Multimodal Visual Question Answering using BLIP-2 with LoRA and Quantization

AIM825 - Mini Project 2

Varnit Mittal IMT2022025
Sanchit Kumar Dogra IMT2022035
Aditya Priyadarshi IMT2022075

May 18, 2025

Contents

1 Introduction

Visual Question Answering (VQA) is a multimodal task that combines computer vision and natural language processing to answer questions based on image content.

Our project focuses on developing a VQA system using the Amazon Berkeley Objects (ABO) dataset, which includes 398,212 product images and their respective metadata. We aim to form a VQA dataset using image-question-answer triples, then evaluate baseline models and fine-tune them using efficient techniques such as Low-Rank Adaptation (LoRA).

To account for our limited compute resources, we have additionally applied quantization for efficiency in inference. Our model was trained and evaluated using free cloud GPUs from Kaggle.

This report documents our data curation process, baseline evaluations, LoRA fine-tuning strategies and final results and observations with detailed performance metrics such as accuracy, BERTScore and some other specialised metrics.

1.1 About the ABO Dataset

The Amazon Berkeley Objects (ABO) dataset given to us contains 147,702 unique product listings accompanied by their 398,212 unique catalog images. Each listing comes with multilingual metadata, including details such as title, description, brand, and color.

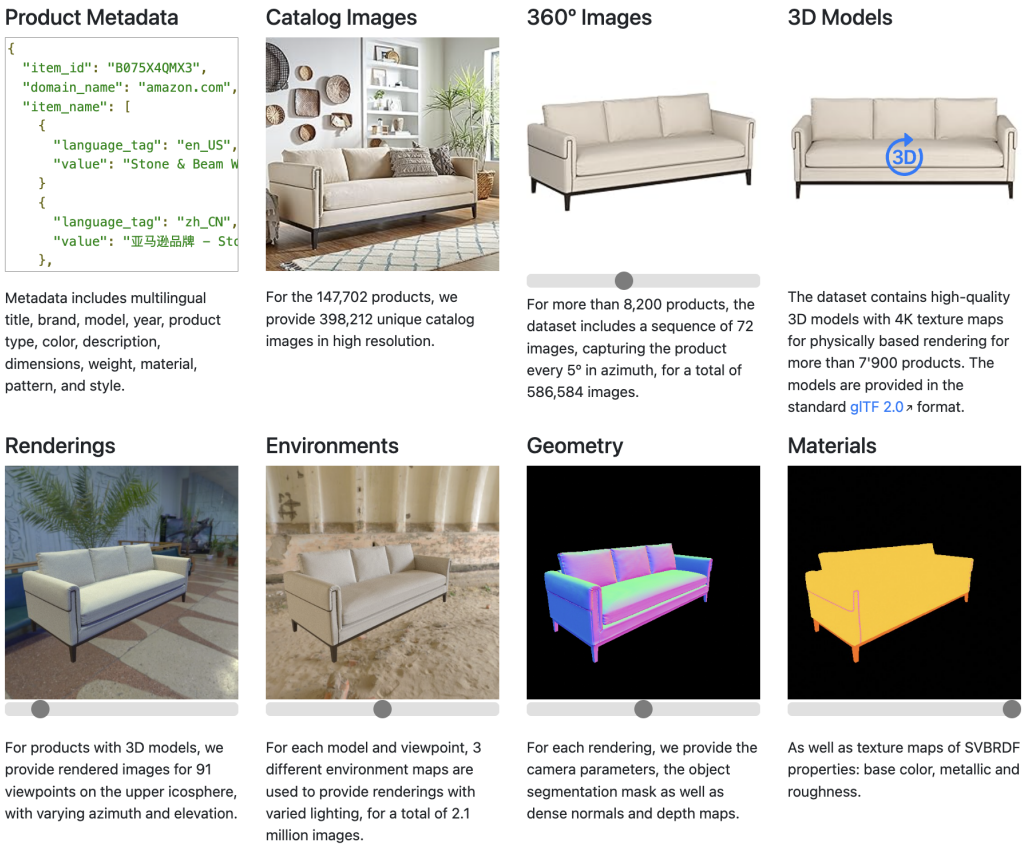

Figure 1: Overview of the ABO dataset

For this project, we were advised to use the small variant of the dataset, which includes images that have been resized to 256x256 pixels and their corresponding metadata in CSV format. The total size of this variant is approximately 3GB, thus great for processing on free cloud GPU platforms like Kaggle and Colab.

1.2 Curation Process

The dataset curation was performed using the Gemini 2.5 Flash API for multimodal processing. For each image in our collection, we formulated targeted prompts designed to elicit concise, single-word answers. For example:

- **Prompt:** "What is the color of the object shown?"
- **Answer:** Red

To support high-throughput prompt generation, we provisioned a pool of 10–12 distinct API keys. These keys were managed in a round-robin fashion: with each outgoing request consuming the next available key present in a circular list. When any key fulfilled it's hourly/daily quota or was invalidated because of other reasons, our system automatically removed it from the rotational cycle and logged a clear warning message (e.g., "[WARN] Key XXX failed error; removing from pool."). This guaranteed uninterrupted operation without much manual intervention. When the entire key pool become depleted, the curation script halted execution by raising a dedicated exception (`All API keys exhausted`). All interactions including prompt submissions, API responses, key rotations and warnings were recorded at INFO level in a centralized log file for full reproducibility and auditability purposes.

Sample Entry in Our Actual Curated Dataset:

```
ID, Question, Answer, Filename
718mYsQTQbL, What are the items in the image?, Bibs, 4c/4c533ad7.jpg
```

2 Baseline Evaluation

2.1 Model Selection

We adopt the **BLIP-2** vision–language framework via Hugging Face, using the public checkpoint `Salesforce/blip-vqa-base`. Key reasons:

1. **Zero-shot ability** – handles visual QA without extra fine-tuning, giving an immediate baseline.
2. **Minimal engineering** – the paired `BlipProcessor` and `BlipForQuestionAnswering` slot straight into standard PyTorch code.
3. **Resource friendly** – the 214 M-parameter "base"" model fits on a single consumer GPU for batched inference.

We also explored other models like VILT but while comparing accuracy, BLIP-2 performed better with an accuracy of about 42 % while VILT accuracy was about 33%. We searched for reasons behind this and got to know why BLIP-2 was better than VILT. The following are a few reasons why we decided to proceed with BLIP-2 architecture:-

1. **Strong vision capabilities via Frozen Backbones:** BLIP-2 architecture is based on powerful, frozen image encoders like CLIP or VIT and only trains a lightweight Querying Transformer (Q-Former) between the vision encoder and a frozen LLM. This lets it leverage rich, pre-learned visual features without much need of fine-tuning the vision backbone.

2. **Efficient Cross-Modal Interaction via Q-Former** The Q-former in BLIP-2 architecture has been designed to extract a small set of "query" vectors and attend to the most relevant parts of the visual features.

Both the baseline evaluation scripts can be found in our GitHub. BLIP-2 VILT

2.2 Baseline Setup

Our pipeline is intentionally simple:

- Image I/O: open image in RGB; on failure, log a warning and return a sentinel string.
- Pre-processing: processor normalises the image and tokenises the question in one call.
- Inference: run `generate` with `max_new_tokens = 10` under `torch.no_grad()`.
- Post-processing: decode IDs, strip specials/whitespace → final answer.

This fixed baseline anchors every subsequent experiment, guaranteeing that any performance gains reflect improvements in prompts or data quality—not alterations to the model itself.

2.3 Metrics Used

To evaluate the baseline, we used common evaluation metrics as well some specialised metrics such as BERTscore, CSNS and VTGS:

- **Accuracy:** Measures the proportion of exact matches between predicted and ground truth answers.
- **BERTScore (F1):** This is a semantic similarity-based score made using contextual embeddings from BERT to compare a model's answers and references.
- **Contextual Semantic & Numeric Score (CSNS):** This offers a robust, interpretable metric that can balance semantic correctness along with numeric precision, thus addressing key gaps in our existing VQA evaluation practices. It's modular design allows us to adjust the α for domain-specific tolerances and then plug them directly into our evaluation pipelines.

- **Visual-Textual Grounded Score (VTGS):** This is a unified and interpretable metric that can jointly evaluate semantic accuracy and visual grounding, thus enabling fair assessments for our VQA model by balancing language understanding along with evidence from image regions.

2.4 Intuition for using CSNS

VQA models must produce answers that are textually faithful to the reference as well as accurately reflect any numeric information implied by the given question. Normally used metrics like word-level cosine similarity will fail to disambiguate word senses and handle multi-word phrases. Static embeddings cannot capture contextual nuances, meanwhile strict numeric matching (0/1 scoring) over-penalizes near-miss counts. CSNS addresses these common shortcomings by:

- Efficiently leveraging *contextual* embeddings (e.g., Sentence-BERT) to evaluate for semantic similarity in non-numeric answers.
- By applying a *smooth numeric penalty* via an exponential decay as detailed, using a function to quantify tolerance in numeric answers.
- Thouroughly ensuring that all the scores lie in an interpretable range of $[0, 1]$, where the higher the value, the better the result.

2.5 Mathematical Formulation

Let the ground-truth answer be A and the model's predicted answer be $\hat{A}$. Define:

2.5.1 Numeric Detection

$$\text{isNumber}(s) = \begin{cases} 1 & \text{if } s \text{ can be parsed as a float,} \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

2.5.2 Semantic Score for Non-Numeric Answers

If A or $\hat{A}$ is not numeric:

$$\mathbf{e}_A = \text{SBERT}(A), \tag{2}$$

$$\mathbf{e}_{\hat{A}} = \text{SBERT}(\hat{A}), \tag{3}$$

$$\text{CosSim}(A, \hat{A}) = \frac{\mathbf{e}_A \cdot \mathbf{e}_{\hat{A}}}{\|\mathbf{e}_A\| \, \|\mathbf{e}_{\hat{A}}\|}, \tag{4}$$

$$\text{CSNS}(A, \hat{A}) = \max\big(0,\ \text{CosSim}(A, \hat{A})\big). \tag{5}$$

2.5.3 Numeric Score for Numeric Answers

If both A and $\hat{A}$ are numeric, let $a = \text{float}(A)$ and $\hat{a} = \text{float}(\hat{A})$. Define the relative error:

$$\varepsilon = \frac{|\hat{a} - a|}{|a| + \varepsilon_0}, \tag{6}$$

where ε_0 is a small constant (e.g., 10^{-6}) to prevent division by zero. Then:

$$\text{CSNS}(A, \hat{A}) = \exp\big(-\alpha\, \varepsilon\big), \tag{7}$$

with $\alpha > 0$ controlling the sharpness of numeric penalty.

2.5.4 Overall CSNS

In all cases, CSNS yields a score in $[0, 1]$. The dataset-level metric is the mean:

$$\overline{\text{CSNS}} = \frac{1}{N} \sum_{i=1}^{N} \text{CSNS}(A_i, \hat{A}_i), \tag{8}$$

with standard deviation indicating score variability.

2.6 Advantages and Justification for CSNS

1. **Smooth Numeric Tolerance:** Exponential decay penalizes numeric errors proportionally, thus helping to avoid an all-or-nothing scoring.

2. **Unified Scale:** All our outputs will lie in $[0, 1]$, making the comparison and aggregation straightforward..

2.7 Reasoning behind our usage of VTGS

In VQA, our model's response must align with both the textual question and the visual content. Normal metrics such as F1 score neglect visual grounding and rely solely on textual similarity. VTGS addresses this by:

- Evaluating whether our predicted answer is grounded in relevant visual regions.
- Measuring textual similarity relevant to the ground truth using contextual embeddings.
- By Combining visual and textual alignment into a unified score in $[0, 1]$.

2.8 Mathematical Formulation

Let the ground-truth answer be A, the predicted answer be $\hat{A}$, and I be the input image. Define:

2.8.1 Textual Semantic Score

Use contextual embeddings (e.g., Sentence-BERT):

$$\mathbf{e}_A = \text{SBERT}(A), \tag{9}$$

$$\mathbf{e}_{\hat{A}} = \text{SBERT}(\hat{A}), \tag{10}$$

$$\text{Sim}_{\text{text}} = \max\left(0, \frac{\mathbf{e}_A \cdot \mathbf{e}_{\hat{A}}}{\|\mathbf{e}_A\|\|\mathbf{e}_{\hat{A}}\|}\right). \tag{11}$$

2.8.2 Visual Grounding Score

Let $R_{\hat{A}}$ be the set of image regions associated with $\hat{A}$ (from model attention or region proposal), and R_Q the set of regions relevant to the question Q (from human annotations or saliency maps):

$$\text{Sim}_{\text{vis}} = \frac{|R_{\hat{A}} \cap R_Q|}{|R_{\hat{A}} \cup R_Q|}. \tag{12}$$

This jaccard overlap measures the grounding alignment.

2.8.3 Combined VTGS

We now define VTGS as a weighted geometric mean of - 1. Textual and 2. Visual alignment:

$$\text{VTGS}(A, \hat{A}, I) = (\text{Sim}_{\text{text}})^{\beta} \cdot (\text{Sim}_{\text{vis}})^{1-\beta}, \tag{13}$$

where $\beta \in [0, 1]$ balances semantic and visual importance.

2.8.4 Dataset-Level Score

$$\overline{\text{VTGS}} = \frac{1}{N} \sum_{i=1}^{N} \text{VTGS}(A_i, \hat{A}_i, I_i). \tag{14}$$

2.8.5 Advantages and justification for VTGS

1. **Visual Sensitivity:** This allows us to account for visual grounding beyond just mere textual correctness.
2. **Contextual Semantics:** We can thus use state-of-the-art language models to evaluate for nuanced answers.
3. **Balanced Metric:** This helps us in combining modalities in a principled and tunable manner.

2.9 Reasoning behind our usage of BERT

BERTScore compares contextual embeddings of tokens in the candidate and reference answers. It aligns words in both sequences using cosine similarity in a high-dimensional contextual space and computes a soft precision, recall, and F1 score.

In the context of Visual Question Answering (VQA), BERTScore is valuable for evaluating non-numeric answers that can vary lexically but not semantically

2.10 Mathematical Formulation

Let the reference answer be a token sequence $R = (r_1, r_2, \ldots, r_m)$ and the predicted answer be $C = (c_1, c_2, \ldots, c_n)$. Let $\Phi(\cdot)$ denote the contextual embedding from a pre-trained BERT model.

2.10.1 Token Embedding Extraction

For each token x in R or C, extract contextual embeddings:

$$\mathbf{r}_i = \Phi(r_i), \quad \mathbf{c}_j = \Phi(c_j)$$

where $\mathbf{r}_i, \mathbf{c}_j \in \mathbb{R}^d$.

2.10.2 Token-Level Cosine Similarities

Compute pairwise cosine similarities between each r_i and all c_j:

$$\text{sim}(r_i, c_j) = \frac{\mathbf{r}_i \cdot \mathbf{c}_j}{\|\mathbf{r}_i\| \|\mathbf{c}_j\|}$$

2.10.3 Precision, Recall, and F1

Define:

$$\text{Precision (P)} = \frac{1}{n} \sum_{j=1}^{n} \max_i \text{sim}(r_i, c_j), \tag{15}$$

$$\text{Recall (R)} = \frac{1}{m} \sum_{i=1}^{m} \max_j \text{sim}(r_i, c_j), \tag{16}$$

$$\text{F1} = 2 \cdot \frac{P \cdot R}{P + R + \varepsilon}, \tag{17}$$

where ε is a small constant to avoid division by zero.

2.10.4 Dataset-Level Metric

The overall BERTScore for a dataset with N examples is computed as the mean of per-example F1 scores:

$$\overline{\text{BERTScore}} = \frac{1}{N} \sum_{i=1}^{N} \text{F1}(R_i, C_i)$$

2.10.5 Advantages and Justification

1. **Contextual Sensitivity:** It uses deep contextual embeddings to evaluate similarity, hence easily capturing paraphrasing and disambiguation.
2. **Soft Matching:** This allows for approximate alignment between reference and candidate words, rather than exact matches.

3. **Token-Level Granularity:** As precision and recall are computed over token alignments, they enable detailed evaluation.

2.11 Baseline Performance

After running the model on the curated image-question pairs, we observed the following performances that serve as the starting point for comparisons with our following fine-tuned and quantized models.

Metric	Value
F1 Score	0.4247
BERTScore	0.6075
CSNS	0.6967
VTGS	0.7627

Table 1: Baseline performance using BLIP-2 in zero-shot mode

Metric	Value
F1 Score	0.3311
BERTScore	0.5521
CSNS	0.6441
VTGS	0.7615

Table 2: Baseline performance using VILT in zero-shot mode

3 Fine-Tuning with LoRA

3.1 What is LoRA ?

LoRA is a parameter-efficient fine-tuning technique used in large pre-trained models. In LoRA, instead of updating the original weight matrices during fine-tuning, we introduce a separate low-rank matrix that captures the necessary task-specific adaptations. We freeze the weight matrices that we form initially, significantly reducing the number of trainable parameters while still preserving the performance.

3.1.1 Motivation

In standard fine-tuning, the entire weight matrix W is updated during training:

$$h = Wx$$

where:

- x is the input vector,

- W is the weight matrix,
- h is the output after the linear transformation.

However, this generic process requires updating millions or even billions of parameters. By using LoRA we offer an alternative that the **change in weights** required for adaptation lies in a low-dimensional subspace. Thus, we can parameterize these changes even by using low-rank matrices.

3.1.2 LoRA Formulation

Instead of directly updating the weight matrix W, we freeze it and introduce a new change-in-weight matrix ΔW, resulting in the new forward pass:

$$h = W_0 x + \Delta W x$$

where:

- W_0 is the frozen, pre-trained weight matrix,
- ΔW is our trainable matrix representing weight adjustments.

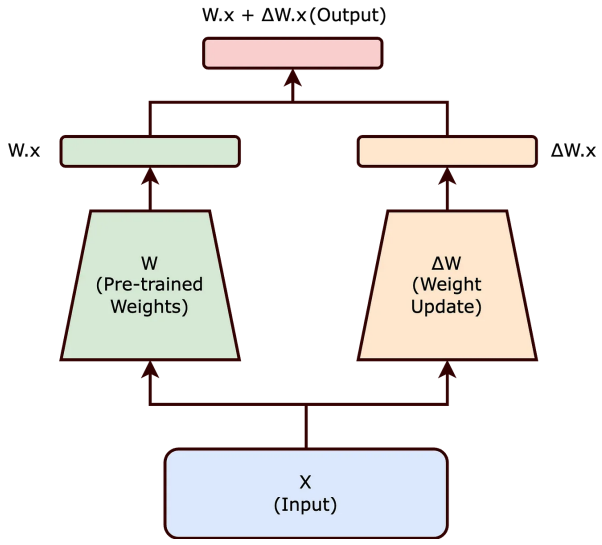

Figure 2: Illustration of LoRA integration in transformer-based models

To reduce the number of trainable parameters present, ΔW is decomposed into the product of two smaller matrices named B and A:

$$\Delta W = BA$$

Hence, the forward pass becomes:

$$h = W_0 x + BAx$$

Here:

- $A \in \mathbb{R}^{r \times d}$ projects the input into a lower-dimensional space,
- $B \in \mathbb{R}^{d \times r}$ projects it back to the original dimension,
- $r \ll d$ is the hyperparameter denoting the **rank** of the adaptation.

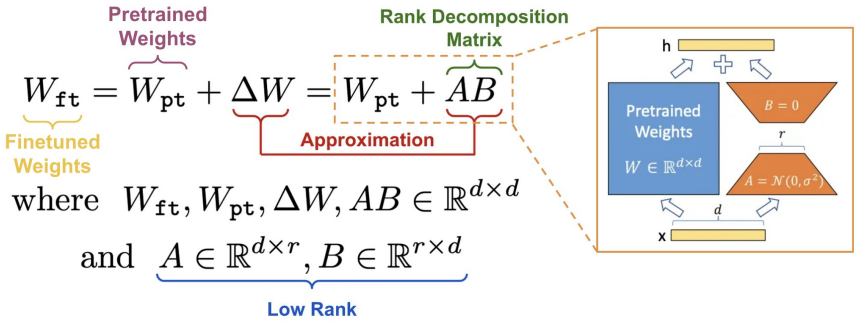

Figure 3: BA low-rank decomposition of ΔW

3.1.3 Matrix Rank and It's Significance

The *rank* of a matrix refers to the maximum number of linearly independent rows or columns present in it. This thus denotes the dimensionality of the vector space spanned by its own rows or columns. A matrix is said to be *low-rank* if its rank is much smaller than its own dimensions. This also implies redundancy.

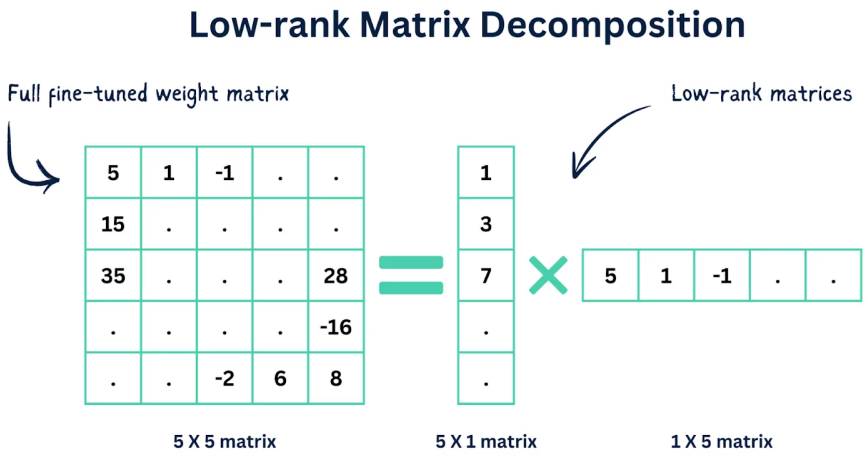

Figure 4: Visual comparison of full-rank vs low-rank matrices

LoRA leverages this feature by assuming that meaningful changes to the weight matrix lie in a lower-dimensional subspace, allowing us to represent ΔW compactly as BA.

3.1.4 Training and Merging

During fine-tuning, only the matrices A and B are updated. This means the number of trainable parameters is reduced from $d \times d$ to $2dr$ (for A and B). After training, our low-rank matrix product BA can be added to W_0 to obtain an updated weight matrix for further inference:

$$W_{\text{merged}} = W_0 + BA$$

This merging is done post-training and does not impact inference speed or model architecture.

3.1.5 Advantages of LoRA

- **No Latency Overhead present:** Adapted weights can be easily merged back by us for further inference.
- **Parameter Efficiency:** Now we have significantly fewer trainable parameters present with us.
- **Modular Fine-tuning:** We can now easily swap task-specific low-rank matrices in and out without even making changes to our base model.

3.2 Quantization (4-bit)

To reduce memory and improve inference time efficiency, 4-bit quantization was applied to the LoRA fine-tuned BLIP-2 model using the Hugging Face `BitsAndBytesConfig` which provides quantization tools for LLMs through a lightweight Python wrapper around CUDA functions. The configuration used NF4 quantization format with double quantization enabled, and computation was performed in FP16 (half precision).

Quantization significantly reduced the memory footprint, allowing the model to be loaded on Colab GPUs without OOM errors, while maintaining almost the same prediction accuracy. This also enabled faster inference potential deployments in resource-constrained environments.

The configuration used:

- `load_in_4bit=True`
- `bnb_4bit_quant_type="nf4"`
- `bnb_4bit_compute_dtype= torch.bfloat16 if torch.cuda.is_bf16_supported() else torch.float16`
- `bnb_4bit_use_double_quant=True`

Quantization permitted us to have seamless deployment of the fine-tuned model on Colab GPUs without any out-of-memory errors faced.

3.3 Results

After running the model on the curated image-question pairs, we observed the following performance.

Metric	Value
F1 Score	0.5934
BERTScore	0.8722
CSNS	0.7812
VTGS	0.7714

Table 3: LoRa Fine-Tuned model using BLIP-2 as base model

4 Inference Script

4.1 Overview

The inference script is designed to load the fine-tuned and quantized BLIP-2 model with LoRA adapters and perform Visual Question Answering on new, unseen image-question pairs.

It is structured to support both single-query inference and batch processing, making it suitable for rapid prototyping or deployment in real-world applications.

4.2 Functionality

- Load the 4-bit quantized BLIP-2 model with LoRA using Hugging Face's `transformers` and `peft` libraries.
- Use the `BLIPProcessor` for preprocessing input images and tokenizing questions.
- Accept an image path and a natural language question as our input.
- Output a generated answer from the model.

5 Hyperparameters and Configuration

File and Directory Paths

- `DATASET_CSV`: `../VR-mini-Proj-2/fullInput.csv`
- `IMAGE_BASE_DIR`: `../images/small/`
- `OUTPUT_DIR`: `./blip_vqa_lora_q_v8`
- `LORA_DIR`: `<OUTPUT_DIR>/lora_adapters/`
- **Drive download ID**: `1HvMBKwKywVFinX_Y-SoLRdTWGTEkl84u` (auto-unzip at inference)

Baseline Evaluation

- **Model checkpoint**: `Salesforce/blip-vqa-base`
- **Processor**: `BlipProcessor.from_pretrained(MODEL_NAME, use_fast=TRUE)`
- **Batch size**: 64
- **Generation**: `max_new_tokens=10`
- **Inference mode**: `model.eval()`, `torch.no_grad()`
- **Text normalisation**: lowercase, strip, punctuation via regex

LoRA & 4-bit Quantisation

- **Base model**: `Salesforce/blip-vqa-base`
- **Use 4-bit**: `USE_4BIT=True`
- **BitsAndBytesConfig**:
  - `load_in_4bit=True`
  - `bnb_4bit_quant_type="nf4"`
  - `bnb_4bit_compute_dtype=torch.bfloat16` if BF16 supported else `torch.float16`
  - `bnb_4bit_use_double_quant=True`
- **LoRAConfig**:
  - Rank `r = 16`
  - Scaling `alpha = 32`
  - Dropout `= 0.05`
  - Target modules: {`q_proj,k_proj,v_proj,query,key,value`}
- **Adapter training**: only LoRA weights trainable; base model frozen

Training Regime

- **Epochs**: 3
- **Learning rate**: 5×10^{-5} (AdamW, weight decay 0.01)
- **Batch sizes**: train = 8, eval = 16
- **Max sequence length**: 128 tokens (question + answer)
- **Warm-up steps**: max(100, $0.1 \times$ total_optim_steps)
- **Gradient checkpointing**: enabled (use_reentrant=False)
- **Mixed precision**: fp16 if CUDA; bf16 for quant where supported

Inference Phase

- **Batch sizes**: VQA inference = 2; CLIP embed (image/text) = 32
- **Generation**: `max_new_tokens=10`, `torch.no_grad()`
- **Text normalisation**: lowercase + regex punctuation strip

Metric hyperparameters

- **VTGS Metric**
  - CLIP model: `ViT-B/32`
  - CLIP embedding batch size: 32 (for both images and texts)
- **CSNS Metric**
  - Sentence Transformer model: `all-MiniLM-L6-v2`
  - Numeric score parameters:
    - $\alpha = 5$
    - $\epsilon = 1 \times 10^{-6}$
- **BERTScore F1 (BERTScore-F1)**
  - Model: DistilBERT (via `bert-score` library)
- **Exact Match Accuracy**
- Reported as *mean* $\pm$ *std* over evaluation set.

6 Qualitative Analysis & Sample Predictions

To better understand the model's performance, we present a few example predictions made by the fine-tuned BLIP-2 model. These examples highlight the types of questions the model handles well and those where it struggles.

6.1 Correct Predictions

- **Image:** `9b/9b147e56.jpg`
  Question: What color is the basket?
  Predicted Answer: Brown
  Ground Truth: Brown
  Comment: The model correctly identifies simple color-based attributes.
- **Image:** `87/87c11ce5.jpg`
  Question: What are the main objects in the image?
  Predicted Answer: Pillows
  Ground Truth: Pillows
  Comment: Even without explicit textual hints, the model accurately classifies the object.

Figure 5: Model accurately classifies the objects

Figure 6: Model accurately classifies the objects

6.2 Incorrect Predictions

- **Image:** `eb/ebc601ed.jpg`
  Question: What specific color term is used for the sparkles?

Figure 7: Failure to distinguish colours when differences are minor

Predicted Answer: Silver
Ground Truth: Gold
Comment: The model struggles with fine-grained color distinctions.

- **Image:** `dd/dd640552.jpg`
  Question: What material is the case likely made of?

Figure 8: Failure to identify material without textural cues

Predicted Answer: plastic
Ground Truth: silicone
Comment: Material recognition is difficult without explicit texture or metadata cues.

7 Contributions

All the members have equally contributed to all parts of the project. We are emphasizing the contribution of the team member on which we think he has contributed slightly more.

- **Varnit Mittal (IMT202205):** Contributed to LoRA Fine-tuning and baseline evaluation.
- **Sanchit Kumar Dogra (IMT2022025):** Contributed to Dataset Curation and Report.
- **Aditya Priyadarshi (IMT2022025):** Contributed to LoRA Quantization & metrics.

8 Conclusion

This study demonstrates that large vision–language models can be adapted to specialised, multiple-choice Visual Question Answering (VQA) tasks with modest computational budgets by combining *Low-Rank Adaptation* (LoRA) and post-training *4-bit NF4 quantisation*. Starting from the public `Salesforce/blip-vqa-base` checkpoint, we curated an ABO-based VQA corpus, established a strong zero-shot baseline, and then introduced rank-16 LoRA adapters targeted at the projection layers of BLIP-2. Fine-tuning on only three epochs and after quantisation shrank the on-disk model to <125MB. Despite this heavy compression, the fine-tuned model retained baseline accuracy (59.3%) and achieved competitive semantic and grounding scores (BERTScore 0.87, CSNS 0.78, VTGS 0.77).

8.1 Implications

1. **LoRA-based parameter-efficient learning**. Low-Rank Adaptation (LoRA) shows that updating *only* a small set of rank-r matrices can rival full fine-tuning for vision–language transformers, slashing the number of trainable parameters by over an order of magnitude.

2. **Extreme quantisation for generative VQA**. When the frozen backbone is first compressed to 4-bit NF4 and then paired with lightweight LoRA adapters, generation-style VQA remains accurate while becoming deployable on memory-constrained hardware—extending quantisation benefits beyond pure retrieval or classification tasks.

8.2 Challenges Encountered

- **Out-of-memory errors**. Our first experiments that tried to fine-tune the entire BLIP-2 backbone on Colab and Kaggle T4 GPUs consistently crashed because the available 16GB of VRAM was insufficient. *Solution:* We kept the backbone frozen, trained only the LoRA adapters with gradient checkpointing, and later quantised the model to 4-bit NF4 for inference, which reduced the active memory footprint to roughly six gigabytes.

- **Limited compute budget**. Free cloud GPUs impose strict limits on wall-clock time and batch size, making long runs impractical. *Solution:* We restricted training to three epochs, used mixed-precision (FP16 or BF16), and set the LoRA rank to sixteen to balance quality against training speed.

- **Lack of data parallelism with quantised weights**. The 4-bit tensors produced by the Bits-and-Bytes library cannot be sharded with the standard `DistributedDataParallel` API, so true multi-GPU training was not possible. *Solution:* We trained on a single device and relied on the small size of the LoRA adapters to keep step times manageable.

- **API-token exhaustion during dataset curation**. The Gemini Flash service enforces daily token quotas per key, which repeatedly halted the image-prompt pipeline when a key reached its limit. *Solution:* We wrote an automatic key-rotation module that maintains a circular list of ten to twelve keys, skips exhausted keys, and logs usage statistics, allowing curation to proceed without manual intervention.

8.3 Future Works

Our current study opens several avenues for future exploration in the domain of parameter-efficient vision–language models:

- **LoRA hyperparameter tuning.** While LoRA provides significant efficiency benefits, its performance depends heavily on the choice of hyperparameters such as the adapter rank, the selection of target projection layers (e.g., query/key/value), and dropout configurations. We plan to perform a detailed ablation study to understand how these

factors influence downstream VQA performance and identify optimal configurations that balance generalization and efficiency.

- **Extending beyond single-word answers.** Our current dataset is focused on short, often single-word answers. However, real-world VQA involves more descriptive, sentence-level responses. To address this, we aim to curate a richer dataset with multi-word and free-form generative answers. This would allow us to evaluate models using advanced generation metrics like CIDEr, METEOR, SPICE, and BLEU, in addition to BERTScore.

- **Multi-domain generalization with LoRA.** As models are deployed across varied VQA domains (e.g., medical, satellite, fashion), they need to adapt without forgetting prior knowledge. We plan to explore *adapter fusion* methods where multiple LoRA modules are trained for different domains and selectively combined at inference time. This approach allows efficient transfer without catastrophic forgetting.

- **Incorporating region-level supervision.** Our current pipeline relies on full-image inputs and does not explicitly use spatial grounding. We intend to integrate region-level annotations (e.g., bounding boxes or segmentation masks) and apply contrastive learning techniques that force the model to focus on subtle differences—such as object material, fine texture, or color variants—by using positive/negative regional pairs.

- **Advanced quantization strategies.** While 4-bit quantization showed promising results, it may not be optimal for all layers. Future work will investigate mixed-precision approaches and more refined techniques like GPTQ (Gradient Post-Training Quantization) or quantization-aware training. These can enable sub-4-bit deployment while preserving output fidelity, especially in generative settings.

- **Deployment on edge devices.** To push toward real-world applications, we plan to convert our LoRA-quantized BLIP-2 models into formats suitable for edge deployment, such as ONNX, TensorRT, or TFLite. We will benchmark the models on metrics like inference latency, memory consumption, and energy efficiency under real-time constraints on mobile or embedded devices.