VR Mini Project - 2
Multimodal Visual Question Answering with Amazon Berkeley Objects Dataset

Team Members: Aryan Mishra(IMT2022502), Md Owais(IMT2022102)

May 18, 2025

Contents

1 Introduction

Visual Question Answering (VQA) is a difficult multimodal task that calls for models to reason and comprehend both visual information and natural language questions. In this paper, we utilize the Amazon Berkeley Objects (ABO) dataset—147,702 product descriptions and 398,212 256×256 catalog images with dense multilingual metadata—to develop a strong multiple-choice VQA pipeline for single-word answers.

The goals of this research are fourfold:

1. **Data Curation**: Create a well-balanced, single-word-answer VQA dataset from ABO through multimodal APIs (e.g., Gemini 2.0) and on-device models (e.g., Ollama) for prompt-based question-answer generation.

2. **Baseline Evaluation**: Test pre-trained VQA models (BLIP VQA Base, ViLT, BLIP-2) on the curated dataset with no further training.

3. **LoRA Fine-Tuning**: Use Low-Rank Adaptation (LoRA) for parameter-efficient fine-tuning, optimizing for our 32GB GPU resource limit.

4. **Evaluation Metrics**: Compare performance on standard metrics (Accuracy, F1 Score) and state-of-the-art semantic metrics (BERTScore), iterating towards better results.

This report outlines the process and results of each step—data curation, baseline testing, LoRA fine-tuning, and metric-driven evaluation—to provide a VQA model that meets accuracy requirements with computational efficiency.

2 Data Curation

2.1 Dataset Structure

The dataset has three major components:

1. Images: Found in dataset/images/small, this folder holds raw product images in JPEG format.

2. Listings Metadata: In dataset/abo-listings/listings/metadata, this contains JSON files (some gzipped) listing product information like main image ID, secondary image IDs, and attributes like item name, product type, and color.

3. Image Metadata: A CSV file named dataset/images/metadata/images.csv contains image-specific metadata such as image ID, height, width, and file path.

2.2 Preprocessing Steps

The filtering approach is about preparing a dataset linguistically and contextually appropriate for VQA. The procedure, as done in filter.py, entails the following:

1. Language-Based Filtering: There are retained only metadata entries in English variants (en_IN, en_US, en_CA, en_GB, en_AU, en_SG). This choice favors English

because of its dominance in the dataset and its ability to create contextually meaningful text embeddings in vision-language models.

The script verifies fields such as item_name, item_keywords, or bullet_point to spot the language tag for linguistic context consistency.

2. Attribute Selection: Out of the large set of metadata attributes, only those that pertain to VQA are kept: bullet_point, color, color_code, fabric_type, item_name, item_shape, material, pattern, product_description, product_type, and style.

   Attributes such as item_weight are dropped, since they are not inferable from images and therefore cannot be considered relevant for visual understanding purposes.

   This filtering ensures that the dataset includes visually verifiable data, making it more appropriate for VQA.

3. Initial Output:

   Filtered entries are written to dataset_curated/curated_listings.jsonl, a JSON Lines document with image paths and chosen metadata attributes.

4. Duplicate Removal:

   The script finds and eliminates duplicate listings by image paths, keeping only the entry with the highest number of attributes to ensure as much contextual information as possible.

   Deduplicated data is written to dataset_curated/unique_listings.jsonl.

5. Data Splitting:

   The deduplicated listings are split into six mutually disjoint splits (S1 to S6), each having 3,000 entries.

   Each split guarantees representation of all 576 distinct product types, ensuring diversity and balance between categories.

   Images are duplicated to split-specific directories, and metadata is stored as JSON files to enable orderly access for downstream tasks.

2.3 Prompt Design

"'Given the product image and the following context: context, generate 5 short question and one-word answer pairs that test visual understanding. Each answer must be a single word (e.g., 'Red', 'Bag', 'Plastic'). Format as: Q: <question>A: <answer>... "'

- The curate.py program uses the Gemini 1.5 Flash API to produce Visual Question Answering (VQA) pairs for each of the splits generated by filter.py. The prompt is assembled by replacing flattened metadata attributes (e.g., color, material, product_type aggregated into a comma-separated string such as "color: blue, material: cotton, product_type: shirt") with the template listed above. This prompt asks the API to generate five question-answer pairs for each image, where every question tests visual comprehension, and the answers are limited to one word for simplicity and consistency.

- Some sample question-answer pairs produced out of the prompt are:

1. .path/bec06177.jpg,What is the metal's color?,Silver
2. .path/bec06177.jpg,What is the main gemstone shape?,Round
3. .path/bec06177.jpg,What is the setting style?,Prong
4. .path/bec06177.jpg,How many rings are in the set?,Two
5. .path/bec06177.jpg,What is the band style of the wedding band?,Baguette

- The prompt guarantees diversity through the inclusion of an array of metadata features, allowing for questions that span diverse visual properties such as color, pattern, texture, and product type. This diversity avoids redundant questions and captures diverse aspects of the image content. Complexity is ensured by asking questions that involve contextual inference, e.g., spotting patterns or styles, which necessitate richer visual examination deeper than mere recognition.

- The code curate.py, can also be run parallelly by using different API keys inside each run, so that the disjoint Si subsets of data can have their VQA pairs generated parallelly, saving time, as the Gemini API key is rate limited.

- Now we have 6 folders, which have csvs of the format image_path, question, and answer.

- For the purpose of running the code, these 6 folders were uploaded to Kaggle separately.

- For the purpose of submission, the path name was changed to only the image name, and the 6 csvs were merged, to produce a single Dataset.csv, which does not have images, only unique image names, and question answer pairs.

3 Baseline Evaluation

3.1 Model Selection

In our baseline testing, we evaluated three off-the-shelf VQA models from Hugging Face:

- **BLIP VQA Base** (Salesforce/blip-vqa-base): around 385M parameters. A vision–language transformer fine-tuned for VQA, trading performance and model size.

- **ViLT** (dandelin/vilt-b32-finetuned-vqa): around 87.4M parameters. Lightweight model that processes image patches and text tokens together—no CNN-based feature extractor—which makes it very parameter-efficient.

- **BLIP-2** (`Salesforce/blip2-flan-t5-xl`): around 3.1B parameters. A more sophisticated BLIP model that leverages a large language model backend (Flan-T5) for more profound multimodal comprehension, albeit at the expense of substantially more computation.

We ran inference on our curated test set without any additional training to establish baseline metrics, as reported in Table 2 in the "Evaluation metrics" section.

3.2 Evaluation Protocol

The evaluation of the models was carried out using(as explained in detail in the "Evaluation Metrics" section):

- **Accuracy**: The proportion of correctly answered questions.
- **F1 Score**: The harmonic mean of precision and recall.
- **BERTScore**: A context similarity measurement comparing generated responses to ground truth via pre-trained language models.

4 Fine-Tuning with LoRA

Among the three baseline models (BLIP VQA Base, ViLT, and BLIP-2), we chose ViLT and BLIP VQA Base for fine-tuning. We did not use BLIP-2 because of its size (around 3 billion parameters), which would run at high costs on the current free GPU resources. We ran LoRA fine-tuning on ViLT with various hyperparameters combinations and got the best results using the following combination:

- Rank (r): 8
- Epochs (e): 3
- Learning Rate: 1e-4
- lora_alpha: 32
- lora_dropout: 0.05
- per_device_train_batch_size: 4,
- gradient_accumulation_steps: 2

This configuration yielded an accuracy of 39.2%, an F1 Score of 13.25%, and a BERTScore of 97.70%.
As this performance was not satisfactory, we chose BLIP VQA Base as our final fine-tuned model, leveraging its stronger baseline results and resource efficiency.

4.1 Finetuning Process Description

We used the Low-Rank Adaptation (LoRA) method to adapt our chosen models in a parameter-efficient way. The underlying concept of LoRA is to lock most of the pre-trained model weights and add small, trainable low-rank matrices to the attention heads. At forward passes, rather than updating the entire weight matrix $W \in R^{d\times d}$, LoRA learns two low-rank matrices $A \in R^{d\times r}$ and $B \in R^{r\times d}$, such that the update $AB = \Delta W$. This factorization decreases the number of trainable parameters from d^2 to $2dr$, where $r \ll d$.

Key steps in our fine-tuning pipeline:

1. **Freeze Base Weights**: All the original model parameters are frozen, keeping pre-learned knowledge intact and minimizing GPU memory consumption.

2. **Insert LoRA Adapters**: In every multi-head attention layer, we insert trainable adapter matrices into the query and key projections.

3. **Set Hyperparameters**: We specify the adapter rank (r), LoRA scaling factor (α), and dropout rate for adapter outputs.

4. **Optimizer Configuration**: We employ AdamW and apply weight decay exclusively to non-adapter parameters, in combination with gradient clipping to ensure training stability.

5. **Training Loop**: For e epochs, we perform regular sequence-to-sequence training steps but only have gradients passing through the adapter matrices A and B.

6. **Early Stopping**: We track validation loss and halt training if it does not decrease for three continuing epochs, avoiding overfitting.

By limiting updates to this low-dimensional subspace, LoRA allows for fast fine-tuning with a fraction of the compute and memory cost of full-model updates. This efficiency was particularly important with our 32GB GPU budget, allowing us to try more hyperparameter settings and train with larger batch sizes.

4.2 LoRA Hyperparameter Experiments for BLIP VQA Base

To evaluate LoRA on ViLT, we varied the rank (r) and epochs (e) and measured three metrics: Accuracy, F1 Score, and BERTScore. Results are shown in Table 1.

Exp.	**Rank (r)**	**Epochs (e)**	**Accuracy**	**F1 Score**	**BERTScore**
1	12	3	64%	19%	97.81%
2	16	3	65%	19%	97.98%
3	16	5	66%	21%	98.11%
4	5	5	66%	21%	98.02%
5	5	7	68%	22%	98.46%
6	24	5	64%	18%	97.54%

Table 1: LoRA hyperparameter experiments on Blip Vqa Base, evaluating Accuracy, F1 Score, and BERTScore.

4.3 Final Model Selection and LoRA Parameters

BLIP VQA Base was selected for final LoRA fine-tuning due to its superior baseline performance. The exact LoRA configuration used for the best-performing model is listed below:

- **Base Model**: Salesforce/blip-vqa-base
- **LoRA Rank (r)**: 5
- **lora_alpha**: 32
- **lora_dropout**: 0.1

- **Epochs** (e): 7
- **weight_decay**: 0.01
- **Learning Rate**: 5e-5
- **per_device_train_batch_size**: 4
- **gradient_accumulation_steps**: 4
- **Optimizer**: AdamW with gradient clipping
- **Early Stopping**: Enabled, monitored on validation loss

This configuration achieved an accuracy of 68%, F1 Score of 22%, and BERTScore of 98.46% on the test set, striking a balance between performance improvement and resource utilization over the baseline, however remaining very similar to the remaining finetuned models.

4.4 Challenges and Mitigations

In fine-tuning, we faced the following challenges:

- **GPU Memory Constraints**: Training even the BLIP VQA Base model with LoRA adapters resulted in increased memory consumption. *Mitigation*: We used gradient checkpointing and smaller batch sizes to stay within our GPU budget.
- **Overfitting on Small Dataset**: Sudden convergence on small validation data resulted in overfitting. *Mitigation*: We turned on early stopping (patience of 3 epochs) and added a dropout of 0.1 within the LoRA layers.

5 Evaluation Metrics

5.1 Accuracy

Accuracy is the ratio of correctly answered questions to all questions asked. It is formally defined as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \times 100\%. \tag{1}$$

Accuracy provides a simple glimpse of overall correctness but is deceptive on imbalanced datasets.

5.2 F1 Score

The F1 Score is the harmonic mean of precision and recall, weighing both false positives and false negatives:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}, \tag{2}$$

$$\tag{3}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}, \tag{4}$$

$$\tag{5}$$

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{6}$$

F1 is particularly valuable when class sizes are highly skewed.

5.3 BERTScore

Accuracy gives a simple snapshot of total correctness, but it can be deceptive on unbalanced datasets and does not recognize near-misses. For instance, a ground-truth response like "two" or "elephant" would be labeled incorrect if the model responds with "2" or "elephants" respectively, as both have the same meaning. To recognize such nuance, we supplement Accuracy with contextual scores such as BERTScore, which estimate semantic similarity instead of exact token matches.
BERTScore uses contextualized embeddings from a pre-trained BERT encoder. For every generated token, it identifies the top cosine similarity with some reference token embedding, and then averages them to produce a semantic similarity score. This more strongly correlates with human evaluation than raw overlap of tokens.

5.4 Evaluation Results

Table 2 shows placeholder results for each model, both before and after fine-tuning:

Model	**Baseline**			**Fine-Tuned**		
	Accuracy	F1 Score	BERTScore	Accuracy	F1 Score	BERTScore
BLIP VQA Base	43.80%	12%	97.26%	68%	22%	98.46%
ViLT	26.60%	08.06%	97.10%	39.2%	13.25%	97.70%
BLIP-2	42.78%	19%	80.23%	–	–	–

Table 2: Baseline vs. fine-tuned evaluation metrics for selected VQA models.

5.5 Insights

Table 2 analysis shows the following important observations:

- **BLIP-2's Baseline Strength vs. Semantic Drift**: BLIP-2 received the strongest baseline F1 Score (19.0%) and comparable Accuracy (42.8%) to BLIP VQA Base, yet its relatively low BERTScore (80.2%) indicates it occasionally gives semantically off-target responses with correct tokens.

- **ViLT's Efficiency Trade-Off**: ViLT's light-weight nature (87M parameters) is accompanied by lower baseline Accuracy (26.6%) and F1 (8.1%). Fine-tuning raised them to 39.2% and 13.3%, still trailing BLIP VQA Base, which suggests limited representational capacity for this task.
- **BLIP VQA Base's LoRA Gains**: Fine-tuning BLIP VQA Base with LoRA improved Accuracy to 68.0% from 43.8% and F1 to 22.0% from 12.0%, confirming that adapter-based tuning can really boost performance on a solid base model.
- **High Semantic Consistency Post-Fine-Tuning:** Both BLIP VQA Base and ViLT have BERTScores over 97% after fine-tuning, which means that even if surface forms vary, model outputs are still extremely semantically consistent with ground truth.

Hence, we can conclude that considering all the factors such as number of parameters and the evaluation metrics, Blip VQA Base is the best model for our use case.