# Multimodal Visual Question Answering with Amazon Berkeley Objects Dataset

Nathan Mathew Verghese
IMT2022022

Divyam Sareen
IMT2022010

Sarvesh Kumar
IMT2022521

## I. Introduction

The task was to build a Visual Question Answering model that takes an image and a related question as input, and responds with a one-word answer. For training, we used the Amazon-Berkeley Objects dataset, along with its metadata, which provided additional details about the items shown in the images.

To approach this, we broke the project down into several key stages: curating the dataset using powerful language models available online, setting up a baseline using lightweight yet effective transformer models, and fine-tuning using popular techniques like Low-Rank Adaptation (LoRA). We regularly evaluated the model's performance throughout the training process to track its progress and understand how well it was learning.

## II. Data Curation:

### A. Dataset Overview

In the first stage, we worked with the [1]Amazon-Berkeley Objects Dataset, which includes product images and their associated metadata. We selected the first 20,000 entries from the listings JSON file to create a dataset of 20,000 images, each paired with 2 to 3 relevant questions and single-word answers of different difficulty, made by an LLM.

### B. Preprocessing

Each entry from the `listings.json` file was matched to its corresponding image using a unique image ID, with the mapping provided in the `images.csv` file. For each listing, we first checked whether the resource language was English. If it was, we extracted relevant details such as `item_keywords`, `color`, and `product-type`. These details, along with the associated image, were then passed to a language model to generate meaningful questions.

### C. Data Curation Process

For each image and its corresponding metadata extracted from the listing, a prompt (explained later) was constructed and sent to Gemini 1.5 Flash via its API. To handle a large number of requests, multiple API keys were used and rotated throughout the process, along with 25-second delays. Gemini returned a Visual Question-Answer (VQA) pair in JSON format, which was then parsed to verify its validity. Valid responses were saved in a CSV file along with the image ID, image path, and the original listing.

### D. The Prompt

To generate Visual Question Answering (VQA) pairs, we used a two-part prompting strategy with Gemini 1.5 Flash: a `system_prompt` to define the task, and a `user_prompt` to pass specific inputs for each image. These prompts were sent to the Gemini API for each image in the dataset.

*a) System Prompt:* The system prompt used is as follows:

> You are an expert Visual Question Answering (VQA) dataset creator, specialising in generating questions about product images. Your goal is to create question-answer pairs based *solely* on visual information present in an image.
> Your Constraints and Guidelines:
>
> - **Visual Grounding:** All questions must be answerable by looking directly at the image provided. No external knowledge is allowed for answering.
> - **Single-Word Answers:** Every answer must be a single, concise word (e.g., colors like "Green", confirmations like "Yes"/"No", materials like "Metal", shapes like "Square").
> - **Diverse Questions:** Generate questions covering various visual aspects, including object identification, color, apparent material, shape, key parts, and basic spatial relationships if applicable.
> - **Clarity:** Questions should be clear, specific, and unambiguous.
> - **Strict Avoidance:** Do NOT generate questions about: subjective qualities, price/brand/origin unless clearly visible as large text, counting numerous small items, reading fine print, or anything requiring external data or numerical output.
> - **Output Format:** You MUST output the results as a JSON formatted list of lists. Each inner list must contain exactly two strings: the question and the single-word answer. Example: `[["Question 1", "Answer 1"], ["Question 2", "Answer 2"]]`

*b) User Prompt:* For each image, we dynamically generated the following user prompt using the image ID and extracted keywords:

Image ID: {`image_id`}
Metadata Keywords: {`keywords`}
Generate 2-3 question-answer pairs based on the image and keywords provided. Follow the JSON output format strictly.

*c) Usage:* These prompts were passed to the Gemini 1.5 Flash model via its API. The model returned a JSON-formatted list of VQA pairs.

## III. Baseline Evaluation Models

We used multiple existing vision-language models to obtain a baseline evaluation on our dataset. The models chosen were Bakllava, BLIP-VQA-Base, BLIP-2, and Granite. Each model was selected for its unique architecture and trade-offs between size, performance, and accessibility. To ensure smooth execution and avoid interruptions during long runs on Kaggle, we implemented several `try-except` blocks throughout our notebooks to gracefully handle issues. Additionally, we included a timeout mechanism to halt the notebook safely before Kaggle's maximum runtime limit of 12 hours was reached. This allowed us to preserve progress and avoid loss of intermediate results.

### A. BLIP-VQA-Base

*Model Description:* [2]BLIP (Bootstrapped Language-Image Pretraining) is a base model for Visual Question Answering. It integrates a Vision Transformer (ViT) for extracting features from the image, a BERT-based text encoder for processing the question, and cross-modal attention layers to fuse visual and textual information. BLIP is relatively lightweight and does decently well on some benchmarks.

*Implementation:* We initialised BLIP with the `Salesforce/blip-vqa-base` weights. GPU acceleration was enabled where possible, and a fast tokeniser was employed for quicker preprocessing. For inference, the model processed each image and corresponding question, which was wrapped in a prompt given as `Based on the image, answer the following question with a single word. Question: {question} Answer:`. The entire pipeline was built to load images, tokenise inputs, predict outputs, and clean answers using regular expressions. Results were stored in a CSV file. The script also handled errors like missing or unreadable images and used `tqdm` for tracking progress.
`Code`

### B. BLIP-2

*Model Description:* BLIP-2 builds upon the original BLIP model by introducing a more modular architecture, including a frozen image encoder and a Q-former (query transformer) for text conditioning. The fused representations are then passed to a large language model (LLM) such as [3]Flan-T5-XL or [4]OPT-2.7B for final answer generation. This decoupled design enables better scalability, flexibility in LLM selection, and improved transferability across tasks.

*Implementation:* We experimented with two language model backbones for BLIP-2: Flan-T5-XL and OPT-2.7B. The input questions were wrapped the same way as in BLIP-VQA Base.

For the Flan-T5-XL variant, we initialised with the `Salesforce/blip2-flan-t5-xl` weights, which is optimised for accuracy and comprehension across VQA tasks. For the OPT-2.7B variant, we used the `Salesforce/blip2-opt-2.7b` weights, offering a smaller and faster alternative for our needs as Kaggle was posing issues with computation power.

In both cases, the inference loop handled image loading, question formatting, tokenisation, and response cleaning. Results were saved in CSV files after applying regex-based postprocessing. Errors such as unreadable images were logged and skipped, with progress tracked using `tqdm`.
`Code (Flan-T5-XL)`
`Code (OPT-2.7B)`

*Quantisation:* **The OPT-2.7B model was having issues while being loaded onto Kaggle as is. So we used 4-bit quantisation to help us work through this issue.**

### C. Bakllava

*Model Description:* [5]Bakllava is an open-source multimodal model capable of strong performance on vision-language tasks. It builds on the LLaVA architecture and uses CLIP for vision feature extraction while leveraging a language model backbone. It supports long-context understanding and is optimised for image-grounded tasks such as VQA.

*Implementation:* The same implementation format was used, similar to the other models. The questions were wrapped in prompts to make the answers one word long. Outputs were parsed from the model's JSON response and filtered for single-word answers using regex-based cleaning. Processed results were stored in a CSV file for analysis.
`Code`

### D. Granite

*Model Description:* [6]Granite is IBM's proprietary vision-language model, designed for tasks like captioning and visual question answering. It features strong alignment between image and text modalities and is optimised for enterprise-grade applications, balancing performance with inference efficiency.

*Implementation:* A similar method of implementation was used as in the other models. A wrapper script sent each image and question to the API, parsed the JSON response, validated the answer format, and stored outputs in structured CSV files.
`Code`

The results and metrics of each model that we ran are given below. Do note that some models only ran half the dataset, some models were quantised because of the computational limitations that we had.

TABLE I: Baseline Model Evaluation Metrics

| Model | Dataset | Quantised | EM Accuracy | EM F1 | BERT F1 |
| --- | --- | --- | --- | --- | --- |
| BLIP-VQA-Base | 100% of total | No | 0.409 | 0.581 | 0.882 |
| BLIP-2 (Flan-T5 XL) | 100% of total | No | 0.488 | 0.656 | 0.889 |
| BLIP-2 (OPT-2.7B) | 50% of total | Yes (4-bit) | 0.508 | 0.674 | 0.899 |
| Bakllava | 100% of total | No | 0.620 | 0.766 | 0.920 |
| Granite | 50% of total | No | 0.673 | 0.805 | 0.906 |

*Note:* EM stands for Exact Match.

## IV. Fine Tuning Approaches

[7]Low-Rank Adaptation (LoRA) is a parameter-efficient fine-tuning technique that modifies the standard training of transformer-based models by introducing low-rank decomposition matrices. Instead of updating the full weight matrices during backpropagation, LoRA freezes the pre-trained weights and injects trainable matrices $A \in \mathbb{R}^{d \times r}$ and $B \in \mathbb{R}^{r \times k}$ such that the original weight matrix $W_0 \in \mathbb{R}^{d \times k}$ is adapted as:

$$W = W_0 + \Delta W = W_0 + AB$$

This significantly reduces the number of trainable parameters, allowing for efficient fine-tuning even on large models. We used LoRA to adapt our models to our dataset. We ran a select few models and tried our best to improve them, as larger models wouldn't be small enough for Kaggle to work properly. The configurations are given further.

## V. Models, Configurations, and Dataset Splits

We evaluated several vision-language models fine-tuned using LoRA. Most models were having issues getting loaded on Kaggle without giving us a memory error. Below are the successful configurations that we managed to run:

- **BLIP-VQA-Base:** Fine-tuned using LoRA with multiple hyperparameter configurations for the rank ($r$): 16 (quantised and non-quantised), 8 (non-quantised), and 32 (non-quantised). The dataset split used was 80-20. `Code`
- **BLIP-VQA-Base (M):** Another change we made to try to make the model better was that we tried to **modify target modules** to see if that gave better results, which it did. The results will be shown later. `Code`
- **BLIP-2 (Flan-T5 XL):** Fine-tuned using LoRA with $r = 8, 16, 32$ (tried) and quantisation enabled. It was trained and tested on a limited rows due to computational bounds. `Code`
- **BLIP-2 (OPT-2.7B):** Fine-tuned using LoRA with $r = 8, 16, 32$ (tried) and quantisation enabled. The dataset split used was 50-50. `Code`

The top models on the test split are given in the table below. The best-performing model is highlighted in bold.

TABLE II: Evaluation Metrics for Fine-Tuned Models Using LoRA

| Model | Train-Test Split | Quantised | LoRA $r$ | EM Acc. | EM F1 | BERT F1 |
| --- | --- | --- | --- | --- | --- | --- |
| BLIP-VQA-Base | 80-20 | No | 8 | 0.485 | 0.653 | 0.902 |
| **BLIP-VQA-Base (M)** | **80-20** | **No** | **16** | **0.570** | **0.726** | **0.921** |
| BLIP-VQA-Base | 80-20 | No | 16 | 0.528 | 0.691 | 0.899 |
| BLIP-VQA-Base (M) | 80-20 | Yes | 16 | 0.562 | 0.720 | 0.922 |
| BLIP-VQA-Base | 80-20 | No | 32 | 0.524 | 0.688 | 0.901 |
| BLIP-2 (Flan-T5 XL) | 300 questions tested | Yes | 8 | 0.367 | 0.537 | 0.775 |
| BLIP-2 (OPT-2.7B) | 50-50 | Yes | 16 | 0.508 | 0.673 | 0.899 |

*Note:* EM stands for Exact Match. (M) signifies that the target modules were modified.

Something to note is that some Blip-2 models were too big for Kaggle to run for all rows on the dataset. Hence, the performance was considerably lower.

## VI. Evaluation Metrics

We used 3 main metrics (but did not limit to just this).

- `Exact-Match Accuracy`: This checks if the predicted words exactly match the ground-truth words after removing special symbols and ignoring capitalisation.

- `Exact-Match F1 score`: Here, we assume precision is 1, meaning everything the model predicted is correct. The recall is then the fraction of the ground-truth tokens that appear in the prediction:

$$\text{Recall} = \frac{\text{Number of correctly predicted tokens}}{\text{Total tokens in the ground truth}}$$

  Since precision $P = 1$, the F1 score simplifies to:

$$F1 = 2 \times \frac{P \times R}{P + R} = \frac{2R}{1 + R}$$

- `BERT F1 score`: Unlike exact matching, BERT F1 uses BERT's contextual embeddings to compare predicted and true tokens. Instead of only checking if tokens are exactly the same, it measures how similar they are in meaning.
  This makes BERT F1 better for cases where the prediction is close in meaning but not a perfect word-for-word match.
  - **Precision** measures how many predicted tokens have a good semantic match in the ground truth.
  - **Recall** measures how much of the ground truth is captured by the prediction's meaning.
  - **F1** combines precision and recall into a single score reflecting overall semantic overlap.

  Because it looks at meaning rather than exact words, BERT F1 usually gives a more useful score when evaluating natural language tasks.

## VII. Conclusion

In this project, we developed a robust multimodal Visual Question Answering pipeline using the Amazon-Berkeley Objects dataset. We curated a high-quality dataset leveraging LLM-generated question-answer pairs grounded in image metadata and established strong baselines with various vision-language models, including BLIP and BLIP-2. Our approach highlights the power of combining structured data, automated prompt engineering, and model fine-tuning techniques like LoRA to address real-world VQA challenges effectively. Future work could explore improved reasoning models and dataset expansion for even richer understanding, and if we had stronger compute, we could have tried larger models as well.

We have learnt quite a bit from this project, not limited to just the course. Also, having to work around the general issues of Kaggle and computational limitations also was quite challenging to get to the end result that we did get to. It was considerably hard to get large models to work on Kaggle well enough for us to train and test for the whole dataset.

Link to github: https://github.com/nathanmathewv/Multimodal-VQA

Link to dataset: https://www.kaggle.com/datasets/nathanmathew/images-with-vqas

## VIII. References

### References

[1] Amazon Berkeley Objects (ABO). Available at: https://amazon-berkeley-objects.s3.amazonaws.com/index.html
[2] BLIP GitHub repository. Available at: https://github.com/salesforce/BLIP.git
[3] BLIP2 Flan-T5-XL model on Hugging Face. Available at: https://huggingface.co/Salesforce/blip2-flan-t5-xl
[4] BLIP2 OPT-2.7B model on Hugging Face. Available at: https://huggingface.co/Salesforce/blip2-flan-t5-xl
[5] Bakllava model on Hugging Face. Available at: https://huggingface.co/llava-hf/bakLlava-v1-hf
[6] Granite model on Hugging Face. Available at: https://huggingface.co/ibm-granite/granite-vision-3.2-2b
[7] Check LoRA out here: https://huggingface.co/docs/diffusers/main/en/training/lora