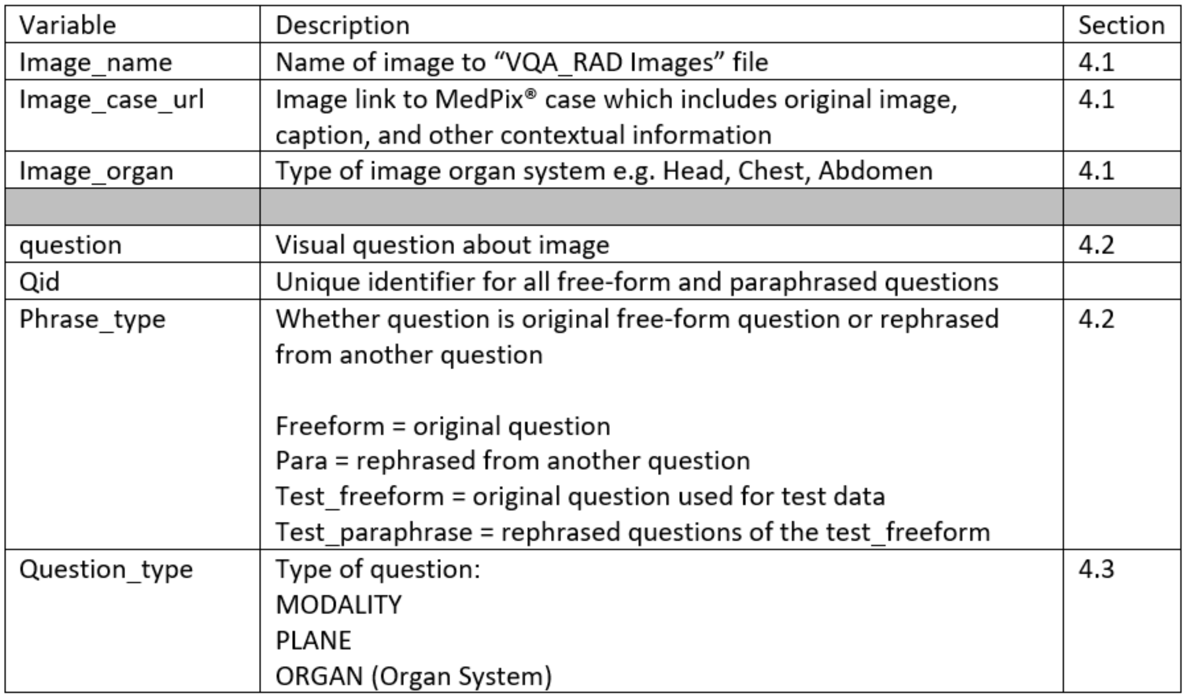
INTER IIT ROUND-2 PS SOLUTION

Problem Description:
Our task involves building a specialized Visual Question Answering (VQA) model tailored to medical images, with a particular focus on radiology. This entails interpreting complex radiological images and comprehending textual questions about them. To tackle this, we rely on the VQA-RAD dataset, consisting of question-answer pairs from 315 radiology images. The dataset includes open-ended and binary "yes/no" questions, making it ideal for training and evaluating Medical Visual Question Answering models. It's important to emphasize that, like all medical datasets, using VQA-RAD requires strict adherence to ethical and privacy standards due to the sensitive patient information involved.

Dataset Description:
The given dataset contains 2248 rows and 14 columns or variables. The VQA-RAD Image folder contains 315 images.



Variable	Description	Section
Image_name	Name of image to "VQA_RAD Images" file	4.1
Image_case_url	Image link to MedPix® case which includes original image, caption, and other contextual information	4.1
Image_organ	Type of image organ system e.g. Head, Chest, Abdomen	4.1
question	Visual question about image	4.2
Qid	Unique identifier for all free-form and paraphrased questions	
Phrase_type	Whether question is original free-form question or rephrased from another question Freeform = original question Para = rephrased from another question Test_freeform = original question used for test data Test_paraphrase = rephrased questions of the test_freeform	4.2
Question_type	Type of question: MODALITY PLANE ORGAN (Organ System)	4.3

	ABN (Abnormality) PRES (Object/Condition Presence) POS (Positional Reasoning) COLOR SIZE ATTRIB (Attribute Other) COUNT (Counting) Other	
Answer	Answer to the question	4.2
Answer_type	type of answer, e.g. closed-ended, open-ended	4.4
Evaluation	Whether question-answer pair was clinically evaluated by a 2nd clinician, e.g evaluated = two clinical annotators reviewed image and QA pair, not evaluated = one clinical annotator	4.5
Question_relation	Relationship between linked question-answer pairs e.g. Strict agreement Loose agreement - Inversion - Conversion - Subsumption Not similar	4.5
Qid_linked_id	Unique identifier for every pair of free-form and paraphrased questions that can be used to link original and rephrasing	
Question_rephrase	Rephrasing of 'question', can be freeform or para, linked through qid_linked_id	4.2
Question_frame	Rephrasing of 'question' following a templated structure	4.2

6.0 DATASET CHARACTERISTICS

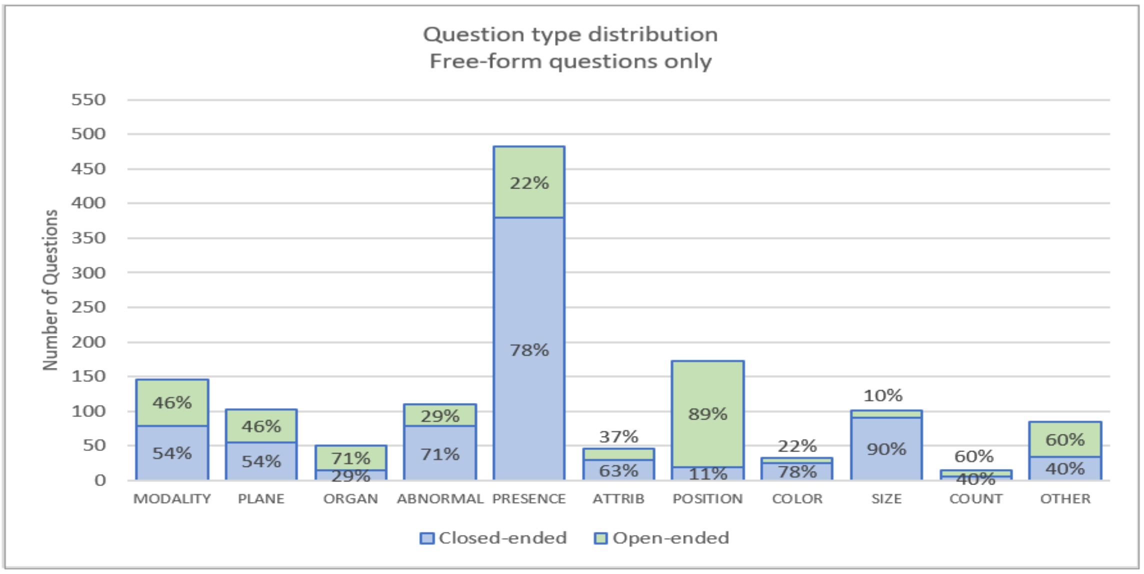


Figure 2. Closed vs Open Ended Questions and Breakdown of different types (free form questions only) Certain questions types more likely to be open-ended: positional, counting questions and other.

Approach:
The code that we used has several dependencies like it is not a single notebook but is dependent on multiple files. Below we provide what the code in each file does:

VQA_RAD_Dataset.py
This code defines a PyTorch dataset class, `VQA_RAD_Dataset`, tailored for the VQA-RAD dataset, which contains clinical questions and answers related to radiology images. It loads question-answer pairs, processes images and text, and provides data suitable for training and testing medical Visual Question Answering (VQA) models. The class initializes with parameters such as file paths and transformation settings. During training, it combines questions and answers, tokenized text, handles sequence padding, and generates labels for the model. For test mode, it returns questions, image paths, images, and answers.

randaugment.py
This code defines a set of image augmentation functions and a class `RandomAugment` for applying random data augmentations to images. The augmentation functions include transformations like identity, auto contrast, equalize, rotate, solarize, color, contrast, brightness, sharpness, shear, translate, posterize, and more. The `RandomAugment` class randomly selects a specified number of these augmentation functions and applies them to an input image, simulating a diverse set of image variations for data augmentation in computer vision tasks. These augmentations are commonly used to increase the diversity of the training dataset and improve model generalization.

blocks.py
The provided code contains various components and building blocks for neural network architectures. We have defined all the possible blocks that we thought can be used in the architecture but at the same time this is not the final architecture, this file holds all the possible types of neural network blocks that can be used.

PMC_CLIP_cfg Class:
This class defines various configuration settings for a model. It allows you to specify the backbone type, the number of layers, patch size, image size, and other hyperparameters for the model. You can configure the model architecture by creating an instance of this class with specific settings.

Bottleneck Class:
The `Bottleneck` class defines a bottleneck block, which is a common building block in architectures like ResNet. It consists of three convolutional layers (with the first layer having a 1x1 kernel, the second layer having a 3x3 kernel, and the third layer having a 1x1 kernel), batch

normalization, and ReLU activation functions. The `Bottleneck` class includes a residual connection to allow the model to learn identity mappings.

AttentionPool2d Class:
This class implements an attention pooling mechanism for 2D data, which can be used to aggregate information from spatial regions of an image. It uses multi-head self-attention to compute attention weights for different parts of the input data, and it aggregates the features accordingly.

ResNet Class:
The `ResNet` class defines a ResNet architecture for image feature extraction. It includes multiple layers composed of `Bottleneck` blocks. The forward method processes input images through the layers, extracts features, and produces output feature vectors. These features can be used for tasks like image classification.

ModifiedResNet Class:
Similar to the `ResNet` class, the `ModifiedResNet` class defines a modified ResNet architecture with specific changes, such as multiple stem convolutions and the use of attention pooling in the final layer.

LayerNorm Class:
This class is a custom implementation of layer normalization. It is used for normalizing the output of intermediate layers in the model. It ensures that the output features have a consistent scale and distribution.

QuickGELU Class:
This class defines a GELU activation function, which can be used as an activation function within the residual attention blocks. GELU is commonly used in deep learning models to introduce non-linearity.

ResidualAttentionBlock Class:
The `ResidualAttentionBlock` class represents a building block of a Transformer-like architecture. It combines multi-head self-attention with MLP layers.
This block can be used to capture complex dependencies between features in the input data and is suitable for various tasks, including image classification.

PatchDropout Class:
The `PatchDropout` class implements patch-level dropout for images. It randomly drops patches from the input data, which can be seen as a regularization technique to prevent overfitting during training.

Transformer Class:
The `Transformer` class defines a custom Transformer architecture. It uses multiple `ResidualAttentionBlock` instances stacked on top of each other. This architecture can capture long-range dependencies in the input data, making it suitable for various computer vision tasks.

In the context of the problem, we use these components to construct a custom neural network architecture. Like according to the need we can add a neural block from this file if required.

<u>QA_model_mlp.py</u>
This code leverages various visual models, such as PMC-CLIP, CLIP, or Scratch, to extract features from medical images. The model also incorporates a large language model (LLM), like Llama, to handle textual questions.During inference, it can generate text-based answers to medical questions, making it suitable for medical Visual Question Answering (VQA) tasks. The use of Prefix-Tuning (PEFT) and pre-trained models enhances the model's adaptability for specific medical VQA applications.

<u>transformer.py</u>
This code defines classes for a transformer-based decoder, allowing for multi-head self-attention and feedforward operations. It includes both forward and backward normalization.

<u>train_downstream.py</u>
The script is designed to be used for training VQA models on different datasets by specifying dataset paths, model hyperparameters, and training settings through command-line arguments. Depending on the dataset used, it appends dataset-specific information to the run_name and output_dir.

<u>Models:</u>
We used the MedVInT-TD MedVQA model architecture for this task. Here's a brief description of each unit in the model architecture:

Visual Encoder:
- Responsible for processing input images.
- Utilizes a pre-trained ResNet-50 architecture from PMC-CLIP.
- Enhances the pre-trained visual features using a trainable projection module.

Text Encoder:
- Handles textual inputs, such as questions.
- Modeled as a simple embedding layer.
- Initialized with parameters from primary language models like GPT.
- Produces embedding features for textual data.

Multimodal Decoder:
- Serves as the transformer decoder-based language model.
- The decoder's output format is free-form text.
- Combines image and text features as input to answer questions.
- Pre-trained on the PMC-OA dataset to ensure alignment between the image and text spaces for VQA tasks.

Each unit in the architecture plays a specific role in processing and combining visual and textual information to answer questions in a VQA context.

Other Models We Tried:
hi-VQA Architecture based on Rad-ReStruct dataset.

Results:
Open Accuracy(Accuracy on open ended questions): 72.4%
Close Accuracy(Accuracy on open ended questions): 85.8%
Overall Accuracy: ~80%

Architecture Backbones:
Vision backbones: CLIP & PMC-CLIP architectures
Language backbones: LLaMA, PMC-LLaMA, PubMedBERT, LLaMA-ENC & PMC-LLaMA-ENC.

Contributions(Team 2):
Vaastavi Kumar: NLP segment of the project.
Mohit Agrawal: CV segment of the project.
Ajit Priyadarshi: Overall(including CV, NLP & code integration).
Mitra Pidaparti: CV segment of the project.
Aryan Arya: CV segment of the project.

References:
https://arxiv.org/pdf/2307.05766.pdf
https://arxiv.org/pdf/2305.10415v5.pdf
https://bmcmedimaging.biomedcentral.com/articles/10.1186/s12880-022-00800-x