AIM825 Course Project

Multimodal Visual Question Answering with Amazon Berkeley Objects Dataset

Project Submitted by:

Abhishek Kumar Singh (MT2024006)

Naval Kishore Singh Bisht (MT2024099)

Table of Contents

1. Visual Question Answering (VQA) Dataset Creation

The data curation process transformed the Amazon Berkeley Objects (ABO) dataset into a Visual Question Answering (VQA) dataset through four steps: processing image metadata, extracting product listings, linking images with metadata, and generating question-answer pairs. Each step uses specific Python tools and techniques to ensure data quality and usability for VQA tasks. Below is a detailed breakdown of each step, including objectives, inputs, processes, tools, outputs, and screenshots to illustrate the results.

Step 1: Processing Image Metadata (process_images.py)

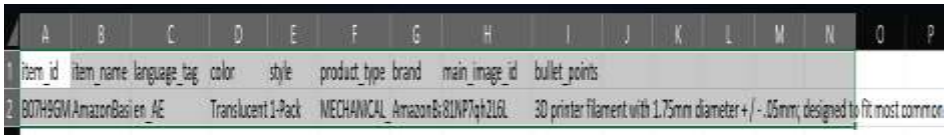
- **Objective**: Validate and prepare image metadata from the ABO dataset to ensure images are accessible and correctly referenced for downstream tasks.
- **Input**:
    - images.csv.gz: A compressed CSV file containing image IDs and relative paths.
    - Image directory: images/small, containing the actual image files.
- **Process**:
    - **Load Metadata**: Use Pandas to read images.csv.gz into a DataFrame, using compression="gzip" to handle the compressed format efficiently.
    - **Construct Full Paths**: Generate image paths by combining the image directory with relative paths using os.path.join, ensuring platform-independent paths.
    - **Verify Image Existence**: Check if each image file exists using os.path.exists, adding an exists column to flag missing files.
    - **Visual Validation**: Open and display a sample image using PIL's Image.open and img.show() to confirm image integrity and content.
    - **Save Processed Metadata**: Export the DataFrame (with full paths and existence flags) to processed_images.csv, preserving all original fields.
    - **Console Output**: Print summary statistics, including total images, number of found images, and the first 5 rows of key columns (image_id, full_path, exists).
- **Tools**:
    - **Pandas**: Loads, manipulates, and saves CSV data.
    - **os**: Provides path construction and file existence checks.
    - **PIL (Python Imaging Library)**: Facilitates image loading and display.
- **Output**:

    - processed_images.csv: Metadata with paths and existence flags.
    - Console output summarizing dataset size and sample data.

Step 2: Processing Product Listings (process_listings_custom.py)

- **Objective**: Extract structured English-language product metadata from JSON files for VQA tasks.
- **Input**:
    - Directory: listings/metadata, containing multiple listings_*.json.gz files with product details in JSON Lines format.
- **Process**:
    - **Locate Files**: Use glob to identify all listings_*.json.gz files for batch processing.
    - **Read JSON Lines**: Process compressed JSON files line by line using gzip.open with UTF-8 encoding.
    - **Extract Item ID**: Extract item IDs from fields like itemId, item_id, id, productId, or asin, skipping entries without IDs.
    - **Filter English Metadata**: Use has_english_value to ensure item_name has English language tags (e.g., en, en_US, en_GB).
    - **Extract Fields**: Use extract_field to retrieve fields like item_name, color, style, product_type, and brand, prioritizing English values.
    - **Process Bullet Points**: Use extract_bullet_points to concatenate English bullet points into a string separated by |.
    - **Save Metadata**: Write metadata to all_listings_metadata.csv using csv.DictWriter with UTF-8 encoding.
    - **Log Issues**: Save details of skipped entries (non-English, missing fields, or JSON errors) to missing_fields_log.json.
    - **Console Output**: Report processed rows, skipped non-English entries, and missing item IDs.
- **Tools**:
    - **json**: Parses JSON Lines data.
    - **gzip**: Reads compressed files.
    - **glob**: Identifies input files.

    - **csv**: Writes metadata to CSV.
- **Output**:
    - all_listings_metadata.csv: English product metadata, including item_id, item_name, color, style, product_type, brand, main_image_id, and bullet_points.
    - missing_fields_log.json: Logs skipped entries.
    - Console output detailing processing statistics.
- **Screenshot**



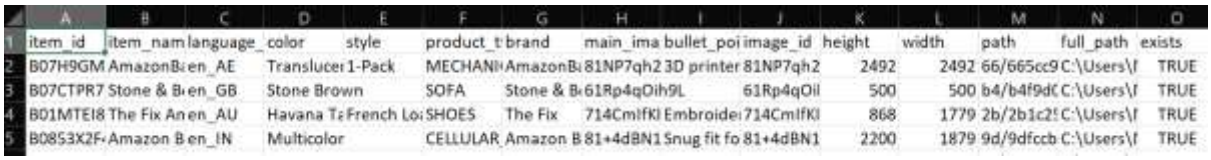
item_id	item_name	language_tag	color	style	product_type	brand	main_image_id	bullet_points
B07H9GM	AmazonBasi	en_AE	Translucent 1-Pack		MECHANICAL_	AmazonBa	81JNP7qh2LGL	3D printer filament with 1.75mm diameter +/- .05mm; designed to fit most common

Step 3: Linking Images with Metadata (link_data.py)

- **Objective**: Merge image metadata with English product metadata to create a unified dataset of valid image-metadata pairs.
- **Input**:
    - images.csv.gz: Image metadata from Step 1.
    - all_listings_metadata.csv: Product metadata from Step 2.
    - Image directory: images/small.
- **Process**:
    - **Load Image Metadata**: Read images.csv.gz into a DataFrame, construct full image paths, and verify existence with os.path.exists.
    - **Chunked Listing Loading**: Read all_listings_metadata.csv in 50,000-row chunks to manage memory.
    - **Merge Data**: Perform an inner merge on main_image_id (from listings) and image_id (from images).
    - **Filter Quality Data**: Keep rows with existing images (exists=True), non-empty item_name, and English language tags (starting with en).
    - **Combine Chunks**: Use pd.concat to combine valid chunks into a single DataFrame.
    - **Save Linked Data**: Export to valid_dataset.csv, preserving all fields.

- **Console Output**: Display the number of valid products and the first 5 rows of key columns (item_id, item_name, main_image_id, full_path, color).

- **Tools**:
  - **Pandas**: Manages data loading, merging, filtering, and concatenation.
  - **os**: Handles path construction and file existence checks.
- **Output**:
  - valid_dataset.csv: Linked image-metadata pairs for English products with valid images.
  - Console output summarizing valid entries and sample data.
- **Screenshot**



item_id	item_nam	language_	color	style	product_t	brand	main_ima	bullet_poi	image_id	height	width	path	full_path	exists
B07H9GM	AmazonBa	en_AE	Translucer	1-Pack	MECHANI	AmazonBa	81NP7qh2	3D printer	81NP7qh2	2492	2492	66/665cc9	C:\Users\I	TRUE
B07CTPR7	Stone & B	en_GB	Stone Brown		SOFA	Stone & B	61Rp4qOih9L		61Rp4qOil	500	500	b4/b4f9d(	C:\Users\I	TRUE
B01MTEI8	The Fix An	en_AU	Havana Ta	French Lo	SHOES	The Fix	714CmlfKl	Embroide	714CmlfKl	868	1779	2b/2b1c2!	C:\Users\I	TRUE
B0853X2F	Amazon B	en_IN	Multicolor		CELLULAR	Amazon B	81+4dBN1	Snug fit fo	81+4dBN1	2200	1879	9d/9dfccb	C:\Users\I	TRUE

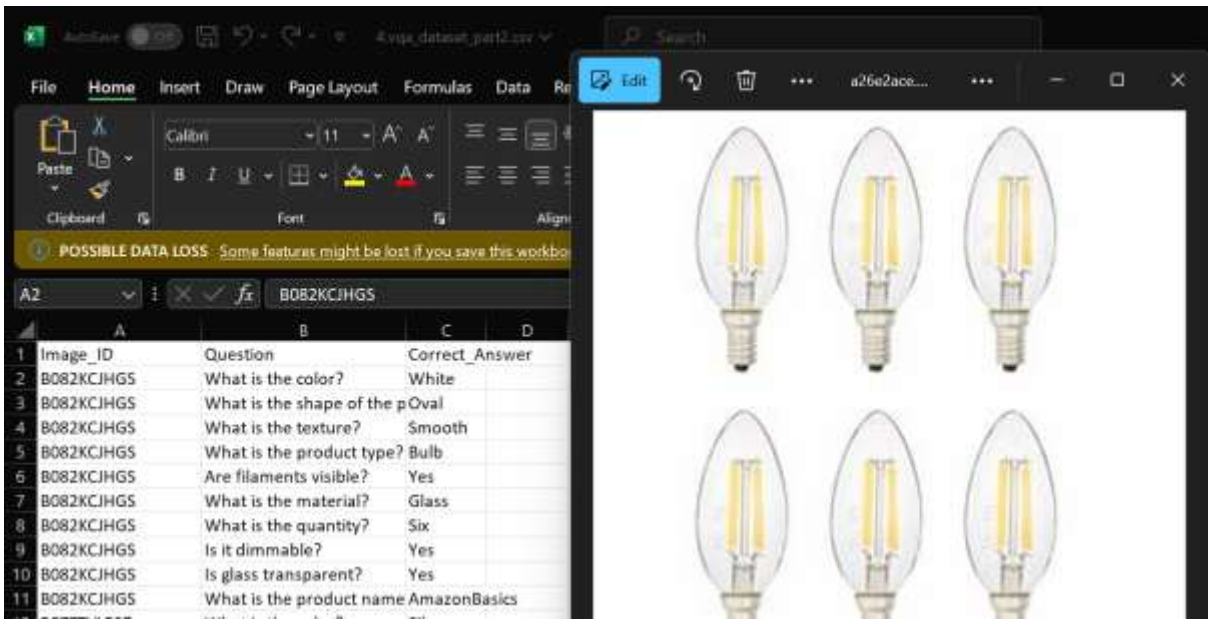
Step 4: Generating Question-Answer Pairs (generate_vqa2.py)

- **Objective**: Generate visual question-answer (VQA) pairs for product images using the Google Gemini-1.5-Flash API.
- **Note**: As we are group of 2, so we decided to split the valid_dataset.csv into 2 part so that both members can generate question answers simultaneously. So in relality in code the files are named as valid_dataset_part1.csv or valid_dataset_part2.csv. So we are explaining the code using valid_dataset_part2.csv. later on we have merged our data into one file.
- **Input**:
  - valid_dataset_part2.csv: Linked image-metadata pairs from Step 3.
  - vqa_dataset_part2.csv: Existing VQA dataset to track processed IDs.
  - Image directory: images/small.
- **Process**:
  - **Filter Unprocessed Items**: Load valid_dataset.csv and exclude processed item_id entries from vqa_dataset_part2.csv.
  - **Validate API Keys**: Test API keys with a sample request to the Gemini API.
  - **Encode Images**: Convert images to base64 format using base64.b64encode for API payloads.

- **Generate Questions**: Request 10 unique visual questions per image with single-word answers via the Gemini API, including five mandatory questions:
    - What is the color?
    - What is the texture?
    - What is the product name?
    - What is the product type?
    - What is the shape of the product?
    - Five additional questions on diverse aspects (e.g., material, pattern, brand visibility).
- **Rate Limiting**: Enforce 30 requests per minute (2-second delay) and 300 daily requests per API key using time.sleep.
- **Retry Logic**: Retry failed requests (e.g., HTTP 429 errors) up to 3 times with a 10-second delay.
- **Parse Responses**: Extract JSON question-answer pairs using regex (re.search).
- **Parallel Processing**: Use ThreadPoolExecutor with workers limited to the number of valid API keys.
- **Incremental Saving**: Save VQA pairs to vqa_dataset_part2.csv every 100 API calls, appending to existing data.
- **Error Logging**: Log failed items to failed_items.csv and progress/errors to vqa_processing.log.
- **Console Output**: Display processed item counts, API key usage, and total questions.

- **Prompt**:
    - *“Given this product image (base64) and metadata {metadata}, generate exactly 10 unique visual questions with single-word answers about the product’s appearance, its description and information in bullet_points and product_type columns also. Include the following questions: 'What is the color?' and 'What is the texture?' and 'What is the product name?' and 'What is the product type?' and 'What is the shape of the product?' For the other questions, choose diverse visual aspects (e.g., material, pattern, style, brand visibility, or suitability for its product type). Examples: 'What is the material?', 'Is the brand logo visible?', 'Does the design suit its product type?'. Avoid repetitive questions and vary the phrasing. Format: JSON array of objects with 'question' and 'answer'.”*

- **Tools**:
    - **Pandas**: Manages dataset filtering, concatenation, and CSV output.
    - **requests**: Sends API requests.
    - **base64**: Encodes images.
    - **json, re**: Parses API responses.
    - **os**: Handles file operations.
    - **logging**: Logs progress and errors.
    - **concurrent.futures**: Enables parallel API calls.
    - **time**: Implements rate limiting and retries.
- **Output**:
    - vqa_dataset_part2.csv: Columns Image_ID, Question, and Correct_Answer with VQA pairs.
    - failed_items.csv: Logs failed items with error details.
    - vqa_processing.log: Records API usage and errors.
    - Console output summarizing processed items, API usage, and total questions.
- **Screenshot**



(the questions and its corresponding image (here Image_ID actually refers to item_id)

Tools and Technologies

- **Python Libraries**:
    - **Pandas**: Data manipulation and CSV handling.
    - **PIL**: Image validation and display.
    - **requests**: API communication.
    - **base64**: Image encoding.
    - **json, re**: Response parsing.
    - **os, glob, gzip, csv**: File operations.
    - **logging**: Error and progress tracking.
    - **concurrent.futures**: Parallel processing.
    - **time**: Rate limiting and retries.
- **External Services**:
    - **Google Gemini-1.5-Flash API**: Generates question-answer pairs based on images and metadata.

2. Data Merging and Preprocessing

The data merging and preprocessing process creates a Visual Question Answering (VQA) dataset by merging two previously split dataset parts, combining metadata with question-answer pairs, filtering for items with exactly 10 questions, and organizing corresponding images into a new directory. This involves three steps: merging metadata and questions, filtering the dataset, and copying filtered images. Each step uses Python tools to ensure data consistency for VQA tasks. Below is a detailed breakdown of each step, including objectives, inputs, processes, tools, outputs, and screenshot placeholders.

Step 1: Merging Metadata and Questions (1.merge.py)

- **Objective**: Merge metadata and question-answer pairs from two dataset parts to create a unified VQA dataset.
- **Input**:
    - valid_dataset_filtered_cleaned.csv: Product metadata with item_id, main_image_id, and path.
    - vqa_dataset_combined_filtered_cleaned.csv: Question-answer pairs linked to item_id.
- **Process**:
    - **Load Metadata**: Use Pandas to read valid_dataset_filtered_cleaned.csv, selecting item_id, main_image_id, and path.
    - **Prepend Path Prefix**: Add images/small/ to the path column for correct image paths.
    - **Load Questions**: Read vqa_dataset_combined_filtered_cleaned.csv containing question-answer pairs.
    - **Merge DataFrames**: Perform a left merge on item_id to combine metadata and questions.
    - **Save Merged Data**: Export to merged_dataset.csv without an index.
    - **Console Output**: Confirm CSV creation.
- **Tools**:
    - **Pandas**: Data loading, merging, and CSV output.
- **Output**:
    - merged_dataset.csv: Merged dataset with item_id, main_image_id, path, and question-answer pairs.

- Console output confirming save.

- **Screenshot (of merged_dataset.csv)**

	item_id	main_image_id	path	Question	Correct_Answer
2	B07H9GMYXS	81NP7qh2L6L	images/small/66/665cc994.jpg	What is the color?	Yellow
3	B07H9GMYXS	81NP7qh2L6L	images/small/66/665cc994.jpg	What is the texture?	Smooth
4	B07H9GMYXS	81NP7qh2L6L	images/small/66/665cc994.jpg	What is the product type?	Components
5	B07H9GMYXS	81NP7qh2L6L	images/small/66/665cc994.jpg	What is the product name?	Filament
6	B07H9GMYXS	81NP7qh2L6L	images/small/66/665cc994.jpg	What is the shape of the product?	Round
7	B07H9GMYXS	81NP7qh2L6L	images/small/66/665cc994.jpg	Is the brand logo visible?	Yes
8	B07H9GMYXS	81NP7qh2L6L	images/small/66/665cc994.jpg	How is the filament arranged?	Coiled
9	B07H9GMYXS	81NP7qh2L6L	images/small/66/665cc994.jpg	What is the spool material?	Plastic
10	B07H9GMYXS	81NP7qh2L6L	images/small/66/665cc994.jpg	Is the spool opaque?	No
11	B07H9GMYXS	81NP7qh2L6L	images/small/66/665cc994.jpg	What is the style?	1-Pack

Step 2: Filtering Dataset for 10 Questions (2.filter_questions.py)

- **Objective**: Filter the merged dataset to keep only items with exactly 10 question-answer pairs per item_id.
- **Input**:
  - merged_dataset.csv: Merged dataset from Step 1.
- **Process**:
  - **Load Dataset**: Read merged_dataset.csv using Pandas.
  - **Count Questions**: Group by item_id and count questions, creating a DataFrame with item_id and question_count.
  - **Identify Valid IDs**: Select item_id values with exactly 10 questions.
  - **Filter Dataset**: Keep rows where item_id matches valid IDs.
  - **Save Filtered Data**: Export to filtered_dataset_10_questions.csv without an index.
  - **Console Output**: Report row count and unique item_id count.
- **Tools**:
  - **Pandas**: Data loading, grouping, filtering, and CSV output.
- **Output**:
  - filtered_dataset_10_questions.csv: Filtered dataset with items having 10 questions.
  - Console output with row and item_id counts.

- **Screenshot of the generated csv**

	item_id	main_image_id	path	Question	Correct_Answer
2	B07H9GMYXS	81NP7qh2L6L	images/small/66/665cc994.jpg	What is the color?	Yellow
3	B07H9GMYXS	81NP7qh2L6L	images/small/66/665cc994.jpg	What is the texture?	Smooth
4	B07H9GMYXS	81NP7qh2L6L	images/small/66/665cc994.jpg	What is the product type?	Components
5	B07H9GMYXS	81NP7qh2L6L	images/small/66/665cc994.jpg	What is the product name?	Filament
6	B07H9GMYXS	81NP7qh2L6L	images/small/66/665cc994.jpg	What is the shape of the product?	Round
7	B07H9GMYXS	81NP7qh2L6L	images/small/66/665cc994.jpg	Is the brand logo visible?	Yes
8	B07H9GMYXS	81NP7qh2L6L	images/small/66/665cc994.jpg	How is the filament arranged?	Coiled
9	B07H9GMYXS	81NP7qh2L6L	images/small/66/665cc994.jpg	What is the spool material?	Plastic
10	B07H9GMYXS	81NP7qh2L6L	images/small/66/665cc994.jpg	Is the spool opaque?	No
11	B07H9GMYXS	81NP7qh2L6L	images/small/66/665cc994.jpg	What is the style?	1-Pack

Step 3: Copying Filtered Images (3.filtered_images_corrected.py)

- **Objective**: Copy images for the filtered dataset into a new directory, preserving the original structure.
- **Input**:
    - filtered_dataset_10_questions.csv: Filtered dataset with main_image_id and path.
    - Original images directory: Contains ABO dataset images.
    - Filtered images directory: New directory for filtered images.
- **Process**:
    - **Load Dataset**: Read filtered_dataset_10_questions.csv using Pandas.
    - **Extract Unique Images**: Select unique main_image_id and path pairs, dropping duplicates.
    - **Verify Count**: Print number of unique images to copy.
    - **Create Directory**: Create filtered images directory with os.makedirs.
    - **Copy Images**:
        - Iterate over unique paths with tqdm for progress.
        - Build source and destination paths.
        - Create subdirectories in the filtered directory.
        - Copy images with shutil.copy2 if source exists.
        - Warn for missing images.
    - **Console Output**: Confirm completion and report missing images.
- **Tools**:
    - **Pandas**: Dataset loading and processing.

    - **os**: Directory and path operations.
    - **shutil**: Image copying with metadata.
    - **tqdm**: Progress bar.
- **Output**:
    - Filtered images directory: Copied images in images/small/ structure.
    - Console output confirming completion and missing images.

Tools and Technologies

- **Python Libraries**:
    - **Pandas**: Data manipulation, merging, filtering, and CSV handling.
    - **os**: Directory and path operations.
    - **shutil**: Image copying with metadata.
    - **tqdm**: Progress bar.

3. Model Selection and Baseline Evaluation

Model Choices

Selected Model: Salesforce/blip-vqa-base (BLIP)

- **Description**: BLIP is a vision-language model designed for tasks like VQA, combining a vision transformer (ViT) and a BERT-based text encoder, pre-trained on diverse vision-language datasets.
- **Rationale for Selection**:
    - **Task-Specific Pre-Training**: Pre-trained on VQA datasets (e.g., VQA v2), making it well-suited for question-answering tasks on the ABO dataset.
    - **Resource Efficiency**: Smaller model size (374M parameters) enables faster fine-tuning and lower computational requirements, ideal for constrained hardware.
    - **Performance**: Demonstrates strong baseline performance on VQA tasks, with flexibility for fine-tuning to adapt to the ABO dataset's product-focused images and questions.
    - **Fine-Tuning Feasibility**: Supports efficient fine-tuning with techniques like LoRA, reducing memory and time costs while maintaining performance.
    - **Community Support**: Extensive documentation and community resources facilitate implementation and troubleshooting.
- **Implementation Plan**:
    - Fine-tune using LoRA to adapt to ABO dataset's visual and textual characteristics.
    - Leverage curated VQA pairs to optimize for product-specific questions (e.g., color, texture, shape).

Alternative Considered: Salesforce/blip2-opt-2.7b (BLIP-2)

- **Description**: BLIP-2 is an advanced vision-language model integrating a ViT with a larger OPT-2.7B language model, pre-trained on a broader range of vision-language tasks.
- **Evaluation**:
    - **Strengths**:

        - Enhanced performance due to larger model size (2.7B parameters) and advanced pre-training (e.g., on COCO, Visual Genome).
        - Better handling of complex reasoning tasks, potentially beneficial for diverse ABO question types.
    - **Weaknesses**:
        - High computational cost requires significant GPU memory and longer fine-tuning time, less feasible for limited resources.
        - Overkill for the ABO dataset, where questions are relatively straightforward (single-word answers).
        - Limited VQA-specific pre-training compared to BLIP, requiring more extensive fine-tuning.
- **Reason for Rejection**:
    - Resource constraints prioritize efficiency; BLIP-2's large size is impractical.
    - BLIP's VQA-specific pre-training aligns better with the task, reducing fine-tuning effort.
    - Simpler ABO questions don't necessitate BLIP-2's advanced reasoning capabilities.

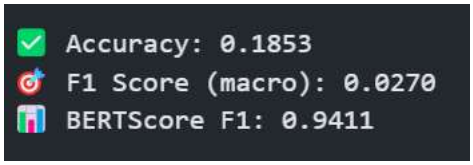
Decision Summary

- **Chosen Model**: Salesforce/blip-vqa-base selected for fine-tuning.
- **Key Factors**:
    - Efficient resource usage suits constrained environments.
    - Strong VQA pre-training aligns with ABO dataset needs.
    - LoRA fine-tuning ensures adaptability to product-specific questions.
- **Alternatives**: Salesforce/blip2-opt-2.7b rejected due to high computational demands and less task-specific pre-training.

BLIP Baseline

```
========== BASELINE EVALUATION RESULTS ==========
Accuracy: 0.2748
F1 Score: 0.2339
Average BLEU Score: 0.0498
Average ROUGE-1 F1 Score: 0.3056
Average ROUGE-L F1 Score: 0.3056
Average BERTScore F1: 0.9457
================================================
========== BASELINE EVALUATION ENDED ==========
```

BLIP2 Baseline



So based on these results and resource constraints we decided to fine tune BLIP model.

4. Fine-Tuning Runs with LoRA on various dataset size and Evaluation

We have done two fine-tuning experiments using the Low-Rank Adaptation (LoRA) technique on the Salesforce/blip-vqa-base model for a visual question-answering (VQA) task. The analysis is based on validation results from "fine-tuning-final.ipynb" (Run 1) and "fine-tuning-final_large.ipynb" (Run 2), focusing on the LoRA methodology, hyperparameters, and performance outcomes.

Fine-Tuning Process Overview

The fine-tuning process employs LoRA, a parameter-efficient fine-tuning (PEFT) method, to adapt the pre-trained Salesforce/blip-vqa-base model to a specific VQA dataset. LoRA minimizes the number of trainable parameters by introducing low-rank updates to the model's weight matrices, enabling efficient adaptation without altering the entire parameter set.

LoRA Methodology and Hyperparameters

How LoRA Works

LoRA enhances efficiency by keeping the original model weights frozen and introducing trainable low-rank updates. For a weight matrix (W), LoRA adds an update ($\Delta W = A.B$), where (A) and (B) are matrices with a rank (r). Only (A) and (B) are trained, while (W) remains unchanged. During inference, the effective weights are ($W + \Delta W$), which can be efficiently integrated into the model.

Hyperparameters

- Rank ((r)): 16
    - Defines the rank of matrices (A) and (B). A rank of 16 balances capacity and efficiency.
- Alpha: 32
    - Scales the low-rank update (ΔW), amplifying its effect. Set to 32 to enhance adaptation strength.
- Dropout: 0.05
    - Applied to the low-rank matrices during training to reduce overfitting.

- Learning rate: 5e-5, and adamW optimiser is used.
- Target Modules: Specified in the PEFT configuration, typically including query and value projections in attention layers.

Efficiency Metrics

- Trainable Parameters: 2,359,296
- Total Parameters: 387,031,868
- Percentage Trainable: 0.6096%
- Significance: Training only 0.6096% of parameters reduces memory and computational demands significantly.

Dataset Description

- **Source: The "naval2024099/vqa-vr" dataset, consisting of image-question-answer (QnA) triples.**
- **Dataset Sizes:**
    - **Run 1: 10,000 QnA pairs with 1,000 unique images.**
    - **Run 2: 50,000 QnA pairs with 5,000 unique images.**
- **Training Scale: Both runs used 5 epochs, with Run 2 leveraging a larger dataset.**

Evaluation Metrics and Results

Metrics Used

- Accuracy: Percentage of exact matches between predicted and ground-truth answers.
- F1 Score: Harmonic mean of precision and recall for answer correctness.
- BLEU Score: Measures n-gram overlap between predicted and ground-truth answers (using nltk.translate.bleu_score with smoothing).
- ROUGE-1 & ROUGE-L F1: Evaluates word and sequence overlap (rouge_score).
- BERTScore F1: Assesses semantic similarity using RoBERTa-large (bert_score).

Validation Results

- Run 1 (10,000 QnA pairs):

    - Accuracy: 0.4850
    - F1 Score: 0.4285
    - Average BLEU Score: 0.0862
    - Average ROUGE-1 F1: 0.5025
    - Average ROUGE-L F1: 0.5025
    - Average BERTScore F1: 0.9587
- Run 2 (50,000 QnA pairs):
    - Accuracy: 0.5538
    - F1 Score: 0.5043
    - Average BLEU Score: 0.0993
    - Average ROUGE-1 F1: 0.6130
    - Average ROUGE-L F1: 0.6130
    - Average BERTScore F1: 0.9692

Analysis

- **Performance Gains: Run 2 outperforms Run 1 across all metrics, attributable to the larger dataset.**
- **Key Improvements:**
    - Accuracy: Increased from 0.4850 to 0.5538 (+0.0688 or 6.88%).
    - F1 Score: Rose from 0.4285 to 0.5043 (+0.0758).
    - ROUGE Scores: Both ROUGE-1 and ROUGE-L F1 improved from 0.5025 to 0.6130 (+0.1105).
    - BERTScore F1: Advanced from 0.9587 to 0.9692 (+0.0105), reflecting better semantic alignment.

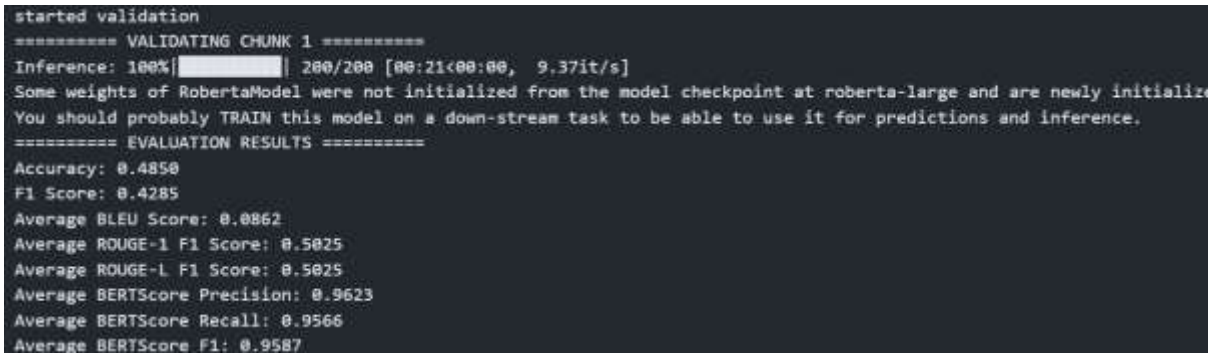
Tools and Packages Used

- pandas: Data loading and manipulation.
- torch: PyTorch for training and inference.
- transformers: Model and processor loading from Hugging Face.
- peft: Implementation of LoRA.

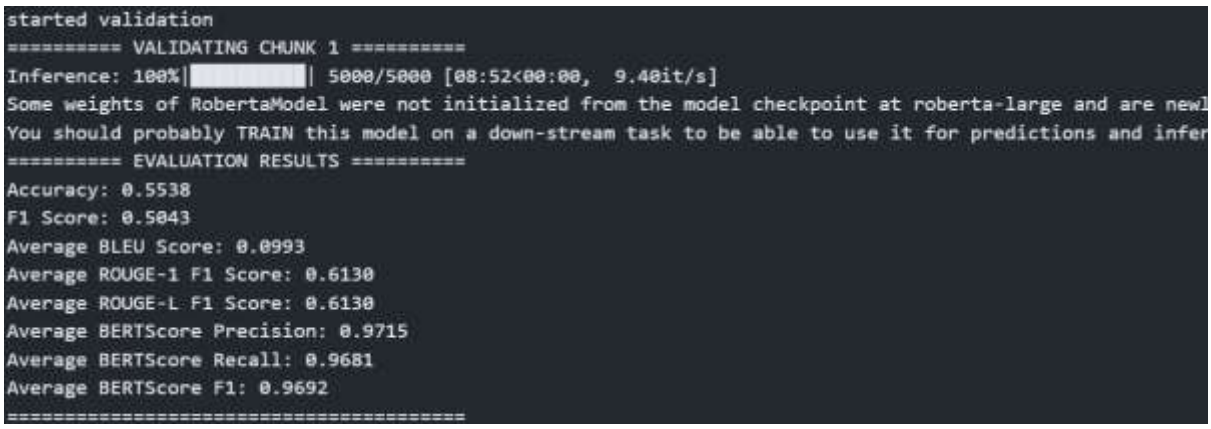
- PIL: Image preprocessing.
- sklearn.metrics: Computation of accuracy and F1 scores.
- nltk: BLEU score calculation.
- rouge_score: ROUGE metrics computation.
- bert_score: BERTScore evaluation.
- tqdm: Progress visualization during training.

Screenshots

- **Validation result on run 1**



- **Validation results on run 2**



Summary

Run 2, utilizing a larger dataset of 50,000 QnA pairs, demonstrates superior performance compared to Run 1 with 10,000 QnA pairs across all evaluation metrics. The LoRA method, with a rank of 16, alpha of 32, and dropout of 0.1, ensures efficiency by training only 0.6096% of the model's 387,031,868 parameters. This highlights LoRA's effectiveness in adapting large pre-trained models to specific tasks with minimal resource overhead.

Execution of inference script on the sample provided by teaching assistants on LMS. For more details follow the instructions.