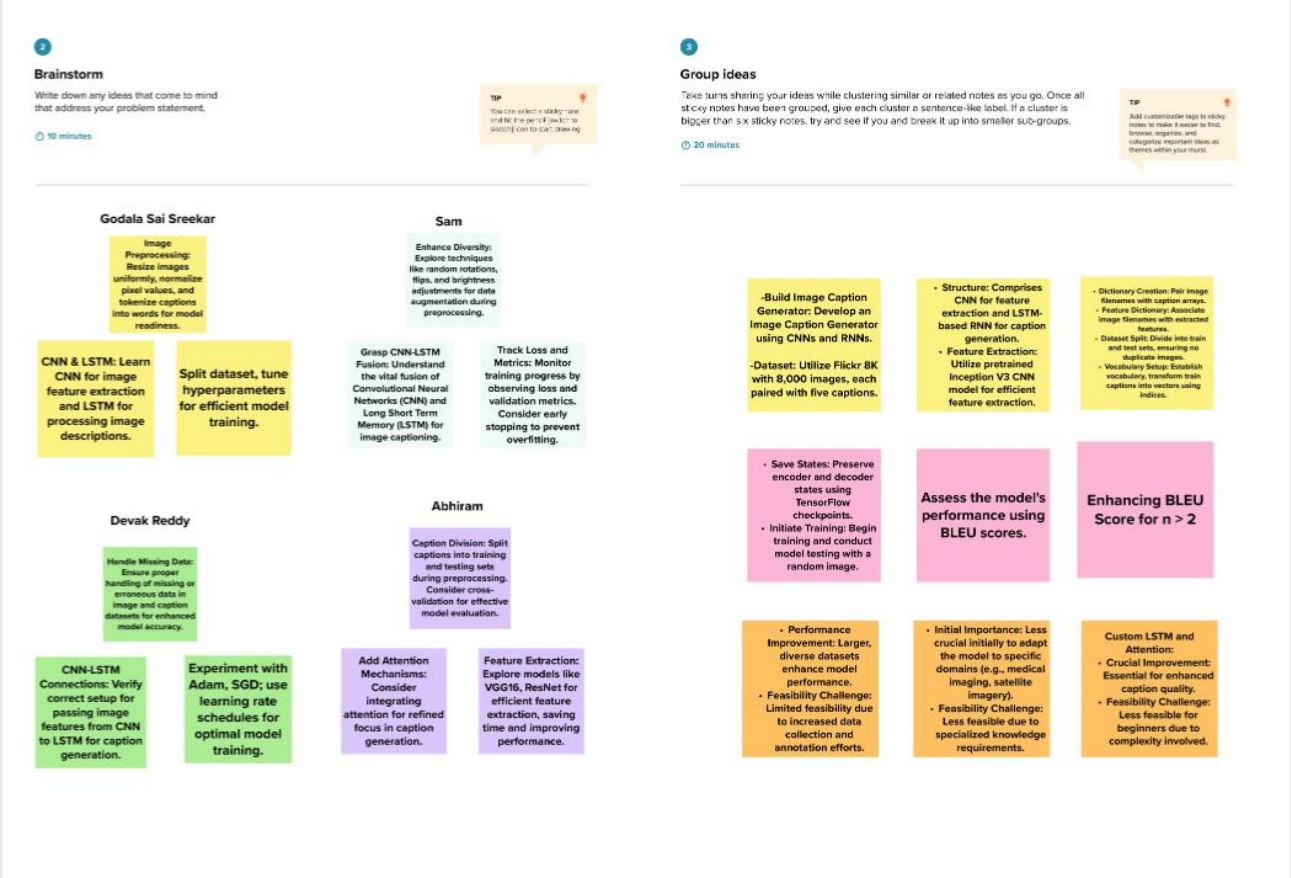
An empathy map is a straightforward and visually intuitive representation that compiles information about a user's actions and mindset. This tool serves as a valuable resource for teams seeking to gain deeper insights into their users. To devise a successful solution, it is

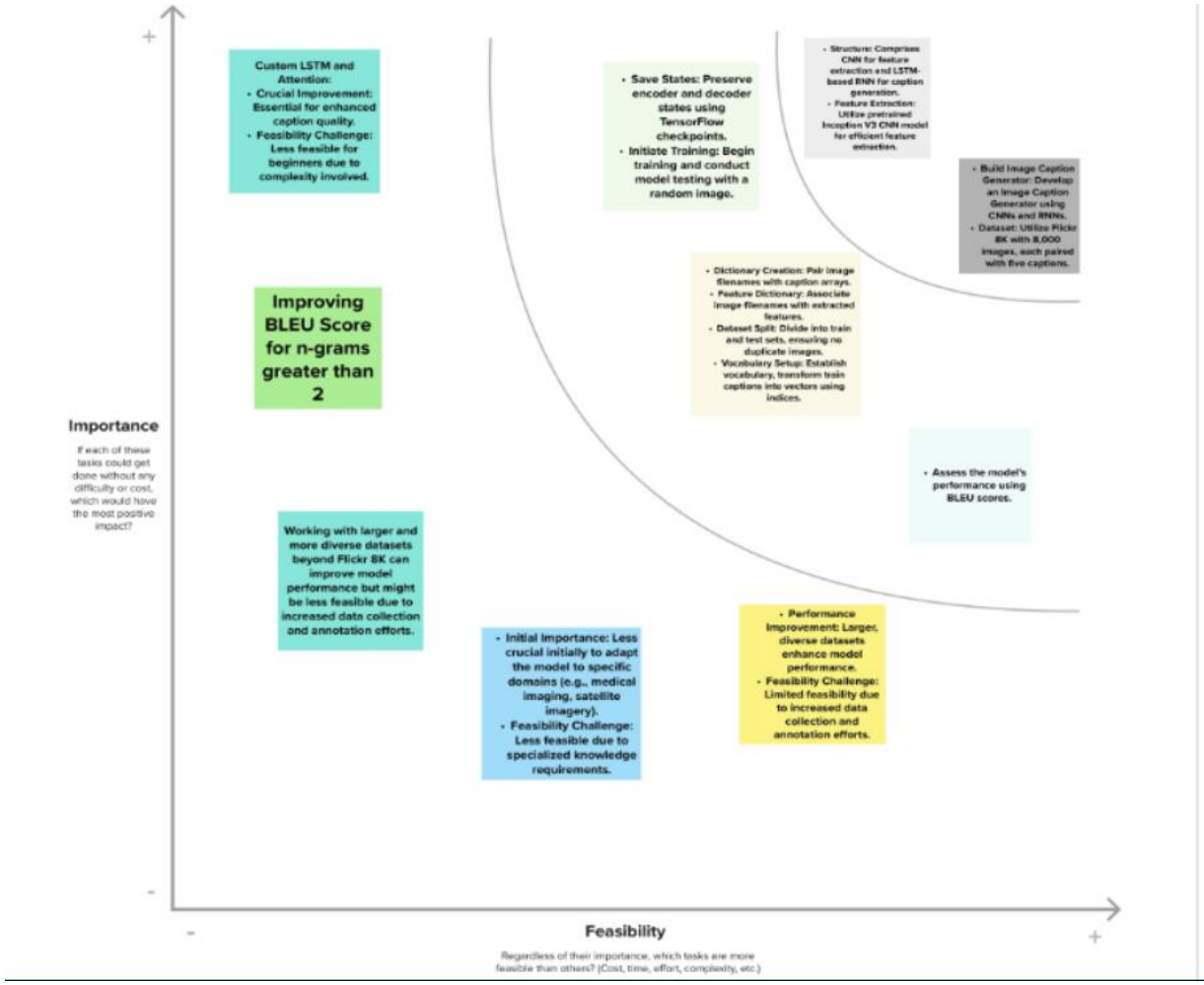
crucial to grasp the actual issue at hand and gain a comprehensive understanding of the individual undergoing it. The process of constructing this map encourages participants to view matters through the user's lens, taking into account their objectives and obstacles.



3.2 Ideation & Brainstorming

Brainstorming cultivates a liberated and inclusive atmosphere where all team members are prompted to engage in the creative thinking process essential for problem-solving. Emphasizing quantity of ideas over their immediate worth, unconventional suggestions are embraced and refined. The collaborative spirit among participants is actively promoted, facilitating the generation of a diverse array of innovative solutions. Utilize this framework in your own brainstorming sessions to empower your team in unleashing their creative potential and shaping ideas, even when they are not physically present in the same space.




+
-
Importance
If each of these tasks could get done without any difficulty or cost, which would have the most positive impact?
Custom LSTM and Attention:
• Crucial Improvement: Essential for enhanced caption quality.
• Feasibility Challenge: Less feasible for beginners due to complexity involved.
• Save States: Preserve encoder and decoder states using TensorFlow checkpoints.
• Initiate Training: Begin training and conduct model testing with a random image.
• Structure: Comprises CNN for feature extraction and LSTM-based RNN for caption generation.
• Feature Extraction: Utilize pretrained Inception V3 CNN model for efficient feature extraction.
• Build Image Caption Generator: Develop an Image Caption Generator using CNNs and RNNs.
• Dataset: Utilize Flickr 8K with 8,000 images, each paired with five captions.
Improving BLEU Score for n-grams greater than 2
• Dictionary Creation: Pair image filenames with caption arrays.
• Feature Dictionary: Associate image filenames with extracted features.
• Dataset Split: Divide into train and test sets, ensuring no duplicate images.
• Vocabulary Setup: Establish vocabulary, transform train captions into vectors using indices.
• Assess the model's performance using BLEU scores.
Working with larger and more diverse datasets beyond Flickr 8K can improve model performance but might be less feasible due to increased data collection and annotation efforts.
• Initial Importance: Less crucial initially to adapt the model to specific domains (e.g., medical imaging, satellite imagery).
• Feasibility Challenge: Less feasible due to specialized knowledge requirements.
• Performance Improvement: Larger, diverse datasets enhance model performance.
• Feasibility Challenge: Limited feasibility due to increased data collection and annotation efforts.
-
+
Feasibility
Regardless of their importance, which tasks are more feasible than others? (Cost, time, effort, complexity, etc.)

REQUIREMENTS ANALYSIS

4.1 Functional Requirements

1. Image Feature Extraction

- The system should employ Convolutional Neural Networks (CNNs) for extracting high-level features from input images.

- It must support the integration of pre-trained models like ResNet50 for efficient and effective feature extraction.

2. Caption Generation Model:

- The system should implement Recurrent Neural Networks (RNNs), specifically Long Short-Term Memory (LSTM) networks, for generating coherent and contextually relevant captions.

- It must include a captions generation sub-model that seamlessly integrates with the image feature extraction process.

3. Dataset Handling

- The system must be capable of ingesting and processing image-caption datasets, with support for standard formats like COCO or Flickr8k.

- It should include functionalities for data pre-processing, tokenization, and sequence conversion to facilitate model training.

4. Training and Model Optimization:

- The system should provide mechanisms for training the neural network model using the specified architecture.

- It must incorporate optimization techniques, such as backpropagation and gradient descent, to enhance the model's accuracy and convergence.

5. Custom Generator Function:

- A custom generator function should be implemented to feed image features, previously generated text, and the next word to predict into the model during training.

6. Model Persistence:

- The system must support the saving and loading of trained models, including both the image feature extraction sub-model and the captions generation sub-model.

7. Vocabulary Management:

- It should include functionalities for managing the vocabulary dictionary, including updates based on the dataset and ensuring consistency during training and inference.

8. Prediction and Evaluation:

- The system should enable the generation of image captions for new, unseen images using the trained model.

- It must provide evaluation metrics, such as BLEU score or METEOR, to assess the quality and relevance of generated captions.

9. Scalability and Adaptability:

- The system should be designed to scale efficiently with varying dataset sizes and complexities.

- It must demonstrate adaptability to different visual recognition tasks and evolving requirements in the field of image caption generation.

10. User Interface (Optional):

- If applicable, the system may include a user interface for users to interact with the model, input images, and view generated captions.

4.2 Non – Functional Requirements

1. Performance:

- The system should demonstrate low-latency response times for both training and inference, ensuring efficient processing of large image datasets.

- It must handle multiple simultaneous requests with minimal impact on performance.

2. Scalability:

- The architecture should be scalable to accommodate an increasing volume of images and captions, allowing the system to grow with expanding datasets.

3. Reliability:

- The system should operate reliably under varying conditions, including diverse image types, sizes, and caption complexities.

- It must handle errors gracefully and recover without compromising the overall functionality.

4. Security:

- The system should implement security measures to protect sensitive data, including images and captions, from unauthorized access or modification.

- It must adhere to best practices for securing neural network models and training data.

5. Usability:

- The user interface, if applicable, should be intuitive and user-friendly, facilitating easy interaction with the image caption generation system.

- The system should provide clear feedback and instructions for users, including guidance on interpreting and improving generated captions.

6. Maintainability:

- The codebase should be well-documented, modular, and adhere to coding standards to facilitate ease of maintenance and future updates.

- Regular updates and patches should be straightforward to implement without major disruptions to the system.

7. Compatibility:

- The system should be compatible with popular deep learning frameworks, ensuring flexibility for users who may prefer specific frameworks for model development and deployment.

8. Adaptability:

- The system should be adaptable to different hardware configurations, allowing deployment on a variety of computing environments, from local machines to cloud-based infrastructure.

9. Ethical Considerations:

- The system should adhere to ethical guidelines, promoting fair and unbiased image caption generation.

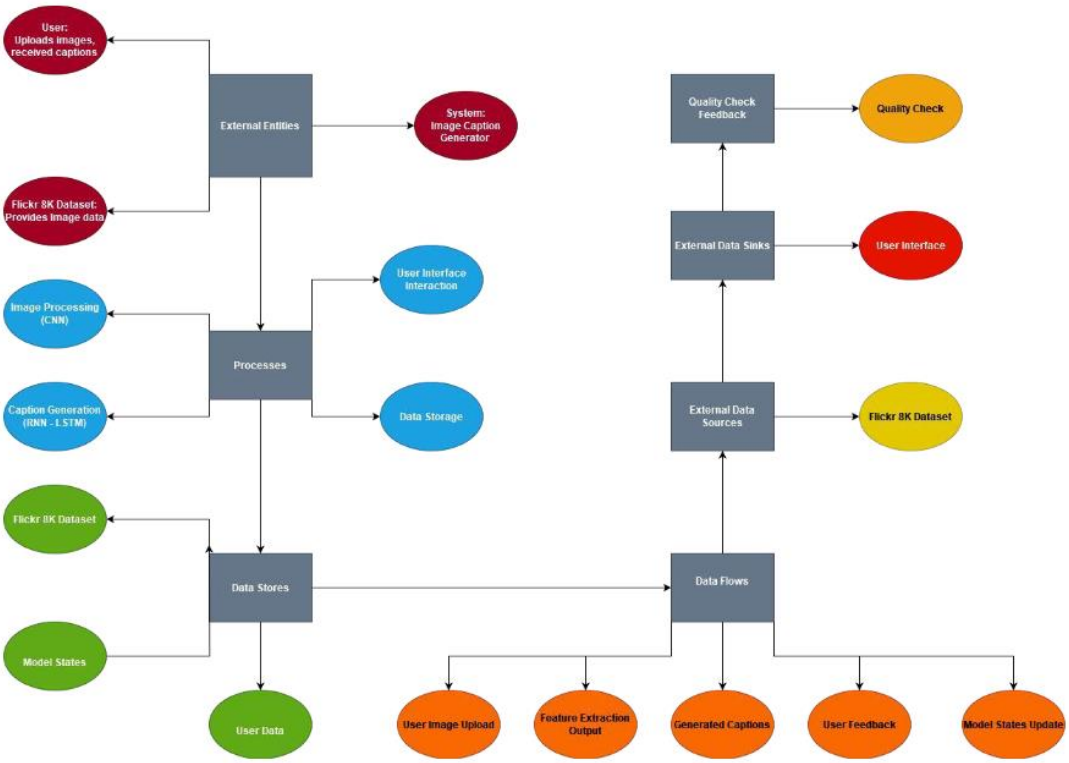
- It should avoid generating offensive or inappropriate content, and mechanisms should be in place to prevent biased outcomes.

10. Documentation:

- Comprehensive documentation should be provided, including user manuals, developer guides, and model architecture descriptions, to facilitate understanding and utilization of the system by different stakeholders.

PROJECT DESIGN

5.1 Data Flow Diagrams & Solution Architecture



User Stories

User Stories

Use the below template to list all the user stories for the product.

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile User)	Upload Image for Caption	USN-1	As a mobile user, I can upload an image to the application for generating captions.	I can access my account / dashboard	High	Sprint-1
	View Generated Captions	USN-2	As a mobile user, I can view the captions generated by the ML/DL model for my uploaded images	I can scroll through or download the captions.	High	Sprint-1
	Provide Feedback on Captions	USN-3	As a mobile user, I can provide feedback on the accuracy and relevance of captions generated by the ML/DL model.	I can submit comments or ratings for each caption.	Medium	Sprint-2
Customer (Web User)	Registration/Login	USN-7	As a user, I can register for the application by entering my email, password, and confirming my password., I can log into the application by entering email and password.	I can access my acount/dashboard	High	Sprint-1
	Web Image Upload	USN-4	As a web user, I can upload images to the application for generating captions using the ML/DL model	I can view and download captions generated for my uploaded images.	High	Sprint-1
	Explore Caption History	USN-5	As a web user, I can explore the history of captions generated by the ML/DL model for my uploaded images.	I can filter, search, or organize the caption history.	Medium	Sprint-2
	Customize Caption Preferences	USN-6	As a web user, I can customize preferences for how captions are generated by the ML/DL model (e.g., tone, length).	I can apply these preferences to future image uploads.	Medium	Sprint-2
Customer Care Executive	Review User Feedback	CCE-1	As a Customer Care Executive, I can access and review user feedback on generated captions.	I can prioritize and address user concerns or issues related to caption accuracy.	High	Sprint-2
	Provide Support for Caption Issues	CCE-2	As a Customer Care Executive, I can respond to user feedback regarding caption inaccuracies.	I can provide clarifications, assistance, or escalate issues to the development team.	High	Sprint-2
	Monitor Caption Quality Trends	CCE-3	As a Customer Care Executive, I can monitor trends in user feedback to identify patterns in caption quality.	I can collaborate with the development team to improve the ML/DL model.	Medium	Sprint-3
Administrator	Manage User Access and Roles	ADM-1	As an Administrator, I can manage user roles and access levels within the application.	I can assign roles such as Customer, Customer Care Executive, and Administrator.	High	Sprint-1
	Monitor System Performance	ADM-2	As an Administrator, I can monitor the performance of the ML/DL model and overall system.	I can identify potential issues and ensure optimal system functionality.	High	Sprint-1
	Update ML/DL Model	ADM-3	As an Administrator, I can initiate updates to the ML/DL model.	I can coordinate with the development team to implement improvements or modifications.	High	Sprint-2
	Manage User Data	ADM-4	As an Administrator, I can manage user data, ensuring compliance with privacy regulations.	I can oversee data storage, retention policies, and user data security.	High	Sprint-1

Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

• Find the best tech solution to solve existing business problems.

• Describe the structure, characteristics, behavior, and other aspects of the software to project stakeholders.

• Define features, development phases, and solution requirements.

• Provide specifications according to which the solution is defined, managed, and delivered.

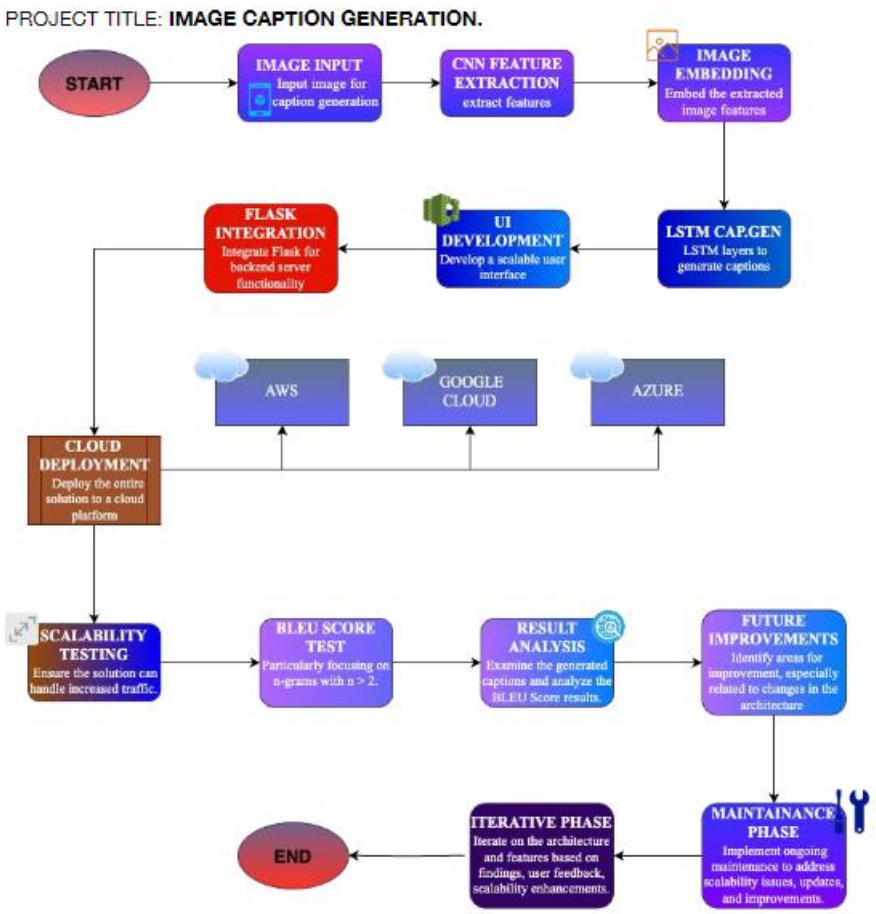


Figure 1: Architecture and data flow of the voice patient diary sample application

PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture

Technical Architecture:

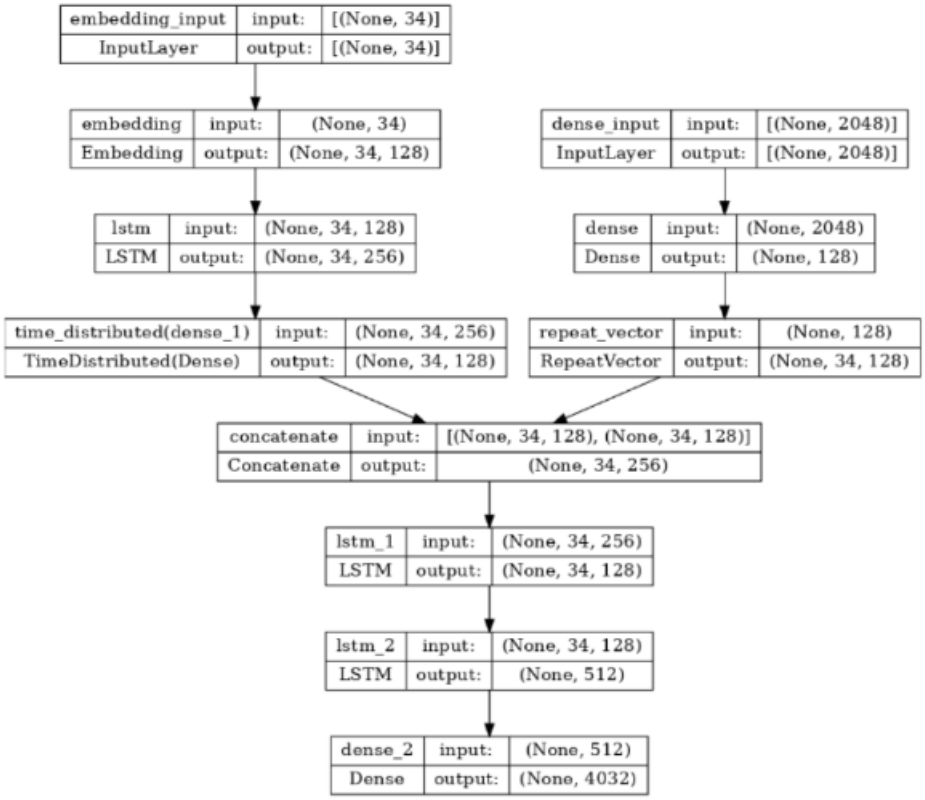


Table-1 : Components & Technologies:

Sl.no	Component	Description	Technology
1.	User Interface	How user interacts with application: User will have a feature to upload the image and get the caption of the image.	HTML, CSS, JavaScript / React/ Bootstrap.
2.	Application Logic-1	combination of pre-trained CNN and LSTM models to extract features and generate a descriptive caption	Python
3.	Application Logic-2	Utilize IBM Watson services to analyze and interpret user-provided data	IBM Watson Application
4.	Application Logic-3	Flask routes to handle incoming requests, process data, and return appropriate responses.	Flask
5.	Database	Data Type, Configurations etc.	MySQL, NoSQL
6.	Cloud Database	Database Service on Cloud	IBM , IBM Cloudant
7.	Machine Learning Model	A machine learning model learns from data to make predictions, automating decision-making for enhanced accuracy and efficiency	LSTM,CSS
8.	Infrastructure (Server / Cloud)	Application Deployment on Local System / CloudLocal Server Configuration	Local, Cloud Foundry, Kubernetes, etc

Table-2: Application Characteristics:

Sl.no	Characteristics	Description	Technology
1.	Open-Source Frameworks	List the open-source frameworks used	Flask,Bootstrap,Scikit-learn
2.	Scalable Architecture	Growing demands without compromising performance, ensuring seamless expansion as needed	Docker for containerization To maintain consistency across different environments.
3.	Availability	The application ensures high availability through the implementation of load balancers, distributed server architecture, and redundant systems	Deploying the application on distributed servers Or using a serverless Architecture.
4.	Performance	Design consideration for the performance of the application (number of requests per sec, use of Cache, use of CDN's) etc.	Caching mechanisms to store frequently accessed data and reduce response times, Optimizing machine learning model performance for real-time classification

6.2 Sprint Planning & Estimation

Product Backlog, Sprint Schedule, and Estimation (4 Marks)

Use the below template to create product backlog and sprint schedule

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Data collection	USN-1	Gather and process transaction data for the model training.	3	High	Abhiram , Devak
Sprint-1	Model Development	USN-2	Build a machine learning model for the image caption generation	5	High	Sreekar, Sam
Sprint-1	Testing the Model accuracy	USN-3	Testing the prepared model,by uploading different images and verifying their captions	1	Medium	Abhiram
Sprint-2	Flask Integration	USN-4	Integrate the ml model with the help of Flask for online interface	3	Medium	Sreekar
Sprint-2	Html,Bootstrap UI	USN-5	Building a website which is user-friendly that interacts the system and displays the desired output	2	Low	Sam
Sprint-3	Deployment on IBM Cloud	USN-6	Deploy the system on IBM cloud for scalability	4	High	Sreekar, Sam
Sprint-3	Documentation	USN-7	Document the project for future purpose	2	Low	Devak

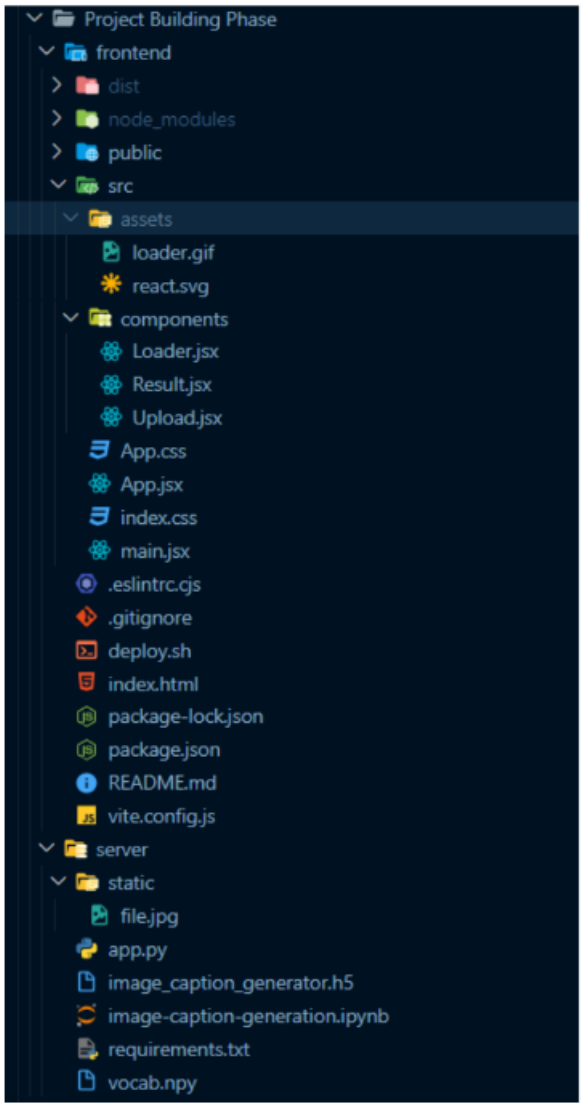
Project Tracker, Velocity & Burndown Chart: (4 Marks)

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date(Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date(Actual)
Sprint-1	9	14 days	27 Oct 2023	01 Nov 2023	9	01 Nov 2023
Sprint-2	20	6 Days	10 Nov 2023	17 Nov 2023	5	16 Nov 2023
Sprint-3	20	6 Days	17 Nov 2023	20 Nov 2023	6	20 Nov 2023

CODING AND SOLUTION

Project Structure:

Create the Project folder which contains files as shown below



Pre-requisites

Software and Tools

- Anaconda Navigator: Free and open-source distribution of Python and R for data science and machine learning applications.
- Jupyter Notebook and Spyder: Tools provided by Anaconda for development.
- Conda: Cross-platform package management system.
- Packages: Install the following packages using Anaconda prompt as administrator:
  - NumPy: `pip install numpy`
  - Pandas: `pip install pandas`
  - Scikit-learn: `pip install scikit-learn`
  - TensorFlow: `pip install tensorflow==2.3.2`
  - Keras: `pip install keras==2.3.1`
  - Flask: `pip install Flask`

Deep Learning Concepts

- Convolutional Neural Network (CNN): A class of deep neural networks for visual imagery analysis.
- Flask: A Python web framework for building web applications.

Project Objectives

By the end of this project, you will:

- Understand fundamental concepts and techniques of Convolutional Neural Network.
- Gain a broad understanding of image data.
- Learn how to pre-process/clean data using different techniques.
- Know how to build a web application using the Flask framework.

Project Flow

Data Collection

- Collect images of events with 5 captions each, organized into subdirectories.
- Download the dataset: Flickr8k Dataset.

Image Pre-processing and Model Building

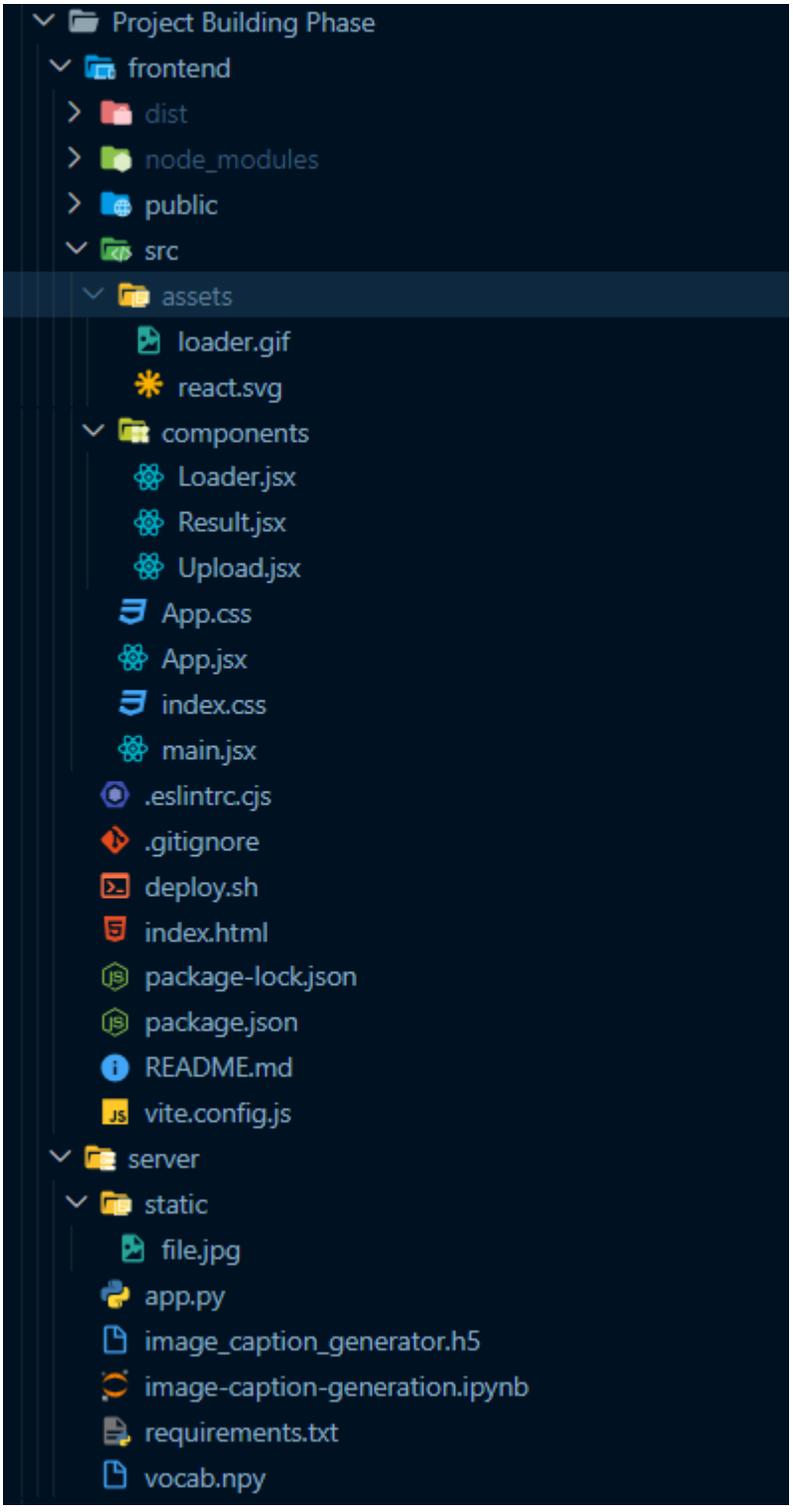
1. Importing Model Building Libraries
    - Import necessary libraries for model building.
2. Exploratory Data Analysis
    - Analyze and explore the dataset.
3. Initializing the Model
    - Use Keras Sequential class to define and initialize the model.
4. Configure the Learning Process
    - Compile the model with a loss function, optimizer, and metrics.
5. Training the Model
    - Train the model using image dataset.
6. Save the Model
    - Save the trained model in H5 format.
7. Test the Model
    - Evaluate the model's performance using test data.

Application Building

1. Create HTML Pages
    - Create HTML pages for home, introduction, and upload functionalities.
2. Python Flask Script (app.py)
    - Write Python script using Flask to run the project.

Project Structure

- Project Folder:
    - Dataset: Contains training and testing images.
    - Flask Application:
        - Templates Folder: Contains HTML pages (index.html, prediction.html).
        - Python Script (app.py): For server-side scripting.
    - Model: Saved as model.h5.
    - Captions: Stored as tokenizer.pkl.


Project Building Phase
frontend
dist
node_modules
public
src
assets
loader.gif
react.svg
components
Loader.jsx
Result.jsx
Upload.jsx
App.css
App.jsx
index.css
main.jsx
.eslintrc.cjs
.gitignore
deploy.sh
index.html
package-lock.json
package.json
README.md
vite.config.js
server
static
file.jpg
app.py
image_caption_generator.h5
image-caption-generation.ipynb
requirements.txt
vocab.npy

Milestones

Milestone 1: Collection of Data

- Collect images and captions based on the project structure. We have used Flickr 8k Dataset

Milestone 2: Image Pre-processing and Model Building

- Import libraries, explore data,

Importing the relevant libraries

+ Code + Markdown

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from glob import glob
from tqdm.notebook import tqdm
tqdm.pandas()
import cv2, warnings
warnings.filterwarnings('ignore')
import tensorflow as tf
from tensorflow.keras.layers import Dense, Flatten, Input, Add, Dropout, LSTM, TimeDistributed, Embedding, R
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.utils import to_categorical, plot_model
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import ModelCheckpoint
```

Loading the images

[ ]:

```
img_path = '/kaggle/input/flickr8k/Images/'
images = glob(img_path+'*.jpg')
images[:5]
```

[ ]:

```
len(images)
```

Loading the captions

```
captions = open('/kaggle/input/flickr8k/captions.txt','rb').read().decode('utf-8').split('\n')
captions[:5]
```

```
len(captions)
```

Visualizing images along with their captions

```
for i in range(5):
    plt.figure(figsize=(5,5))
    img = cv2.imread(images[i])
    img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    plt.imshow(img);
```

Preprocessing the captions text

```
captions = captions[1:]
captions[:5]
```

```
captions[8].split(',')[1]
```

```
captions_dict = {}

for cap in captions:
    try:
        img_name = cap.split(',')[0]
        caption = cap.split(',')[1]
        # Each image has 5 captions
        if img_name in img_features:
            if img_name not in captions_dict:
                captions_dict[img_name] = [caption] # Storing the first caption
            else:
                captions_dict[img_name].append(caption) # Adding the remaining captions
    except:
        break
```

```
len(captions_dict)
```

```
def text_preprocess(text):
    modified_text = text.lower() # Converting text to lowercase
    modified_text = 'startofseq ' + modified_text + ' endofseq' # Appending the special tokens at the beginning and ending of text
    return modified_text
```

```
# Storing the preprocessed text within the captions dictionary
for key, val in captions_dict.items():
    for item in val:
        captions_dict[key][val.index(item)] = text_preprocess(item)
```

- Initialize and configure the model, train, save, and test.

Downloading the ResNet50 inception model

[ ]:
```
inception_model = ResNet50(include_top=True)
inception_model.summary()
```

[ ]:
```
last = inception_model.layers[-2].output # Output of the penultimate layer of ResNet model
model = Model(inputs=inception_model.input,outputs=last)
model.summary()
```

Extracting features from images

[ ]:
```
img_features = {}
count = 0

for img_path in tqdm(images):
    img = cv2.imread(img_path)
    img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    img = cv2.resize(img,(224,224)) # ResNet model requires images of dimensions (224,224,3)
    img = img.reshape(1,224,224,3) # Reshaping image to the dimensions of a single image
    features = model.predict(img).reshape(2048,) # Feature extraction from images
    img_name = img_path.split('/')[-1] # Extracting image name
    img_features[img_name] = features
    count += 1
    # Fetching the features of only 1500 images as using more than 1500 images leads to overloading memory
    if count == 1500:
        break
    if count % 50 == 0:
        print(count)
```

Establishing the model architecture

[ ]:
```
embedding_len = 128
MAX_LEN = max_len
vocab_size = len(count_words)

# Model for image feature extraction
img_model = Sequential()
img_model.add(Dense(embedding_len,input_shape=(2048,),activation='relu'))
img_model.add(RepeatVector(MAX_LEN))

img_model.summary()

# Model for generating captions from image features
captions_model = Sequential()
captions_model.add(Embedding(input_dim=vocab_size+1,output_dim=embedding_len,input_length=MAX_LEN))
captions_model.add(LSTM(256,return_sequences=True))
captions_model.add(TimeDistributed(Dense(embedding_len)))

captions_model.summary()

# Concatenating the outputs of image and caption models
concat_output = Concatenate()([img_model.output,captions_model.output])
# First LSTM Layer
output = LSTM(units=128,return_sequences=True)(concat_output)
# Second LSTM Layer
output = LSTM(units=512,return_sequences=False)(output)
# Output Layer
output = Dense(units=vocab_size+1,activation='softmax')(output)
# Creating the final model
final_model = Model(inputs=[img_model.input,captions_model.input],outputs=output)
final_model.compile(loss='categorical_crossentropy',optimizer='RMSprop',metrics='accuracy')
final_model.summary()
```

Model training

[ ]:
```
mc = ModelCheckpoint('image_caption_generator.h5',monitor='accuracy',verbose=1,mode='max',save_best_only=True)

final_model.fit([X,y_in],
                y_out,
                batch_size=512,
                callbacks=mc,
                epochs=200)
```

[ ]:
```
# Creating an inverse dictionary with reverse key-value pairs
inverse_dict = {val: key for key,val in count_words.items()}
```

Saving the final trained model and the vocabulary dictionary

[ ]:
```
final_model.save('image_caption_generator.h5')
```

[ ]:
```
np.save('vocab.npy',count_words)
```

Generating sample predictions

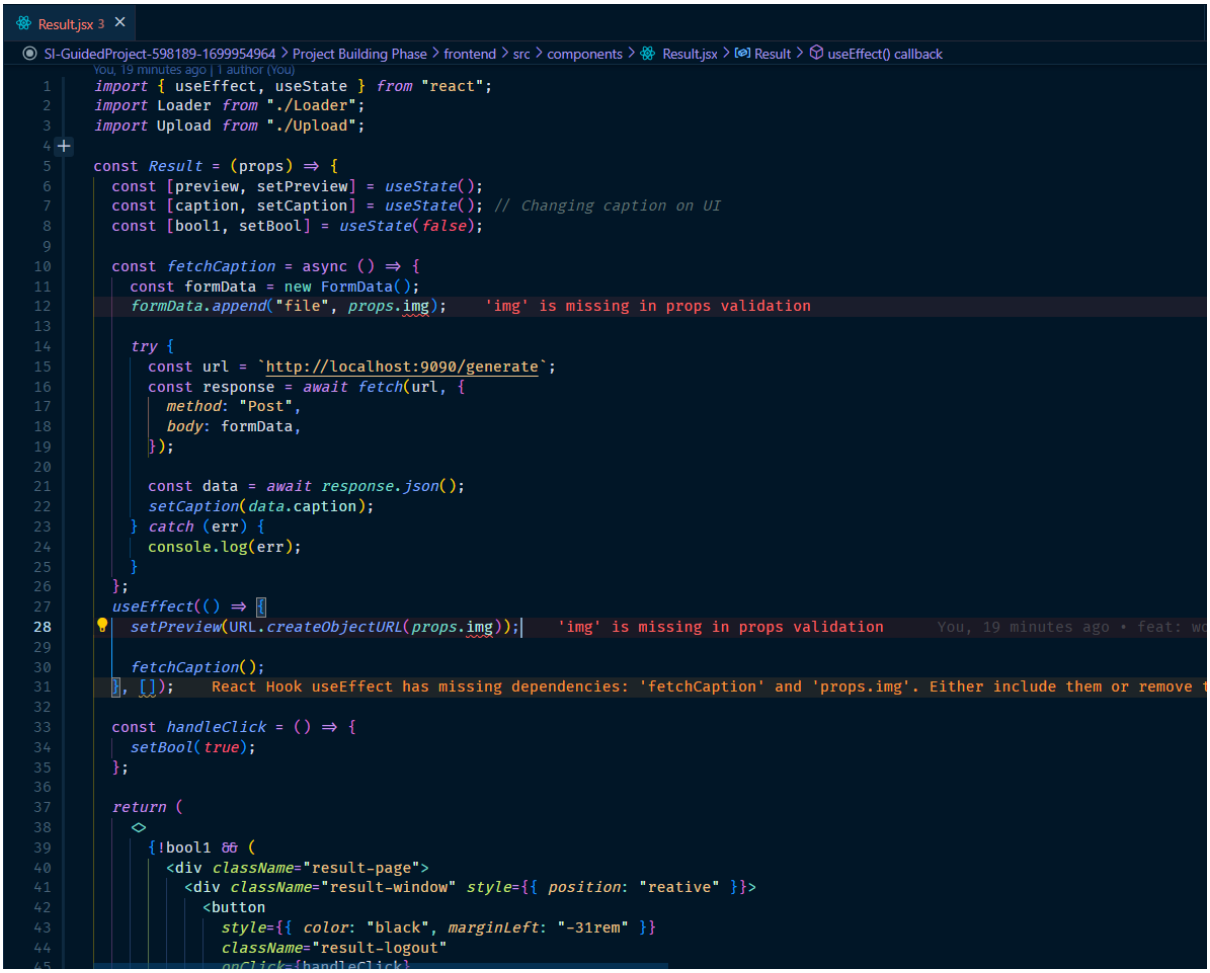
```
[ ]:
# Custom function for extracting an image and transforming it into an appropriate format
def getImage(idx):
    test_img_path = images[idx]
    test_img = cv2.imread(test_img_path)
    test_img = cv2.cvtColor(test_img,cv2.COLOR_BGR2RGB)
    test_img = cv2.resize(test_img,(224,224))
    test_img = np.reshape(test_img,(1,224,224,3))
    return test_img
```

```
for i in range(10):
    random_no = np.random.randint(0,1501,(1,1))[0,0]
    test_feature = model.predict(getImage(random_no)).reshape(1,2048)
    test_img_path = images[random_no]
    test_img = cv2.imread(test_img_path)
    test_img = cv2.cvtColor(test_img,cv2.COLOR_BGR2RGB)
    pred_text = ['startofseq']
    count = 0
    caption = '' # Stores the predicted captions text

    while count < 25:
        count += 1
        # Encoding the captions text with numbers
        encoded = []

        for i in pred_text:
            encoded.append(count_words[i])

        encoded = [encoded]
        # Padding the encoded text sequences to maximum length
        encoded = pad_sequences(encoded,maxlen=MAX_LEN,padding='post',truncating='post')
        pred_idx = np.argmax(final_model.predict([test_feature,encoded])) # Fetching the predicted word index having the maximum probability of occurr
        sampled_word = inverse_dict[pred_idx] # Extracting the predicted word by its respective index
        # Checking for ending of the sequence
        if sampled_word == 'endofseq':
            break
        caption = caption + ' ' + sampled_word
        pred_text.append(sampled_word)

    plt.figure(figsize=(5,5))
```

Milestone 3: Application Building

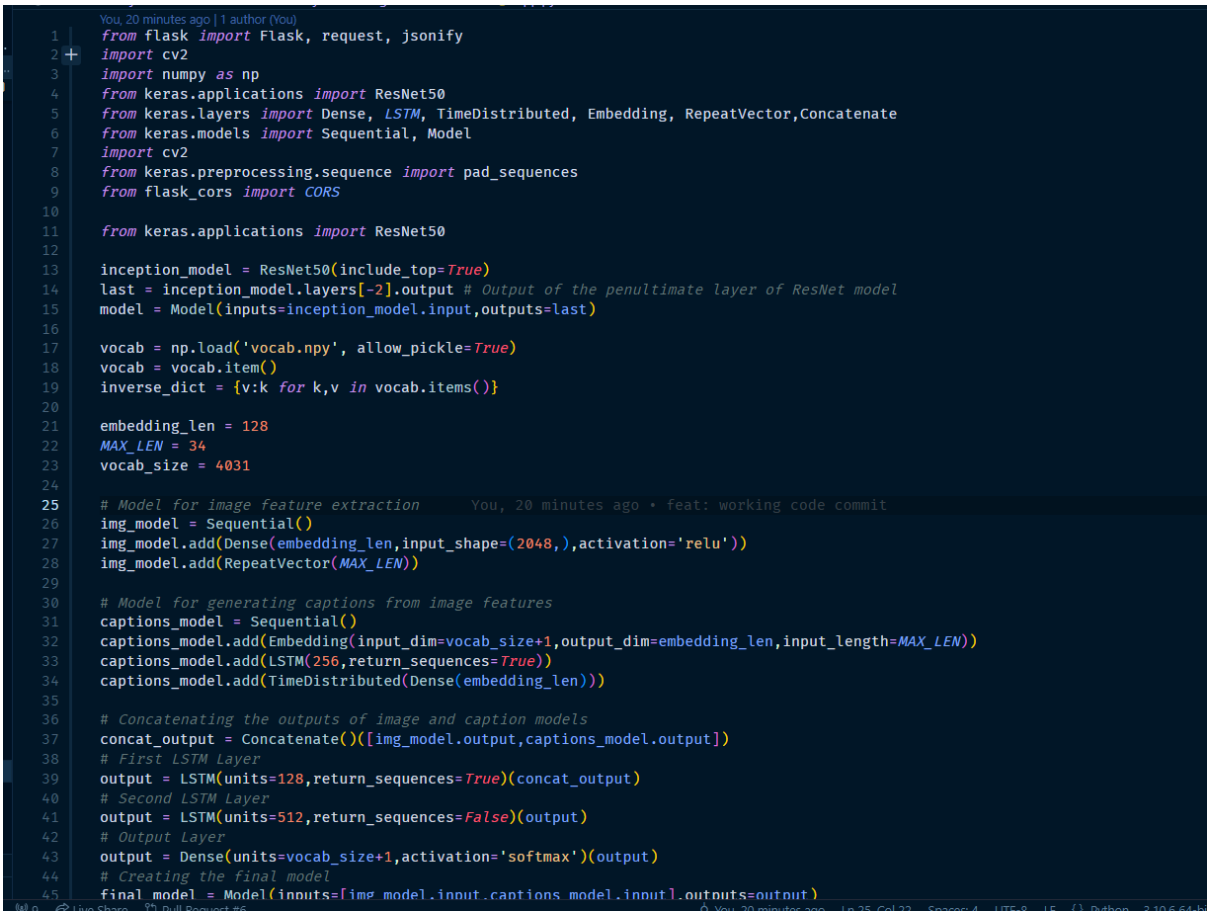
- Create HTML pages using React Framework

```
You, 1 second ago | 1 author (You)
import { useState } from "react";
import "../index.css";
import Result from "./Result";

const ImageCaptionGenerator = () => {
  const [selectedFile, setSelectedFile] = useState("");
  const [bool, setBool] = useState(false);

  const handleImageChange = (event) => {
    const img = event.target.files[0];
    setSelectedFile(img);
  };

  const handleGenerateCaption = () => {      You, now • Uncommitted changes
    if (selectedFile) setBool(true);
    else {
      window.alert("Select image first");
    }
  };

  return (
    <>
      <div>
        {!bool && (
          <>
            <input
              type="file"
              style={{ color: "black" }}
              onChange={handleImageChange}
            />
            <div>
              <button onClick={handleGenerateCaption}>Generate Caption</button>
            </div>
          </>
        )}

        {bool && <Result img={selectedFile} />}
      </div>
    </>
  );
};

export default ImageCaptionGenerator;
```

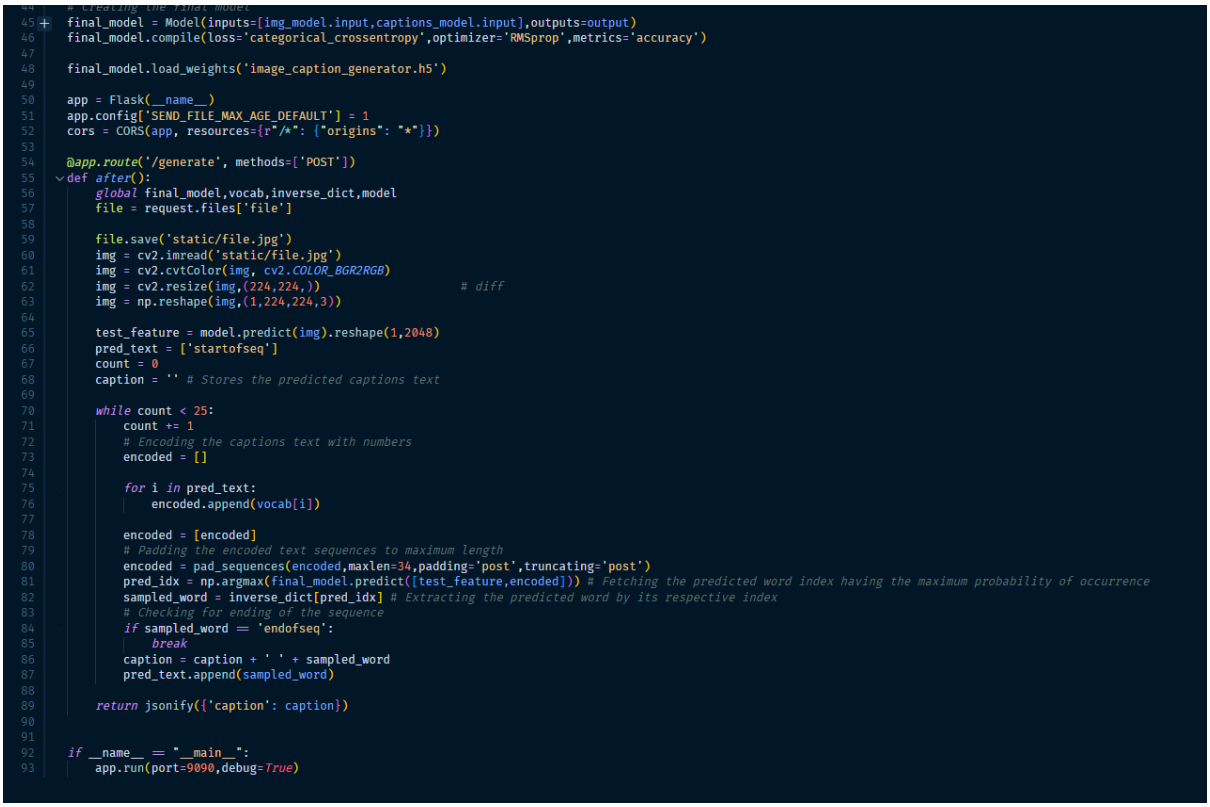

Result.jsx 3
SI-GuidedProject-598189-1699954964 > Project Building Phase > frontend > src > components > Result.jsx > Result > useEffect() callback
You, 19 minutes ago | 1 author (You)
import { useEffect, useState } from "react";
import Loader from "./Loader";
import Upload from "./Upload";
const Result = (props) => {
const [preview, setPreview] = useState();
const [caption, setCaption] = useState(); // Changing caption on UI
const [bool1, setBool] = useState(false);
const fetchCaption = async () => {
const formData = new FormData();
formData.append("file", props.img);
'img' is missing in props validation
try {
const url = `http://localhost:9090/generate`;
const response = await fetch(url, {
method: "Post",
body: formData,
});
const data = await response.json();
setCaption(data.caption);
} catch (err) {
console.log(err);
}
};
useEffect(() => {
setPreview(URL.createObjectURL(props.img));
'img' is missing in props validation
You, 19 minutes ago • feat: wo
fetchCaption();
}, []);
React Hook useEffect has missing dependencies: 'fetchCaption' and 'props.img'. Either include them or remove t
const handleClick = () => {
setBool(true);
};
return (
<>
{!bool1 && (

<div className="result-window" style={{ position: "reative" }}>
<button
style={{ color: "black", marginLeft: "-31rem" }}
className="result-logout"

- Writing Flask Script



```
You, 20 minutes ago | 1 author (You)
from flask import Flask, request, jsonify
import cv2
import numpy as np
from keras.applications import ResNet50
from keras.layers import Dense, LSTM, TimeDistributed, Embedding, RepeatVector,Concatenate
from keras.models import Sequential, Model
import cv2
from keras.preprocessing.sequence import pad_sequences
from flask_cors import CORS

from keras.applications import ResNet50

inception_model = ResNet50(include_top=True)
last = inception_model.layers[-2].output # Output of the penultimate layer of ResNet model
model = Model(inputs=inception_model.input,outputs=last)

vocab = np.load('vocab.npy', allow_pickle=True)
vocab = vocab.item()
inverse_dict = {v:k for k,v in vocab.items()}

embedding_len = 128
MAX_LEN = 34
vocab_size = 4031

# Model for image feature extraction      You, 20 minutes ago • feat: working code commit
img_model = Sequential()
img_model.add(Dense(embedding_len,input_shape=(2048,),activation='relu'))
img_model.add(RepeatVector(MAX_LEN))

# Model for generating captions from image features
captions_model = Sequential()
captions_model.add(Embedding(input_dim=vocab_size+1,output_dim=embedding_len,input_length=MAX_LEN))
captions_model.add(LSTM(256,return_sequences=True))
captions_model.add(TimeDistributed(Dense(embedding_len)))

# Concatenating the outputs of image and caption models
concat_output = Concatenate()([img_model.output,captions_model.output])
# First LSTM Layer
output = LSTM(units=128,return_sequences=True)(concat_output)
# Second LSTM Layer
output = LSTM(units=512,return_sequences=False)(output)
# Output Layer
output = Dense(units=vocab_size+1,activation='softmax')(output)
# Creating the final model
final_model = Model(inputs=[img_model.input,captions_model.input],outputs=output)
```



```
# Creating the final model
final_model = Model(inputs=[img_model.input,captions_model.input],outputs=output)
final_model.compile(loss='categorical_crossentropy',optimizer='RMSprop',metrics='accuracy')

final_model.load_weights('image_caption_generator.h5')

app = Flask(__name__)
app.config['SEND_FILE_MAX_AGE_DEFAULT'] = 1
cors = CORS(app, resources={r"/*": {"origins": "*"}})

@app.route('/generate', methods=['POST'])
def after():
    global final_model,vocab,inverse_dict,model
    file = request.files['file']

    file.save('static/file.jpg')
    img = cv2.imread('static/file.jpg')
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img,(224,224,))                    # diff
    img = np.reshape(img,(1,224,224,3))

    test_feature = model.predict(img).reshape(1,2048)
    pred_text = ['startofseq']
    count = 0
    caption = '' # Stores the predicted captions text

    while count < 25:
        count += 1
        # Encoding the captions text with numbers
        encoded = []

        for i in pred_text:
            encoded.append(vocab[i])

        encoded = [encoded]
        # Padding the encoded text sequences to maximum length
        encoded = pad_sequences(encoded,maxlen=34,padding='post',truncating='post')
        pred_idx = np.argmax(final_model.predict([test_feature,encoded])) # Fetching the predicted word index having the maximum probability of occurrence
        sampled_word = inverse_dict[pred_idx] # Extracting the predicted word by its respective index
        # Checking for ending of the sequence
        if sampled_word == 'endofseq':
            break
        caption = caption + ' ' + sampled_word
        pred_text.append(sampled_word)

    return jsonify({'caption': caption})


if __name__ == "__main__":
    app.run(port=9090,debug=True)
```

Milestone 4: Final Implementation

- Build and run the Flask application locally.

Browse... Screenshot 2023-11-20 202850.png

Generate Caption

Go back

Result page

a boy is surfing

PERFORMANCE TESTING

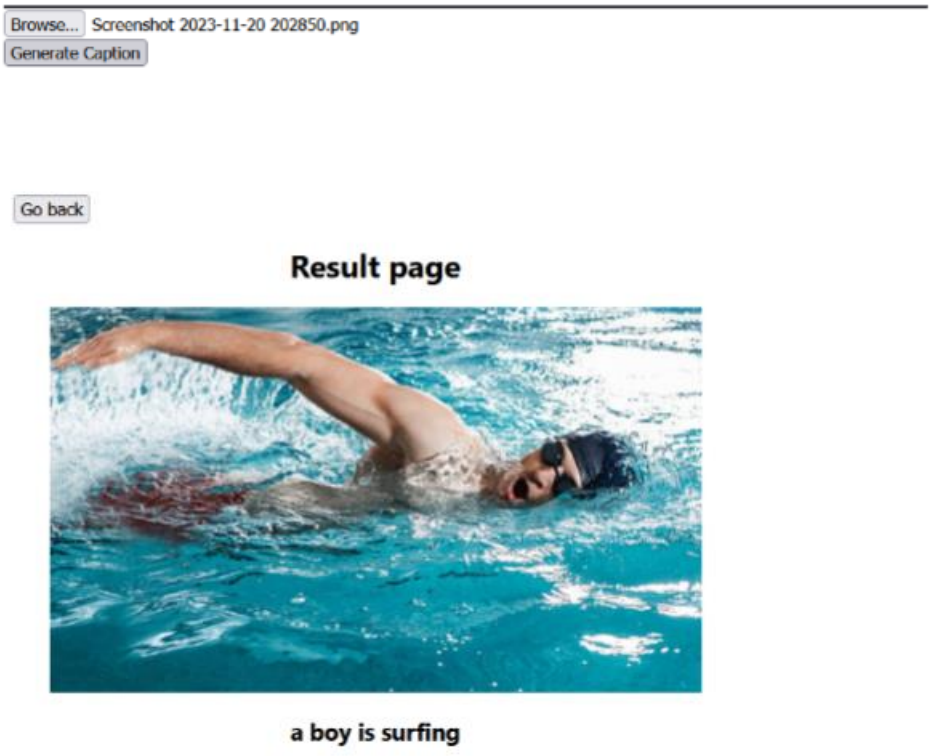
8.1 Performance Metrics

S.No.	Parameter	Values	Screenshot
1.	Metrics	**Regression Model:** MAE - , MSE - , RMSE - , R2 score - **Classification Model:** Confusion Matrix - , Accuray Score- & Classification Report -	Blue Score - 0.14 Accuracy - 89.6
2.	Tune the Model	Hyperparameter Tuning - Validation Method -	Epochs - 200 Beam Search

RESULTS

9.1 Output Screenshots

- Build and run the Flask application locally.



Go back

Result page

A dog playing in the grass.

Go back

Result page

A child in a red jacket playing street hockey guarding a goal

ADVANTAGES AND DISADVANTAGES

Advantages:

1. Automated Captioning:

- The system provides a fully automated solution for generating descriptive captions for images, reducing the need for manual annotation**.**

2. Learning Image Representations:

- The combination of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) enables the model to learn intricate image representations and contextual relationships within captions.

3. Versatility:

- The model is versatile and can be trained on diverse datasets, making it applicable to various domains and types of images.

4. Reduced Workload:

- Content creators and annotators benefit from reduced workload, as the system autonomously generates captions, saving time and effort.

5. Improved Accessibility:

- The system enhances accessibility for visually impaired individuals by providing descriptive captions for images shared on digital platforms.

Disadvantages

1. Data Dependency:

- The model's performance heavily relies on the quality and diversity of the training dataset. Inadequate or biased data may lead to inaccurate or biased captions.

2. Overfitting:

- There is a risk of overfitting to the training dataset, resulting in captions that may not generalize well to unseen images.

3. Interpretability Challenges:

- Understanding how the model generates captions can be challenging, leading to potential difficulties in diagnosing and correcting errors or biases.

4. Resource Intensive:

- Training deep learning models, especially those combining CNNs and RNNs, can be computationally expensive and require substantial computing resources.

5. Limited Context Understanding:

- The model's understanding of context is limited to the training data, and it may struggle with nuanced or abstract concepts not well-represented in the dataset.

6. Inability to Verify Accuracy:

- Unlike human annotators, the model may generate captions that are contextually incorrect or lack factual accuracy, and verifying the correctness of every generated caption can be challenging.

CONCLUSION

In conclusion, the Image Caption Generation project represents a significant stride in leveraging deep learning to enhance the interpretability of visual content. The implementation successfully combines Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) to automate the generation of descriptive captions for images. The use of the Flickr8k dataset and the ResNet50 pre-trained model ensures a robust foundation for feature extraction, and the Long Short-Term Memory (LSTM) network excels in capturing sequential dependencies within captions.

One of the project's notable achievements is its ability to reduce the manual workload involved in annotating images, offering a valuable tool for content creators and those working with large visual datasets. The versatility of the model allows for adaptation to various domains, contributing to its broader applicability in real-world scenarios.

However, challenges persist, particularly in addressing biases and ensuring accuracy in caption generation. The model's dependence on training data quality raises concerns about potential biases and the need for diverse, representative datasets. Further research and refinement are crucial to enhancing the model's interpretability and reducing the risk of overfitting to specific datasets.

As technology evolves, and with continued advancements in deep learning, the Image Caption Generation project lays the groundwork for future innovations in computer vision and natural language processing. The project's success in automating a complex task underscores the potential of artificial intelligence to transform the way we interact with and understand visual information.

FUTURE SCOPE

The Image Caption Generation project opens avenues for several exciting future developments and enhancements:

1**. Multimodal Capabilities:** Extending the model to incorporate additional modalities, such as audio and video, could result in a more comprehensive understanding of multimedia content. This expansion would require adapting the architecture to process diverse data types and enriching the model's contextual awareness.

2. **Fine-tuning for Specialized Domains:** Tailoring the model for specific domains, such as medical imaging or industrial applications, presents an opportunity to address domain-specific challenges. Fine-tuning on specialized datasets can enhance the model's accuracy and relevance in these contexts.

3. **Addressing Bias and Ethical Considerations:** Future work should focus on mitigating biases present in both training data and generated captions. Implementing ethical guidelines and refining the model to avoid perpetuating stereotypes is essential for responsible AI deployment.

4. **Real-Time Captioning**: Exploring techniques for real-time caption generation could have significant applications in live events, video streaming, and augmented reality. This requires optimizing the model for low-latency processing without compromising on caption quality.

5. **User-Driven Customization**: Allowing users to customize and influence the tone, style, or content of generated captions would enhance the model's utility across diverse user preferences and applications.

6. **Integration with Content Creation Platforms:** Integrating the model into popular content creation tools or social media platforms could empower users to effortlessly generate engaging captions for their visual content.

Continued research and collaboration across disciplines, including computer vision, natural language processing, and ethics in AI, will play a pivotal role in unlocking the full potential of image caption generation and ensuring its responsible and equitable deployment in various domains.

APPENDIX

GIT Repository link: Github