Multimodal Visual Question Answering with Amazon Berkeley Objects Dataset

Github Link: https://github.com/truptikhodwe/Visual-Question-Answering

Project Idea and Problem Statement:

This Project involves creating a multiple-choice Visual Question Answering (VQA) dataset using the Amazon Berkeley Objects (ABO) dataset, evaluating baseline models, fine-tuning using Low-Rank Adaptation (LoRA), and assessing performance using standard metrics. At the end of the project, the evaluator or the user on giving some image and some question as the input, the model will give an answer for the same after analyzing.

Methodology:

We completed the project by following the steps in below order:

- Curation of data
    - In the amazon berkeley dataset, we have images. But our task is VQA, so we want to generate a dataset such that for each image we also have questions and answers, so that the model can be then trained on this new generated dataset.
- Data pre-processing and cleaning, do some EDA
- Training the datasets using the existing models for baseline evaluation.
- Fine tuning the models using LoRA.
- Comparing the results of the models using different evaluation metrics.

Data Curation:

We noticed the Dataset had roughly 4,00,000 entries. We decided to divide the dataset into 6 parts from where each team member worked on 2 of the parts of the dataset. We had initially decided for each member to process 20,000 images with 3 questions of varying difficulty per image. But, due to the constraints in the training compute and the model size, we finally went ahead and followed these steps for the curation of the dataset.

- We decided to reduce it to a total of 20,000 images with 3 questions per image across all 3 members.
- To generate the question-answer pairs we tried experimenting with multiple models. We initially tried to work with Mistral AI, though it generated really good questions and answers, the request rate limit was quite less. So, we decided to switch to gemini. We decided not to use offline models being run locally due to laptop GPU constraints. The response time of the gemini api was much faster. (Links for all the apis have been added at the end)
- We leveraged the listings.tar file: which mapped the metadata of the product to the image id.

- We also used images.csv from the metadata folder; present within abo-small to map the image-id to the relative path of the image. We used this metadata along with the image as input to form the questions.
- The questions are of easy, medium and hard difficulty level categorized accordingly within the generated dataset.
- The prompt used was as follows :
    - "You are a Visual QA dataset assistant: for each image, generate exactly three question-answer pairs, with a single word as the answer - in Easy (basic visual attributes), Medium (advanced visual features), and Hard (logical inferences) difficulty - covering diverse types, answerable solely from the image. Format: Question: ..., Answer: ...\n\n”
      f"Product metadata:\n{metadata_str}".
- The code for the same can be found on github.

Data Cleaning and Preprocessing:

The output retrieved from the gemini api was cleaned and preprocessed to the best of our abilities.

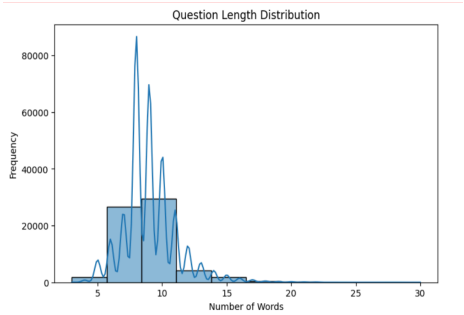
- Characters which frequent the response output from gemini but are not important to the answer are removed.
- Removed asterisks which are used for markdown
- Remove excessive newlines
- Collapse all white space
- Remove non-alphanum from the end
- Convert everything to lowercase
- Ensuring no missing/null values
- Ensuring label consistency for difficulty (only allowed values: Easy/Medium/Hard)

This ensures the dataset is clean, consistent, and ready for training or evaluation in downstream tasks. Thereafter it is written to the dataset CSV and used.

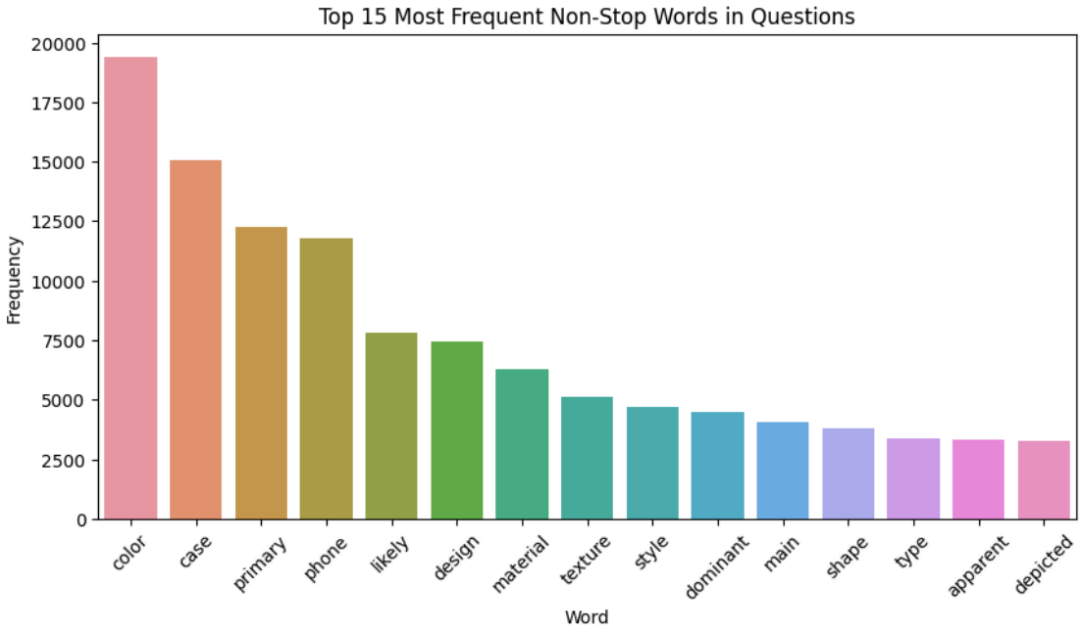
Exploratory Data Analysis:

Plotted some graphs which helped in the cleaning and preprocessing steps.

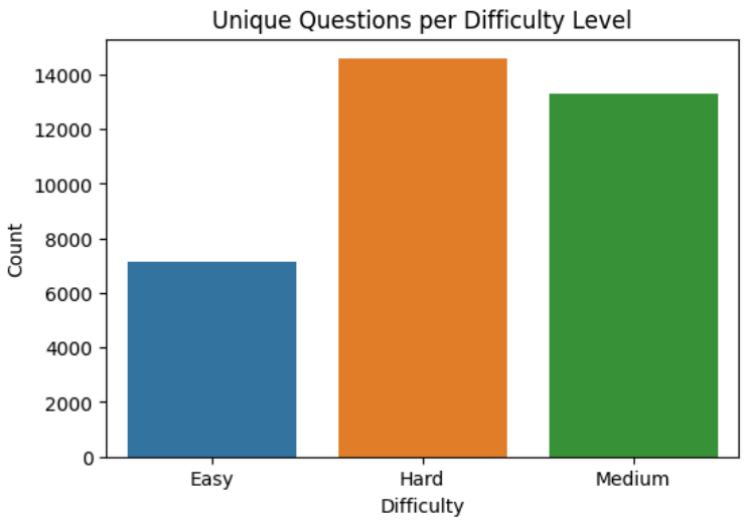
1. Question Length Distribution:
    a. Shows the complexity or verbosity of the questions
    b. Why it matters: Short questions may be easy to parse; long ones may require reasoning.



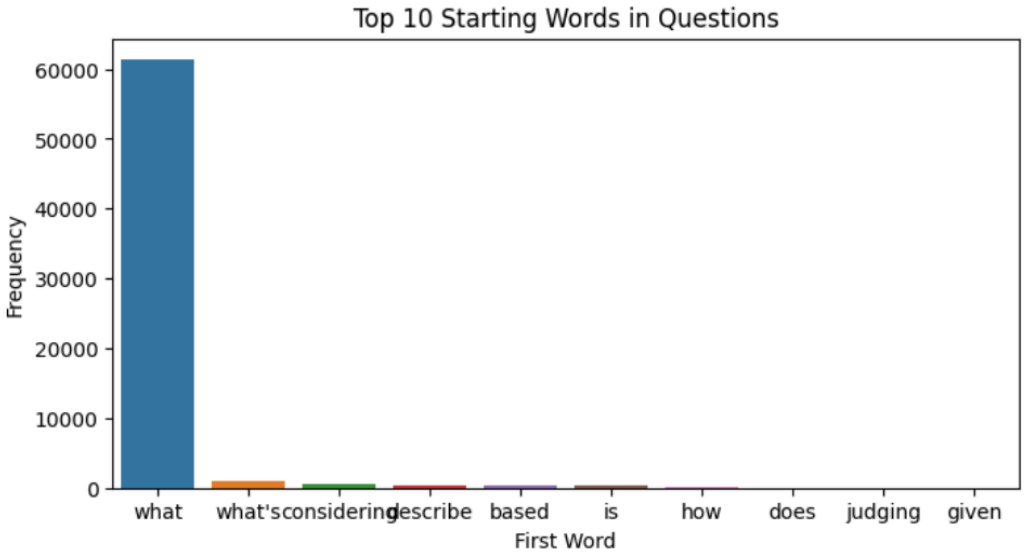
2. Top N Most Frequent Words in Questions:
    a. Shows the Common terms and topics.
    b. Why it matters: Helps in understanding dataset bias or scope (e.g., if most questions are "what color").



3. Unique questions per difficulty:
    a. It shows how distinct question phrasing is across difficulties.
    b. Why it matters: Ensures that questions aren't just copy-pasted with minor tweaks.



4. Vocabulary Size:
    a. It shows the number of unique words across all questions.
    b. Vocabulary size (unique words): 5075
5. Question Start Word Distribution (e.g., “what”, “how”, “where”):
    a. It shows the type of questions (descriptive, locative, quantitative).
    b. Why it matters: Reveals reasoning or factual bias.



This EDA shows the diversity in the data created which will be useful for training the model. The first words and frequent words helped us in making some modifications to the prompt which we were giving to the gemini API.

Training the dataset And Results:

This section describes how we utilized pre-trained Visual Question Answering (VQA) models to evaluate our dataset. We experimented with models listed below. While each model differed in architecture and performance, the core pipeline and evaluation strategy remained consistent.

- Salesforce/Blip2: A powerful vision-language model that enables open-ended VQA by leveraging pretrained image and language encoders with a Q-Former module.
- Salesforce/Blip_VQA_base: A BLIP base model fine-tuned specifically for the VQA task, offering solid accuracy on standard question-answer benchmarks.
- dandelin/vilt-b32-finetuned-vqa: Smaller, transformer-based vision-language model with good VQA accuracy on smaller datasets.
- Salesforce/blip-vqa-capfilt-large: A larger BLIP model fine-tuned on caption-filtered data for enhanced visual grounding and VQA performance
- Salesforce/instructblip-flan-t5-xl :Is an instruction-tuned Vision-Language model that combines BLIP with FLAN-T5-XL, enabling strong performance on tasks like Visual Question Answering (VQA) by following natural language prompts.

General Training and Evaluation Pipeline. This part applies to all pre-trained models we tested:

1. Dataset Preparation:
    a. The image metadata (images.csv) was loaded and linked with VQA entries from the VQA_Dataset.csv.
    b. Images were filtered based on their presence in the VQA dataset.
    c. Image paths were resolved using a mapping between image_id and its file path.

2. Model and Processor Initialization:
    a. For each model, we used the corresponding processor and model from the HuggingFace Transformers library.
    b. The Accelerator library was used to handle device-agnostic model deployment (GPU/CPU).

3. Inference Loop:
    a. For each test question:
        i. The corresponding image was loaded.
        ii. Image-question pair was passed to the model processor.
        iii. Model inference was performed to generate the predicted answer.
        iv. Ground-truth and predicted answers were stored for evaluation.

4. Evaluation Metrics: We used both exact and approximate matching metrics:
    a. Exact Match Accuracy: Measures if the predicted answer exactly matches the ground truth, relevant for strict correctness.
    b. Substring Match Accuracy: Useful for checking partial correctness when exact phrasing might differ.
    c. Token-level Macro F1: Captures overlap at the word level, balancing precision and recall for semantic similarity.
    d. Exact Match F1 / Precision / Recall: Quantifies how often predictions perfectly match the answers, considering both over- and under-prediction.
    e. Substring Match F1 / Precision / Recall: Evaluates partially correct answers that still contain the key information.

f. BERTScore F1: Uses contextual embeddings to assess semantic similarity, even when phrasing differs.
g. ROUGE-L: Measures longest common subsequences, reflecting the structural overlap between prediction and reference.
h. BLEU (1 to 4): Captures n-gram overlap, commonly used in generation tasks to assess fluency and relevance.
i. METEOR: Considers synonymy and word order, making it suitable for varied but semantically correct responses.
j. Levenshtein Similarity: Measures character-level edit distance, helpful for analyzing spelling or minor wording errors.

5. Logging and Analysis:
    a. Evaluation results were saved at regular intervals (every 100 samples).
    b. Skipped or errored samples (due to missing images, corrupt files, etc.) were logged for debugging.

Results for the same (%):

Metrics | Models	Blip2	Blip_vqa_base	vilt-b32-finetuned-vqa	blip-vqa-capfilt-large	Instruct Blip
Exact Match Accuracy	4	36	0	34.34	21
Substring Match Accuracy	14	38.67	0.11	35.62	33
Token-level Macro F1	8.10	37	0.02	34.90	23.63
Exact Match F1	7.69	52.94	0	51.13	34.71
Exact Match Precision	100	100	0	100.00	100
Exact Match Recall	4	36	0	34.34	21
Substring Match F1	24.56	55.77	0.22	52.53	49.62
Substring Match Precision	100	100	100	100.00	100
Substring Match Recall	14	38.67	0.11	35.62	33
BERTScore F1	83.07	95.65	80.92	95.42	89.19

ROUGE-L F1	9.13	38.50	0	36.03	25.85
BLEU-1	3.38	34.77	0	31.86	5.88
BLEU-2	0.83	2.56	0	0.48	0.94
BLEU-3	0.48	0.90	0	0.10	0.45
BLEU-4	0.30	0.42	0	0.04	0.25
METEOR Score	6.36	19.90	0	18.92	15.41
Levenshtein Similarity	16.56	48.37	2.53	45.48	32.32

Fine-Tuning:

After the training on pre-trained models was done, we did fine tuning of some of the models above using LoRA. Low-Rank Adaptation(LoRA) is a parameter-efficient fine-tuning technique that significantly reduces the number of trainable parameters by injecting trainable low-rank matrices into each layer of a pre-trained model, instead of updating all parameters. This allows large vision-language models (like BLIP) to be adapted to downstream tasks like VQA with minimal compute and memory overhead, while still achieving strong performance.

The LORA configuration was as follows:

```
# Define LoRA configuration
lora_config = LoraConfig(
    r=16,  # Rank of low-rank matrices
    lora_alpha=32,  # Scaling factor
    target_modules=["query", "value"],  # Target attention layers in BLIP-1
    lora_dropout=0.1,  # Dropout for regularization
    bias="none"  # No bias adaptation
)
```

This LoRA (Low-Rank Adaptation) configuration is designed to fine-tune specific attention layers of the BLIP-1 model efficiently by injecting trainable low-rank matrices into them. Here's a brief description:

- **r=16**: Sets the rank of the low-rank matrices, determining the degree of adaptation capacity added to the model.
- **lora_alpha=32**: A scaling factor applied to the LoRA updates, controlling their influence during training.

- **target_modules=["query", "value"]**: Specifies that only the attention layers corresponding to "query" and "value" projections in the transformer will be adapted. Improves cross modal-alignment. In VQA, successful reasoning requires the model to align relevant visual regions with the question content. Fine-tuning query helps the model better interpret the question, while modifying value helps it retrieve more appropriate visual/textual features.
- **lora_dropout=0.1**: Adds dropout to the LoRA layers during training to improve generalization and reduce overfitting.
- **bias="none"**: Indicates that no additional bias terms are adapted, keeping the parameter count minimal.

Results for the same (%):

Metrics | Models	Blip_vqa_base	blip-vqa-capfilt-large
Exact Match Accuracy	48.67	50.74
Substring Match Accuracy	49	51.41
Token-level Macro F1	48.89	50.79
Exact Match F1	65.47	67.32
Exact Match Precision	100	100.00
Exact Match Recall	48.67	50.74
Substring Match F1	65.77	67.91
Substring Match Precision	100	100.00
Substring Match Recall	49	51.41
BERTScore F1	96.73	96.87
ROUGE-L F1	49.22	52.06
BLEU-1	48.68	50.68
BLEU-2	3.04	1.22

BLEU-3	1	0.30
BLEU-4	0.45	0.11
METEOR Score	25.15	26.40
Levenshtein Similarity	56.90	59.13

We got the best results after fine tuning, from the blip-vqa-capfilt-large model and we proceeded with this one. The final weights after training are uploaded here.

How to run the code:

1. Using the requirements.txt file, create a new environment (Python 3.9) and install the dependencies mentioned in the file.
   - `conda create --name vr_proj_env python=3.9 (if not already created)`
   - `conda activate vr_proj_env`
   - `pip install -r requirements.txt`

2. Run the inference.py file using the following command:
   `python inference.py --image_dir <PATH-TO-IMAGE-DIR> --csv_path <PATH-TO-IMAGE_METADATA-CSV>`

Materials used as reference and learnings:

- https://aistudio.google.com/prompts/new_chat
- https://medium.com/cyberark-engineering/how-to-run-llms-locally-with-ollama-cb00fa55d5de?utm_source=chatgpt.com
- https://discuss.huggingface.co/t/example-for-fine-tuning-clip-or-blip2-for-vqa/50772?utm_source=chatgpt.com
- https://gautam75.medium.com/fine-tuning-vision-language-models-using-lora-b640c9af8b3c?utm_source=chatgpt.com
- https://huggingface.co/docs/peft/en/developer_guides/quantization
- https://github.com/NielsRogge/Transformers-Tutorials/tree/master/BLIP-2
- https://github.com/dino-chiio/blip-vqa-finetune
- https://colab.research.google.com/gist/Vishesht27/1ebbe4c01138c6bfc100740b1fe76bf5/lora_applications.ipynb#scrollTo=rLS3HNTq5WU_
- https://openrouter.ai/mistralai/mistral-small-3.1-24b-instruct:free