AIM 825 - Visual Recognition: End-Term Project Report

Samyak Jain (IMT2022071, Samyak.Jain@iiitb.ac.in)
Shashwat Chaturvedi (IMT2022118, ShashwatChaturvedi@iiitb.ac.in)
Nupur Dhananjay Patil (IMT2022520, Nupur.Patil@iiitb.ac.in)

May 18, 2025

Abstract

This report presents our end-term project for AIM 825, focusing on curating a Visual Question Answering (VQA) dataset from the Amazon Berkeley Objects (ABO) dataset, evaluating baseline models, and fine-tuning chosen models using Low-Rank Adaptation (LoRA). We detail dataset curation, model performance, and optimization strategies, achieving significant improvements. We also did additional experiments with KV Cache and Flash Attention enhanced inference efficiency.

Contents

1 Dataset Curation

1.1 Objective

The primary goal was to construct a high-quality, balanced Visual Question Answering (VQA) dataset from product images and their associated metadata. Each image is accompanied by three question-answer (QA) pairs, corresponding to three difficulty levels:

- **Easy**: Based on basic visual attributes (e.g., color, shape).
- **Medium**: Based on finer visual details or patterns (e.g., texture, brand symbols).
- **Hard**: Requiring logical inference (e.g., usage, compatibility).

All answers are constrained to single-word responses to ensure concise and structured outputs.

1.2 Source Data

We utilized a curated subset of the smaller version of the Amazon Berkeley Objects (ABO) dataset, consisting of:

- RGB product images.
- Metadata for each product image, including multilingual entries across many fields. We used selected fields relevant for question generation.

First, we combined all the JSON files into a single file named `listings.json`. Each JSON entry contained `main_image_id` and `other_image_id` fields. We then created a dictionary that mapped each image ID (from both the `main_image_id` and `other_image_id` fields) to the corresponding metadata JSON object. This dictionary was uploaded to Kaggle for later use. This approach prevents the need to search through the entire JSON file to find the metadata associated with a particular image ID, thus improving lookup efficiency. Then, this metadata was extracted and filtered to ensure it was:

- **English-only**: Many fields contained multilingual entries. We implemented `extract_english_value()` to retrieve only entries tagged with language codes starting with `en_`.
- **Readable and formatted**: Nested dictionary fields such as `item_dimensions` were normalized into human-readable strings.
- **Cleaned and concise**: Markdown artifacts, excessive whitespace, and non-alphanumeric edge characters were removed using a custom `clean()` function, standardizing all input and output text.

1.3 Prompt Design

A carefully constructed prompt was designed to guide the generative model (Gemini 1.5 Flash) toward producing high-quality QA pairs. The prompt included the following components:

- **Role specification:** "You are a Visual QA dataset assistant..."
- **Answer format constraint:** Single-word answers to enforce uniformity.
- **Difficulty-level guideline:** A description of what constitutes Easy, Medium, and Hard questions.

- **Metadata injection:** A block of formatted product metadata appended to the prompt to help ground the QA generation in product-relevant context. The following metadata fields were used to aid the prompt:

  ```
  brand, bullet_point, color, fabric_type, finish_type, item_dimensions,
  item_keywords, item_shape, item_weight, material, model_name, pattern,
  product_description, style
  ```

  These fields were selected because they contained enough information to prevent hallucinations and allow the model to ask more useful questions that might not be possible with just the image as a prompt. Other fields did not contain meaningful information for the prompt.

Example prompt format:

> You are a Visual QA dataset assistant: for each image, generate exactly three question-answer pairs, with a single word as the answer—covering Easy (basic visual attributes), Medium (advanced visual features), and Hard (logical inferences) difficulty levels. Ensure diversity in question types, and make sure all questions are answerable solely from the image.
>
> Product metadata:
> Brand: Adidas
> Color: Black
> Item Dimensions: Height: 10 cm, Width: 20 cm
> Style: Sporty
> ...

1.4 Model Selection and Rationale

We selected **Gemini 1.5 Flash** via the Google Generative AI API based on a balance of performance, accessibility, and practical constraints. Table 1 summarizes the rationale.

Table 1: Rationale for selecting Gemini 1.5 Flash for VQA generation.

Factor	Rationale
Token Efficiency	Offered 1500 API calls/day, highest among free-tier models. Gemini 2.5 Flash provides only 500/day, and Gemini 2.0 Flash-Lite lacks a usable model string.
Latency vs Cost	Balanced speed and quality. Local inference (e.g., LLaVA/BLIP via Ollama) was too slow (3 min/image on CPU, no team GPU).
Multimodal Capability	Supported image + text input, crucial for image-grounded VQA generation.

1.5 Generation Workflow

The VQA curation pipeline was designed to balance throughput, stability, and API rate limits. The core generation loop followed these steps:

1. Load image from dataset.
2. Clean metadata and format.

3. Generate a prompt using the image and metadata.
4. Call `model.generate_content([image, prompt])`.
5. Parse output using regex to extract exactly 3 QA pairs.
6. Clean and store results in a CSV buffer.

Rate Limiting Strategy

To comply with the Gemini 1.5 Flash API limits (5 requests/min), the following throttling measures were implemented:

- A sleep interval of 60 seconds was introduced after processing every 4 images (`pause_every_n_images = 4`) to avoid hitting the minute limit.
- A maximum of 1500 images were processed per session, matching the daily quota.

CSV Output

The output was saved incrementally to a CSV file with the following fields:

Table 2: CSV Output Format for Curated VQA Samples

Field	Description
`image_id`	Identifier for the source image
`question`	Generated question from Gemini
`answer`	Corresponding answer from Gemini
`difficulty`	Annotated difficulty level (easy, medium, hard)

1.6 Quality Control

To ensure robustness in data generation and integrity of the final dataset, multiple control mechanisms were implemented:

- **Skipping rules:** The script skipped any image that failed to open, lacked valid metadata, or did not yield valid QA pairs from the model.
- **Error logging:** Skipped cases printed detailed messages to the console to aid in debugging and ensure traceability.
- **Regex validation:** The raw response from Gemini was parsed using a carefully constructed regular expression to extract question-answer pairs and strip any extraneous markup or artifacts.

1.7 Post Processing

After we had compiled all the generated csv's, some answers still contained multiple words, so we just wrote a python script to just take the first word from all the answers to ensure one word answers. This did not lead to any semantic loss because we observed that whenever the answers were of more than one word, the words after the first word were usually just explanations from the model as to why it gave that as that answer.

1.8 Dataset Size and Splits

64408 QA pairs were generated and stored in `VQADataset.csv`. The data was then randomly split into training (80%) and test (20%) sets for inference and training (used random seed 118)(for reproducability).

Table 3: Sample QA pairs from `VQADataset.csv`.

Image Path	Question	Answer	Difficulty
"41PBKVD918L"	"What is the primary color of the swing"	"Green"	"Easy"
"41PBKVD918L"	"What material appears to be used for the swing's canopy"	"Nylon"	"Medium"
"41PBKVD918L"	"What is the main purpose of this product"	"Relaxing"	"Hard"

2 Baseline Model Evaluation

2.1 Model Choices

We evaluated four prominent Vision-Language models using our curated QA dataset via `scripts/evaluation.i`

- **BLIP-VQA-Base** — 380M parameters; pretrained specifically for VQA.
- **ViLT** — 117M parameters; a compact transformer with no CNN backbone.
- **Granite-Vision (IBM)** — 2B parameters; part of IBM's Granite series for vision-language understanding.
- **BLIP2-FLAN-T5-XL** — a large, generative model combining BLIP2 with FLAN-T5 for multi-modal tasks.

2.2 Evaluation Metrics

We used multiple metrics to capture different aspects of model performance, both in terms of exact matches and semantic similarity:

- **Exact Match Accuracy:** Measures strict correctness by comparing the predicted answer exactly with the ground truth.
- **Exact Match Precision:** Precision for exact string matches.
- **Exact Match Recall:** Recall for exact string matches.
- **Exact Match F1:** F1 score combining exact match precision and recall.
- **Substring Match Accuracy:** Considers whether the prediction appears as a substring in the ground truth.
- **Substring Match Precision:** Precision for substring-level matches.
- **Substring Match Recall:** Recall for substring-level matches.
- **Substring Match F1:** F1 score for substring-based evaluation.
- **Token-level Macro F1:** Computes the F1 score at the token level and averages across all classes (macro-average).

- **BERTScore F1:** Evaluates semantic similarity using contextualized embeddings from BERT.
- **ROUGE-L F1:** Measures the longest common subsequence between predicted and reference answers.
- **BLEU Scores:** Evaluates n-gram overlaps between predicted and ground-truth answers.
  - BLEU-1: Unigram overlap.
  - BLEU-2: Bigram overlap.
  - BLEU-3: Trigram overlap.
  - BLEU-4: Four-gram overlap.

2.3 Results and Comparison

Table 4 summarizes the most relevant metrics. **BLIP-VQA-Base** achieved the best performance in both exact and semantic evaluations, likely due to its VQA-specific pretraining. **ViLT** was a strong lightweight contender, while **Granite-Vision** showed good substring-level matching but failed at exact matches. **BLIP2-FLAN-T5-XL** underperformed overall, likely due to its generative tendencies not being optimal for single-word answers.

Table 4: Comprehensive baseline evaluation metrics for VQA models.

Metric	BLIP-VQA-Base	ViLT	Granite-Vision	BLIP2-FLAN-T5-XL
Exact Match Accuracy (%)	34.96	24.10	0.00	4.30
Exact Match Precision (%)	100.00	100.00	0.00	100.00
Exact Match Recall (%)	34.96	24.10	0.00	4.30
Exact Match F1 (%)	51.80	38.84	0.00	8.24
Substring Match Accuracy (%)	36.23	24.55	50.44	17.07
Substring Match Precision (%)	100.00	100.00	100.00	100.00
Substring Match Recall (%)	36.23	24.55	50.44	17.07
Substring Match F1 (%)	53.19	39.43	67.05	29.16
Token-level Macro F1 (%)	35.51	24.25	2.58	9.72
BERTScore F1 (%)	95.40	95.86	78.59	82.90
ROUGE-L F1 (%)	36.48	24.85	2.43	9.75
BLEU-1 (%)	32.41	23.90	0.85	4.10
BLEU-2 (%)	0.48	0.42	0.02	0.11
BLEU-3 (%)	0.10	0.09	0.00	0.03
BLEU-4 (%)	0.04	0.03	0.00	0.01

2.4 Conclusion & Insights

Our evaluation across four diverse VQA models reveals distinct strengths and weaknesses, as summarized below:

- **BLIP-VQA-Base**:
  - Demonstrates the best overall performance in terms of exact match and F1 score.
  - Aligns well with the expected single-word answer format, making it highly suitable for our dataset.
- **ViLT** and **BLIP2**:
  - Produce semantically meaningful answers with high BERTScore.

  - However, their generative nature often leads to verbose or multi-word answers, reducing exact match accuracy.

- **Granite-Vision**:
  - Shows strong performance in substring matching.
  - Fails to provide exact or token-level accurate answers, suggesting a need for further fine-tuning on task-specific data.

- **Evaluation Strategy**:
  - Our metric suite (Exact Match, Substring Match, Token-level F1, BERTScore, BLEU, and ROUGE) enables a multi-angle assessment.
  - This holistic view captures both strict correctness and semantic alignment with human expectations.

3 Fine-Tuning with LoRA

3.1 Overview

This section outlines the fine-tuning process of two pre-trained models—ViLT and BLIP-VQA-Base—using Low-Rank Adaptation (LoRA) on a custom VQA dataset curated from the Amazon-Berkeley Objects (ABO) dataset. The objective was to enhance model performance for our domain-specific VQA task while maintaining efficiency in terms of memory and compute usage.

3.2 Model Selection

We selected two medium-sized models aligned with our resource constraints:

- **ViLT (dandelin/vilt-b32-finetuned-vqa)**: A transformer-based model for vision-and-language tasks with a fixed vocabulary of 3129 answers. It uses a classification head to predict answers by scoring each entry in the vocabulary. Approximate parameter count: 117M (470MB model).
- **BLIP-VQA-Base (Salesforce/blip-vqa-base)**: A generative model that produces answer text autoregressively instead of classifying from a fixed vocabulary, offering more flexibility but at a higher computational cost. Approximate parameter count: 385M (1.54GB model).

Both models were loaded using the Hugging Face `transformers` library. LoRA enabled us to fine-tune these models efficiently by adapting only a subset of weights.

3.3 Dataset and Preprocessing

A custom dataset was constructed using a subset of the ABO dataset. Separate preprocessing pipelines were used for each model:

- **ViLT:** A custom `VQADataset` class was implemented to:
  - Load and resize images to 384×384 pixels.
  - Tokenize questions (max length: 40).
  - Encode answers into binary vectors matching the 3129-label vocabulary.
  - Log unmapped answers for analysis in `unmapped_answers.log`.

- **BLIP:** The same dataset class was adapted with:
  - Image loading in RGB mode; resizing handled internally by `BlipProcessor`.
  - Autoregressive tokenization of answers (max length: 32) using BLIP's tokenizer.

3.4 Error Handling

To ensure uninterrupted training, any unreadable or missing images were replaced with blank RGB images of appropriate size (384×384 for ViLT, 224×224 for BLIP). This mitigated crashes during data loading.

3.5 Training Configuration

LoRA was configured with:

- `rank = 16`, `lora_alpha = 32`, `lora_dropout = 0.1`
- Learning rate: 5×10^{-5}, batch size: 4, precision: `fp16`
- Training epochs: 3 initially; further experiments explored epochs {3, 5, 7} and ranks {8, 16, 24}

3.6 Results and Observations

Fine-tuning significantly improved BLIP-VQA-Base's performance, while ViLT saw modest gains.

Table 5: Performance of BLIP-VQA-Base before and after LoRA fine-tuning.

Metric	Without LoRA	With LoRA
Exact Match Accuracy (%)	34.96	47.37
Exact Match Precision (%)	100.00	100.00
Exact Match Recall (%)	34.96	47.37
Exact Match F1	51.80	64.29
Substring Match Accuracy (%)	36.23	47.98
Substring Match Precision (%)	100.00	100.00
Substring Match Recall (%)	36.23	47.98
Substring Match F1	53.19	64.85
Token-level Macro F1 (%)	35.51	47.40
BERTScore F1 (%)	95.40	96.53
ROUGE-L F1 (%)	36.48	48.63
BLEU-1 (%)	32.41	47.19
BLEU-2 (%)	0.48	0.59
BLEU-3 (%)	0.10	0.12
BLEU-4 (%)	0.04	0.04

3.7 Why Did BLIP Improve More Than ViLT?

1. Architecture: BLIP has a more expressive design with Q-Former and strong vision-language fusion, making it more adaptable to LoRA. ViLT has a simpler fusion mechanism with fewer effective LoRA injection points.

2. Pretraining: BLIP is pretrained on VQA-style tasks, aligning well with our dataset. ViLT's general-purpose pretraining lacks the same level of semantic grounding.

3. LoRA Compatibility: BLIP's architecture allows LoRA to adapt key components like Q-Former and cross-attention, leading to better gains. ViLT benefits less due to limited capacity and shallow integration.

Table 6: Performance of ViLT before and after LoRA fine-tuning.

Metric	Without LoRA	With LoRA
Exact Match Accuracy (%)	24.10	24.73
Exact Match Precision (%)	100.00	100.00
Exact Match Recall (%)	24.10	24.73
Exact Match F1	38.84	39.66
Substring Match Accuracy (%)	24.55	25.00
Substring Match Precision (%)	100.00	100.00
Substring Match Recall (%)	24.55	25.00
Substring Match F1	39.43	40.00
Token-level Macro F1 (%)	24.25	24.81
BERTScore F1 (%)	95.86	96.23
ROUGE-L F1 (%)	24.85	25.27
BLEU-1 (%)	23.90	24.62
BLEU-2 (%)	0.42	0.31
BLEU-3 (%)	0.09	0.06
BLEU-4 (%)	0.03	0.02

4. Answer Style: BLIP generates concise, accurate answers suited for our short-answer dataset. ViLT often outputs longer, less precise text, lowering exact-match performance.

4 Inference Time Optimization

4.1 KV Cache: Reducing Redundant Computation

To improve inference efficiency for the generative BLIP-VQA-Base model, we investigated the use of KV Cache, which stores intermediate key and value tensors during sequence generation. This reduces redundant computations by reusing cached states across decoding steps, thus accelerating autoregressive answer generation. Since ViLT is a classification-based model, KV Cache is not applicable; therefore, we performed standard inference benchmarking on ViLT to establish a baseline.

BLIP-VQA-Base with KV Cache

- **Model and Data:** Fine-tuned BLIP-VQA-Base tested on 10,000 ABO VQA samples.
- **Inference Setup:** Two runs were conducted:
  1. Without KV Cache (`use_cache=False`)
  2. With KV Cache (`use_cache=True`)
- **Metrics:** Average inference time per sample and answer consistency were measured.

ViLT Inference Benchmark

- **Model and Data:** Fine-tuned ViLT tested on 5,000 ABO VQA samples.
- **Inference Setup:** Two standard inference runs (no KV Cache).
- **Metrics:** Average time per sample and answer consistency recorded.

KV Cache reduced BLIP's inference time by approximately 20% without any loss in answer consistency. ViLT's inference times remained stable and deterministic across runs, consistent with its classification architecture.

Table 7: Inference Performance: BLIP-VQA-Base with and without KV Cache

Configuration	Avg. Time per Sample (s)	Answer Consistency
Without KV Cache	0.1234	10,000 / 10,000 identical
With KV Cache	**0.0987**	10,000 / 10,000 identical

Table 8: Inference Performance: ViLT Standard Inference Benchmark

Run	Avg. Time per Sample (s)	Answer Consistency
Run 1	0.0567	5,000 / 5,000 identical
Run 2	0.0568	5,000 / 5,000 identical

4.2 Flash Attention: Accelerated Attention Mechanism

To further accelerate inference for BLIP-VQA-Base, we implemented Flash Attention using PyTorch's `torch.compile` functionality, which compiles model code into optimized lower-level instructions. This replaced the standard attention mechanism with Flash Attention during inference.

Two BLIP model versions were benchmarked on the ABO VQA dataset:

- **Standard Attention:** Baseline with KV Cache enabled.
- **Flash Attention:** Flash Attention enabled, no KV Cache.

Table 9: BLIP-VQA-Base Inference Time with Standard vs. Flash Attention

Method	Avg. Inference Time per Sample (s)
Standard Attention (with KV Cache)	0.1148
Flash Attention (no KV Cache)	**0.1083**

Flash Attention improved inference speed by approximately 6%, demonstrating a clear efficiency gain over the standard attention mechanism.

4.3 8-bit Quantization: Reducing Memory Footprint and Inference Time

To further optimize inference efficiency, we applied 8-bit quantization to the BLIP-VQA-Base model using the `bitsandbytes` library. This reduced the model's memory footprint and enabled faster computation by loading model weights in 8-bit precision instead of the default 16/32-bit formats.

We used the quantized model in conjunction with the Hugging Face Transformers library by setting `load_in_8bit=True` during model loading. The quantized model was evaluated on the same ABO VQA test set and compared to the standard full-precision model in terms of inference speed, memory usage, and output consistency.

- **Standard Model:** Full-precision BLIP-VQA-Base (FP16/FP32)
- **Quantized Model:** 8-bit version using `bitsandbytes`

The 8-bit quantized model achieved a 20% reduction in inference time and 45% lower memory usage, with a negligible drop in answer consistency. This demonstrates the practical benefits of post-training quantization for deploying VQA models on resource-constrained systems.

Table 10: BLIP-VQA-Base: Inference Performance with 8-bit Quantization

Method	Avg. Inference Time (s)	Peak Memory (GB)	Answer Consistency
Full Precision	0.1121	7.8	10,000 / 10,000
8-bit Quantized	**0.0895**	**4.3**	9,998 / 10,000

4.4 Conclusion

Our experiments highlight the importance of matching optimization techniques to model architecture:

- **KV Cache** significantly accelerates inference in generative models like BLIP-VQA-Base, reducing time by around 20% without affecting output quality.
- **Flash Attention** provides additional speed-ups (6%) by replacing standard attention with a compiled, memory-efficient variant.
- **8-bit Quantization** achieves up to 20% faster inference and 45% memory savings with minimal impact on answer consistency, making it a strong choice for resource-constrained deployment.
- For classification-based models like ViLT, inference is already efficient and deterministic; hence, techniques like KV Cache are not applicable.

These results emphasize the value of selecting inference optimizations based on a model's architecture and use case, enabling better performance and scalability in VQA systems.

5 Inference Script Description

This script performs inference using our fine-tuned BLIP-VQA model with LoRA adapters. It accepts two inputs via command-line arguments: a directory containing images and a CSV file containing corresponding questions and image filenames.

The main steps are as follows:

- **Downloading the LoRA-adapted model:** If not already present locally, the script downloads the fine-tuned LoRA adapter folder (zipped) from Google Drive and extracts it.
- **Loading the base BLIP-VQA model and processor:** It loads the pretrained `Salesforce/blip-vqa-b` model and its processor.
- **Cleaning adapter configuration:** The script reads the LoRA adapter configuration file, retains only the essential keys required for compatibility, and overwrites the config file with this minimal version.
- **Loading the LoRA adapter:** The adapter weights are loaded into the base model via the `PeftModel` wrapper, and the combined model is moved to the appropriate device (GPU if available).
- **Inference loop:** For each row in the input CSV, the script loads the corresponding image, prepares the input encoding (image and question), performs generation to predict the answer, and extracts only the first word (lowercased) to maintain single-word answers.

- **Result saving:** Finally, it appends the generated answers as a new column to the input dataframe and saves the results to `results.csv`.

This pipeline facilitates straightforward evaluation or deployment of our fine-tuned BLIP-VQA model on new image-question pairs without requiring manual setup of the adapter files.

6 Conclusion

This report detailed the creation of a Visual Question Answering dataset derived from the Amazon Berkeley Objects dataset, baseline evaluations of multiple VQA models, and fine-tuning using LoRA. Our best-performing model, BLIP-VQA-Base, achieved a significant accuracy improvement of 47.37%. Future work will focus on further optimizing ViLT, exploring larger datasets, and integrating inference efficiency techniques such as KV Cache and Flash Attention.

Code Repository: `https://github.com/sassy2711/VR_MiniProject2_IMT2022071_118_520`

Link to zip file containing lora weights files for fine-tuned blip-vqa-base:
`https://drive.google.com/file/d/1E5QIO1yiLtJ1uIMRwDZgDiIA0cHkOL6O/view?usp=sharing`