AIM 825: Visual Recognition

Multimodal Visual Question Answering with Amazon Berkeley Objects Dataset

Project Report

Submitted by:

- Shannon Muthanna I B (IMT2022552)
- Aayush Bhargav (IMT2022089)
- Praveen Peter Jay (IMT2022064)

---

1 Introduction

Visual Question Answering (VQA) is an interdisciplinary research problem at the intersection of computer vision and natural language processing (NLP). It involves developing models that can answer questions related to images, combining visual understanding with natural language processing. The task of VQA has gained significant attention due to its wide applications in areas like human-computer interaction, autonomous systems, robotics, and accessibility for visually impaired individuals.

In this project, we aim to contribute to the VQA domain by leveraging the Amazon Berkeley Objects (ABO) dataset, a large-scale collection of images with associated annotations. The primary goal of this assignment is to develop a multiple-choice VQA dataset, which will serve as a benchmark for evaluating baseline models and fine-tuning state-of-the-art approaches.

2 Dataset and VQA Generation Pipeline

2.1 Dataset

The VQA models developed in this project are based on the small version of the **Amazon Berkeley Objects (ABO)** dataset. This dataset includes 398,212 downscaled catalog images (256×256) alongside an `images.csv` file that provides descriptions of the images. Additionally, product metadata was utilized, distributed across sixteen JSON files. These files include attributes such as color, material, product type, keywords, and shape.

2.2 Curation

The ABO dataset is large and inherently imbalanced across product categories. To enable fair experimentation and ensure a representative distribution of product types, a curated and balanced subset was created. The curation involved merging image data from `images.csv` with metadata from the sixteen JSON files in the `listings.tar` archive. The output of this process was a new file named `balanced_dataset.csv`, which includes the following fields:

- `image_path`
- `name`
- `product_type`
- `color`

- keywords

A maximum of 5,000 items per product type was enforced to achieve balance.

2.3 Methodology

The dataset curation was performed using a Python script parser.py, and involved the following steps:

1. Loading Image Data The script loads images.csv, which contains metadata such as image_id and path for each image. A total of 398,212 image IDs were processed and used as primary keys to link image data with metadata.

2. Processing Metadata from JSON Files

- The 16 JSON files in listings/metadata/ were parsed.
- Each file was processed line-by-line, extracting fields like item_name, product_type, color, and item_keywords.
- Products were matched with main_image_id from the image list.
- Metadata was extracted using custom functions:
  - extract_english_value: Retains only English values based on language tags (en, en_).
  - extract_all_keywords: Deduplicates and joins English keywords.
- Metadata was successfully collected for 123,551 images.

3. Creating the Initial Dataset

- Combined image paths with extracted metadata.
- Filtered out:
  - Specific problematic image IDs: 518Dk4FOzZL, 719hoe+OvIL, 71Qbh8wmhnL
  - Entries with missing or non-ASCII metadata.
  - Null values (via drop_nulls=True).
- Final initial dataset: 72,427 entries with 314 unique product types.

4. Balancing the Dataset

- Capped product types to a maximum of 5,000 items.
- Used fixed seed (random_state=42) for reproducibility.
- Reduced dataset to 14,798 entries while retaining 314 unique product types.

5. Saving the Balanced Dataset

- Sorted entries by image_path.
- Saved as balanced_dataset.csv.
- Fields include: path, name, product_type, color, keywords.

2.4 Results and Statistics

- **Total Images Processed:** 398,212
- **Images with Metadata:** 123,551
- **Initial Dataset Size:** 72,427 entries
- **Final Balanced Dataset Size:** 14,798 entries
- **Unique Product Types:** 314

This curation process ensured a high-quality, diverse, and balanced subset suitable for downstream tasks such as Visual Question Answering and product classification.

2.5 VQA Dataset Generation Using Gemini API

Overview

The script `amazon_vqa_generator.py` generates a VQA dataset using Google's Gemini API. It processes `balanced_dataset.csv` to produce `vqa.csv`, which includes image paths, three questions per image, and single-word answers based on both image content and metadata.

```
    prompt = f"""
    I need you to generate 3 diverse Visual Question Answering (VQA) pairs for this product image.

    Product Information:
    - Name: {product_info['name']}
    - Type: {product_info['product_type']}
    - Color: {product_info['color']}
    - Keywords: {product_info['keywords']}

    Instructions:
1. Analyze the provided image and the metadata.,
2. Generate exactly 3 distinct questions about prominent visual features, objects, colors, materials, or attributes clearly visible in the image.,
3. Each question MUST have a single-word answer directly verifiable from the image.,
4. The 3 questions generated MUST be different from each other, and MUST be answerable just by looking at the image.,
5. Vary the difficulty levels across questions, but they should be answerable just by looking at the image.

    Format your response as a JSON array:
    [
        {{"question": [Your first question here], "answer": [Your first answer here]}},
        {{"question": [Your second question here], "answer": [Your second answer here]}},
        {{"question": [Your third question here], "answer": [Your third answer here]}}
    ]

    Only respond with the JSON array.
    """
```

Figure 1: Prompt used for VQA generation

Key Features and Methodology

1. **Gemini API Integration**
   - Uses Gemini 2.0 Flash API for image analysis.
   - Authenticates via API key.
   - Returns 3 question-answer pairs per image in JSON format.
2. **Prompt Design**
   - Instructs the API to generate diverse questions using visual and metadata cues.
   - Enforces single-word answers.
   - Specifies JSON response format for consistent parsing.
3. **Checkpoint System for Collaboration**
   - Saves last processed index to a checkpoint file.
   - Enables resumption and parallel team collaboration.
4. **Managing API Limits**
   - Enforces 4-second intervals to stay within 15 requests/minute limit.
5. **Error Handling and Saving Progress**
   - Retries failed API calls up to three times.
   - Skips missing or faulty data.
   - Saves progress every 5 images or at the end.
6. **API Key Usage**
   - API key is embedded in the script.
   - Typically obtained from Google's API console after project setup.

How It All Fits Together

- The sleep mechanism ensures compliance with API rate limits.
- Checkpoints facilitate robust and team-friendly usage.
- Structured prompts ensure high-quality and predictable output.
- Frequent saving and error handling offer reliability for large-scale dataset creation.

2.6 Conclusion

The `amazon_vqa_generator.py` script offers a reliable and scalable pipeline for generating a VQA dataset. Through intelligent prompt engineering, rate control, checkpointing, and structured output, it effectively processes the balanced ABO dataset to produce high-quality VQA data suitable for model training and evaluation. This pipeline lays a solid foundation for downstream tasks in product-related visual reasoning and analysis.

3 Key Metrics

The evaluation of a VQA model requires various metrics to assess different aspects of the model's performance. Below is a summary of the key metrics used for the evaluation of the various models.

3.1 F1 Score

The F1 Score is the harmonic mean of precision and recall calculated at the token level. It balances false positives and false negatives, offering a robust measure of answer quality, especially in scenarios where predictions are only partially correct. F1 is particularly informative in multi-token answers where token-level granularity matters.

3.2 BERTScore

BERTScore uses contextual embeddings from a pre-trained BERT model to compute precision, recall, and F1 scores. This allows it to evaluate semantic similarity beyond surface-level token matching, capturing subtle differences like synonyms and paraphrases. It is especially valuable for evaluating answers that are semantically valid but lexically different.

3.3 BARTScore

BARTScore assesses the fluency and relevance of generated text using the BART model, based on the negative log-likelihood of the target under a pre-trained generative model. Unlike other metrics, its scores are unbounded and reflect the naturalness and grammatical quality of predictions. It is widely used for evaluating generative language tasks where coherence and fluency are essential.

3.4 METEOR Score

The METEOR metric accounts for synonymy, stemming, and word order when comparing predictions with references. It is more flexible than exact match metrics and is widely used in machine translation for its ability to capture partial matches and semantic similarities. METEOR's alignment-based scoring makes it particularly robust for short, varied responses.

3.5 ROUGE-L F1 Score

ROUGE-L measures the longest common subsequence (LCS) between predicted and reference answers, emphasizing fluency and sequential overlap without requiring exact token matches. While it balances precision and recall, its effectiveness diminishes on short answers, especially single-word responses. Nonetheless, it is valuable for evaluating longer, sentence-level outputs.

3.6 Word Overlap F1 Score

This metric computes the F1 score based on the intersection of words in the predicted and reference answers. It provides partial credit for overlap even when exact matches are absent, making it useful for evaluating open-ended responses with diverse phrasing. Its simplicity and interpretability make it a practical supplement to more complex semantic metrics.

3.7 Wu-Palmer Similarity

Wu-Palmer Similarity quantifies semantic relatedness by comparing word senses in the WordNet hierarchy. It captures synonymy and conceptual similarity, making it especially suitable for evaluating models fine-tuned on VQA tasks where semantic alignment is critical. The score reflects the depth of shared meaning between prediction and reference, not just lexical overlap.

4 Baseline Models

4.1 BLIP-VQA-Base

To establish a reliable quantitative baseline for VQA, we first utilized the `Salesforce/blip-vqa-base` model—an encoder-decoder architecture that tightly integrates visual and textual modalities for multimodal reasoning.

Based on a Vision Transformer (ViT) as the image encoder and a language model decoder, BLIP (Bootstrapping Language-Image Pretraining) excels in zero-shot and few-shot VQA tasks by leveraging pretraining on large-scale vision-language datasets. Its design enables strong alignment between image content and natural language queries, making it well-suited for tasks requiring semantic grounding and contextual understanding.

4.1.1 Evaluation Setup

The preprocessing pipeline was handled by the Hugging Face `BlipProcessor`, which performs two primary functions:

- It transforms raw image inputs into normalized pixel tensors that are compatible with the ViT-based visual encoder.
- It tokenizes the input question into tensor representations that can be processed by the language decoder.

All images were explicitly converted to RGB using the Python Imaging Library (PIL) to ensure consistent input format and to avoid channel mismatches that can occur with grayscale, CMYK, or other non-standard image modes. This conversion was a crucial standardization step, preventing runtime errors during tensor conversion and ensuring compatibility with the ViT-based encoder within the BLIP model. We followed this method of transformations for subsequent models as well.

After standardization, inference was carried out using the processed image-question pairs. The model generated natural language answers using autoregressive decoding, with outputs subsequently decoded into text for evaluation. This pipeline ensured robust handling of diverse input formats and streamlined downstream answer generation.

4.1.2 Results and Observations

Quantitative results from the evaluation are summarized in Figure 2. We report performance across two dataset subsets for inference—25% and 50% of the full set—using a range of metrics including Exact Match (Accuracy), BERTScore, ROUGE-L F1 and Word Overlap F1.

These results reveal consistent trends in answer quality and semantic fidelity as the sample size increases. A marginal improvement is observed with the 50% subset, indicating that model stability and evaluation reliability benefit from larger sample sizes.

```
Baseline Performance Metrics for BLIP-VQA-base (25% Sample):
Exact Matching Accuracy: 0.5089
BERT Score F1: 0.9650
ROUGE-L F1: 0.5234
Word Overlap F1: 0.5148
```

```
Baseline Performance Metrics for BLIP-VQA-base (50% Sample):
Exact Matching Accuracy: 0.5106
BERT Score F1: 0.9652
ROUGE-L F1: 0.5248
Word Overlap F1: 0.5162
```

Figure 2: Comparative performance of BLIP-VQA-Base across different dataset sample sizes

4.2 BLIP2-opt-2.7b

The next model we used for evaluation was the `Salesforce/blip2-opt-2.7b` model. BLIP-2 represents a significant architectural advancement, incorporating a lightweight Querying Transformer (Q-Former) that serves as a vision-language interface. This design enables more efficient cross-modal alignment and improves performance on open-ended VQA tasks by leveraging large pre-trained language models with minimal additional training overhead.

4.2.1 Evaluation Setup

The input preprocessing for BLIP-2 was managed using the Hugging Face `Blip2Processor`, which is responsible for preparing both image and text modalities in a format compatible with the model's dual-stream architecture, in a manner akin to `BlipProcessor`.

Following preprocessing, each image-question pair was passed through the model in inference mode. The corresponding question was prepended with a prompt: `"Question: <question> Answer in one word:"` to steer the model toward concise answers. The `generate()` method produced a sequence of output tokens representing the model's answer, which was then decoded into natural language text.

4.2.2 Results and Observations

Figure 3 provides a comprehensive overview of the quantitative evaluation outcomes. Multiple performance metrics are reported across three inference subsets representing 25%, 50%, and 100% of the full dataset, enabling a nuanced analysis of model behavior at varying scales.

For each subset, predictions were generated and recorded for all samples in csv files, with detailed scoring applied on a per-record basis to ensure thorough assessment.

BLIP-2 demonstrated robust and consistent performance across all dataset partitions, with metric scores showing minimal variation between subsets. Notably, BARTScore emerged as a particularly informative metric, highlighting aspects of generation quality that became increasingly apparent in comparisons with subsequent model evaluations.

```
=== Average Scores for BLIP2 using 25% of dataset ===
F1 Score:              0.4092
Accuracy:              0.4028
BERTScore Precision:   0.7936
BERTScore Recall:      0.7899
BERTScore F1:          0.7914
BARTScore:             -3.9160
ROUGE-L F1:            0.4235
METEOR:                0.2241
```

(a)

```
=== Average Scores for BLIP2 using 50% of dataset ===
F1 Score:              0.4109
Accuracy:              0.4043
BERTScore Precision:   0.7904
BERTScore Recall:      0.7869
BERTScore F1:          0.7882
BARTScore:             -3.8912
ROUGE-L F1:            0.4258
METEOR:                0.2261
```

(b)

```
=== Average Scores for BLIP2 using entire dataset ===
F1 Score:              0.4104
Accuracy:              0.4039
BERTScore Precision:   0.7891
BERTScore Recall:      0.7854
BERTScore F1:          0.7868
BARTScore:             -3.8840
ROUGE-L F1:            0.4256
METEOR:                0.2258
```

(c)

Figure 3: Comparative results of BLIP-2-opt-2.7b on inference sets of various sizes

4.3 CLIP

We established another zero-shot VQA baseline by implementing an evaluation pipeline using the Contrastive Language-Image Pretraining (CLIP) model. We specifically utilized the `ViT-B/32` architecture with OpenAI pretraining weights, loaded via the `open_clip` library. This approach enables the model to match image-question pairs against textual prompts without requiring task-specific training.

4.3.1 Evaluation Setup

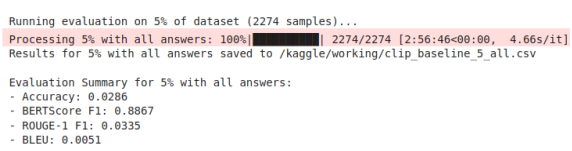
For each VQA sample, textual prompts were generated by concatenating the question with candidate answers in the format: `"Question: {question} Answer: {candidate}"`. This formulation enabled CLIP to jointly embed the image and semantically enriched text prompt, effectively framing answer selection as a text-to-image alignment task. Each candidate prompt was tokenized with the model-specific tokenizer, encoded via the CLIP text encoder, and L2-normalized. Correspondingly, input images were preprocessed with resizing, cropping, and normalization before being passed through the image encoder to obtain embeddings.

To evaluate the model under varying constraints on answer sets, two candidate regimes were defined: a **fixed set** of 100+ common answers (including basic colors, numbers, spatial relations, and common objects) and a **full set** comprising all unique ground-truth answers in the dataset, allowing for both constrained and open-set evaluations. For computational efficiency, we sampled 2% and 5% subsets of the dataset, running each evaluation twice—once per answer regime—resulting in four total experimental configurations.

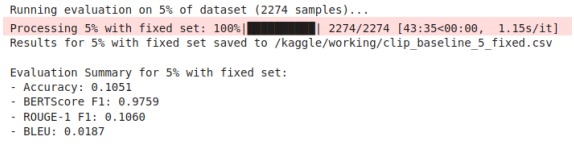
4.3.2 Results and Observations

The zero-shot CLIP baseline was evaluated across all experimental configurations, with detailed results logged separately for each dataset subset and answer regime. Performance was consistently stronger with the fixed candidate set, reflecting the benefits of constraining answer space. However, the open-set evaluation revealed the model's capacity for semantic generalization despite the lack of fine-tuning.

Across sampled subsets, cosine similarity-based predictions demonstrated reasonable alignment between images and candidate prompts, though performance naturally declined as answer set complexity increased. Accuracy, BERTScore F1 and ROUGE-L F1 metrics established a solid reference point for future comparisons, highlighting both the strengths and limitations of zero-shot CLIP for visual question answering tasks.



(a)



(b)

Figure 4: Comparative results of CLIP on a 5% dataset across the two answer regimes

4.4 LLaVA-1.5-7b-hf

Another interesting model we evaluated was LLaVA-1.5 in a zero-shot setting. LLaVA integrates CLIP-based visual encoding with LLaMA's language generation capabilities to support open-ended multimodal reasoning. We employed the `llava-hf/llava-1.5-7b-hf` checkpoint via the HuggingFace `transformers` library.

4.4.1 Evaluation Setup

The model and processor were instantiated using `AutoProcessor` and `LlavaForConditionalGeneration`, and deployed with mixed-precision inference (`torch.float16`) using `device_map="auto"` to optimize GPU utilization. To mitigate memory fragmentation, garbage collection and CUDA cache clearing were invoked between batches.

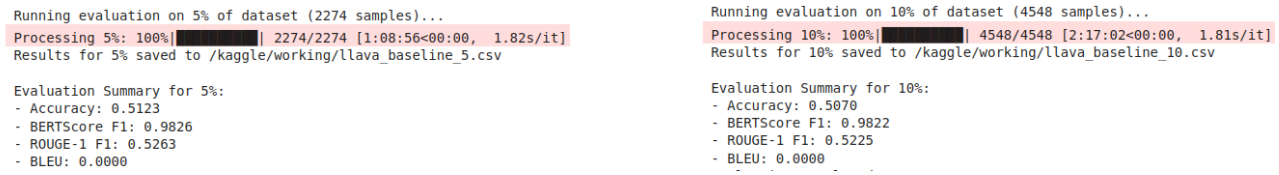
Evaluation was conducted on two random dataset samples (5% and 10%). Each prompt followed the template: `"<image> Question: {question} Answer:"`, enabling the model to generate free-form textual responses. The final answer was extracted by isolating the last word in the generated output, with punctuation removed to conform to VQA answer formatting conventions.

4.4.2 Results and Observations

The LLaVA model produced coherent responses across both dataset subsets, but results appeared to diminish as the inference set grew, with a noticeable decrease from just 5% to 10%. Structured CSV logs captured model outputs and associated metrics, including Accuracy, BERTScore F1, ROUGE-1 F1 and BLEU.

Although LLaVA was not fine-tuned on the evaluation dataset, it demonstrated an extremely strong capacity for semantic alignment between image content and question intent. The model's generative nature allowed it to

infer plausible answers beyond the surface-level cues. Overall, LLaVA serves as a compelling zero-shot baseline, matching the best performance BLIP-VQA-Base across many metrics.



(a) (b)

Figure 5: Comparative results of LLaVA-1.5-7b-hf on 5% and 10% of the dataset

4.5 ViLT-b32-finetuned-vqa

The Vision-and-Language Transformer (ViLT) employs a unified transformer architecture to jointly encode visual and textual inputs without convolutional or object detection modules. We utilized the `dandelin/vilt-b32-finetuned-vqa` checkpoint from HuggingFace, which is fine-tuned specifically for VQA tasks. While ViLBERT was not readily accessible for our setup, we opted for this model of ViLT that offered compatibility and allowed for effective experimentation.

4.5.1 Evaluation Setup

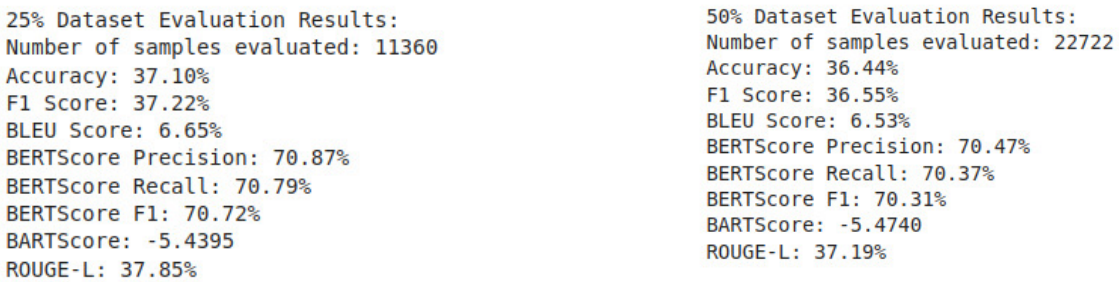
Each VQA example consisted of an image and a question, jointly processed using the ViLT processor to produce multimodal input tensors. Images were resized and normalized, while text was tokenized using the associated tokenizer. Inference was performed under `torch.no_grad()` on available GPUs, with the predicted answer selected as the highest-probability label from the model's fixed vocabulary.

Two evaluation subsets—25% and 50% of the dataset—were selected using deterministic sampling. The evaluation pipeline featured dynamic image loading, explicit memory management with `gc.collect()` and `torch.cuda.empty_cache()`, and robust exception handling to skip corrupted samples. Output predictions and metrics were written to CSV files for subsequent analysis.

4.5.2 Results and Observations

ViLT performed reliably across both dataset sizes, with performance scaling slightly negatively with increased data volume. The model showed a decent baseline accuracy.

Despite its relatively lightweight architecture, ViLT demonstrated effective multimodal fusion and generalization across diverse question types. Performance was notably stable in smaller samples, indicating robustness even under limited data conditions. These results establish ViLT as a competitive baseline and a strong candidate for fine-tuning and domain-specific adaptation in future work.



(a) (b)

Figure 6: Comparative results of ViLT-b32-finetuned-vqa on 25% and 50% of the dataset

5 Fine-Tuning BLIP-VQA-Base with LoRA

In this section, we describe the process of fine-tuning the BLIP-VQA-Base model for the task of VQA. Given that the BLIP-VQA-Base model showed one of the best baseline results on our dataset, and our knowledge on this model due to it being covered in depth as part of the course (as opposed to LLaVA), we decided to use it as the base model for further fine-tuning with Low-Rank Adaptation (LoRA). This approach allows us to adapt a pre-trained model to our specific VQA task more efficiently by leveraging fewer parameters, which is particularly useful when working with limited computational resources.

5.1 Dataset Preparation

We first created a custom dataset class, `VQADataset`, which takes in the VQA data, processes the images and questions, and prepares them for training. This class ensures that the input to the model is in the correct format, combining both the image and its corresponding question. The answers are also tokenized and used as labels during training.

- **Image Processing:** Each image is loaded, converted to RGB, and processed using the BLIP processor, which prepares the image-question pair for the model.
- **Question Processing:** Each question is paired with its corresponding image, tokenized, and prepared as input for the model.
- **Answer Tokenization:** The answer is tokenized to produce the labels. We also ensure that the tokenized answers don't exceed a predefined length (e.g., `max_length=20`).

5.2 Model Configuration with LoRA

LoRA (Low-Rank Adaptation) is applied to the BLIP-VQA-Base model to facilitate more efficient fine-tuning by reducing the number of trainable parameters. The `configure_lora()` function initializes the LoRA configuration, setting parameters such as `r=16` (rank), `lora_alpha=16` (scaling factor), and `lora_dropout=0.05` (dropout rate).

- **LoRA Configuration:**
  - `r = 16`: The rank is set to 16, which determines the dimensionality of the low-rank adaptation. A value of 16 was chosen as a balance between memory efficiency and model expressiveness. This rank allows sufficient capacity for adaptation while ensuring that the model remains computationally feasible.
  - `lora_alpha = 16`: The scaling factor is set to 16. This value was chosen because it provides a good trade-off between stability and expressiveness. A larger `alpha` would have introduced unnecessary complexity, while a smaller one would have constrained the model's ability to learn from the task.
  - `lora_dropout = 0.05`: Dropout is set to 5%, which is a common value for preventing overfitting without losing too much model capacity. This helps the model generalize well to unseen data.
- **LoRA Application:** LoRA is applied specifically to the `query` and `value` modules of the attention mechanism. These modules were selected because they play a critical role in the model's ability to focus on the relevant parts of the image when answering the question. By applying LoRA to these modules, we reduce the number of parameters that need to be adapted, which helps in achieving efficient fine-tuning.
- **Forward Method Customization:** A custom forward method is used to ensure that only supported inputs are passed to the base model. This prevents any unnecessary overhead during training, ensuring optimal performance.

5.3 Training Setup

The fine-tuning is performed using the `fine_tune_with_lora()` function, which manages the entire training pipeline, including dataset loading, model configuration, optimizer setup, and training loop.

- **Optimizer and Learning Rate Scheduler:** We use the AdamW optimizer with a learning rate of `5e-5`. The AdamW optimizer was selected due to its effectiveness in handling sparse gradients and its strong performance in transformer-based architectures. The learning rate was set at `5e-5` after experimenting with different values. This value strikes a balance between convergence speed and stability during training, ensuring that the model fine-tunes efficiently without overshooting the optimal solution.

- **Batch Size:** The batch size is set to 8. A smaller batch size was chosen due to memory constraints on the available GPU, but it was large enough to allow for effective gradient updates. The batch size of 8 provides a good trade-off between memory usage and the stability of gradient estimates.
- **Epochs:** We fine-tuned the model for 3 epochs. This number was chosen after experimentation, where it was observed that the model starts to converge after 3 epochs, beyond which performance gains diminish.
- **Automatic Mixed Precision (AMP):** We utilized AMP to speed up training and reduce memory usage without sacrificing model quality. AMP allows the model to perform calculations with reduced precision (float16) while keeping the model's accuracy intact. This results in faster training times, particularly on GPUs with tensor cores, such as NVIDIA's V100 and A100.

5.4 Automatic Mixed Precision (AMP)

The use of AMP provides a substantial performance boost, reducing both memory usage and computation time while preserving the accuracy of the model. The `torch.cuda.amp.GradScaler()` is used to scale gradients when training with AMP, ensuring that numerical stability is maintained during backpropagation. The use of AMP ensures that the training process is both faster and more memory-efficient.

5.5 Caching and Pinning Memory

To further improve performance, we enabled **data caching** and **pinning memory**. Pinning memory allows for faster data transfer between the CPU and GPU by ensuring that the data is allocated in page-locked memory. This makes the data loading process more efficient, especially when using high throughput training, by reducing data transfer bottlenecks.

- **Pinning Memory:** This ensures that data is transferred more efficiently to the GPU, which speeds up the training process by reducing wait times for data loading.
- **Use of Cache:** The `use_cache` argument is set to `True` in the model inputs. This enables caching of certain intermediate computations during inference and training, improving efficiency when performing repeated operations over the same data.

5.6 Training Loop

The training loop runs for a predefined number of epochs (3 in this case) and iterates over the dataset in batches. **Eighty percent** of our handcrafted dataset was utilized to assist in fine-tuning the model. During each epoch:

- **Forward Pass:** For each batch, the model performs a forward pass where the image-question pairs are processed.
- **Backward Pass:** The loss is calculated, and gradients are backpropagated to update the model's parameters.
- **AMP Support:** If AMP is enabled, the forward and backward passes are done within a mixed-precision context, which allows the model to train faster without sacrificing model quality.
- **Loss Calculation and Optimization:** The loss is calculated, and the optimizer updates the model parameters accordingly. We use a learning rate scheduler to ensure that the learning rate decays appropriately during training.

The training time per epoch is tracked, and the average loss is computed to monitor progress. At the end of each epoch, the model's performance is evaluated, and the model is saved periodically.

5.7 Novelties Introduced

Several novelties were introduced during the fine-tuning process:

- **Caching and Pinning Memory:** By enabling caching of intermediate computations and pinning memory, we improved the efficiency of the data loading pipeline, which is crucial for handling large datasets and speeding up the training process.

- **Automatic Mixed Precision (AMP):** The use of AMP provided a substantial performance boost, reducing both memory usage and computation time while preserving the accuracy of the model.

As a direct consequence of the aforementioned optimizations, we observed a substantial acceleration in training throughput, enabling us to complete the process efficiently.

In particular, the duration of epoch 1 was reduced by a remarkable **54.13%**, with the epoch time dropping from approximately `2794.53s` to `1281.78s`, yielding an effective speed-up of **2.18**×. In other terms, the per-sample processing efficiency exhibited a significant gain: throughput improved from roughly `13 samples/second` to an impressive `28 samples/second`—more than doubling the rate.

It is worth emphasizing that these improvements were achieved without compromising model quality: both training configurations resulted in nearly identical average loss values across an epoch. These observations, while presented for epoch 1, were representative of broader performance trends across subsequent epochs. The non-optimized setup was not run to completion due to its impractical training time, underscoring the critical impact of the adopted optimizations on experimental feasibility.

```
Epoch 1/3 - Average Loss: 7.9930
Epoch Time: 2794.53s - Samples Processed: 36373
```

(a) Epoch duration prior to optimization

```
Epoch 1/3 - Average Loss: 7.9926
Epoch Time: 1281.78s - Samples Processed: 36373
```

(b) Epoch duration after applying efficiency enhancements

Figure 7: Comparative visualization of training epoch durations before and after optimization

This substantial performance uplift reflects the compounded benefits of architectural streamlining, asynchronous data loading, and hardware-aware parallelism. Not only did this lead to faster training cycles, but it also enabled more responsive experimentation and real-time model iteration, significantly accelerating our research velocity.

5.8 Model Evaluation and Saving

After completing the fine-tuning process, the model is saved along with the processor. This allows the model to be easily loaded for inference and evaluation on the test dataset. The fine-tuned model is stored in a specified directory, ensuring that all necessary components (such as the processor and model weights) are preserved for future use.

The final model is evaluated on a hidden test dataset, made up of the remaining **twenty percent** of our original dataset (not used for fine-tuning), where its performance is assessed using standard metrics, such as accuracy, F1, BERTScore, METEOR score, and others. These metrics help gauge the effectiveness of the fine-tuned model in answering questions related to images in the ABO dataset.

By applying these optimizations, we ensure that the fine-tuning process is both efficient and effective, ultimately leading to improved performance on the VQA task. The evaluation results comprising of few of the important scores have been displayed in Figure 8.

```
--- BLIP-VQA Evaluation Summary ---

Total samples: 9110

Exact Match         : 0.6709
Token Match         : 0.6709
Wup Score           : 0.8339
F1 Score            : 0.6712
Meteor Score        : 0.3436
Bertscore F1        : 0.9855
Bart Score          : -4.5013
```

Figure 8

5.9 Qualitative and Quantitative Visualization Analysis

In addition to quantitative metrics, we conducted a series of visual analyses to understand the performance characteristics, limitations, and interpretability of our VQA model. This section presents five key visualizations and discusses their implications on the overall evaluation.

5.9.1 Metric Correlations

Figure 9 provides a heatmap of Pearson correlation coefficients among the different metrics. High correlation ($r > 0.90$) with Exact Match shows that Token Match, F1 Score and Meteor Score indicate that these metrics didn't provide much additional insights. Conversely, metrics like BERT Score, WUP Score and and BART Score show moderate correlation with other metrics, indicating that they provide a unique perspective in their own ways by offering a blend of surface and deep semantic sensitivity.

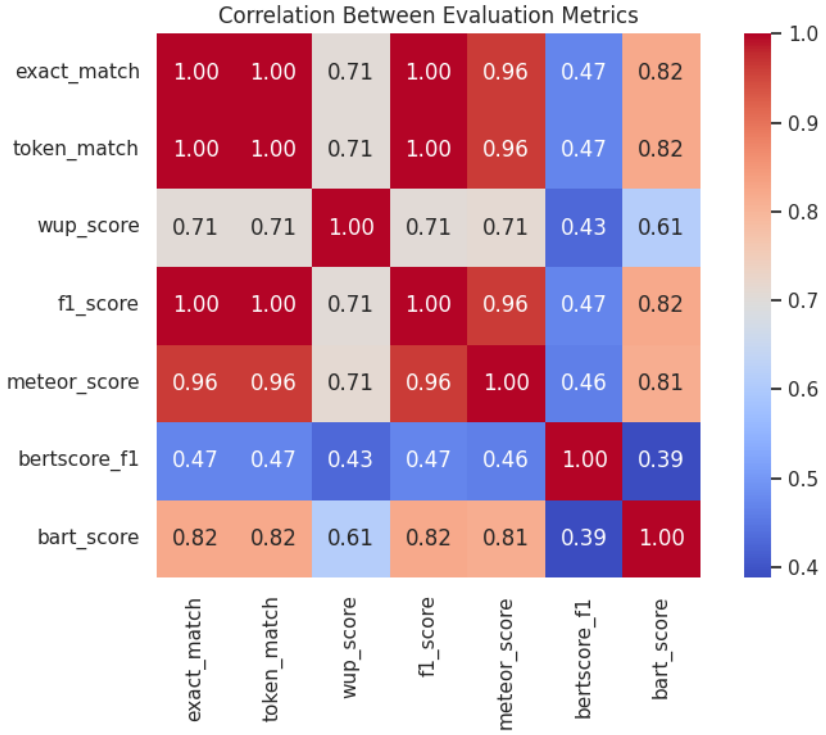


Figure 9: Metric Correlation Heatmap

These results informed us on what major metrics to consider while plotting the next couple of graphs, since they were capturing the essence and contributions of the other metrics.

5.9.2 Metric Distributions

Figure 10 visualizes the distribution of evaluation metrics such as Exact Match, F1 Score, METEOR, BERTScore, and BARTScore. The plot highlights the skewed nature of several metrics, where a significant portion of predictions receive low scores (e.g., EM), while soft metrics like BERTScore and METEOR show higher central tendencies. This suggests that while the model struggles with exact lexical matching comparitively, it frequently produces semantically plausible answers. The long-tailed nature of F1 and BARTScore reflects variability in output quality, which is essential for identifying outlier cases.

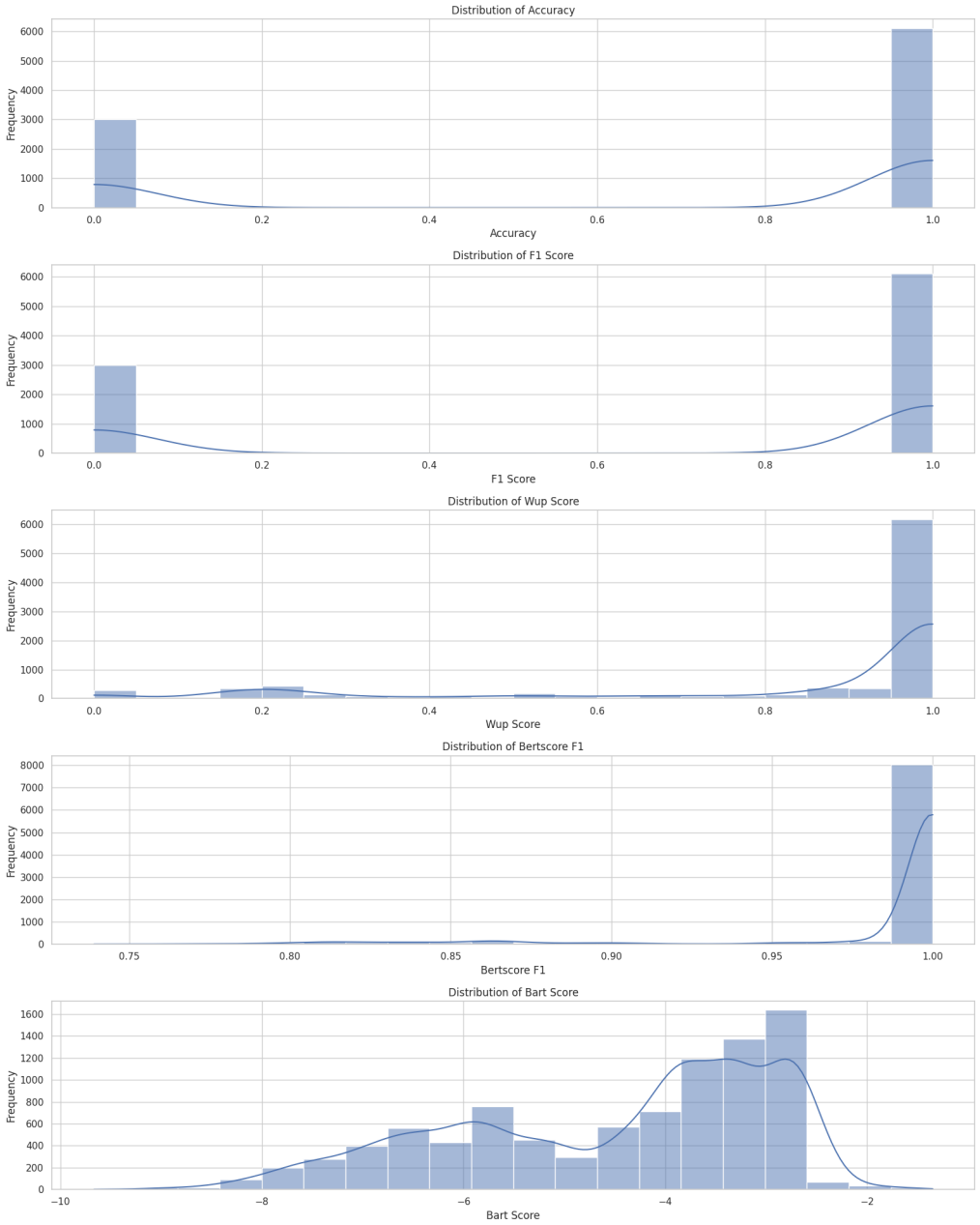


Figure 10: Metric distributions

These insights prompted us to emphasize semantic metrics in our evaluation, ensuring that the model's strengths in contextually appropriate answers are not undervalued.

5.9.3 Performance by Question Type

Figure 11 demonstrates that the model performs relatively better on color-based and counting-based questions, whereas performance on open-ended "what kind" questions lags behind. This reflects limitations in reasoning and abstraction, often attributed to insufficient grounding or contextual ambiguity.

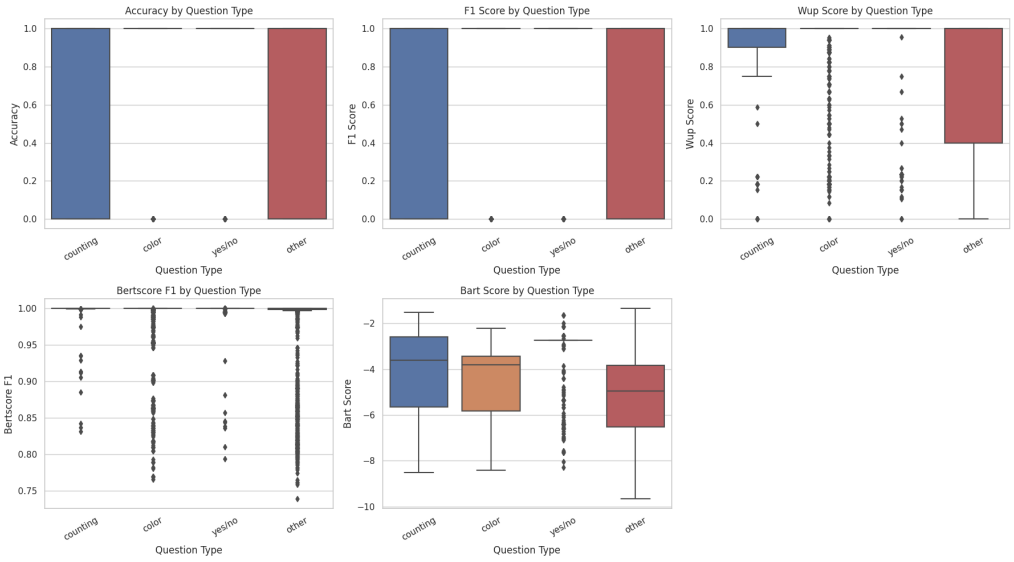


Figure 11: Metric By Question Type

These findings show that more work can be done to make a robust model capable of answering advanced and difficult questions, while also demonstrating promising results to handle a variety of questions with a wide-array of applications.

5.9.4 F1 Score vs Answer Length

As shown in Figure 12, there is a noticeable inverse relationship between answer length and F1 score. Shorter answers (1–3 characters) generally score higher, while longer answers exhibit declining accuracy. This analysis may be sensitive to question type bias, as certain question types inherently demand longer answers. This further illustrates the notable performance for "Yes/No" type of questions suggesting that while the model is adept at generating concise responses, it struggles relatively with maintaining fluency and relevance in longer outputs.

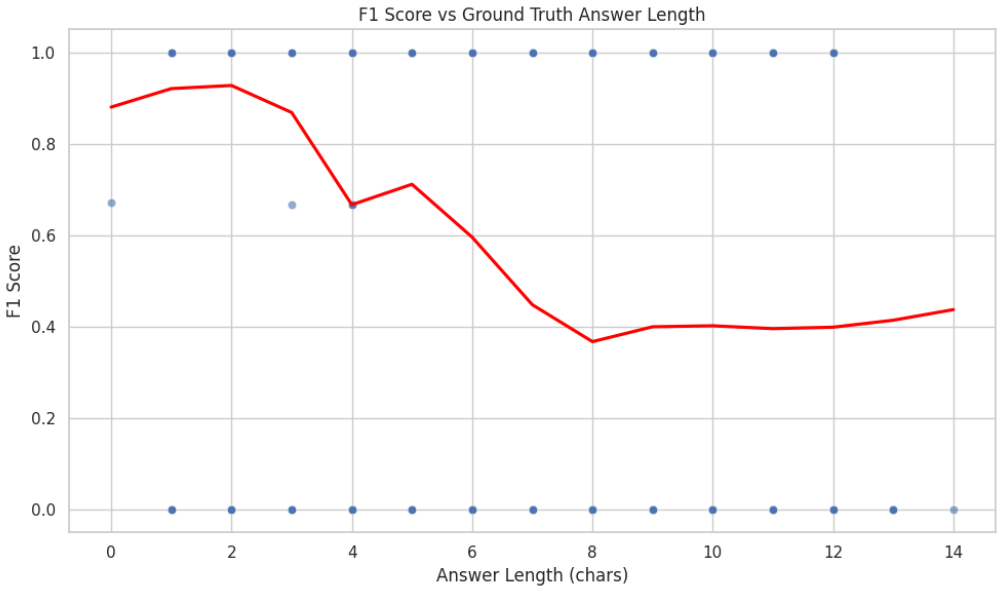


Figure 12: F1 vs Length

5.9.5 Error Patterns

Figure 13 visualizes frequent substitution or omission errors made by the model. High-intensity cells along off-diagonal positions indicate systematic token-level errors, such as confusing similar colors (e.g., "red" vs

"orange") or misidentifying object categories (e.g., "dog" vs "cat"). This highlights deficiencies in fine-grained visual discrimination or grounding.

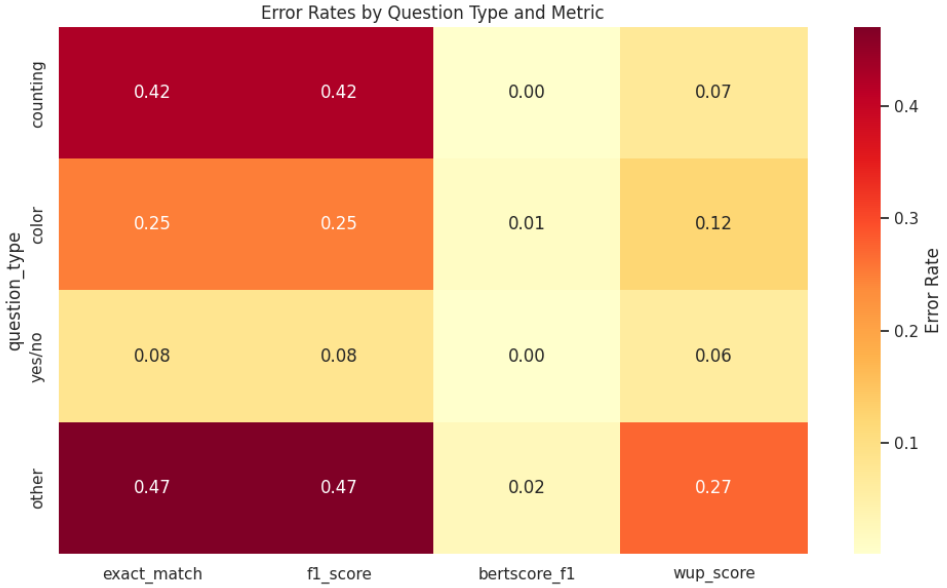


Figure 13: Error Heatmap

These patterns justify the need for enhanced visual feature granularity, potentially via higher-resolution embeddings or multimodal fusion improvements.

In summary, these visual analyses not only complement the quantitative evaluation but also uncover latent weaknesses in the model's architecture and training regime. Future work will incorporate these insights into model refinement, targeted data augmentation, and evaluation metric selection.

5.10 Challenges Faced During Fine-Tuning

Despite the overall success in fine-tuning the BLIP-VQA-Base model using LoRA, we encountered several practical and technical challenges that required troubleshooting and adaptation:

- **Limited GPU Availability and Compute Budget:** The fine-tuning was performed on a constrained GPU environment (free Kaggle notebooks), which imposed restrictions on total GPU hours and memory. We had to carefully balance batch sizes, sequence lengths, and model configuration to avoid out-of-memory errors and training interruptions.
- **Kaggle Runtime Instability:** Kaggle notebooks frequently timed out or became unresponsive during longer training sessions. This made it difficult to run multi-epoch training without interruption. We adapted by modularizing code into smaller cells when applicable and using shorter training loops for experimentation.
- **LoRA with BLIP Architecture:** While integrating LoRA into the BLIP model, we initially encountered errors related to undefined inputs like `inputs_embeds`. The BLIP-VQA model did not expose all expected fields by default, requiring us to override or wrap the forward method to ensure compatibility with LoRA's expected input types.
- **Incompatibility with Quantization Modules:** We attempted to use the `bitsandbytes` library for parameter quantization to further reduce memory usage. However, on Kaggle, the latest module failed to install or be recognized for reasons we weren't entirely sure of, despite multiple attempts. This limited our ability to explore 4-bit or 8-bit quantization strategies.
- **Limited Experimentation Bandwidth:** Due to time and hardware constraints, we were limited in the number of hyperparameter tuning runs and ablation studies. As a result, certain design decisions (e.g., dropout rate, learning rate) were based on references on internet resources than exhaustive experimentation.

These challenges highlight the complexities involved in adapting large-scale vision-language models under constrained environments. Nonetheless, the experience reinforced the value of modularity, incremental testing, and efficient debugging in low-resource research settings.

6 Summary of results

While several evaluation metrics were computed and analyzed, two emerged as central to our study: **Accuracy** and **BERTScore F1**. These metrics serve as cornerstones in assessing model performance from both lexical and semantic standpoints.

The complementary nature of these two metrics—surface-level exactness versus deep semantic fidelity—makes them jointly essential for a holistic evaluation. For this reason, both metrics were computed for every model and experiment configuration, forming the backbone of our comparison framework. Other metrics such as METEOR, BARTScore, ROUGE-L, and WUPS were computed selectively, where we felt relevant, to provide additional interpretability or to highlight specific strengths of individual model variants (e.g., fluency, word overlap, or visual-semantic alignment).

The table below presents a concise summary of these evaluation metrics across various models, showcasing our fine-tuning efforts and the resulting performance improvements.

Model	**Inference Sample Size**	**Accuracy**	**BERTScore F1**
BLIP-VQA-Base	11360	0.5089	0.9650
	22722	0.5106	0.9652
BLIP2-opt-2.7b	11360	0.4028	0.7914
	22722	0.4042	0.7822
	45482	0.4039	0.7868
CLIP (all)	909	0.0374	0.8879
	2274	0.0286	0.8867
	11360	0.0479	0.5115
CLIP (fixed)	909	0.1045	0.9770
	2274	0.1051	0.9759
	11360	0.0992	0.5772
LLaVA-1.5-7b-hf	2274	0.5123	0.9826
	4548	0.5070	0.9822
ViLT-b32-finetuned-vqa	11360	0.3710	0.7072
	22722	0.3644	0.7031
BLIP-VQA-Base (Fine-tuned)	9110	**0.6712**	**0.9855**

Table 1: Comparison of Accuracy and BERTScore F1 across models