Advanced Machine Learning (WOA7015) (OCC4)

Alternative Assessment (Preliminary Report)

2025/2026 Sem 1

From Discriminative Fusion to Vision–Language Generation: A Comparative Study of Medical Visual Question Answering on VQA-RAD

Group: backPropagation

Team members & Matrix No.:

Ooi Li Yoong (24226057)

Yan ZhenYang (23090967)

BACKGROUND

Medical Visual Question Answering (Med-VQA) seeks to develop artificial intelligence systems capable of answering natural-language questions based on medical images such as X-rays, computed tomography (CT), and magnetic resonance imaging (MRI) scans (Li, Yang, et al., 2025). This capability supports healthcare professionals by assisting in disease diagnosis, medical image interpretation, and informed clinical decision-making (Li, Man, Zhou, & Liang, 2025). In addition, Med-VQA serves as a valuable educational tool, facilitating interactive clinical scenario-based learning for students and trainees, while also helping patients better understand their medical conditions (Lu, Zeng, Lu, Chen, & Xia, 2025).

Unlike conventional image classification tasks, Med-VQA requires models to jointly understand complex visual patterns and clinical language, making it a challenging multimodal problem (Li, Yang, et al., 2025). To support research in this area, the VQA-RAD dataset was introduced by Lau, Gayen, Ben Abacha, and Demner-Fushman (2018), providing radiology images paired with clinically generated question–answer annotations. Furthermore, the dataset includes both closed-ended questions (e.g., yes/no) and open-ended questions that require descriptive or explanatory answers, making it a challenging benchmark for evaluating multimodal reasoning in the medical domain.

Early approaches to Med-VQA, including those evaluated in the original VQA-RAD study, predominantly formulated the task as a multi-class classification problem (Lau et al., 2018). These models typically consisted of three components: a convolutional neural network (CNN) for image feature extraction, a recurrent neural network such as a Long Short-Term Memory (LSTM) for question encoding, and a multimodal fusion module to combine visual and textual representations. While simple feature concatenation can be used, subsequent work explored more advanced fusion

mechanisms such as multimodal compact bilinear pooling and attention-based fusion to better align image regions with relevant question semantics (Liu, Su, Guo, & Zhu, 2022).

Despite these architectural improvements, the limited size of VQA-RAD (2244 instances) poses significant challenges for model training and fine-tuning (Li, Man, et al., 2025). Models pretrained on large-scale natural image datasets, such as ImageNet, often suffer from a semantic gap when applied to medical images, as radiological patterns and clinical terminology differ substantially from those seen during pretraining. This has motivated research into medical-specific transfer learning, where models are pretrained on domain-relevant image–text data, such as radiology reports, to learn more suitable visual and textual representations (Gong, Chen, Liu, Yu, & Li, 2021). Frameworks such as Biomedical Vision-Language (BioViL) exemplify this direction by aligning medical images with corresponding clinical language during pretraining, thereby providing more informative features for downstream Med-VQA tasks. Additionally, data augmentation techniques like Mixup are commonly adopted to partially alleviate data scarcity by increasing input variability and improving model robustness (Li, Yang, et al., 2025).

In parallel with improvements in representation learning and fusion strategies, researchers have also questioned the fundamental task formulation of Med-VQA. Classification-based models rely on a fixed set of predefined answers, which limits their flexibility and makes them unsuitable for open-ended questions requiring descriptive or nuanced responses. More recent Vision–Language Models (VLMs), such as Bootstrapped Language-Image Pretraining (BLIP), address this limitation by treating Med-VQA as a text generation problem. By jointly modeling vision and language and generating answers token by token, these models offer greater expressiveness and better handling of open-ended clinical queries.

Motivated by these developments, this study investigated Med-VQA with a primary focus on multimodal fusion strategies and task formulation. Using the VQA-RAD dataset as a common benchmark, we systematically compared classical CNN–LSTM-based discriminative models employing different fusion mechanisms with a pretrained generative vision–language model. In doing so, transfer learning through large-scale pretraining was considered as an enabling factor rather than the main variable of interest. Furthermore, data augmentation techniques were utilized to address data scarcity. Through this comparative analysis, the study aimed to elucidate the strengths and limitations of discriminative fusion versus vision–language generation approaches, and to provide empirical insights into effective Med-VQA system design under limited-data conditions.

OBJECTIVE

The objectives of this study were as follows:

1. To evaluate the impact of data augmentation on Med-VQA performance under limited-data conditions using the VQA-RAD dataset
2. To compare the performance of different multimodal fusion mechanisms within classical CNN–LSTM-based discriminative models for medical visual question answering using the VQA-RAD dataset
3. To compare the performance between discriminative classification-based models and generative vision–language models in handling closed-ended and open-ended Med-VQA tasks

METHODS

Dataset Description

This study utilized the VQA-RAD dataset, a publicly available benchmark specifically designed for Medical Visual Question Answering tasks (Lau et al., 2018). We loaded it via https://huggingface.co/datasets/flaviagiammarino/vqa-rad. The dataset consists of 2,244 question–answer pairs associated with radiology images, including X-ray, CT, and MRI modalities. Each image was paired with one or more clinically generated questions and corresponding ground-truth answers, reflecting realistic diagnostic and educational scenarios encountered in clinical practice. A sample of data is shown in Figure 1; in this sample, a chest X-ray image was paired with the question of "What type of image is this?", while the ground-truth answer is "X-ray".

```
sample = train_split[78]
print('Question:', sample['question'])
plt.imshow(sample['image'].convert('RGB'))
print('Answer:', sample['answer'])
```

```
Question: what type of image is this?
Answer: x-ray
```

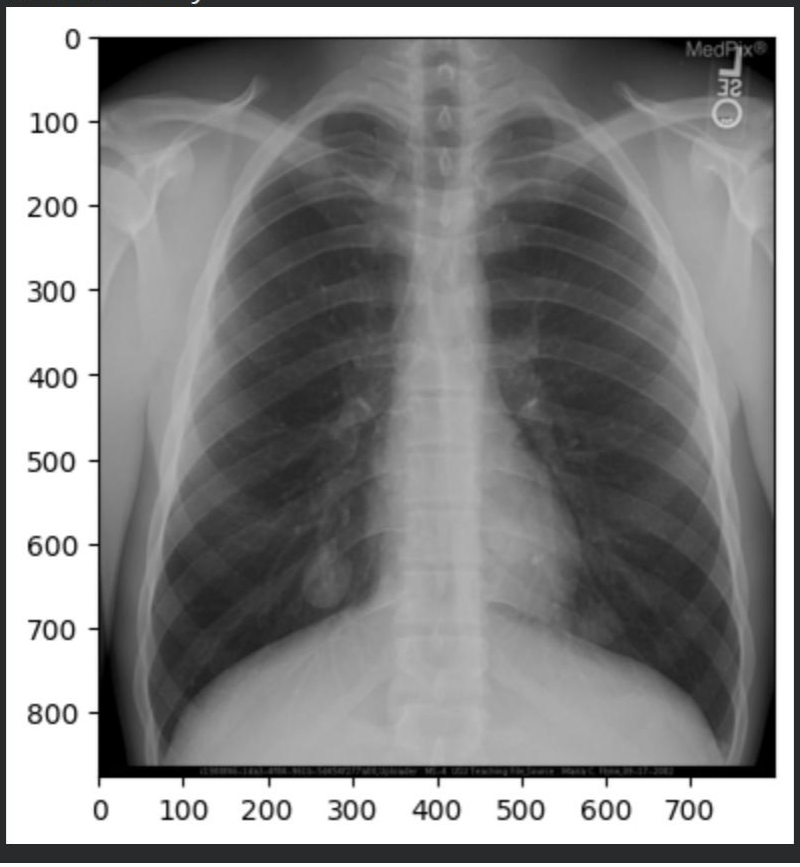

Figure 1: Example sample from the VQA-RAD dataset

The questions in VQA-RAD were categorized into closed-ended and open-ended types. Closed-ended questions typically required binary responses (e.g., yes or no), while open-ended questions demanded more descriptive or explanatory answers related to anatomical structures, abnormalities, or imaging findings. This mixed-question format caused VQA-RAD to be a challenging benchmark for evaluating both classification-based and generative Med-VQA models. One sample from each categories are provided in Table 1.

Table 1: Examples of question types in the VQA-RAD dataset

Question Type	**Example**	**Answer**
Closed-ended	are there >5 lymph nodes located near the stomach?	yes
Open-ended	what is meant by tram-track?	thickening of bronchi

Furthermore, based on the answer distribution shown in Figure 2, it was evident that the majority of highly frequent responses in the VQA-RAD dataset were dominated by binary answers such as “yes” and “no”, while the remaining answers were more diverse and descriptive in nature. This clear separation supports a simple yet effective rule for categorizing question types, where answers belonging to the set {“yes”, “no”} were treated as closed-ended, and all other responses were considered open-ended. Given that single-word binary answers overwhelmingly correspond to closed-ended questions, and longer, multi-word answers correspond to explanatory or descriptive queries, this heuristic was sufficient for reliably partitioning the dataset into closed-ended and open-ended subsets for evaluation purposes in this study. As a result, the testing data contained 251 closed-ended and 200 open-ended questions.

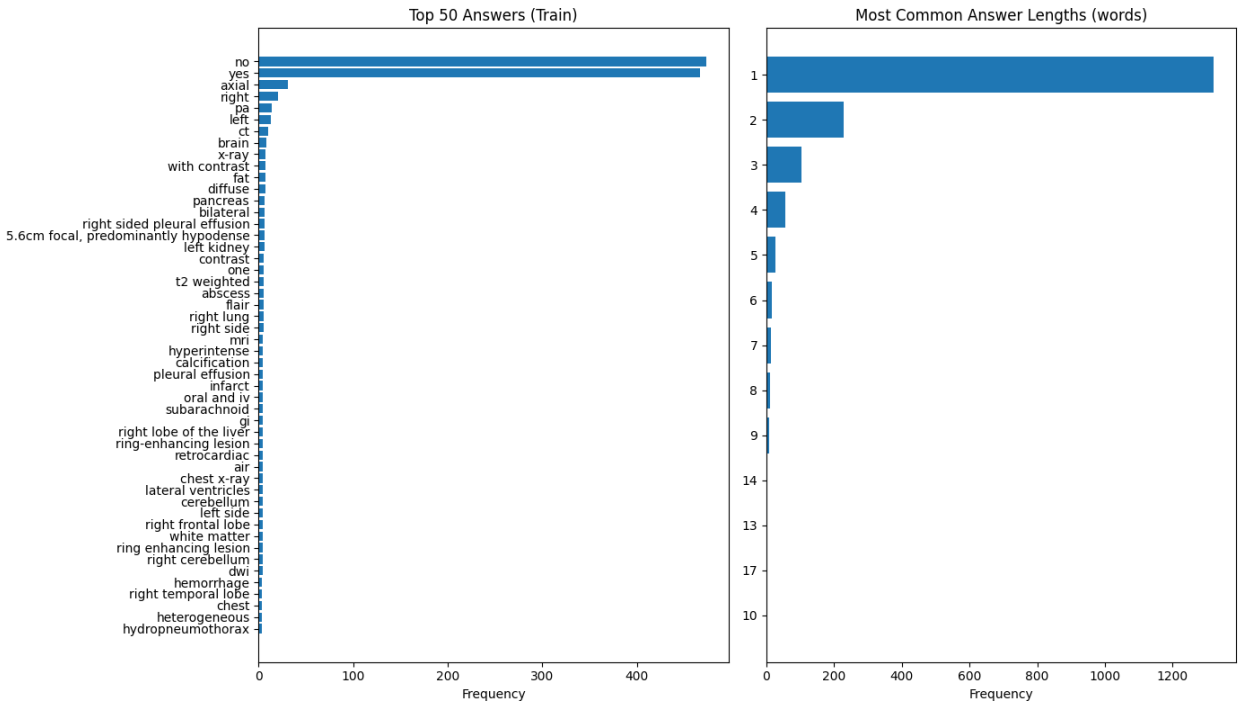

Figure 2: Answer distribution in the VQA-RAD training set

In the original split provided by the dataset authors, there were 1,793 training samples and 451 testing samples. Due to the relatively small dataset size, VQA-RAD presents inherent challenges for model generalization and robustness, motivating the exploration of data augmentation techniques and transfer learning strategies in this study.

Data Preprocessing

Before model training, a series of data preprocessing steps was applied to ensure that the inputs were compatible with the respective model architectures. As this study investigated two fundamentally different Med-VQA pipelines, namely, classical discriminative models based on CNN–LSTM architectures with explicit multimodal fusion, and a generative VLM, their preprocessing requirements were inherently different. In particular, the classical models relied on fixed-size tensor inputs and explicit vocabulary construction, whereas the VLM operated on

processor-managed image–text inputs and generated answers autoregressively. Consequently, image preprocessing for the classical models included normalization aligned with the CNN's pretraining regime, while the VLM employed in this study handled normalization internally and operated directly on PIL images. Similarly, text preprocessing for the classical models involved manual tokenization and vocabulary construction for question encoding and answer classification, whereas our VLM leveraged its pretrained tokenizer without requiring an explicit vocabulary definition. These differences motivated the use of separate but conceptually aligned preprocessing pipelines, which are detailed in the following sections.

(i) Image Preprocessing

A summary of the image transformations employed in this study is illustrated in Table 2.

Table 2: Image Transformation Summary

Pipeline	**Resize (224x224)**	**ToTensor**	**Normalize (ImageNet)**	**RandomResizeCrop**	**RandomRotation**
Classical	✓	✓	✓		
Classical with data augmentation		✓	✓	✓	✓
BLIP	✓				
BLIP with data augmentation				✓	✓

“Resize (224×224)” would force all images to a uniform spatial resolution, ensuring batch consistency and matching the expected input size of both the CNN-based models and the VLM. “ToTensor” would convert a PIL image into a PyTorch tensor with shape (C, H, W) and scale pixel values to the [0, 1] range, enabling numerical processing by convolutional networks. “Normalize (ImageNet)” would standardize each channel using the ImageNet mean and standard deviation, which is crucial for the classical CNN-based models because the ResNet encoder was pretrained on ImageNet and therefore assumes inputs with this distribution; applying the same normalization would help preserve the validity of the pretrained features. Notably, the BLIP preprocessing pipeline does not include explicit “ToTensor” or “Normalize” operations. Instead, BLIP operated directly on resized PIL images, with all tensor conversion and normalization handled internally by the BLIP processor. This design choice reflects the end-to-end nature of VLMs, where image and text preprocessing is tightly coupled to the pretrained model and abstracted away from the user, in contrast to classical pipelines where such steps must be explicitly defined.

As previously discussed, data augmentation was incorporated in this study to mitigate overfitting arising from the limited size of the VQA-RAD dataset. Two augmentation strategies were employed. Firstly, "RandomResizedCrop" would perform a random spatial crop followed by resizing to 224×224; introducing scale and translation variability while preserving the required input resolution. Secondly, "RandomRotation" was used to apply a random in-plane rotation within a small range (±10° in this study), introducing slight viewpoint variation that can improve generalization to slight differences in image acquisition or patient positioning. Notably, horizontal or vertical flipping was deliberately excluded. In medical imaging, left–right or up–down orientation often carries semantic meaning, and flipping could therefore

produce anatomically implausible samples and introduce label noise. Similarly, the rotation range was kept small to avoid generating unrealistic views that could distort clinically meaningful structures. Together, these augmentation choices aim to balance increased data diversity with preservation of medical validity.

Each model was therefore trained and evaluated under two separate image preprocessing pipelines (one without augmentation and one with augmentation) to explicitly assess the impact of data augmentation on model performance. The code snippet for these pipelines is shown in Figure 3.

```python
# Base transform (no augmentation)
image_transform_base = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std =[0.229, 0.224, 0.225]
    )
])

# Aug transform
image_transform_aug = transforms.Compose([
    transforms.RandomResizedCrop(224, scale=(0.90, 1.0)),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std =[0.229, 0.224, 0.225]
    )
])

# For BLIP
blip_pil_base = transforms.Compose([
    transforms.Resize((224, 224)),
])

blip_pil_aug = transforms.Compose([
    transforms.RandomResizedCrop(224, scale=(0.90, 1.0)),
    transforms.RandomRotation(10),
])
```

Figure 3: Image preprocessing and data augmentation pipelines

(ii) Text Preprocessing

Text preprocessing was different for the classical discriminative models and the generative VLM, reflecting their distinct task formulations.

For the classical CNN–LSTM–based discriminative models, text preprocessing followed a common natural language processing pipeline. All questions were first

lowercased and tokenized using a regular-expression–based tokenizer that extracts alphanumeric tokens. A vocabulary was then constructed from the training questions only, mapping each token to a unique integer index (stoi), with a corresponding inverse mapping (itos). Two special tokens were included: <pad> for sequence padding and <unk> for unseen or rare tokens. Each question was encoded as a fixed-length sequence of token indices (e.g., maximum length of 20), where longer questions were truncated and shorter ones were padded. Since these models formulate Med-VQA as a classification task, answers were also preprocessed by constructing an answer set and mapping each unique answer string to a class index. Model training was then performed using a cross-entropy loss over the resulting multi-class output space. The relevant code snippet is displayed in Figure 4.

Text Preprocessing (tokenize, vocab, encode)

```
def tokenize(text: str):
    text = text.lower().strip()
    return re.findall(r"[a-z0-9]+", text)

def build_vocab(questions, min_freq=1):
    counter = Counter()
    for q in questions:
        counter.update(tokenize(q))

    stoi = {"<pad>": 0, "<unk>": 1}
    for w, f in counter.items():
        if f >= min_freq and w not in stoi:
            stoi[w] = len(stoi)

    itos = {i: s for s, i in stoi.items()}
    return stoi, itos

def encode_question(q: str, stoi, max_len=20):
    ids = [stoi.get(t, stoi["<unk>"]) for t in tokenize(q)][:max_len]
    ids += [stoi["<pad>"]] * (max_len - len(ids))
    return torch.tensor(ids, dtype=torch.long)

train_questions = [s["question"] for s in train_split]
stoi, itos = build_vocab(train_questions, min_freq=1)
vocab_size = len(stoi)
```

Answer Space (Classification labels)

```
all_answers = [s["answer"] for s in train_split] + [s["answer"] for s in test_split]
unique_answers = sorted(set(all_answers))

answer2idx = {a: i for i, a in enumerate(unique_answers)}
idx2answer = {i: a for a, i in answer2idx.items()}
```

Figure 4: Code snippet illustrating the classical text preprocessing pipeline used for CNN–LSTM–based discriminative models

In contrast, VLM would rely on its pretrained tokenizer and does not require manual vocabulary construction. In our case, we chose BLIP as the VLM. Questions were tokenized using the BLIP processor, which produces input_ids and attention_mask tensors, with padding and truncation applied to a fixed maximum length. Ground-truth answers were tokenized separately as decoder targets, also with padding and truncation. To ensure correct

loss computation during training, padding token IDs in the answer sequences were replaced with -100 so that they are ignored by the sequence-to-sequence loss function. At inference time, BLIP generated answers autoregressively using a decoding procedure (model.generate), and the resulting token sequences were decoded back into natural-language answers rather than mapped to predefined classes. The relevant code snippet is shown in Figure 5; the important part is encapsulated within the green rectangle.

```python
class VQARadBLIPDataset(Dataset):
    def __init__(self, split, processor, pil_transform=None, max_q_len=32, max_a_len=16):
        self.ds = split
        self.processor = processor
        self.pil_transform = pil_transform
        self.max_q_len = max_q_len
        self.max_a_len = max_a_len

    def __len__(self):
        return len(self.ds)

    def __getitem__(self, idx):
        s = self.ds[idx]
        image = s["image"].convert("RGB")
        if self.pil_transform is not None:
            image = self.pil_transform(image)

        question = s["question"]
        answer = s["answer"]

        enc = self.processor(
            images=image,
            text=question,
            padding="max_length",
            truncation=True,
            max_length=self.max_q_len,
            return_tensors="pt"
        )

        tok = self.processor.tokenizer(
            answer,
            padding="max_length",
            truncation=True,
            max_length=self.max_a_len,
            return_tensors="pt"
        )

        decoder_input_ids = tok.input_ids  # keep PAD ids here (valid token ids)
        labels = decoder_input_ids.clone()
        pad_id = self.processor.tokenizer.pad_token_id
        labels[labels == pad_id] = -100     # mask PAD for loss only

        enc = {k: v.squeeze(0) for k, v in enc.items()}
        enc["decoder_input_ids"] = decoder_input_ids.squeeze(0)
        enc["labels"] = labels.squeeze(0)
        enc["gt_answer"] = answer
        return enc
```

Figure 5: Code snippet illustrating the text preprocessing pipeline used for BLIP models

Model Architecture

This study compared two paradigms on VQA-RAD, which are classical discriminative CNN–LSTM models that predict an answer class from a fixed answer set, and a pretrained VLM that produces answers as text.

(i) Classical Discriminative Pipeline (CNN–LSTM + Fusion + Classifier)

Within this category, all models shared a common backbone consisting of an image encoder, a question encoder, a multimodal fusion module, and a classifier. Images were processed using a ResNet-50 backbone pretrained on ImageNet, with the final pooling and classification layers removed. The encoder produced both a global image representation via adaptive average pooling and a spatial feature map, enabling different fusion strategies. Questions were encoded using a word embedding layer followed by a unidirectional LSTM, where the final hidden state represented the question semantics in a fixed-dimensional vector space. Batch size was fixed as 16. The relevant code snippet is as illustrated in Figure 6.

```
class ResNetEncoder(nn.Module):
    def __init__(self):
        super().__init__()
        backbone = models.resnet50(weights=models.ResNet50_Weights.IMAGENET1K_V1)
        # feature map before avgpool/fc: (B, 2048, 7, 7)
        self.features = nn.Sequential(*list(backbone.children())[:-2])
        self.pool = nn.AdaptiveAvgPool2d((1, 1))
        self.out_dim = 2048

    def forward(self, x):
        fmap = self.features(x)                # (B,C,H,W)
        vec = self.pool(fmap).flatten(1)       # (B,C)
        return vec, fmap

class LSTMEncoder(nn.Module):
    def __init__(self, vocab_size, embed_dim=300, hidden=512):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=0)
        self.lstm = nn.LSTM(embed_dim, hidden, batch_first=True)
        self.out_dim = hidden

    def forward(self, q_ids):
        emb = self.embedding(q_ids)
        _, (h, _) = self.lstm(emb)
        return h[-1]  # (B, hidden)
```

Figure 6: Image and text encoder architectures used in the classical models

Three fusion mechanisms were explored within this framework. The first model (relevant code is shown in Figure 7) employed simple feature concatenation, in which the global image vector and question vector are concatenated and passed through a multilayer perceptron (MLP) to predict an answer class. The second model (relevant code is shown in Figure 8) replaced concatenation with Multimodal Compact Bilinear (MCB) pooling, which approximated bilinear interactions between image and text features using CountSketch projections and Fourier-domain multiplication, followed by normalization and classification. The third model (relevant code is shown in Figure 9) adopted a stronger fusion strategy based on cross-attention: image feature maps were flattened into region-level tokens, while the LSTM outputs token-level question representations. A Transformer-style multi-head cross-attention module allowed question tokens to attend to image regions, producing a question representation that was explicitly conditioned on relevant visual content. The attended question tokens were pooled and fed into an MLP classifier. All classical models formulated Med-VQA as a multi-class classification task over a fixed answer vocabulary and are trained using cross-entropy loss. Additionally, each architecture was evaluated under two training configurations: one using the base image preprocessing pipeline without data augmentation, and another incorporating data augmentation.

```python
class VQA_Concat(nn.Module):
    def __init__(self, img_enc, txt_enc, num_answers):
        super().__init__()
        self.img_enc = img_enc
        self.txt_enc = txt_enc
        fusion_dim = img_enc.out_dim + txt_enc.out_dim
        self.classifier = nn.Sequential(
            nn.Linear(fusion_dim, 512),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(512, num_answers)
        )

    def forward(self, images, q_ids):
        img_vec, _ = self.img_enc(images)
        q_vec = self.txt_enc(q_ids)
        fused = torch.cat([img_vec, q_vec], dim=1)
        return self.classifier(fused)
```

Figure 7: Concatenation-based multimodal fusion architecture

```python
class CountSketch(nn.Module):
    def __init__(self, in_dim, out_dim, seed=123):
        super().__init__()
        g = torch.Generator()
        g.manual_seed(seed)

        h = torch.randint(low=0, high=out_dim, size=(in_dim,), generator=g)
        s = torch.randint(low=0, high=2, size=(in_dim,), generator=g) * 2 - 1  # +/-1

        self.register_buffer("h", h)
        self.register_buffer("s", s)
        self.in_dim = in_dim
        self.out_dim = out_dim

    def forward(self, x):
        # x: (B, in_dim)
        B, D = x.shape
        out = x.new_zeros((B, self.out_dim))
        # add signed values into hashed bins
        out.scatter_add_(1, self.h.unsqueeze(0).expand(B, -1), x * self.s.unsqueeze(0))
        return out

class MCBFusion(nn.Module):
    def __init__(self, img_dim, txt_dim, mcb_dim=8000):
        super().__init__()
        self.cs_img = CountSketch(img_dim, mcb_dim, seed=1)
        self.cs_txt = CountSketch(txt_dim, mcb_dim, seed=2)
        self.mcb_dim = mcb_dim

    def forward(self, img_vec, txt_vec):
        # CountSketch
        a = self.cs_img(img_vec)  # (B,m)
        b = self.cs_txt(txt_vec)  # (B,m)

        # FFT -> multiply -> iFFT
        fa = torch.fft.rfft(a, dim=1)
        fb = torch.fft.rfft(b, dim=1)
        f = fa * fb
        out = torch.fft.irfft(f, n=self.mcb_dim, dim=1)

        # signed sqrt + l2 norm (common in bilinear pooling)
        out = torch.sign(out) * torch.sqrt(torch.abs(out) + 1e-8)
        out = F.normalize(out, dim=1)
        return out

class VQA_MCB(nn.Module):
    def __init__(self, img_enc, txt_enc, num_answers, mcb_dim=8000):
        super().__init__()
        self.img_enc = img_enc
        self.txt_enc = txt_enc
        self.mcb = MCBFusion(img_enc.out_dim, txt_enc.out_dim, mcb_dim=mcb_dim)
        self.classifier = nn.Sequential(
            nn.Linear(mcb_dim, 512),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(512, num_answers)
        )

    def forward(self, images, q_ids):
        img_vec, _ = self.img_enc(images)
        q_vec = self.txt_enc(q_ids)
        fused = self.mcb(img_vec, q_vec)
        return self.classifier(fused)
```

Figure 8: MCB fusion architecture

```python
class VQA_CrossAttention(nn.Module):
    def __init__(self, img_enc, vocab_size, num_answers,
                 embed_dim=300, lstm_hidden=512, max_q_len=20,
                 d_model=512, n_heads=8, dropout=0.1):
        super().__init__()
        self.img_enc = img_enc
        self.max_q_len = max_q_len

        # ---- Question token encoder (keeps token outputs) ----
        self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=0)
        self.lstm = nn.LSTM(embed_dim, lstm_hidden, batch_first=True, bidirectional=False)

        # Project both modalities into the same d_model
        self.q_proj = nn.Linear(lstm_hidden, d_model)

        # CNN feature map is (B, C=2048, 7, 7) -> tokens (B, 49, 2048)
        self.img_proj = nn.Linear(img_enc.out_dim, d_model)

        # ---- Cross-attention block (Transformer-style) ----
        self.cross_attn = nn.MultiheadAttention(embed_dim=d_model, num_heads=n_heads,
                                                dropout=dropout, batch_first=True)

        self.ln1 = nn.LayerNorm(d_model)
        self.ff = nn.Sequential(
            nn.Linear(d_model, 4 * d_model),
            nn.GELU(),
            nn.Dropout(dropout),
            nn.Linear(4 * d_model, d_model),
            nn.Dropout(dropout),
        )
        self.ln2 = nn.LayerNorm(d_model)

        # Classifier on pooled question representation
        self.classifier = nn.Sequential(
            nn.Linear(d_model, 512),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(512, num_answers),
        )

    def forward(self, images, q_ids):
        # ---- Image tokens ----
        _, fmap = self.img_enc(images)                  # (B, 2048, 7, 7)
        B, C, H, W = fmap.shape
        img_tokens = fmap.flatten(2).transpose(1, 2)    # (B, 49, 2048)
        img_tokens = self.img_proj(img_tokens)          # (B, 49, d_model)

        # ---- Question tokens ----
        q_emb = self.embedding(q_ids)                   # (B, T, embed_dim)
        q_out, _ = self.lstm(q_emb)                     # (B, T, lstm_hidden)
        q_tokens = self.q_proj(q_out)                   # (B, T, d_model)

        # Padding mask for questions (True means "ignore")
        q_pad_mask = (q_ids == 0)                       # (B, T)

        # ---- Cross-attention: Q attends to image ----
        # Query = q_tokens, Key/Value = img_tokens
        attn_out, _ = self.cross_attn(
            query=q_tokens, key=img_tokens, value=img_tokens,
            key_padding_mask=None,          # image tokens no padding
            attn_mask=None
        )
        x = self.ln1(q_tokens + attn_out)

        # ---- Feedforward (Transformer) ----
        x = self.ln2(x + self.ff(x))

        # ---- Pool (masked mean over question tokens) ----
        mask = (~q_pad_mask).float().unsqueeze(-1)      # (B, T, 1)
        x_sum = (x * mask).sum(dim=1)                   # (B, d_model)
        denom = mask.sum(dim=1).clamp(min=1.0)          # (B, 1)
        pooled = x_sum / denom

        return self.classifier(pooled)
```

Figure 9: Cross-attention–based multimodal fusion architecture

(ii) VLM

In this study, the VLM is implemented using BLIP. BLIP adopted an end-to-end encoder–decoder architecture that would jointly model visual and textual inputs and generate answers autoregressively as text. Images and questions were processed using a pretrained BLIP processor, which would handle tokenization, padding, and alignment internally, eliminating the need for manual vocabulary construction. During training, ground-truth answers were tokenized as decoder targets, and padding tokens were masked in the loss to enable correct sequence-to-sequence learning. At inference time, BLIP would produce answers via text generation rather than class prediction, allowing it to naturally handle both closed-ended and open-ended questions. This architectural contrast highlights the difference between discriminative fusion-based Med-VQA models and modern generative vision–language approaches. Notably, BLIP architecture was evaluated under two training configurations: one using the image preprocessing pipeline without data augmentation, and another incorporating data augmentation. Meanwhile, batch size was fixed as 4.

(iii) Summary

A summary of the models constructed in this study is shown in Table 3.

Table 3: Summary of Models

Model	Architecture Type	Image Encoder	Text Encoder	Fusion/Interaction Mechanism	Task Formulation	Data Augmentation
1	Classical	ResNet-50 (ImageNet pretrained)	LSTM	Feature Concatenation	Classification	
2	Classical	ResNet-50 (ImageNet pretrained)	LSTM	Feature Concatenation	Classification	✓
3	Classical	ResNet-50 (ImageNet pretrained)	LSTM	MCB Pooling	Classification	
4	Classical	ResNet-50 (ImageNet pretrained)	LSTM	MCB Pooling	Classification	✓
5	Classical	ResNet-50 (ImageNet pretrained)	LSTM	Cross-Attention	Classification	

6	Classical	ResNet-50 (ImageNet pretrained)	LSTM	Cross-Attention	Classification	✓
7	VLM	BLIP visual encoder	BLIP text encoder	Joint Vision–Language Modeling	Generation	
8	VLM	BLIP visual encoder	BLIP text encoder	Joint Vision–Language Modeling	Generation	✓

Model Evaluation

The performance of all constructed models was evaluated separately for closed-ended and open-ended questions in order to account for their fundamentally different answer characteristics.

For closed-ended questions, model performance was assessed using classification accuracy. A prediction was considered correct if the generated or predicted answer exactly matched the ground-truth answer after normalization (lowercasing and removal of punctuation). Accuracy was computed as the proportion of correctly answered closed-ended questions over the total number of closed-ended samples in the test set.

For open-ended questions, performance was evaluated using the BLEU-1 score, which measures unigram overlap between the predicted answer and the reference answer (Lau et al., 2018). BLEU-1 was chosen due to the typically short and concise nature of medical answers in VQA-RAD, where higher-order n-gram statistics are often less informative. To ensure fair comparison, both predicted and ground-truth answers were normalized prior to scoring.

For generative models such as BLIP, answers were produced using autoregressive decoding and then evaluated using the same closed/open criteria as discriminative models. This unified evaluation protocol allowed for a direct comparison between classification-based and generation-based Med-VQA approaches despite differences in task formulation.

Overall performance was reported in terms of closed-ended accuracy, open-ended BLEU-1 score, providing a comprehensive and interpretable assessment of model behavior across question types.

The relevant code snippet is illustrated in Figure 10. Notably, the driver code responsible for model training, inference, and evaluation is omitted here due to its length and will be included in the GitHub repository, where the link will be provided in the final report.

```python
def normalize_text(s: str) -> str:
    s = s.lower().strip()
    s = re.sub(r"[^a-z0-9 ]", "", s)
    s = re.sub(r"\s+", " ", s)
    return s

def bleu1(reference: str, hypothesis: str) -> float:
    ref = normalize_text(reference).split()
    hyp = normalize_text(hypothesis).split()
    if len(hyp) == 0:
        return 0.0

    ref_cnt = Counter(ref)
    hyp_cnt = Counter(hyp)

    matches = sum(min(hyp_cnt[w], ref_cnt.get(w, 0)) for w in hyp_cnt)
    precision = matches / len(hyp)

    r, c = len(ref), len(hyp)
    bp = 1.0 if c >= r else float(np.exp(1 - r / max(c, 1)))
    return bp * precision

def eval_closed_open(gt_answers, pred_answers):
    closed_correct = 0
    closed_total = 0
    open_bleu_sum = 0.0
    open_total = 0

    for gt, pr in zip(gt_answers, pred_answers):
        if is_closed_ended(gt):
            closed_total += 1
            if normalize_text(gt) == normalize_text(pr):
                closed_correct += 1
        else:
            open_total += 1
            open_bleu_sum += bleu1(gt, pr)

    return {
        "closed_acc": closed_correct / max(closed_total, 1),
        "open_bleu1": open_bleu_sum / max(open_total, 1),
        "closed_n": closed_total,
        "open_n": open_total
    }
```

Figure 10: Evaluation Protocol

PRELIMINARY RESULTS

Table 4 shows a summary of performance of the constructed models.

Table 4: Summary of Results

Model	Accuracy (Closed-ended)	BLEU-1 (Open-ended)
1	0.59	0.01
2	0.61	0.01
3	0.61	0.04
4	0.58	0.08
5	0.65	0.13
6	0.65	0.15
7	0.56	0.14
8	0.67	0.18

The results revealed clear performance trends across different model formulations, fusion mechanisms, and task paradigms. Among the classical CNN–LSTM-based models, progressively stronger fusion strategies consistently improved performance. Simple feature concatenation yielded moderate closed-ended accuracy and very limited open-ended performance, indicating that naive fusion struggled to capture fine-grained multimodal interactions. MCB pooling improved open-ended BLEU-1 scores, suggesting better interaction modeling between image and question features, although gains in closed-ended accuracy remained modest and sensitive to data augmentation.

The cross-attention based model achieved the best overall performance among discriminative architectures, attaining the highest closed-ended accuracy and substantially higher BLEU-1 scores

compared to concatenation and MCB. This highlights the importance of region-level visual–textual alignment, particularly for open-ended questions that require more detailed semantic reasoning. Data augmentation provided marginal but consistent improvements for the cross-attention model, especially for open-ended answers, indicating improved robustness under limited-data conditions.

In contrast, the generative BLIP model demonstrated a different performance profile. Without augmentation, BLIP underperformed classical models on closed-ended accuracy but achieves competitive open-ended BLEU-1 scores, reflecting its strength in free-form answer generation rather than fixed-label prediction. Notably, when combined with data augmentation, BLIP achieved the highest overall performance on both closed-ended and open-ended questions, surpassing all discriminative models. This suggested that the pretrained VLM had effectively leveraged augmented visual diversity, even when fine-tuned on small medical datasets.

Overall, these findings indicated that while advanced fusion mechanisms significantly enhanced discriminative Med-VQA models, generative VLM, when paired with appropriate data augmentation, offered superior flexibility and performance for mixed closed- and open-ended question settings. This supported the shift from purely discriminative fusion-based architectures toward generative vision–language formulations for medical visual question answering under limited-data regimes.

AUTHORS CONTRIBUTION

Student Name	Student ID	Contribution
Ooi Li Yoong	24226057	Data preprocessing, model, and report
Yan ZhenYang	23090967	Data collection, model, report, and slide

REFERENCES

1. Gong, H., Chen, G., Liu, S., Yu, Y., & Li, G. (2021). Cross-modal self-attention with multi-task pre-training for medical visual question answering. In Proceedings of the 2021 international conference on multimedia retrieval (pp. 456–460).
2. Lau, J. J., Gayen, S., Ben Abacha, A., & Demner-Fushman, D. (2018). A dataset of clinically generated visual questions and answers about radiology images. Scientific data, 5(1), 1–10.
3. Li, Y., Man, J., Zhou, Y., & Liang, L. (2025). Caption-augmented reasoning model with hierarchical rank lora finetuing for medical visual question answering. Journal of Biomedical Informatics, 104964.
4. Li, Y., Yang, Z., Lee, L.-K., Wang, F. L., Qu, Y., & Hao, T. (2025). Inference enhanced model with answer refinement for medical visual question answering. Multimedia Systems, 31(6), 422.
5. Liu, L., Su, X., Guo, H., & Zhu, D. (2022). A transformer-based medical visual question answering model. In 2022 26th international conference on pattern recognition (icpr) (pp. 1712–1718).
6. Lu, Z., Zeng, Q., Lu, M., Chen, G., & Xia, Y. (2025). Bridging the semantic gap in medical visual question answering with prompt learning. IEEE Transactions on Medical Imaging.