AI6002

Capstone for Artificial Intelligence

Proposal Team-2

Visual Question Answering Web-Application for Scene Understanding

Faculty of Engineering and Applied Science

MEMORIAL UNIVERSITY

Team-2 Members:

Thevananthan Thevarasa	- MUN# 202094858
Aayush Rijal	- MUN# 202292045
MBA Roldan	- MUN# 202285725

Table of Contents

Table of Figures

1. Problem Statement

In the Last few years, deep neural networks have had a significant influence on the disciplines of computer vision and natural language processing. We can now create models that accurately identify items in the photos. But our knowledge of pictures is still far below that of humans. When we look at images as humans, we don't only observe the things; we also comprehend how they interact and may infer their state and characteristics. VQA is particularly intriguing since it enables us to comprehend what our models actually perceive.

Our Capstone focuses on Developing **Visual Question Answering (VQA) Chat application** based on scene understanding, wherein the system must harness the synergy of computer vision, natural language processing, and commonsense knowledge to accurately respond to user-generated questions related to visual content.

Our proposed VQA system provides solution for Scene understanding for images that are captured around our outdoor or Indoor surroundings.

1.1. Key Components and Objectives:

Three Major Challenges:

1. **Visual Understanding:** The system should have the capability to analyze and comprehend the content of images and videos, extracting relevant information about objects, people, actions, and spatial relationships.
2. **Natural Language Processing:** The application must be able to process and understand natural language questions posed by users. This includes handling a variety of question types, such as "What," "Where," "Why," and "How."
3. **Commonsense Reasoning:** The system should integrate a repository of commonsense knowledge to enable it to answer questions that may not have explicit information in the visual content. For instance, it should be able to infer answers to questions like “Why are the men jumping?”, “What is the kid doing?”

Other System Design Challenges:

1. **Multimodal Integration:** The application must seamlessly fuse the understanding of visual content and the interpretation of natural language questions to provide coherent and contextually relevant answers.

2. **User Interaction:** The user interface should be user-friendly, allowing users to upload images or input URLs, pose questions in natural language, and receive timely and accurate responses.
3. **Accuracy and Efficiency:** The system should strive to provide accurate answers in real-time or with minimal delay. It should handle a wide range of questions and images, ensuring high precision and recall.
4. **User Feedback and Learning:** Incorporate mechanisms for users to provide feedback on the system's responses, enabling iterative improvement and learning from user interactions.

The successful development of this Visual Question Answering web application, combining vision, language, and commonsense knowledge, will not only enhance our ability to understand and interact with visual content but also have practical applications in fields like education, accessibility, content retrieval, and more.

1.2. Potential Applications

Summary:

A single universal system with one VQA model wouldn't be able to handle all the potential applications and scenarios listed in this section.

Our proposed VQA system provides a solution for Scene understanding for images that are captured around our outdoor or Indoor surroundings.

But the followings Listed potential applications would require model changes, different training Dataset, different fine tuning and validation methodologies based on the application Criteria, Field of Application and Level of scene understanding and Language understanding demanded.

A. **Educational Assistance:**

This VQA system can be used to create interactive educational content. Teachers and students can upload images or videos related to a particular topic or concept, and students can ask questions about the content. The system will provide informative answers, helping students better understand the material. It can also be used in e-learning platforms for self-paced learning, making educational resources more engaging and effective.

B. **Accessibility Support for the Visually Impaired:**

Visually impaired individuals can use this system to gain a deeper understanding of their surroundings. They can take a picture with a smartphone, ask questions about it (e.g., "What's in the picture to my left?"), and receive spoken answers, enabling them to navigate their environment and interact with visual content in a meaningful way. This promotes greater independence and inclusion for the visually impaired community.

C. **Content-Based Image and Video Search:**

Instead of relying on metadata or manual annotations, a VQA system can be used to search for images or videos based on their content. Users can describe what they are looking for in natural language (e.g., "Find images of a red sunset over a beach with palm trees") and get results that closely match their descriptions. This simplifies content retrieval in image and video databases.

D. **Virtual Assistants and Chatbots:**

Integrating VQA into virtual assistants or chatbots can enhance their ability to assist users with visual and context-dependent inquiries. For instance, a virtual assistant could help users plan a trip by analyzing and answering questions about travel photos, weather, landmarks, and more, all through a conversational interface.

E. **Retail and E-commerce:**

E-commerce websites can employ VQA to improve the shopping experience. Users can upload a picture of an item they're interested in, and the system can answer questions like "Where can I buy this?" or "Are there any discounts available?" This simplifies product discovery and enhances the customer's shopping journey.

F. **Healthcare Diagnosis and Training:**

In the healthcare sector, a VQA system can aid in diagnosing medical conditions from medical imagery (e.g., X-rays, MRI scans) by answering questions like "What abnormalities are present in this X-ray?" Additionally, it can be used in training medical professionals by providing insights into medical images and procedures, improving medical education.

G. **Content Moderation and Reporting:**

On social media and content-sharing platforms, a VQA system can assist in content moderation. It can identify and report inappropriate or harmful content by analyzing images and answering questions like "Is this content safe for all audiences?" This helps maintain a safer online environment.

H. **Tourism and Travel Planning:**

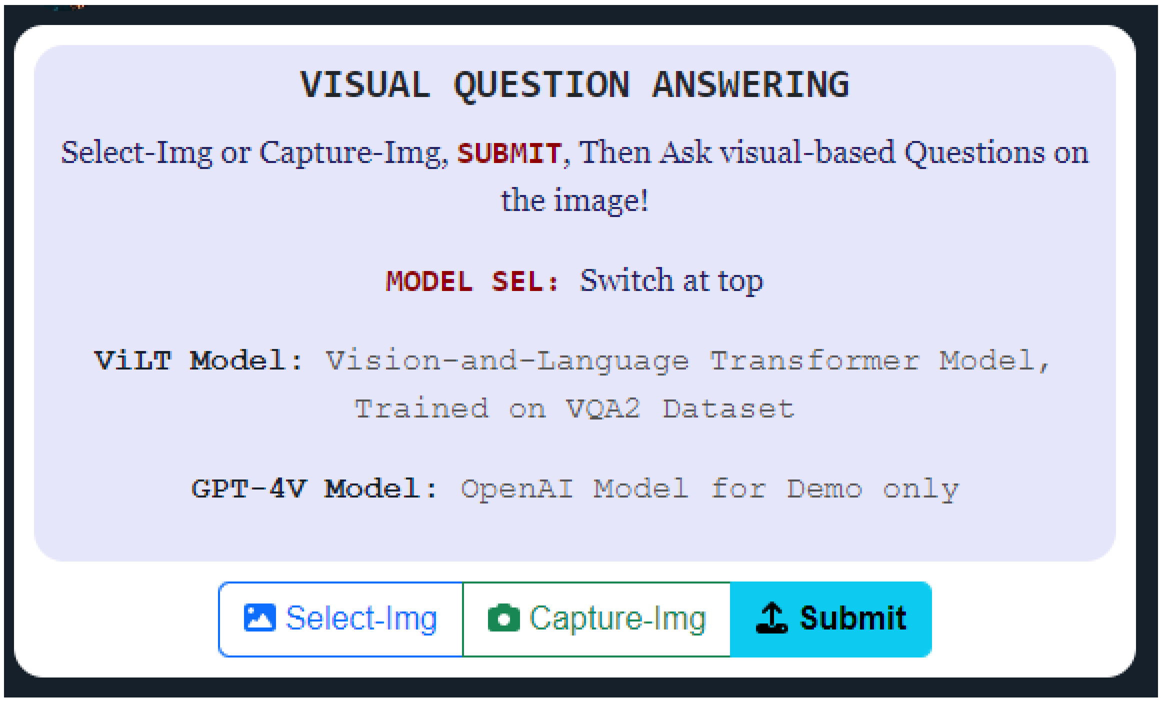
Travelers can use the VQA system to plan their trips. They can upload images of destinations or landmarks and ask questions like "What are the must-visit places in this city?" The system can provide recommendations based on visual input and contextual questions.

2. Cloud Deployed VQA Web-App

Cloud URL: http://ec2-18-219-100-175.us-east-2.compute.amazonaws.com

Shrinked URL: https://tinyurl.com/vqat2

- use **Mobile** or **laptop**
- Create a New Account using **sign-up**
- **Login** with username(not email!) and password.
- **Select an Image** from Gallery or PC or Camera
- **Submit** the Image, Wait for Response..
- **Ask a Question on image**, Wait for Response!
- Provide **Feedback**!



VISUAL QUESTION ANSWERING

Select-Img or Capture-Img, **SUBMIT**, Then Ask visual-based Questions on the image!

MODEL SEL: Switch at top

ViLT Model: Vision-and-Language Transformer Model, Trained on VQA2 Dataset

GPT-4V Model: OpenAI Model for Demo only

Figure 1 - VQA Chat app guide notes

3. Brief-Summary of Relevant Works Referred.

We referred to some of the top papers from the journal i.e., Conference on Computer Vision and Pattern Recognition (CVPR) and find out state of art techniques in the field of VQA. The relevant works and some takeaways from the papers are described below: -

3.1. Making the V in VQA Matter

The VQA model referenced demonstrates an integrative approach combining convolutional neural networks (CNNs) for visual feature extraction and Long Short-Term Memory (LSTM) networks for processing sequential text data. The model architecture involves a hierarchical processing pipeline where image features extracted via a CNN are normalized and then combined with text features from a 2-layer LSTM through point-wise multiplication. This joint representation is then condensed through successive fully-connected layers, culminating in a softmax layer that outputs a probability distribution over potential answers. [1]

Performance logs indicate that the model is trained over 30 epochs, showing a consistent decrease in training loss and a gradual increase in accuracy for both experiments (Exp1 and Exp2), evidencing the model's learning and improvement. Moreover, the validation loss remains relatively stable across epochs, suggesting robust generalization capabilities. The related works section would emphasize the model's proficiency in learning representations and highlight its steady progress in training as reflected in the training logs, underscoring its relevance and contribution to the field of Visual Question Answering.

3.2. Vision-and-Language Transformer.

Recent advancements in the domain of Visual Question Answering (VQA) have been epitomized by the introduction of the Vision-and-Language Transformer (ViLT) model. This innovative model represents a paradigm shift in multimodal learning, eschewing the computationally intensive convolutional operations and region-specific annotations traditionally relied upon in VQA tasks. [2]

ViLT leverages a streamlined architecture that processes raw image patches and text tokens through shallow embedding layers, analogous in computational simplicity. The central component of ViLT is a transformer encoder which concurrently processes the sequences of text and image patch embeddings. This single-stream approach contrasts with dual-stream methodologies, enabling a more parameter-efficient model that fosters robust inter-modality interactions without additional complexity.

The model's technical specifications are noteworthy, featuring a configuration known as ViT-B/32, with a hidden size of 768, a depth of 12 layers, a patch size of 32, an MLP size of 3072, and 12 attention heads. Pre-trained on ImageNet, ViLT's design facilitates a substantial reduction in computational load while maintaining high performance in VQA benchmarks.

Training objectives for ViLT include Image Text Matching (ITM), where the model discerns the relevance of a given image to the accompanying text, and Masked Language Modeling (MLM), predicting masked words within a textual input by considering its contextual relationship with the visual data. A novel Word Patch Alignment (WPA) technique also enhances the model's capability to align textual tokens with corresponding image patches, utilizing optimal transport theory for improved correspondence measures.

ViLT's approach to visual and textual embedding and its modality interaction offers insightful perspectives for VQA tasks. Its design principles and training strategies present valuable considerations for future research in multimodal learning frameworks, underscoring the potential of transformer-based models to redefine benchmarks in VQA and related fields.

3.3. Grad-CAM.

This paper "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization" presents a novel approach called Grad-CAM (Gradient-weighted Class Activation Mapping) for generating visual explanations from deep neural networks, with a particular focus on its application in Visual Question Answering (VQA) systems. In VQA tasks, understanding why a model makes a certain prediction is crucial for transparency and interpretability. Grad-CAM addresses this need by providing a heatmap that highlights the regions in an input image that contribute most to the model's decision. This heatmap can be superimposed onto the image, making it easier for humans to interpret the model's reasoning. [3]

Grad-CAM employs gradient-based localization, using gradients from the final convolutional layer to identify crucial image regions for specific predictions, establishing a direct link between model outputs and influential visual areas. It further aggregates these gradients through global average pooling to align them spatially with the input image, preserving the interpretability of1 the explanation. Notably, Grad-CAM is versatile, as it can be applied to various neural network architectures without model-specific modifications, enhancing interpretability in Visual Question Answering (VQA) by offering human-understandable explanations by overlaying localization heatmaps on images, revealing which image regions influenced the model's responses to questions.

3.4. Bottom-Up and Top-Down Attention

This paper "Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering" addresses the field of Visual Question Answering (VQA) by introducing an attention mechanism that combines both bottom-up and top-down approaches to improve model performance in generating answers to questions based on images. In VQA tasks, understanding and reasoning about images are crucial for providing accurate answers to questions. The paper proposes a novel approach that leverages both bottom-up and top-down attention mechanisms. [4]

Bottom-Up Attention analyzes the image to identify important regions and objects, creating image features with spatial information to emphasize relevant image parts. In contrast, Top-Down Attention processes the question to identify crucial words and dynamically adjusts attention on image features based on the question's semantics, enabling the model to adapt its focus in response to the question's content, enhancing its reasoning ability.

The key contributions are threefold. Firstly, it enhances model understanding by combining bottom-up and top-down attention mechanisms, enabling the model to better grasp the relationship between image content and the posed question, resulting in improved answer accuracy. Secondly, the paper promotes effective reasoning by integrating visual information and question semantics, enhancing the model's ability to reason about images and provide more accurate answers across a wide range of questions. Lastly, the proposed approach is versatile, applicable to both image captioning and VQA tasks, highlighting its effectiveness across various domains within computer vision and natural language processing.

4. Dataset

In the development of our "Visual Question Answering Web-Application for Scene Understanding," the utilization of high-quality and meticulously curated datasets is paramount to achieving accurate, robust, and dependable results. One such indispensable resource that we have harnessed for training, validation, and testing is the VQA 2.0 Dataset. The VQA 2.0 Dataset is a rich collection of annotated visual content, questions, and answers that plays a fundamental role in enabling our system to comprehend and respond to user-generated queries about images and videos.

4.1. VQA 2.0 Dataset Overview:

The VQA 2.0 Dataset comprises various essential components, including VQA Annotations, VQA Input Questions, VQA Input Images, and Complementary Pairs List. This comprehensive dataset has undergone meticulous post-processing, ensuring the highest data quality and consistency.

Leveraging this dataset, our system will be trained and fine-tuned to provide users with precise, contextually relevant, and insightful answers to their queries, fostering a more intelligent and interactive user experience.

The multi-modal VQA system demands the Several type of Dataset to Train, Validate and Test the System. Each type of dataset is explained in the following sections.

4.2. VQA Annotations:

Spelling Correction: The dataset undergoes a thorough spelling correction process, employing Bing Speller to rectify any typographical errors present in the question-and-answer strings. This step is critical to maintain data accuracy and consistency.

Question Normalization: To ensure uniformity, questions are normalized, with the first character in uppercase and the last character always being a question mark.

Answer Normalization: Answers are normalized, with all characters converted to lowercase. Periods are retained only as decimal points, and number words are translated into digits. Articles such as 'a,' 'an,' and 'the' are stripped to enhance data cleanliness and consistency.

Apostrophe Addition: If a contraction is detected without an apostrophe, it is added to rectify contractions and ensure linguistic correctness (e.g., converting "dont" to "don't").

The annotations are stored using the JSON file format.
The annotations format has the following data structure:

```
{
"info" : info,
"data_type": str,
"data_subtype": str,
"annotations" : [annotation],
"license" : license
}
```

data_type: source of the images (mscoco or abstract_v002).
data_subtype: type of data subtype (e.g. train2014/val2014/test2015 for **mscoco**).
question_type: type of the question determined by the first few words of the question.
answer_type: type of the answer. Currently, "yes/no", "number", and "other".
multiple_choice_answer: most frequent ground-truth answer.
answer_confidence: subject's confidence in answering the question.

4.3. VQA Input Questions:

This component of the dataset includes a diverse array of natural language questions related to the visual content. Questions vary in complexity, context, and focus, providing a wide spectrum of queries that users may pose to the application.

Input Questions Format
The questions are stored using the JSON file format.

The questions format has the following data structure:

```
{
"info" : info,
"task_type" : str,
"data_type": str,
"data_subtype": str,
"questions" : [question],
"license" : license
}
```

task_type: type of annotations in the JSON file (OpenEnded).
data_type: source of the images (**mscoco** or **abstract_v002**).
data_subtype: type of data subtype (e.g. train2014/val2014/test2015 for mscoco).

4.4. VQA Input Images from COCO:

The COCO dataset is known for its high quality and large scale, making it a valuable resource for training and evaluating machine learning models, particularly in tasks related to object recognition and scene understanding.

The VQA Input Images from COCO are a rich repository of visual content, comprising images and potentially videos. These images encompass a range of scenes, objects, and activities, offering a robust visual context for users to query.

- Training images: 82,783 images
- Validation images: 40,504 images
- Testing images: 81,434 images

4.5. Training, Validation, Test partitioning of Dataset

The Dataset are Divided into Training, Validation and Testing. The purpose of each partition of dataset is explained below.

Training Dataset:

- **Model Learning:** The training dataset is used to train the machine learning model. It provides examples of input data and their corresponding known output, allowing the model to learn the underlying patterns and relationships in the data.
- **Parameter Tuning:** During training, the model's parameters or weights are adjusted to minimize the difference between its predictions and the actual target values in the training data.
- **Feature Extraction:** The model learns to extract relevant features from the input data, helping it make accurate predictions.

Validation Dataset:

- **Hyperparameter Tuning:** The validation dataset is used to fine-tune hyperparameters, such as learning rates or regularization terms. It helps optimize the model's performance without overfitting to the training data.
- **Model Selection:** Multiple models or variations of the same model can be compared using the validation dataset to determine which one performs the best.

- **Early Stopping:** Validation data is used to monitor the model's performance during training. If performance on the validation data starts to degrade, early stopping can be employed to prevent overfitting.

Testing Dataset:

- **Performance Assessment:** The testing dataset serves as an independent dataset that the model has never seen during training or validation. It's used to evaluate the model's performance in a real-world, generalization scenario.
- **Generalization Assessment:** Testing helps ensure that the model hasn't overfit the training data and that it can make accurate predictions on new, unseen examples.
- **Benchmarking:** The testing dataset allows for benchmarking the model's performance, making it possible to compare different models or algorithms on a level playing field.

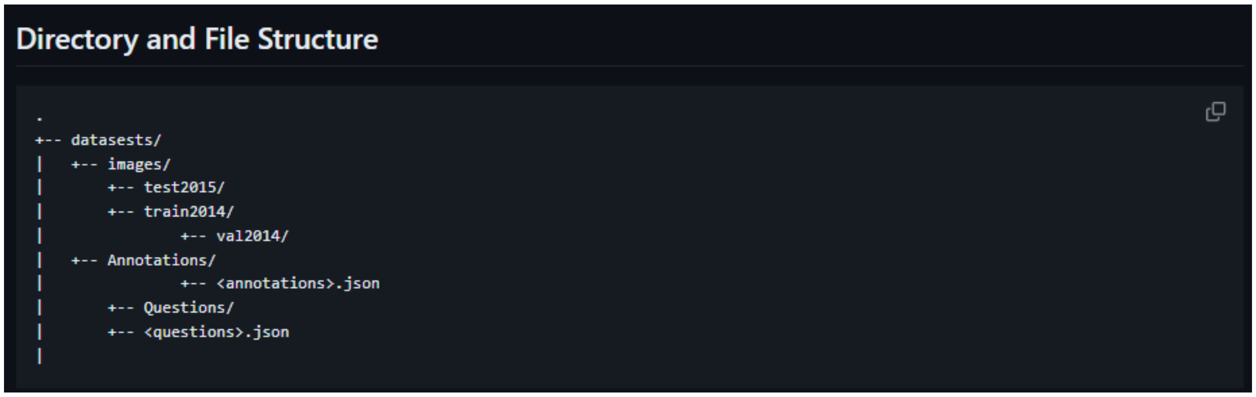
4.6. Python script to download and extract the Dataset.

Training and Validation of the model cannot be done in the local computer due to the Size of Dataset and the time taken to train on the local GPU.

Therefore, we used Colab-Pro computing space and its terminal to run training and validation of VQA Model1. uploading the huge dataset from local PC to colab is highly expensive in-terms of time taken and loosing computing units for the GPU runtime.

Instead having a Dataset downloader and Extractor Script would much cost-effective and time effective.

Script: https://github.com/AI6002/capstone-project-team2/tree/ac2579c3dda0f5fe44779832bbdcee0fe386a344/Datasets


Directory and File Structure

```
.
+-- datasests/
|   +-- images/
|       +-- test2015/
|       +-- train2014/
|               +-- val2014/
|   +-- Annotations/
|               +-- <annotations>.json
|       +-- Questions/
|       +-- <questions>.json
|
```

Figure 2 - VQA2.0 Dataset download structure

5. VQA Models

5.1. VQA is a Multimodal Task.

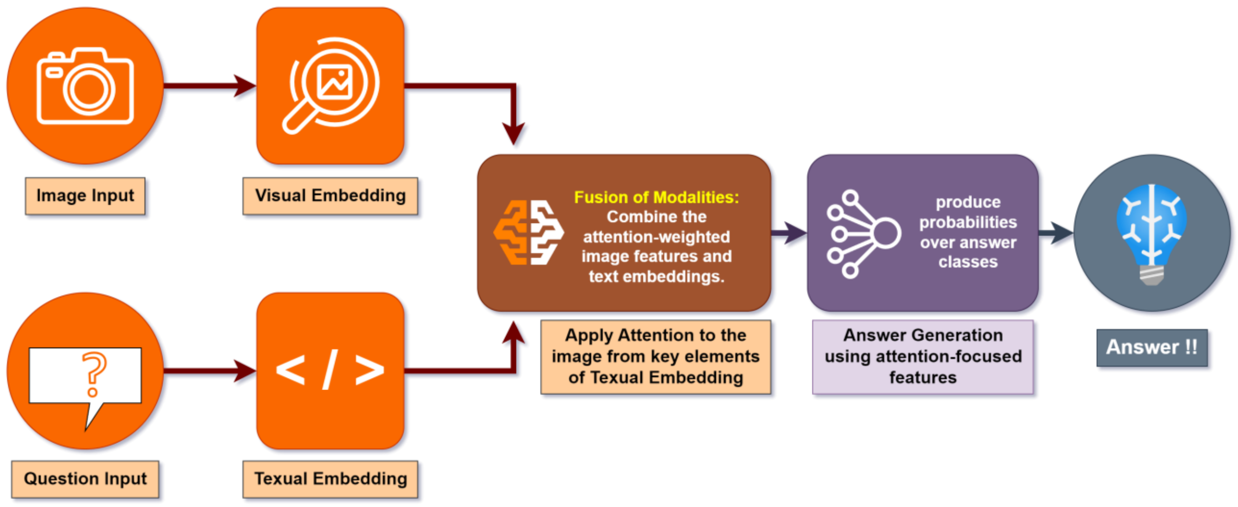


Image Input
Visual Embedding
Question Input
< / >
Texual Embedding
Fusion of Modalities: Combine the attention-weighted image features and text embeddings.
Apply Attention to the image from key elements of Texual Embedding
produce probabilities over answer classes
Answer Generation using attention-focused features
Answer !!

Figure 3 - VQA Multimodal approach diagram

<u>**Computer Vision**</u>

- **Image Preprocessing**
- **Feature Extraction:** Using CNNs like ResNet/VGGNet/EfficientNet/DenseNet
- **Object Detection and Recognition:** labels objects, detect attributes and relationships.
- **Scene Understanding**: interprets the overall scene, actions or context.
- **Semantic Feature Encoding:** into semantic vector space to Integrate with Lang Model

<u>**Natural Language Processing**</u>

- **Preprocessing:** tokenization, cleaning.
- **Text Embedding:** Convert the processed text into numerical vectors or embeddings.
- **Language Understanding:** of semantic and syntactic structure using either legacy RNN/GRU/LSTM/CNN or SOTA Transformer Models BERT, GPT Series, Google's T5 etc.
- **Contextual and Commonsense Reasoning:** not directly present in the image/ question.

Integration and Answer Generation

- **Apply Attention:** aligning relevant parts of the image with key elements of the text embeddings.
- **Fusion of Modalities:** Combine the attention-weighted image features and text embeddings.
- **Answer Generation:** using the integrated, attention-focused features

5.2. LSTM, VGG19 Based Stacked Attention Model

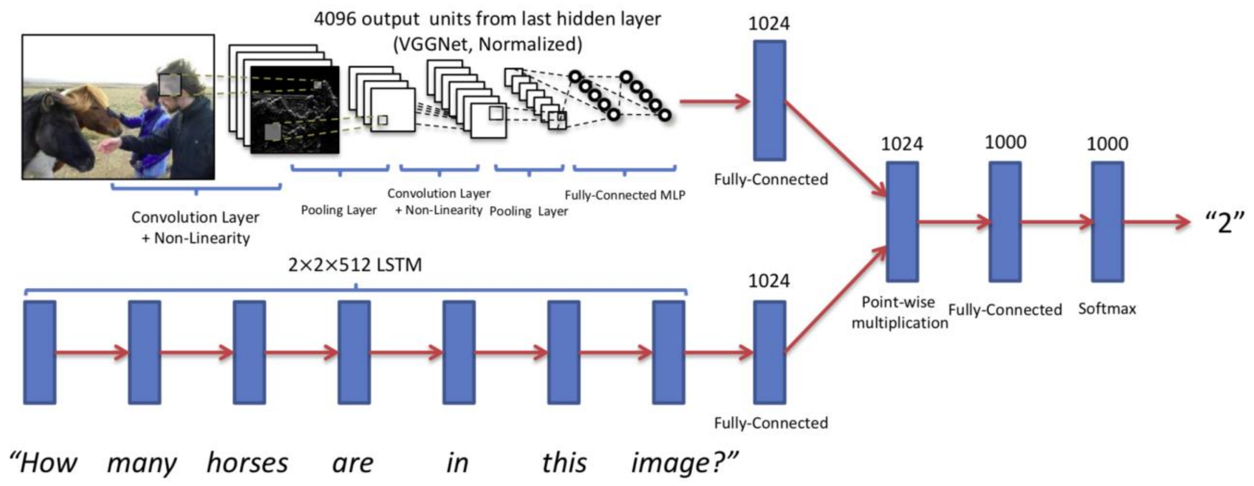

4096 output units from last hidden layer
(VGGNet, Normalized)
1024
Convolution Layer + Non-Linearity
Pooling Layer
Convolution Layer + Non-Linearity
Pooling Layer
Fully-Connected MLP
Fully-Connected
1024
1000
1000
"2"
2×2×512 LSTM
1024
Point-wise multiplication
Fully-Connected
Softmax
Fully-Connected
"How many horses are in this image?"

Figure 4 - Stacked Attention Model CVPR 2019

- Paper CVPR May 2017: "Making the V in VQA Matter: Elevating the Role of Image Understanding in Visual Question Answering"
- Image feature extraction & Obj Detection: **VGG19-CNN**
- NLP Model for Text Embedding: **Long Short-Term Memory (LSTM)**
- Attention and Answer Generation: **Hierarchical Co-attention (HieCoAtt).** Its co-attends to both the image and the question.

Above Diagram shows an Architecture of our first approach for the VQA Model. Where We treat visual question answering task as a classification problem. Given an image ***I*** and a question ***q*** in the form of natural language we want to estimate the most likely answer ***â*** from a fixed set of answers based on the content of the image.

Approach:

A. **Image Embedding**

We plan to use a pretrained convolutional neural network (**CNN**) model based on residual network architecture (**ResNet**) to compute a higher dimensional tensor representation of the input image.

B. Question embedding

The User question is tokenized and encoded into word embeddings. The embeddings are then fed to a long short-term memory (**LSTM**). The final state of the LSTM to represent the question.

C. Stacked Attention

The concatenated image features and the final state of LSTMs are then used to compute multiple attention distributions over image features.

D. Output Classifier

Finally, Concatenate the image glimpses along with the LSTM state and apply nonlinearities to produce probabilities over answer classes.

5.3. ViLT: Vision-and-Language Transformer

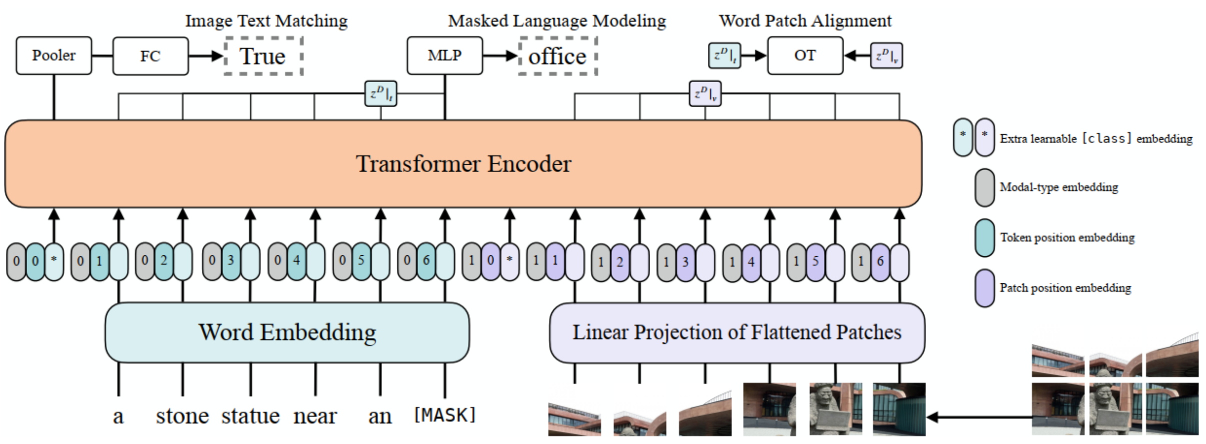


Figure 5 - Vision and Language Transformer ICML 2021

- ICML 2021 paper: "ViLT: Vision-and-Language Transformer Without Convolution or Region Supervision"
- Visual Embedding: **Patch Projection method introduced by ViT (Vision Transformer)**

- Textual Embedding: **Tokenizer from pre-trained BERT (Bidirectional Encoder Representations from Transformers)**
- Modality Interaction: **single-stream approach, self-attention in Transformer Encoder**

Technical Specifications

- The model uses weights from ViT-B/32 pre-trained on ImageNet.
- Hidden size (H): 768
- Layer depth (D): 12
- Patch size (P): 32
- MLP size: 3072
- Number of attention heads: 12.

Training Objectives and Techniques

Image Text Matching (ITM):

- This objective involves replacing the aligned image with a different image with a probability of 0.5.

Word Patch Alignment (WPA):

- Inspired by the word region alignment objective, WPA computes the alignment score between textual and visual subsets.

Masked Language Modeling (MLM):

- The model predicts the ground truth labels of masked text tokens from its contextualized vector.

Whole Word Masking:

- This technique involves masking all consecutive subword tokens that compose a whole word, hypothesized to be particularly crucial for VLP to utilize information from the other modality effectively.

Image Augmentation:

- ViLT employs RandAugment for image augmentation during fine-tuning.

5.4. Gtp-4v Model

- **GPT-4** with vision (**GPT-4V**) enables users to instruct GPT-4 to analyze image inputs provided by the user.
- Incorporating additional modalities (such as image inputs) into large language models (LLMs) is a key frontier in artificial intelligence research and development.
- The largest language models developed by OpenAI, featuring approximately **175 billion parameters**.
- Paper: https://cdn.openai.com/papers/GPTV_System_Card.pdf

6. Front End User Interface

6.1. USER Interface Operation sequence.

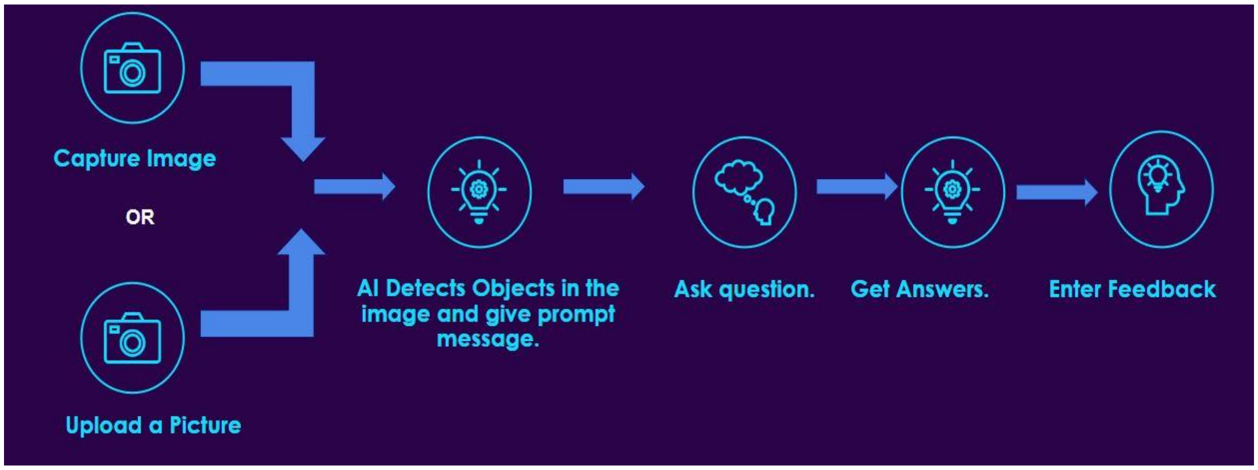


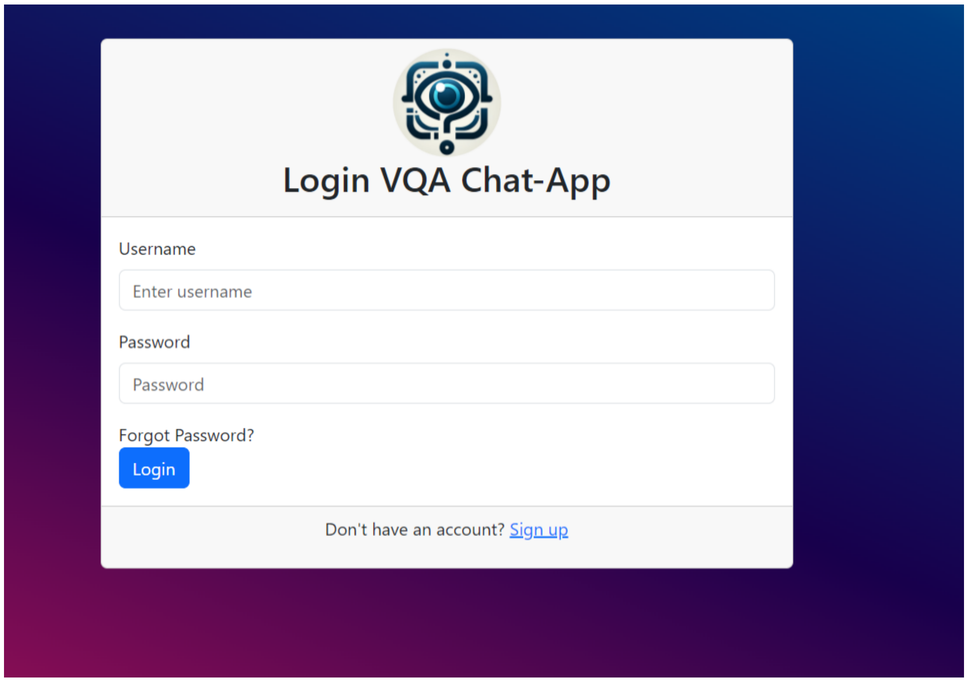
Figure 6 - User Interface Operation Sequence

6.2. login Page.

The Login VQA page is a crucial component of the Visual Question Answering (VQA) Chat-App. This page serves as the entry point for registered users to access the application's features. This Login VQA page aims to provide a user-friendly, secure, and intuitive login experience for users accessing the VQA Chat-App, emphasizing simplicity and functionality in the login process.

- Users access the Login VQA page by navigating to the application's login URL.
- The page structure is designed using HTML and Bootstrap classes.
- It consists of a responsive layout with a card-based user interface, organized into sections (header, form, footer).
- Users enter their login credentials (username and password) into the form fields provided.
- Input fields are mandatory (required attribute) for data submission. Upon completion of entering credentials, users click the 'Login' button to submit the form.
- The form data is sent to the '/login' endpoint using the POST method for server-side processing. Flask handles the form submission, validating and processing the entered credentials.
- If the credentials are valid, the user is redirected to the application's home page. In case of invalid credentials, error messages are flashed (displayed) using Flask's flash message system.

- If login errors occur (e.g., incorrect credentials), error messages are displayed in an alert at the top of the card body.
- Users without an account are prompted to sign up via a hyperlink that redirects them to the signup page using route.


Login VQA Chat-App

Username

Enter username

Password

Password

Forgot Password?

Login

Don't have an account? Sign up

Figure 7 - Frontend page look of login page

6.3. Register Page.

User Registration facilitates the process for new users to join the application by creating an account. Registering is a fundamental step for users to access the functionalities and features provided by the VQA application.

- HTML and Bootstrap classes are utilized to create a responsive layout.
- Users fill out the registration form fields, including username, email address, password, and password confirmation.
- All form fields are mandatory (required attribute) to ensure complete data submission.
- Users click the 'Sign Up' button to submit the registration form, which uses the POST method to send data to the '/signup' endpoint for processing.
- Flask handles the form submission, processing the provided user data for registration purposes.
- Upon successful registration, the user is redirected to the login page to access the application.
- If registration errors occur (e.g., invalid input, existing username or email), error messages are displayed in an alert at the top of the card body.
- Users who are already registered can navigate to the login page.

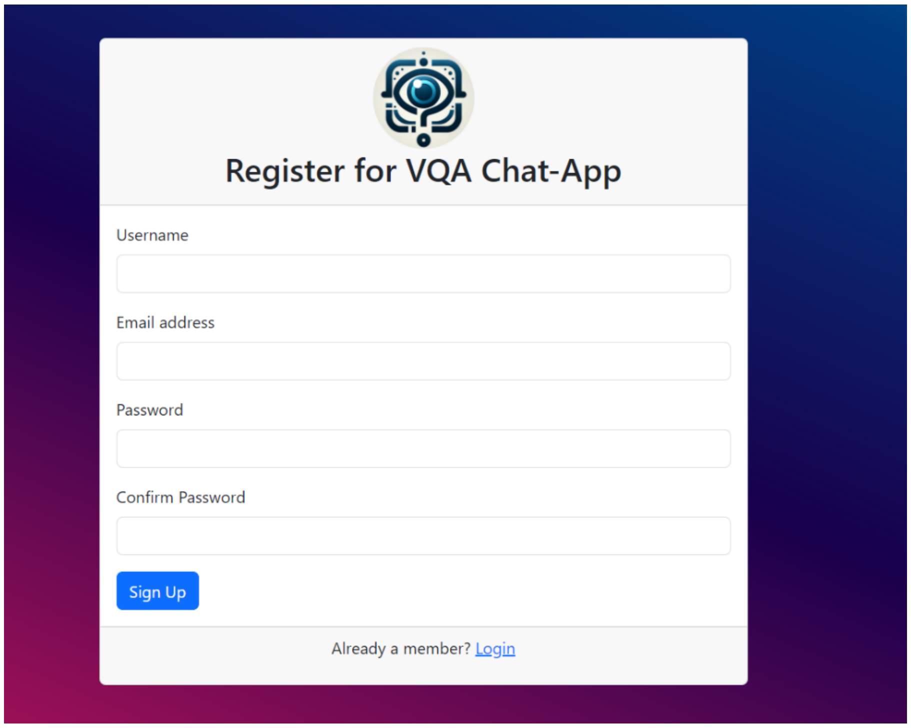


Figure 8 Frontend page look of User Registration Page.

6.4. Home Page.

This page essentially serves as the main interaction space for users to communicate with the VQA system. It enables users to input queries, upload images, switch between models, and view ongoing conversations, fostering a user-friendly environment for interaction and query processing within the VQA Chat-App. Some of the features included in the page are given below: -

a. Profile and Model Selection:
    - It displays user profile information like username.
    - It provides an option to switch between different models (GPT –04 and VLIT) via a toggle switch.
    - It includes icons for additional functionalities like notifications and a dropdown menu.

b. Image Upload Section:
    - It allows users to upload images either by selecting them or capturing them through the camera.
    - It provides guidance on using the application, highlighting the purpose of the VQA models available.

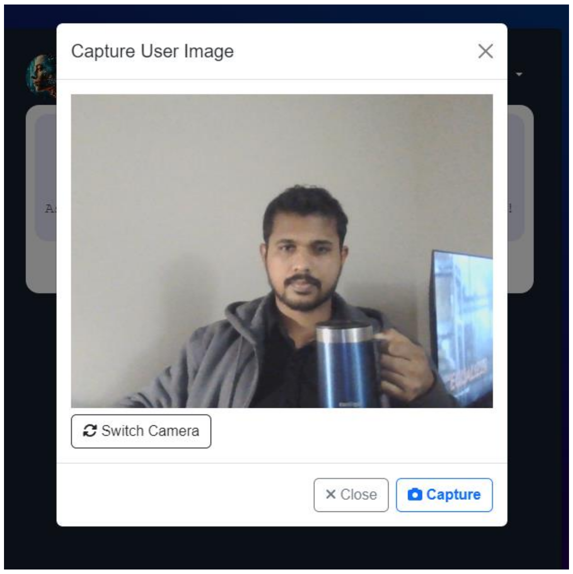


Figure 9: - Capturing image using the webcam.

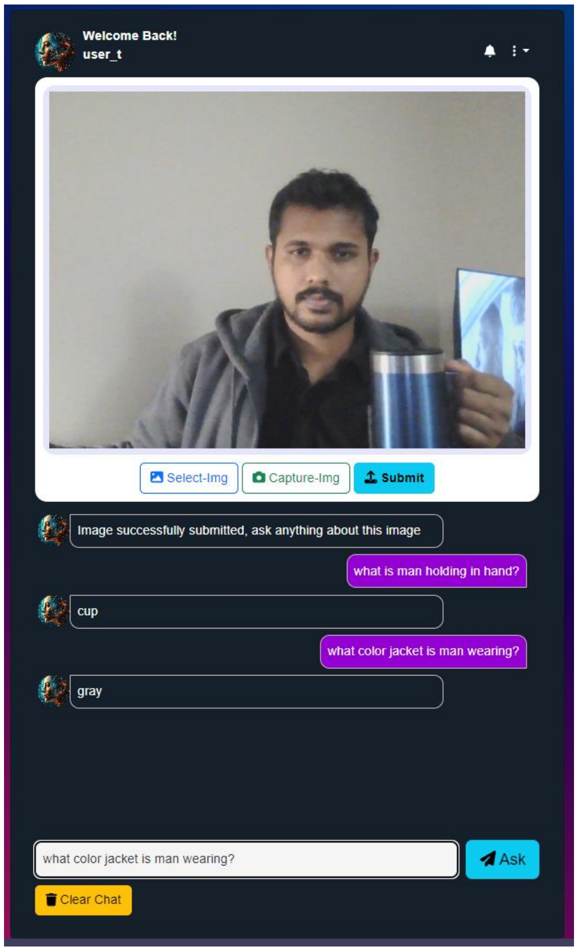


Image successfully submitted, ask anything about this image

what is man holding in hand?

cup

what color jacket is man wearing?

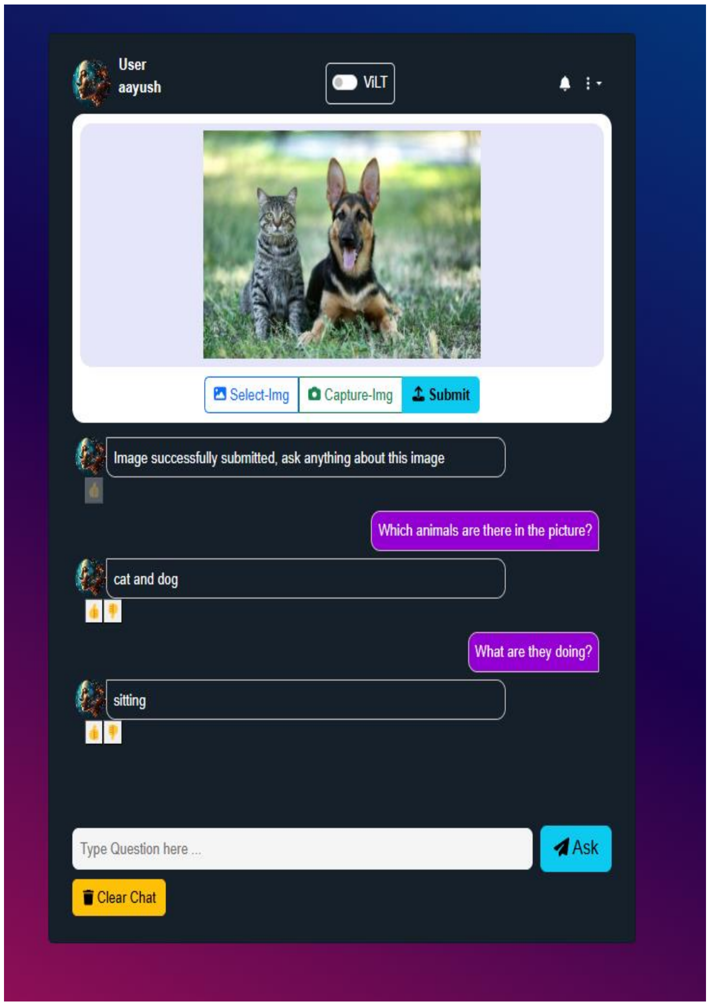
gray

what color jacket is man wearing?

Figure 10: - Demo After uploading the picture..

c. Message Display and Input:

- Shows the conversation between the user and the VQA application in the 'msg_section'.
- Offers a text input form for users to type their questions to the VQA application.
- Includes a 'Submit' button to send the entered question for processing.



User
aayush

Image successfully submitted, ask anything about this image

Which animals are there in the picture?

cat and dog

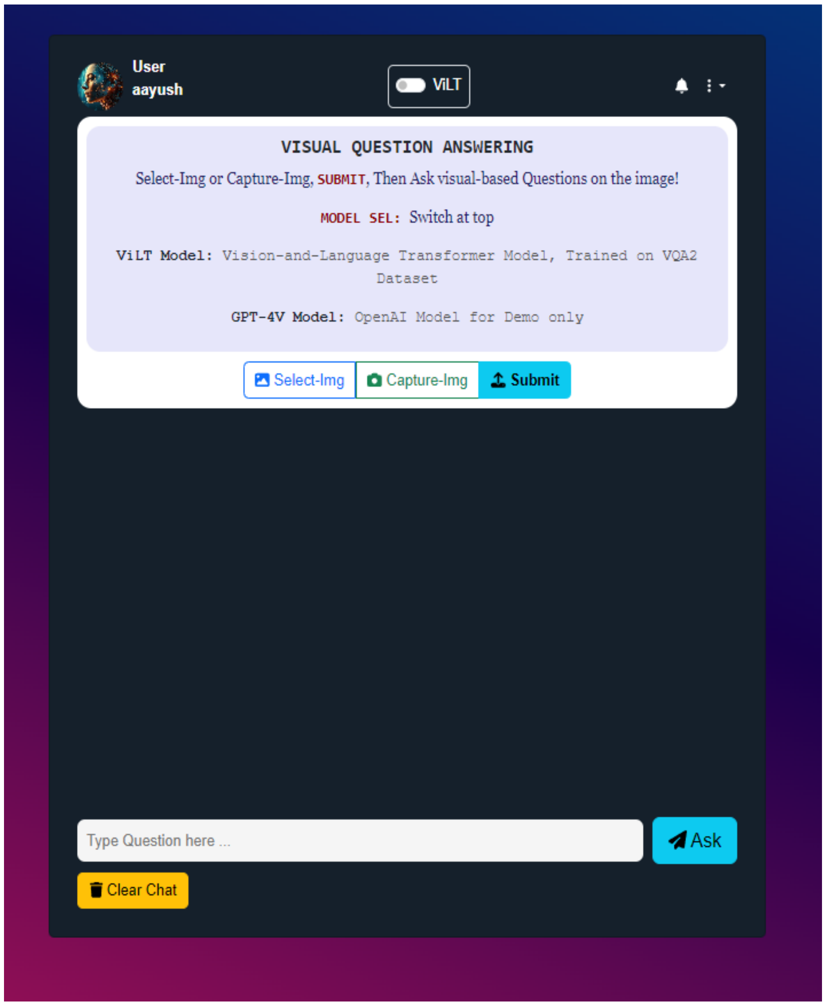
What are they doing?

sitting

Figure 11: - Working of VQA Model and message display.

d. Additional Functionalities:

- 'Clear Chat' button is present at the bottom to reset the conversation.



VISUAL QUESTION ANSWERING

Select-Img or Capture-Img, SUBMIT, Then Ask visual-based Questions on the image!

MODEL SEL: Switch at top

ViLT Model: Vision-and-Language Transformer Model, Trained on VQA2 Dataset

GPT-4V Model: OpenAI Model for Demo only

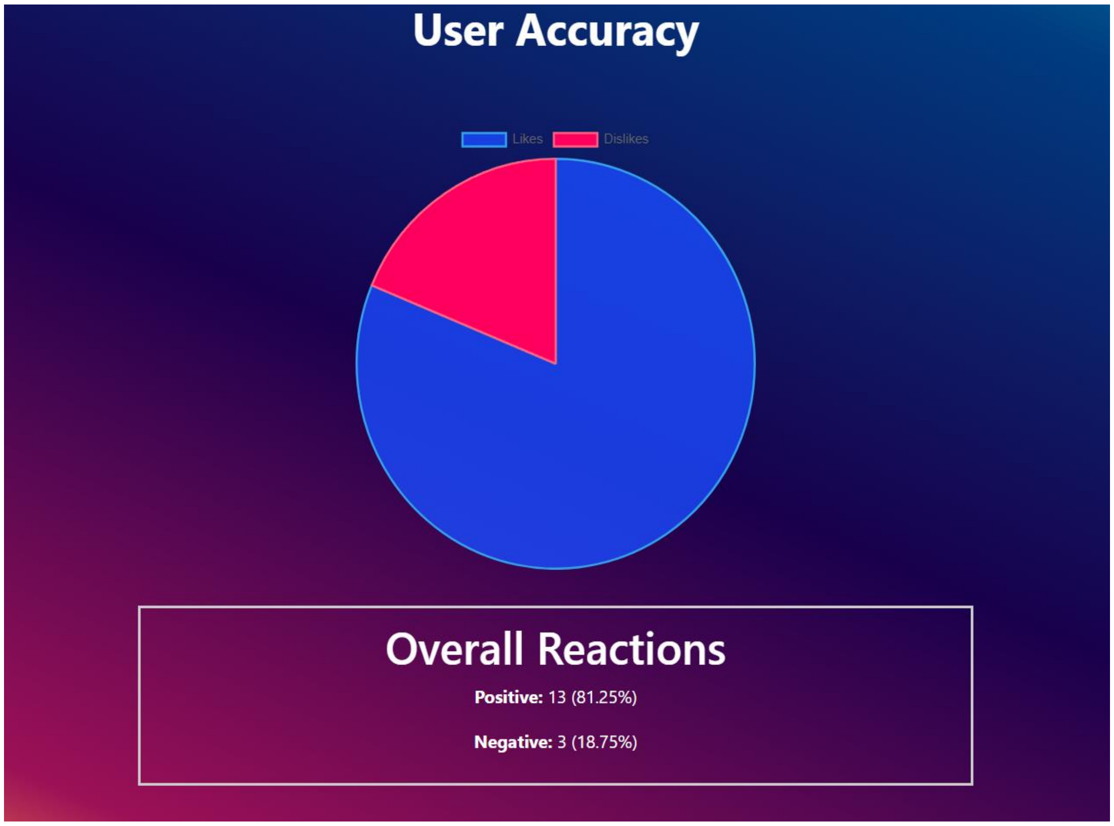
Figure 92 - Frontend page view of Homepage

6.5. User Accuracy Page.

The 'User Accuracy' page within the VQA Chat-App presents a comprehensive overview of user sentiments and engagement. At its core, the page is designed to offer a visual representation of user reactions through a pie chart, prominently displayed in the 'chart-container.' This chart, powered by the Chart.js library, dynamically illustrates the distribution between positive and negative reactions by leveraging Flask variables, specifically likes_count and dislikes_count, to populate the chart data.

Complementing the visual representation, the page includes an 'Overall Reactions' section encapsulated in a styled 'reaction-box,' delivering concise statistics on user sentiment. Here, users can quickly grasp the count and percentage of positive ('likes') and negative ('dislikes') reactions, aiding in a swift understanding of the overall user sentiment towards the application's content.

The page functionality is orchestrated through additional scripts, importing the Chart.js library and defining a JavaScript function named renderPieChart. This function intelligently renders the pie chart by accessing and utilizing data fetched from the Flask backend, ensuring a dynamic and accurate depiction of user reactions. By amalgamating these elements, the 'User Accuracy' page provides an insightful snapshot of user sentiments, facilitating a deeper understanding of user engagement and content reception within the VQA Chat-App.



Overall Reactions

Positive: 13 (81.25%)

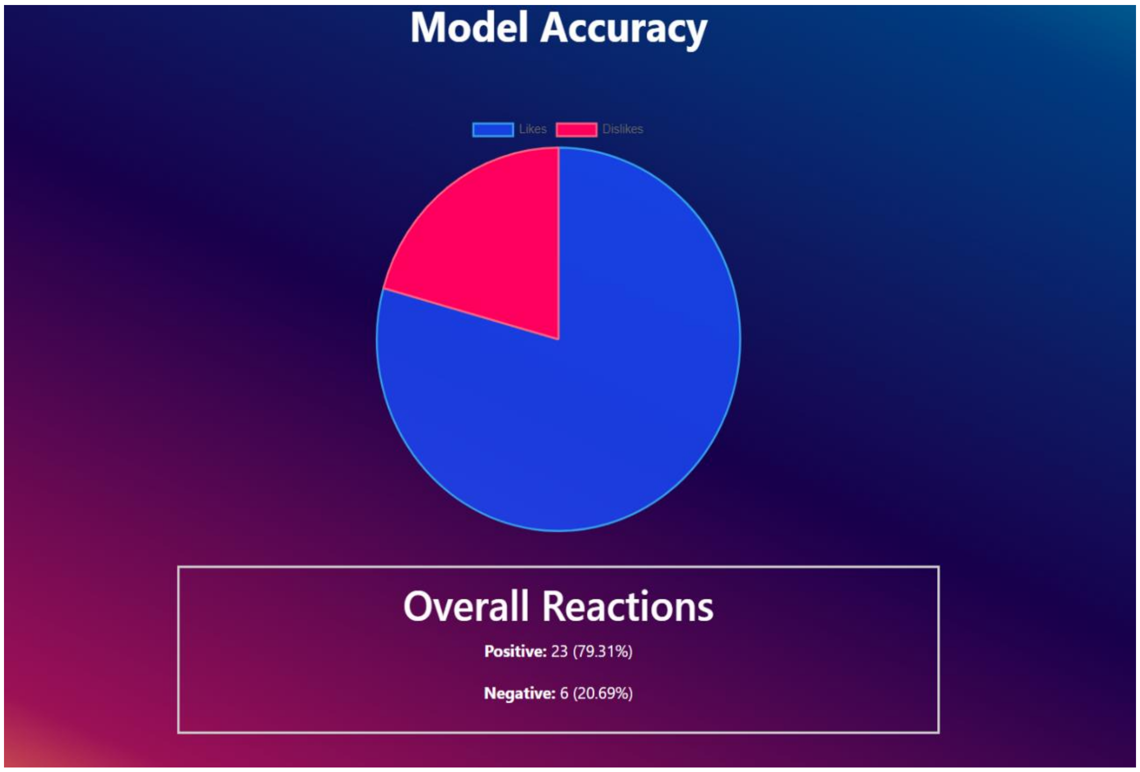
Negative: 3 (18.75%)

Figure 103 - Frontend page view of User Accuracy page

6.6. Model Accuracy Page.

The distinction between the 'Model Accuracy' and 'User Accuracy' pages lies in their respective scopes and focuses within the VQA Chat-App. The 'Model Accuracy' page primarily concentrates on assessing the overall performance and reception of the application's model across the entire user base. It derives insights by aggregating reactions—such as 'likes' and 'dislikes'—across all users, providing a holistic view of how the model is perceived collectively. This evaluation is often visually represented, typically through pie charts, showcasing the distribution of positive and negative sentiments derived from user reactions.

In contrast, the 'User Accuracy' page delves into individual users' interactions and engagements within the app. It analyzes personalized reactions and preferences, offering insights into specific users' content perceptions and engagement patterns. This page tailors its data representation to depict an individual's 'likes' and 'dislikes,' offering a more personalized view of user satisfaction and content interaction. In essence, while 'Model Accuracy' assesses the model's general reception among all users, 'User Accuracy' zooms in to understand individual user preferences and engagement levels within the application.



Overall Reactions

Positive: 23 (79.31%)

Negative: 6 (20.69%)

Figure 14- Frontend page view of User Accuracy page

7. Back-End: Flask Web Application.

The back end of our web application will be supported by Flask that is a small python web framework that allows us to use multiple tools and features making the development of the web application easier and efficiently. This combination of features makes it easier for developers to build a web application faster by just having a single python file. Flask is also extensible and doesn't force a particular directory structure or require complicated boilerplate code before getting started. Flask uses the Jinja template engine to dynamically construct HTML pages with Python concepts like variables, loops, and lists. On the other hand, there are some markup formats that had allow our project to perform a returned to user via HTTP request. These templates will be used as part of this project.

For the efficiency of our project, we will create Flask web application file, that will be composed of HTML, CSS and JavaScript codes that will help achieve and implement our VQA interface. Another important aspect in our back end is the implementation of a Web Server Gateway Interface also known as WSGI, this one is the standard that specifies the communication standards between our web server and a client application. PEP333 contains the specifics of these requirements. These are the advantages that the WSGI bring to our Flask back end wed application:

- Flexibility in the application's components.
- Interoperability across Python frameworks.
- Scalability of the application as the number of users grows.
- Efficiency in terms of development speed.

The main goal for the back-end web application is to provide a strong back bone structure for Vision Question Answer user interface in a way multiple users can have an enjoyable and efficient while interacting with our VQA model.

7.1. Back-End Architecture

The back-end architecture of our VQA web application is one of the most important aspects to secure the functionality of our model. Our back-end endpoints perform all the operations connecting with the Database, authentication, searching etc. For this purpose, we have stablished several blueprints for our Flask to help our structure by grouping several functionalities into more manageable components.

The Flask Blueprint is split in such a way that the code is divided into different modules. In this section, we provide the architect with the previous application to make Blueprints that encapsulate related functionality. In this layout, we provide the most relevant five Flask Blueprints in our project:

- API Blueprint to enable external systems to search and retrieve product information

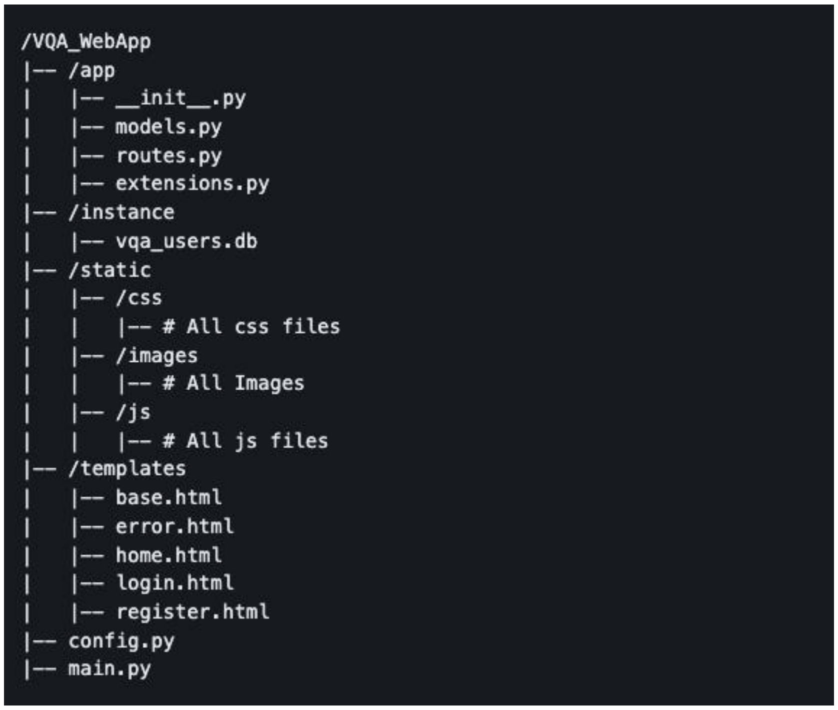
Authentication Blueprint to enable users to log in and recover their password.

- Cart Blueprint for cart and checkout functionality.
- General Blueprint for the homepage.
- Products Blueprint for searching and viewing products.

This structure has made it easier for us to find the code and resources related to a given functionality.

7.1.1. Flask Blueprint

The development of the blueprint for our web application **encapsulates functionality, such as views, templates, and other resources. To get a taste for how a Flask Blueprint would work, you can refactor the previous application by moving the** index view **into a Flask Blueprint. To do so, you have to create a Flask Blueprint that contains the index view and then use it in the application. Bellow we will provide an illustration of our web application structure layout:**



```
/VQA_WebApp
|-- /app
|   |-- __init__.py
|   |-- models.py
|   |-- routes.py
|   |-- extensions.py
|-- /instance
|   |-- vqa_users.db
|-- /static
|   |-- /css
|   |   |-- # All css files
|   |-- /images
|   |   |-- # All Images
|   |-- /js
|   |   |-- # All js files
|-- /templates
|   |-- base.html
|   |-- error.html
|   |-- home.html
|   |-- login.html
|   |-- register.html
|-- config.py
|-- main.py
```

Figure 15- VQA Web application structure layout

Here is the following description of the most relevant files from the structure:

- **main.py:**

Role: Entry point of the Flask application.

Scaling: Maintain for application-wide initializations. Rarely modified.

- **/app/__init__.py:**

Role: Initializes the Flask app and binds components like routes, database, and extensions.

Scaling: Import and register new modules or Blueprints here.

- **/app/models.py**

Role: Contains database models for SQLAlchemy.

Scaling: Define new or update existing models as data requirements evolve.

- **/app/routes.py**

Role: Houses route definitions and view functions.

Scaling: Add new routes for additional pages and features. Consider splitting into multiple files or using Blueprints for organization.

- **/app/extensions.py**

Role: Initializes and configures Flask extensions.

Scaling: Add new extensions or modify existing ones as needed. Keep focused on extensions.

- **config.py**

Role: Defines configuration settings for various environments.

Scaling: Update or add new configurations for new features requiring environment-specific settings.

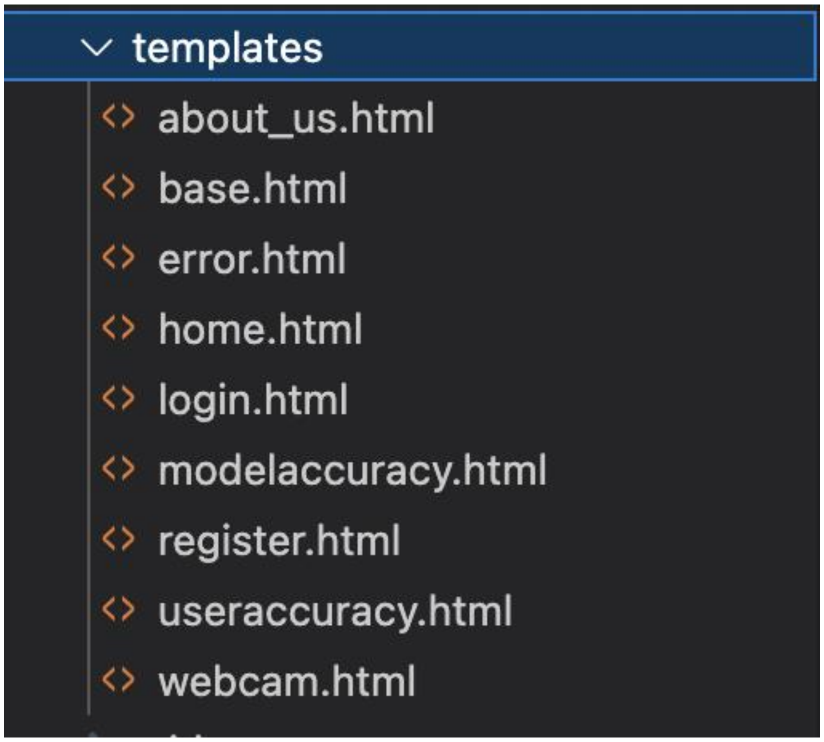
While developing a project in Flask it is important to say that it does not enforce any particular project layout. The layout of this project is structured with the purpose of an efficient Vision Question Answer development.

Also, during the development of this project we made sure that only one has a file for the application logic. This is to avoid ending up with a very large app.py that mixes code that’s nearly unrelated. This can make it hard to navigate and maintain the script.

7.1.2. HTML Templates

As a view renders a template in Flask, the template file is searched in all directories registered in the application's template search path. This path is set to ["/templates"] by default, thus templates are only searched for in the /templates directory within the application's root directory. Once the template_folder parameter is given while creating a Blueprint, the Blueprint's templates folder is added to the application's template search path when the Flask Blueprint is registered.

Here there is an illustration of the html templates in our web application:



templates
about_us.html
base.html
error.html
home.html
login.html
modelaccuracy.html
register.html
useraccuracy.html
webcam.html

Figure 16- Integration of html templates in our web application

The html templates in our work are base.html, home.html, login.html, register.html, about_us.html and others.

6.1.3 Dependencies

For the development of this project there are some crucial dependencies that need to be installed. There are 2 main different dependencies to be installed and should not conflict. These are the following:

Dependencies for Flask:

- **flask:** conda install -c anaconda flask.
- **flask-login:** conda install -c conda-forge flask-login.
- **flask_sqlalchemy:** conda install -c conda-forge flask-sqlalchemy.
- **flask_cors:** conda install -c conda-forge flask_cors.
- **jinja2:** conda install -c anaconda jinja2.
- **bcryp:** conda install -c anaconda bcryp.

Dependencies for VQA:

- ***pytorch torchvision torchaudio:*** conda install pytorch torchvision torchaudio pytorch-cuda=12.1 -c pytorch -c nvidia.
- ***transformers:*** conda install -c huggingface transformers.
- ***PIL:*** conda install -c anaconda pillow.
- ***numpy:*** conda install numpy.

Another point is to create a different or separate environment to develop this project, since it will conflict the packages and the dependencies. The Virtual environments are independent groups of Python libraries, one for each project. Packages installed for one project will not affect other projects or the operating system's packages. To Create an environment the following step. First, is to create a project folder and a .venv folder within:

For macOS/Linux:

- $ mkdir myproject
- $ cd myproject
- $ python3 -m venv .venv

For Windows:

- mkdir myproject
- cd myproject
- py -3 -m venv .venv

To conclude in this section, we have provided all the dependencies that need to be installed with the specific requirement to follow depending on the type of operation system that will be used. All of them are easy to install but it is important the installed them based on the order mentioned below.

7.2. Database Integration

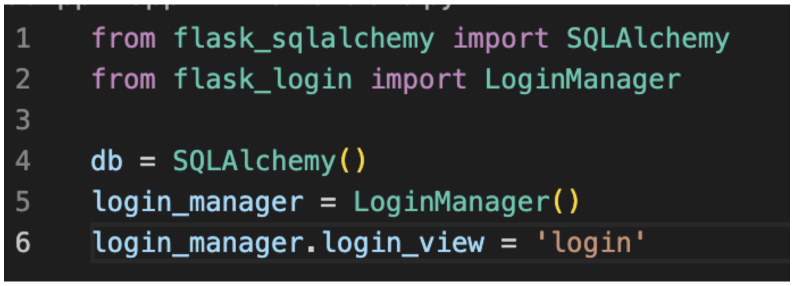
The selection of our database integration will be SQLite for the User interface web application. We decided to use SQLite not only because of her popularity but also simple-to-use relational database system. It has numerous advantages over other relational databases. At the same time is easy to install, create the tables and structure priorities.

This data integration will be based on SQLAlchemy in the back end. This will be used to add database capability to a Flask program. The SQLAlchemy is a Python SQL toolkit and object relational mapper (ORM) that allows Python to interface with your preferred SQL database system, such as MySQL, PostgreSQL, SQLite, and others. An ORM is a program that transforms data across incompatible systems, for instance the object structure in Python and table structure in a SQL database. The SQLAlchemy is essentially a connection between Python and an SQL database.

Although SQLAlchemy is not required to connect with a SQLite database, we chose to utilize it since it provides you with a skill set that can be applied to any SQL database system in the future.

Python and a number of different SQL database systems can be interfaced with by SQLAlchemy some of these systems require the installation of an extra module or library. Since Python 3.x comes with the sqlite3 module, SQLite doesn't need any additional modules.

The implementation of our database in our back end is show below:



```python
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager

db = SQLAlchemy()
login_manager = LoginManager()
login_manager.login_view = 'login'
```

Figure 17 - Illustration of the SQLAlchemy integration in the Back end

In our model.py file we integrated the different classes in the database. The user will compose of an ID number that will be attached every user once the registration is completed. When the registration is completed, the information will be stored in the database such as username, email and password.

Another important class is Reaction**, this one** provided code is to model and manage user reactions (like 'like' or 'dislike') to messages or posts within an application based on the database information. Below we can see an explanation on the structure and the information.

Reaction Class:

- Inherits from **db.Model**.
- Attributes: **id**, **user_id**, **message_id**, **reaction_type**.
- Unique **id** (primary key).
- **user_id** links to the User model (Foreign Key).
- **message_id** and **reaction_type** (like/dislike).
- Relationship with User Model:

In addition, when running the MySQL database locally, a socket string has been included. This string will be very different on MacOS and Windows.

8. Cloud Deployment

Below, we provide an illustration of our back-end architecture and the way it interacts with the cloud. The theme is to illustrate how the client-side interfaces with a sophisticated backend system to deliver a VQA service, which is likely based on machine learning, as our VQA Model is integrated. The use of AWS EC2 helps to scalable and reliable cloud infrastructure for hosting the application.

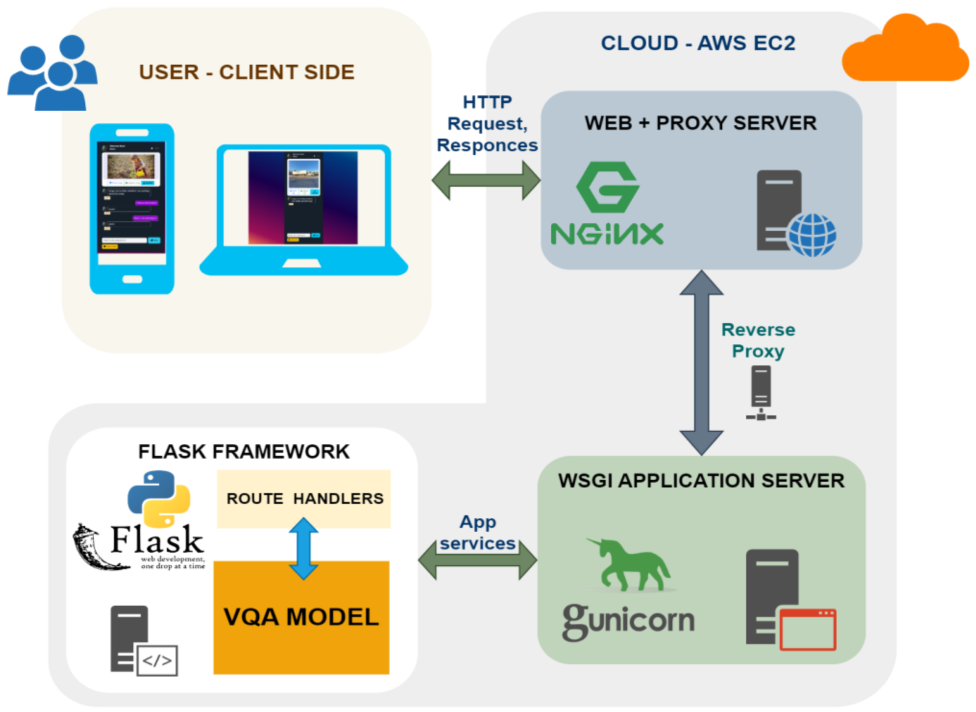


Figure 18 - Cloud Deployment Diagram

The diagram is divided into two main sections, indicating different sides of the application: the User (Client Side) and the Cloud (AWS EC2).

User-Client Side: This part shows mobile and desktop devices, suggesting that the application is accessible on various platforms. The devices display interface elements, hinting at a user-friendly GUI for interacting with the VQA system.

Cloud - AWS EC2: This section outlines the server-side components hosted on an Amazon Web Services (AWS) EC2 instance. It is further divided into layers, each with specific roles:

Web + Proxy Server (NGINX): NGINX is shown as the entry point for HTTP requests and responses. It acts as a reverse proxy, directing requests to the appropriate application server.

WSGI Application Server (Gunicorn): This layer has Gunicorn, which is a Python WSGI HTTP server for UNIX. It's responsible for running Python web applications that adhere to the WSGI standard. It seems to be the middleware that communicates between NGINX and the Flask application.

Flask Framework: Represented at the bottom of the cloud section, the Flask framework is depicted with its logo and internal components such as 'Route Handlers' and 'VQA Model'. This suggests the Flask application is structured to handle routing with specific logic for the VQA functionality.

The arrows indicate the flow of HTTP requests and responses between the client and server, as well as the internal request/response cycle within the server-side infrastructure.

8.1. Server Side

1. **Hardware**
    - **Instance Type**: AWS EC2 t3.small.
    - **Memory**: 16 GB.
    - **CPU**: Capable of handling variable workloads with burstable performance.
    - **Storage**: Based on selected AWS EBS configuration.
    - **Network**: Integrated with AWS's network infrastructure for internet access and scalability.
2. **Software**
    - **Operating System**: Ubuntu.
    - **Web Server**: Nginx, serving as a reverse proxy.
    - **Application Server**: Gunicorn, a WSGI server for running Python applications.
    - **Web Framework**: Flask, for creating the web application.
    - **Cloud Platform**: AWS EC2 for virtual server infrastructure.
3. **Communication Protocol**
    - **HTTP/HTTPS**: For handling web requests and responses.
    - **TCP/IP**: Foundation protocol for internet and intranet communication.

- **DNS Protocol**: For resolving the server's public DNS to its IP address.

4. **Interfacing**
    - **Nginx to Gunicorn**: Internal request routing from Nginx to Gunicorn.
    - **Gunicorn to Flask**: Execution of web application logic in Flask via Gunicorn.

8.2. Client Side

1. **Hardware**
    - **Devices**: Varied (e.g., smartphones, tablets, laptops, desktops).
    - **Specifications**: Diverse in terms of CPU, memory, and storage, depending on the device.
2. **Software**
    - **Web Browser**: Chrome, Firefox, Safari, etc., for accessing the web application.
    - **Operating System**: Various (Windows, macOS, Linux, iOS, Android).
3. **Communication Protocol**
    - **HTTP/HTTPS**: For sending requests to and receiving responses from the server.
    - **TCP/IP**: Underlying protocol for data transmission over the internet.
4. **Interfacing**
    - **User to Web Browser**: Direct interaction via graphical interface for accessing and interacting with the web application.
    - **Web Browser to Server**: Communication through HTTP/HTTPS requests and responses.

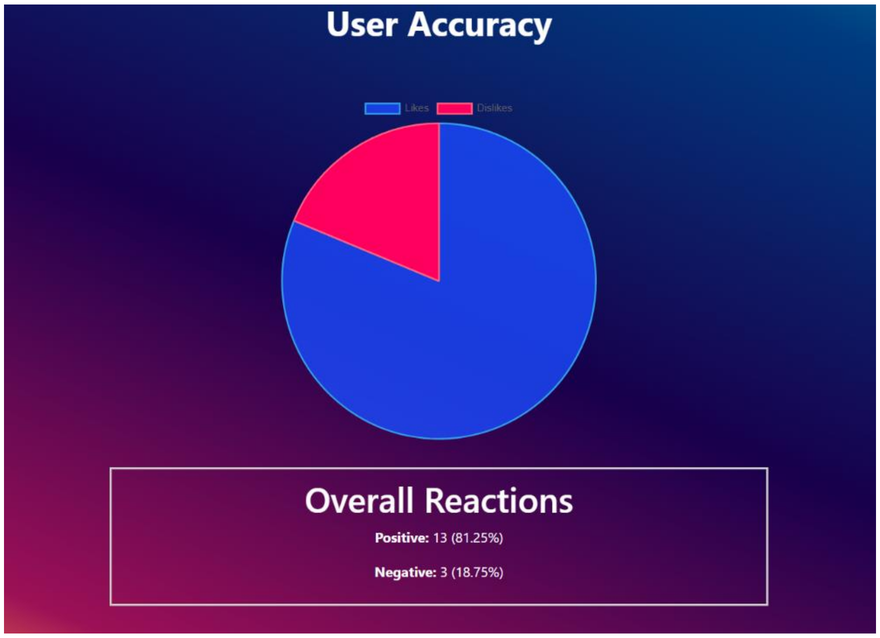
9. System Evaluation - Beta Testing

User Feedback is integral to system evaluation as it provides invaluable insights into the performance, usability, and efficacy of the system. By gathering user feedback, system evaluators can gauge user satisfaction, identify issues, and assess the system's strengths and weaknesses. In this application, (VQA chat App), feedback of the user is gathered in binary format (either likes or dislikes). The detailed implementation is illustrated below: -

9.1. User Feedback collection.

In our application, user feedback is collected through reactions given to bot messages. Users in the app can provide feedback through simple reactions: either "liking" or "disliking" certain messages. Each interaction with the "like" or "dislike" button contributes to user feedback. When a user reacts (likes or dislikes) to a message, an entry is made in the database. Each reaction entry typically includes the user ID, the ID of the message/content reacted to, and the type of reaction (like or dislike).

For user accuracy metrics, the system calculates the count and percentages of likes and dislikes for a particular user. This involves querying the database to count the number of likes and dislikes associated with a user or specific content ID. These metrics are then displayed, showing the user's activity and engagement based on their reactions. User accuracy is calculated based on the ratio of likes or dislikes over the total number of reactions. Feedback data, especially dislikes, could be used to improve the model, informing decisions about what users find less appealing or helpful.



Overall Reactions

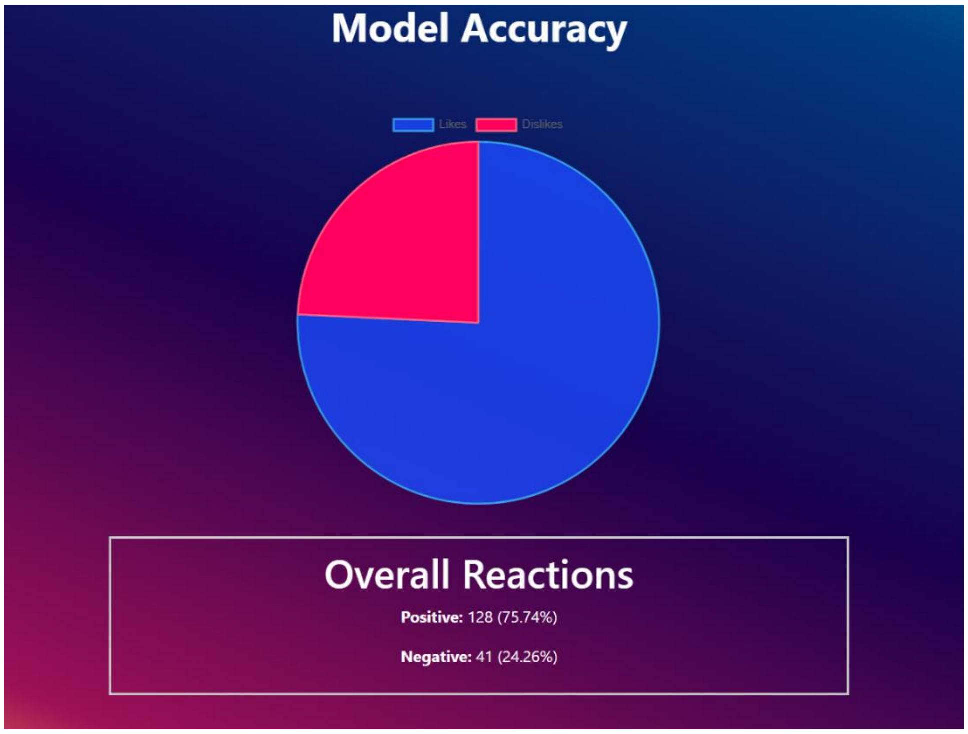
Positive: 13 (81.25%)

Negative: 3 (18.75%)

Figure 19 - Demonstration of UserAccuracy.html page

9.2. Overall Model Accuracy after Beta Testing

- The system calculates the aggregate count of likes and dislikes across all users. This aggregation involves querying the database's Reaction table, specifically filtering to count the total number of likes and dislikes stored in the database.
- The system calculates the total number of reactions by summing up the counts of likes and dislikes obtained in the previous step.
- After computing the total number of reactions, the system calculates the percentage breakdown for both likes and dislikes. This calculation involves determining what proportion of the total reactions corresponds to likes and dislikes.
- **Overall ViLT Model Accuracy after Beta Testing**:
    - **75.7%** Positive feedback of total **161 feedback**.
    - 24.76% Negative feedback of total **161 feedback**.



Overall Reactions

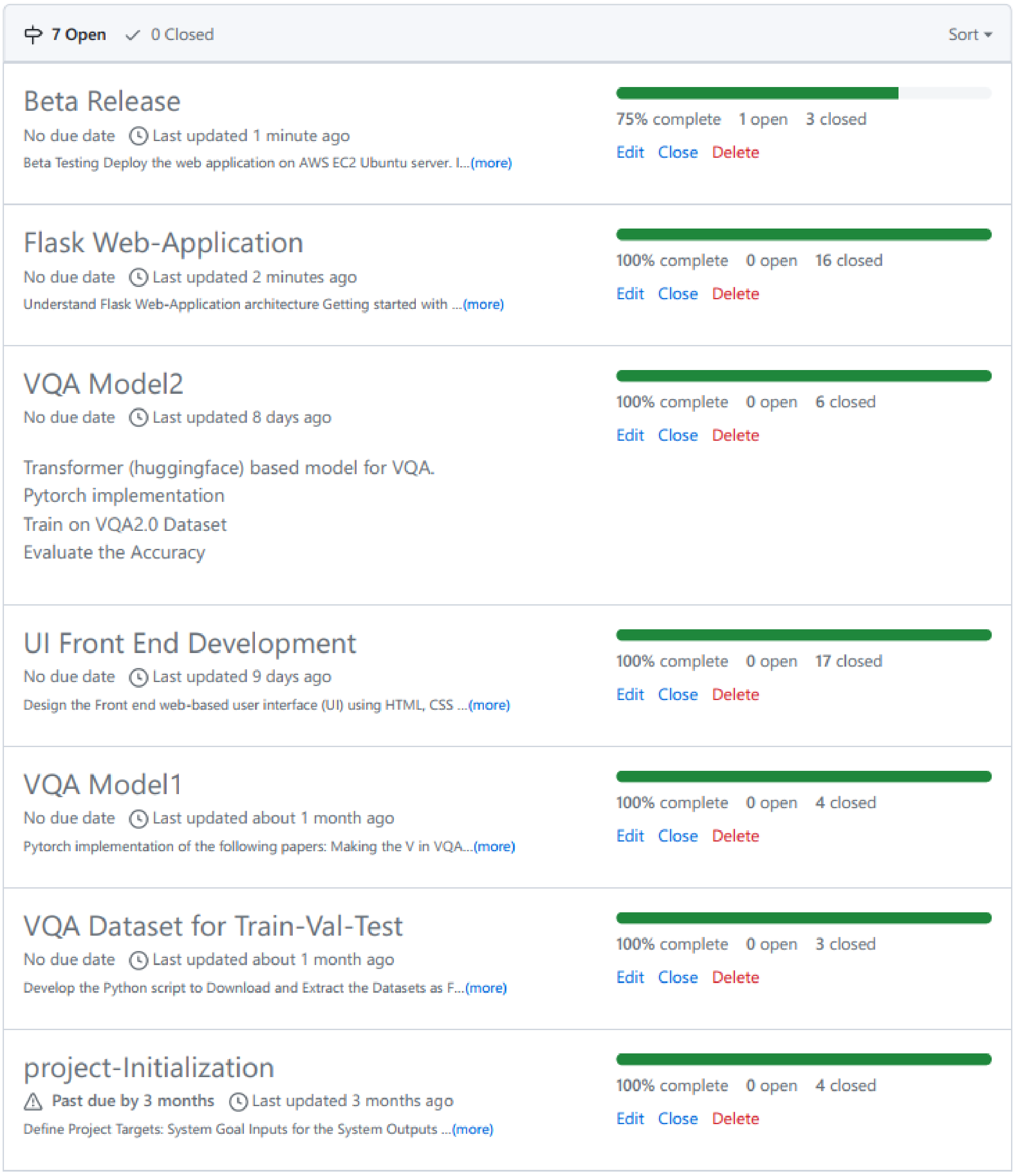
Positive: 128 (75.74%)

Negative: 41 (24.26%)

Figure 20- Overall Model Accuracy after Beta Testing

10.Repository Management

10.1. Milestones



7 Open ✓ 0 Closed Sort

Beta Release
No due date Last updated 1 minute ago
Beta Testing Deploy the web application on AWS EC2 Ubuntu server. I...(more)
75% complete 1 open 3 closed
Edit Close Delete

Flask Web-Application
No due date Last updated 2 minutes ago
Understand Flask Web-Application architecture Getting started with ...(more)
100% complete 0 open 16 closed
Edit Close Delete

VQA Model2
No due date Last updated 8 days ago

Transformer (huggingface) based model for VQA.
Pytorch implementation
Train on VQA2.0 Dataset
Evaluate the Accuracy
100% complete 0 open 6 closed
Edit Close Delete

UI Front End Development
No due date Last updated 9 days ago
Design the Front end web-based user interface (UI) using HTML, CSS ...(more)
100% complete 0 open 17 closed
Edit Close Delete

VQA Model1
No due date Last updated about 1 month ago
Pytorch implementation of the following papers: Making the V in VQA...(more)
100% complete 0 open 4 closed
Edit Close Delete

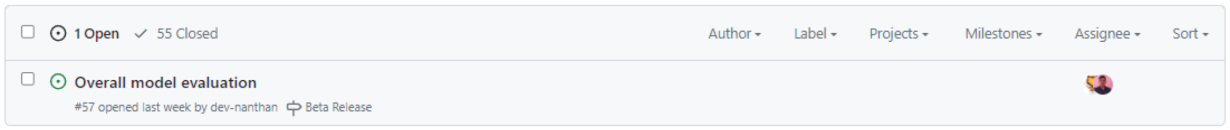
VQA Dataset for Train-Val-Test
No due date Last updated about 1 month ago
Develop the Python script to Download and Extract the Datasets as F...(more)
100% complete 0 open 3 closed
Edit Close Delete

project-Initialization
Past due by 3 months Last updated 3 months ago
Define Project Targets: System Goal Inputs for the System Outputs ...(more)
100% complete 0 open 4 closed
Edit Close Delete

Figure 21- Milestones for Capstone_project_team2

10.2. Issues Status

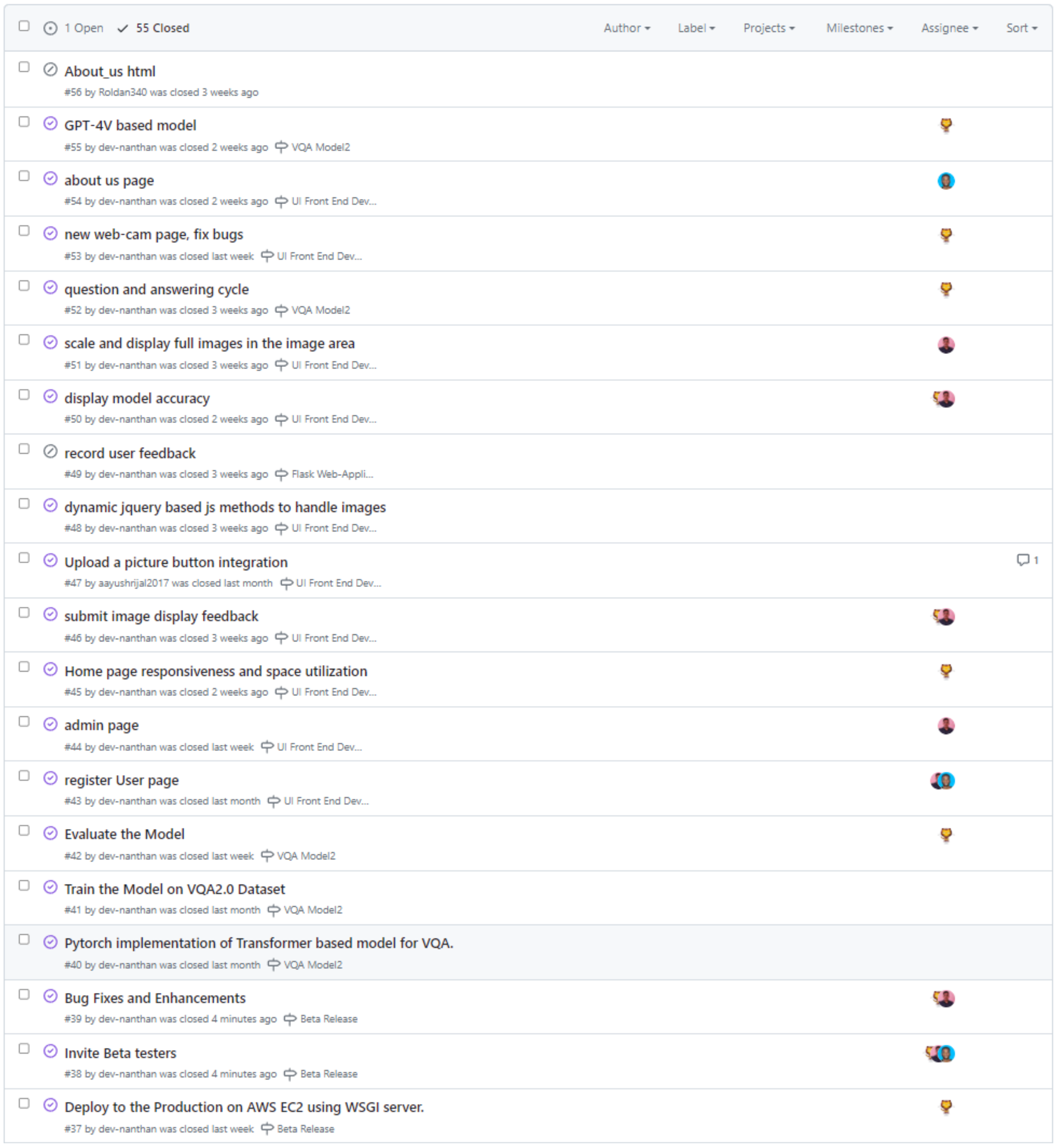
Opened Issues:



1 Open ✓ 55 Closed — Author ▾ Label ▾ Projects ▾ Milestones ▾ Assignee ▾ Sort ▾

Overall model evaluation
#57 opened last week by dev-nanthan — Beta Release

Closed Issues in Reverse Chronological order:



1 Open ✓ 55 Closed — Author ▾ Label ▾ Projects ▾ Milestones ▾ Assignee ▾ Sort ▾

About_us html
#56 by Roldan340 was closed 3 weeks ago

GPT-4V based model
#55 by dev-nanthan was closed 2 weeks ago — VQA Model2

about us page
#54 by dev-nanthan was closed 2 weeks ago — UI Front End Dev...

new web-cam page, fix bugs
#53 by dev-nanthan was closed last week — UI Front End Dev...

question and answering cycle
#52 by dev-nanthan was closed 3 weeks ago — VQA Model2

scale and display full images in the image area
#51 by dev-nanthan was closed 3 weeks ago — UI Front End Dev...

display model accuracy
#50 by dev-nanthan was closed 2 weeks ago — UI Front End Dev...

record user feedback
#49 by dev-nanthan was closed 3 weeks ago — Flask Web-Appli...

dynamic jquery based js methods to handle images
#48 by dev-nanthan was closed 3 weeks ago — UI Front End Dev...

Upload a picture button integration — 1
#47 by aayushrijal2017 was closed last month — UI Front End Dev...

submit image display feedback
#46 by dev-nanthan was closed 3 weeks ago — UI Front End Dev...

Home page responsiveness and space utilization
#45 by dev-nanthan was closed 2 weeks ago — UI Front End Dev...

admin page
#44 by dev-nanthan was closed last week — UI Front End Dev...

register User page
#43 by dev-nanthan was closed last month — UI Front End Dev...

Evaluate the Model
#42 by dev-nanthan was closed last week — VQA Model2

Train the Model on VQA2.0 Dataset
#41 by dev-nanthan was closed last month — VQA Model2

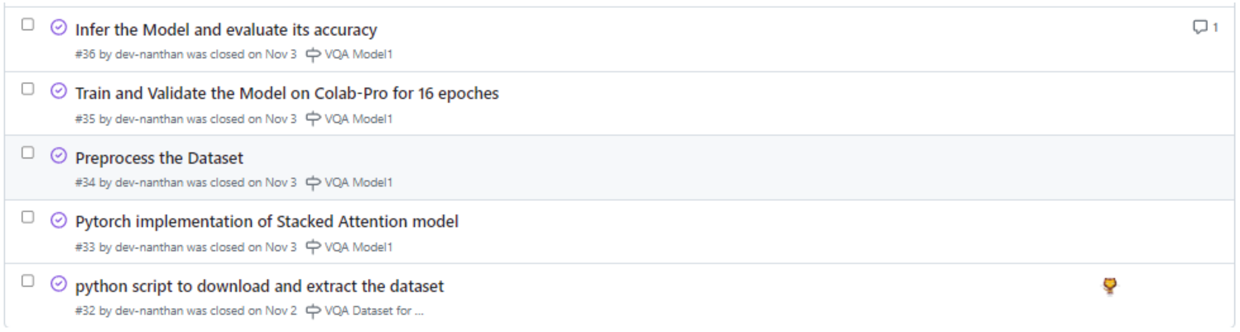
Pytorch implementation of Transformer based model for VQA.
#40 by dev-nanthan was closed last month — VQA Model2

Bug Fixes and Enhancements
#39 by dev-nanthan was closed 4 minutes ago — Beta Release

Invite Beta testers
#38 by dev-nanthan was closed 4 minutes ago — Beta Release

Deploy to the Production on AWS EC2 using WSGI server.
#37 by dev-nanthan was closed last week — Beta Release

Closed Issues Cont.:



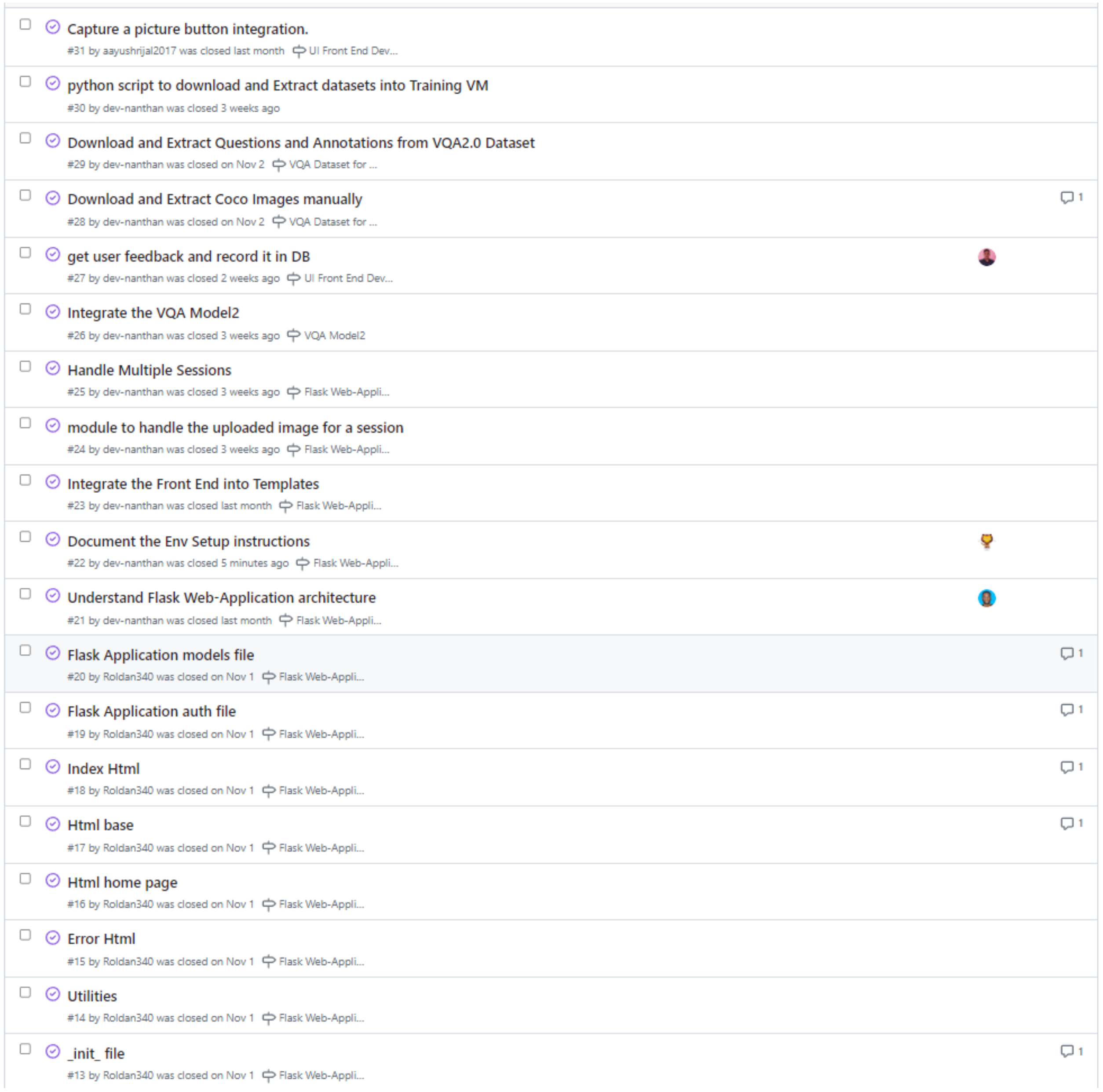
Infer the Model and evaluate its accuracy — 1
#36 by dev-nanthan was closed on Nov 3 VQA Model1

Train and Validate the Model on Colab-Pro for 16 epoches
#35 by dev-nanthan was closed on Nov 3 VQA Model1

Preprocess the Dataset
#34 by dev-nanthan was closed on Nov 3 VQA Model1

Pytorch implementation of Stacked Attention model
#33 by dev-nanthan was closed on Nov 3 VQA Model1

python script to download and extract the dataset
#32 by dev-nanthan was closed on Nov 2 VQA Dataset for ...



Capture a picture button integration.
#31 by aayushrijal2017 was closed last month UI Front End Dev...

python script to download and Extract datasets into Training VM
#30 by dev-nanthan was closed 3 weeks ago

Download and Extract Questions and Annotations from VQA2.0 Dataset
#29 by dev-nanthan was closed on Nov 2 VQA Dataset for ...

Download and Extract Coco Images manually — 1
#28 by dev-nanthan was closed on Nov 2 VQA Dataset for ...

get user feedback and record it in DB
#27 by dev-nanthan was closed 2 weeks ago UI Front End Dev...

Integrate the VQA Model2
#26 by dev-nanthan was closed 3 weeks ago VQA Model2

Handle Multiple Sessions
#25 by dev-nanthan was closed 3 weeks ago Flask Web-Appli...

module to handle the uploaded image for a session
#24 by dev-nanthan was closed 3 weeks ago Flask Web-Appli...

Integrate the Front End into Templates
#23 by dev-nanthan was closed last month Flask Web-Appli...

Document the Env Setup instructions
#22 by dev-nanthan was closed 5 minutes ago Flask Web-Appli...

Understand Flask Web-Application architecture
#21 by dev-nanthan was closed last month Flask Web-Appli...

Flask Application models file — 1
#20 by Roldan340 was closed on Nov 1 Flask Web-Appli...

Flask Application auth file — 1
#19 by Roldan340 was closed on Nov 1 Flask Web-Appli...

Index Html — 1
#18 by Roldan340 was closed on Nov 1 Flask Web-Appli...

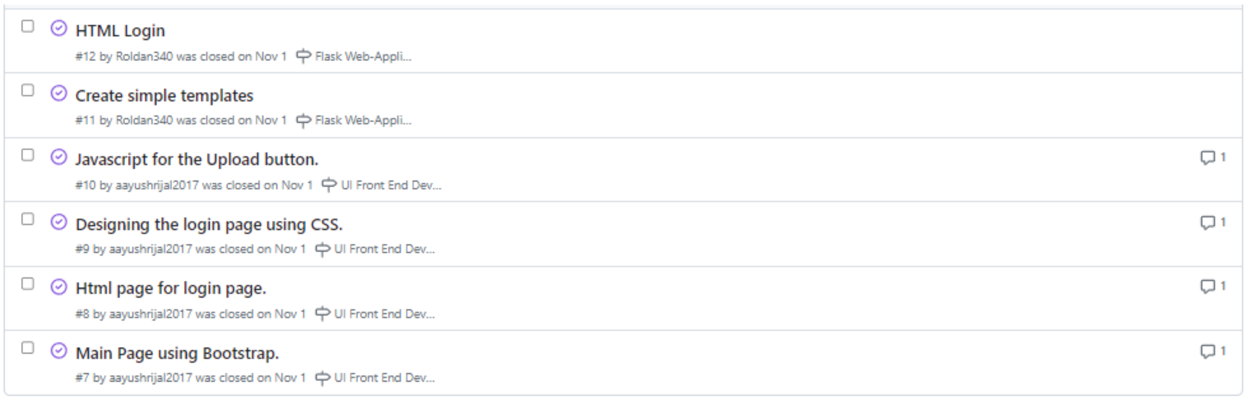
Html base — 1
#17 by Roldan340 was closed on Nov 1 Flask Web-Appli...

Html home page
#16 by Roldan340 was closed on Nov 1 Flask Web-Appli...

Error Html
#15 by Roldan340 was closed on Nov 1 Flask Web-Appli...

Utilities
#14 by Roldan340 was closed on Nov 1 Flask Web-Appli...

init file — 1
#13 by Roldan340 was closed on Nov 1 Flask Web-Appli...



HTML Login
#12 by Roldan340 was closed on Nov 1 Flask Web-Appli...

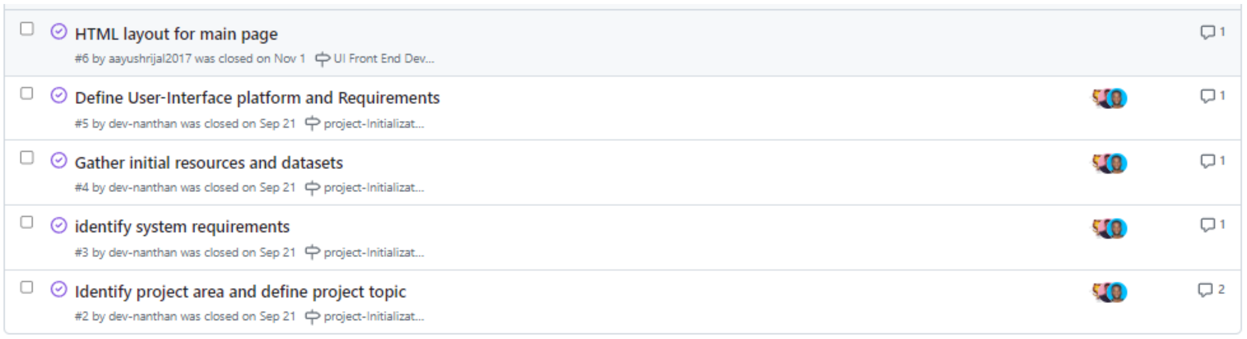
Create simple templates
#11 by Roldan340 was closed on Nov 1 Flask Web-Appli...

Javascript for the Upload button. 1
#10 by aayushrijal2017 was closed on Nov 1 UI Front End Dev...

Designing the login page using CSS. 1
#9 by aayushrijal2017 was closed on Nov 1 UI Front End Dev...

Html page for login page. 1
#8 by aayushrijal2017 was closed on Nov 1 UI Front End Dev...

Main Page using Bootstrap. 1
#7 by aayushrijal2017 was closed on Nov 1 UI Front End Dev...



HTML layout for main page 1
#6 by aayushrijal2017 was closed on Nov 1 UI Front End Dev...

Define User-Interface platform and Requirements 1
#5 by dev-nanthan was closed on Sep 21 project-Initializat...

Gather initial resources and datasets 1
#4 by dev-nanthan was closed on Sep 21 project-Initializat...

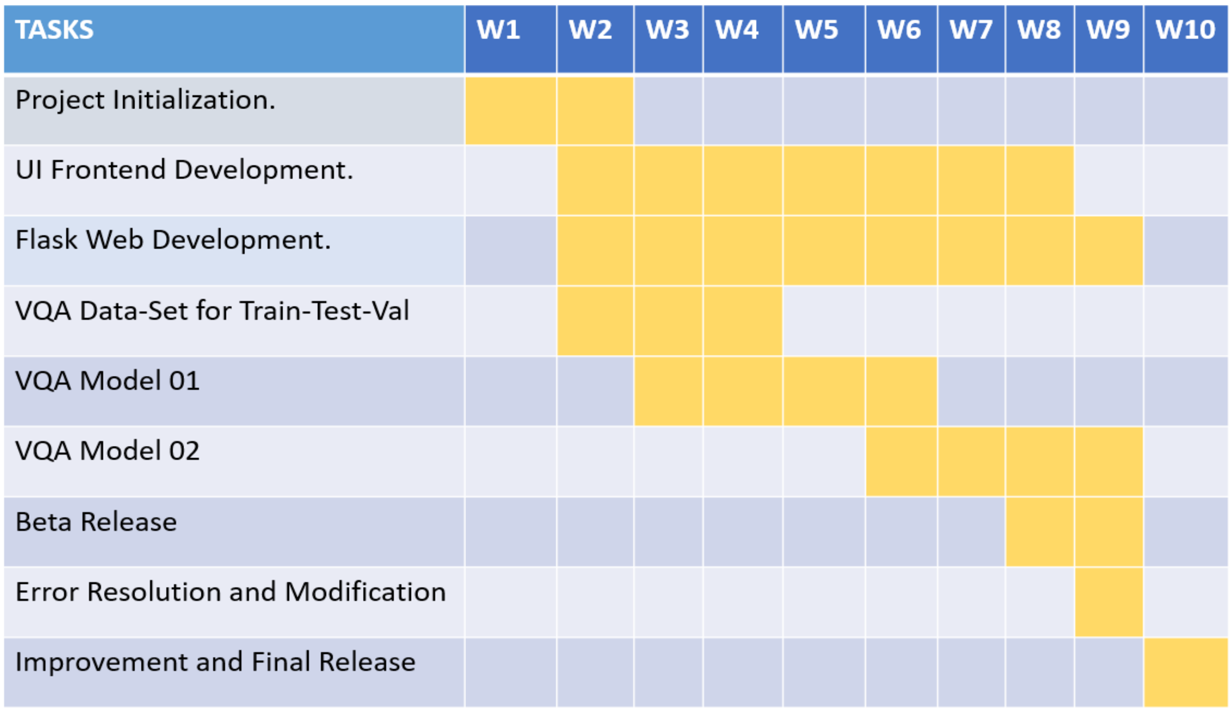
identify system requirements 1
#3 by dev-nanthan was closed on Sep 21 project-Initializat...

Identify project area and define project topic 2
#2 by dev-nanthan was closed on Sep 21 project-Initializat...

11.Project Timeline

For the VQA Web-Application project, we have divided the work into the following categories. Which are essentially set as the milestones in GitHub and being worked on under each issue.

Updated and tracked Timeline of the project:



TASKS	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10
Project Initialization.	🟨	🟨								
UI Frontend Development.		🟨	🟨	🟨	🟨	🟨	🟨	🟨		
Flask Web Development.		🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	
VQA Data-Set for Train-Test-Val		🟨	🟨	🟨						
VQA Model 01			🟨	🟨	🟨	🟨				
VQA Model 02						🟨	🟨	🟨	🟨	
Beta Release								🟨	🟨	
Error Resolution and Modification									🟨	
Improvement and Final Release										🟨

Figure 22 - Project Timeline

12. References

[1] T. K. D. S.-S. D. B. D. P. Yash Goyal, "Making the V in VQA Matter: Elevating the Role of Image Understanding in Visual Question Answering," in *CVPR*, 2017.

[2] B. S. I. K. Wonjae Kim, "ViLT: Vision-and-Language Transformer Without Convolution or Region Supervision," in *ICML* , 2021.

[3] R. S. M. C. A. D. R. Ramprasaath, "Grad-CAM: Visual Explanations from Deep Networks," 2019.

[4] X. H. C. B. D. T. M. J. Peter Anderson, "Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answerin," 2018.