Preliminary report: Advanced Machine Learning Preliminary Report.docx (may make a copy of it, and then start with the report)
Github: https://github.com/liyoongooi/AML/tree/main/Final

Saved files:
https://drive.google.com/drive/folders/1Cj7HEMM8hOpPyYXO7LSdI_E99II8RQ6C?usp=sharing

Dataset

```
ds = load_dataset("flaviagiammarino/vqa-rad")
train_split = ds["train"]
test_split  = ds["test"]

print("Train:", len(train_split), "Test:", len(test_split))
print("Keys:", train_split[0].keys())
```

Train: 1793 samples Test: 451 samples

Closed vs open split rule

```
CLOSED_SET = {"yes", "no"}
def is_closed_ended(answer: str):
    return answer.lower() in CLOSED_SET

# quick counts (train)
closed_n = sum(is_closed_ended(s["answer"]) for s in train_split)
open_n = len(train_split) - closed_n
print("Train closed:", closed_n, "Train open:", open_n)
```

```
Train closed: 940 Train open: 853
```

- We only consider “yes/no” answer as closed set; otherwise, all open-ended

Image Preprocessing

Pipeline	**Resize (224x224)**	**ToTensor**	**Normalize (ImageNet)**	**RandomResizeCrop**	**RandomRotation**
Classical	✓	✓	✓		

Classical with data augmentation		✓	✓	✓	✓
BLIP	✓				
BLIP with data augmentation				✓	✓

Text preprocessing (for classification)

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()}
num_answers = len(answer2idx)
```

- Basically, we are converting each answer to a numerical label so that classifiers can predict
- Notably, there is actually a test-set leakage problem because without including test set answers during the encoding, some lables would not exist and classification will have errors. I did not fix this, because classification just serve as our baseline, and we will focus more on VLM

Text Preprocessing for VLM (BLIP)

- Questions were tokenized using the BLIP processor, which produces input_ids and attention_mask tensors
- Ground-truth answers were tokenized separately as decoder targets

Model Construction

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	✓

Note: All epochs are 10 with learning rate = 5e^-5

Evaluation Metrics

Closed-ended: accuracy

Open-ended: BLEU-1 with critical attribute match

```python
# Critical attributes
ABBR = {"lt": "left", "rt": "right", "l": "left", "r": "right"}
LATERALITY = {"left", "right", "bilateral"}
NEGATION   = {"no", "not", "absent", "negative", "none", "without"}

def norm_tokens(s: str):
    toks = normalize_text(s).split()
    toks = [ABBR.get(t, t) for t in toks]
    return toks

def extract_critical(tokens):
    tset = set(tokens)
    return {
        "laterality": tset & LATERALITY,
        "negation":   tset & NEGATION,
    }

def critical_attribute_match(gt: str, pr: str) -> bool:
    gt_t = norm_tokens(gt)
    pr_t = norm_tokens(pr)
    gt_c = extract_critical(gt_t)
    pr_c = extract_critical(pr_t)

    # Enforce only if GT contains that critical attribute
    if gt_c["laterality"] and (gt_c["laterality"] != pr_c["laterality"]):
        return False
    if gt_c["negation"] and (gt_c["negation"] != pr_c["negation"]):
        return False
    return True

def bleu1_critical_gated(gt: str, pr: str) -> float:
    return bleu1(gt, pr) if critical_attribute_match(gt, pr) else 0.0
```

- For example, if the answer contains left/right as laterality, then the direction must be correct, or else, 0 mark will be awarded

Result

Model	Accuracy	BLEU-1 with critical attribute match
1	0.649	0.046
2	0.610	0.064
3	0.625	0.042
4	0.506	0.035
5	0.625	0.064
6	0.602	0.109
7	0.657	0.198
8	0.666	0.207

- Best model is 8 (aug + BLIP)

Fine-tuning with LoRA

Epoch: 8

Learning rate: 3e^-5

Result:

- Accuracy: 0.673 (improved from 0.666)
- BLEU-1 with critical attribute match: 0.214 (improved from 0.207)