VR Mini Project2

Aryan Singhal IMT2022036
Harsh Vardhan Singh IMT2022101

May 18, 2025

Abstract

This report describes our project on constructing a single-word Visual Question Answering (VQA) dataset using the Amazon Berkeley Objects (ABO) dataset, evaluating baseline VQA models, fine-tuning with Low-Rank Adaptation (LoRA), and assessing performance with standard and novel metrics. We discuss data curation, methodology, experiments and results.

Contents

1 Introduction

Visual Question Answering (VQA) combines computer vision and language to answer questions about images. Using the ABO dataset, we build a single-word-answer VQA benchmark.

In this project, our primary objectives are:

1. **Dataset curation:** Extract a subset of the ABO "small" split smartly, generate questions targeting key visual attributes, and enforce a single-word answer constraint.
2. **Baseline evaluation:** Measure off-the-shelf VQA model performance on our curated dataset to establish a performance baseline.
3. **Parameter-efficient fine-tuning:** Apply Low-Rank Adaptation (LoRA) to a lightweight vision–language model, optimizing for minimal additional parameters while improving VQA accuracy.This has been clubbed with other optimization measures.
4. **Novel metrics:** In addition to standard Exact Match and BertF1, introduce and analyze a new "single-word answer rate" metric and "BertF1 for single word answers" to quantify how often and how well the model's prediction conforms to the one-word requirement.

2 Downloading the Data Amazon Berkeley Objects (ABO) Dataset

- Downloaded the images and metadata csv,
- Converted to Dataset for use on Kaggle.

3 Data Processing and Sampling

3.1 Merge JSON metadata

- Define `json_folder` and glob all `*.json` files.
- Read each JSON, concatenate into `df`, write out CSV.

3.2 Drop irrelevant columns and rows

- We found the columns in the metadata and removed all that we thought were irrelevant for vqa.
- Dropped columns: `brand, item_weight, model_number, country, marketplace, domain_name, node, item_dimensions, model_year, color_code, spin_id, 3dmodel_id, finish_type, item_id, item_weight`.
- Remove duplicate rows and rows where `main_image_id` or `product_type` is `NaN`.

3.3 Consolidate other metadata

- Parse Python literals and keep only English-language values.
- Collapse the entire metadata into a single column in dictionary form:

3.4 Aggregate duplicates by `main_image_id` and `product_type` and copy metadata

- Detected duplicate rows sharing the same entries in `main_image_id` and `product_type` but with one having empty `metadata`.
- To avoid sampling issues, we filled missing metadata by copying from the corresponding non-empty row.

3.5 Dealing with duplicates in `main_image_id`

- We observed that there were still duplicates in the `main_image_id`.
- Our ultimate goal is to have equal images in every `product_type`, therefore we keep the duplicate `main_image_id` in the `product_type` with less number of total images.

3.6 Sample 53 images per `product_type` taking help from `other_image_ids`

- In total there are 576 `product_type`.Initially we thought of curating the data for 30,000 images.
- $30{,}000 \div 576 \approx 53$.Hence the number 53 was chosen.
- For each `product_type` group:
  - If #main images ≥ 53: sample 53 `main_image_id`.
  - Else: take all main and sample remaining needed images from the other image id data.
  - Note: There are still some categories that have less than 53 images but we leave them be.

3.7 Finalize and split

- Finally make a df with `path,product_type,image_id and metadata` as the main columns.We also kept `height and width` of the images in case we needed them in the future.
- Split `final_df` in half for parallel curation using 2 APIs

3.8 Reason For not using oversampling methods:

- **Risk of Spurious Feature Learning:** Some categories have only 1 item in them. Repeating the same image–question pair (e.g., a single red bag shown 53 times) can lead the model to associate "bag" exclusively with "red," rather than learning a more general concept of bag appearance.
- **No New Information Added:** Duplicated samples do not introduce novel visual or contextual cues, so the model gains no additional useful variance to improve its understanding.
- **Resource Constraints on Kaggle:** Oversampling inflates dataset size, which would give a problem with VRAM capacity and run-time limits (12 hours) on Kaggle; this forces reduction of data across product_types in favor of repeats, whereas new images would bring genuine informational gain.

3.9 Other important Observations:

- We noticed that for some photos the metadata does not match the main image but rather some other image from the *other_image_ids*.
- Also the keys names in metadata dictionary did not match with the values.

An Example is shown below:

(a) Main image for this row (b) Image actually referenced by metadata

Figure 1: Mismatch between the row's designated main image and the image referenced in its metadata


```
{
  'bullet_point': None,
  'color': 'Pack of 4',
```

```
  'item_name': 'AmazonBasics - TSA Approved 4 Digit Cable Combination Lock, Pack of 4
  'material': None,
  'model_name': None,
  'item_keywords': None,
  'style': None,
  'fabric_type': None,
  'item_shape': None,
  'pattern': None,
  'product_description': None
}
```

- The main image corresponding to this metadata is shown in Figure 1a. That image shows a single lock and not a pack of 4 locks. The metadata given accurately describes one of the "other" images in that row as shown in Figure 1b, which contains a pack of four locks.
- The metadata also incorrectly uses the `color` key to record "Pack of 4."

If the Gemini API relies solely on this misaligned metadata, with misplaced attributes it will generate incorrect Q&A pairs —so we must account for this in our prompt engineering step.

4 Dataset Curation

4.1 Prompt Engineering

To generate high-quality, single-word question–answer pairs for e-commerce product images, we designed and refined a structured prompt template based on best practices from the Google TechAI Flow whitepaper [1]. Our pipeline integrates image data, metadata, and rigorous validation to ensure consistency and relevance.

- **Prompt Template:** We crafted a single, comprehensive prompt that frames the model as "an expert in generating questions and answers for e-commerce product images." The template specifies:
  - *Inputs:* A product image and its metadata dictionary (e.g. name, category, color, material, brand).
  - *Task:* Produce exactly six questions with one-word answers, of which at least three may leverage metadata cues (if present) (e.g. "What is the color of the shoe?") and at least three must focus purely on visual attributes (e.g. "What is the pattern?").
  - *Constraints:* All answers are single words; questions rely only on observable image content; the model should ignore external knowledge.
  - *Output Format:* A JSON array of objects, each with `question` and `answer` fields.
- **Metadata Verification:** We instruct the model to cross-check the provided metadata against the actual image This was done in order to account for the inconsistencies in metadata discussed in the previous section.

If left unchecked, the API would generate QA pairs based on misaligned metadata attributes. By explicitly asking the model to verify that each metadata field corresponds to what's visible, we guard against incorrect question–answer generation.

- **Implementation:** Using the `google-generativeai` Python client, we load each image via PIL and serialize its metadata to JSON. We then invoke the Gemini 2.0 "flash" model with our prompt and the encoded image, requesting `application/json` output conforming to a Pydantic schema (`QAGenerationOutput`). This schema enforces the presence of exactly six valid question–answer pairs.
- **Validation and Robustness:**
  - *JSON Parsing:* We attempt to parse the model's response text and raise an error on invalid JSON.
  - *Schema Validation:* Pydantic ensures adherence to our expected structure, with explicit errors if the model deviates.
  - *Retry Logic:* In our batch loop over $\sim$ 22,000 images, failed indices are re-tried until successful, with a 0.5 s pause between calls to respect rate limits.
- **Literature Guidance:** Our prompt design and iterative refinement draw on principles outlined in the Google TechAI Flow whitepaper on prompt engineering, particularly the emphasis on explicit task framing, schema enforcement, and metadata-aware question generation [1].

4.2 Iterative Improvement in Prompt Engineering

After generating an initial QA batch, we performed a manual audit and identified two recurring issues in the model's outputs:

1. **Incorrect binary responses:** Questions phrased with *Is*/*Are*/*Does*/etc. sometimes produced the predicate itself (e.g. "Foldable") instead of the required Yes or No.
2. **Overly specific material labels:** When asked "What material is this?", the model often returned fine-grained subtypes (e.g. "Copper", "Aluminium") rather than a high-level category.

To enforce the desired output formats, we augmented our prompt with two explicit rules:

- **Binary (Yes/No) Questions:** "If a question begins with *Is*, *Are*, *Does*, *Do*, *Has*, or *Have*, the answer must be exactly Yes or No, with no other variations."
- **Material Abstraction:** "For any question asking 'What material...?', the answer must be one of the following generic categories only: metal, plastic, wood, glass, ceramic, fabric, leather, rubber, paper, cardboard. Do not use more specific subtypes."

Embedding these instructions directly into the prompt immediately corrected both issues: all binary questions began returning strictly "Yes" or "No," and material questions were abstracted to the prescribed, high-level categories. This iterative refinement significantly improved the consistency and usability of our curated QA pairs.

5 Test Data Curation

To evaluate both our off-the-shelf and fine-tuned VQA models under realistic conditions, we built a held-out test set of 1 000 product images and their metadata. Our process mirrors the previous section in curation part but begins with external public data.

5.1 Data Acquisition

1. **Source and sampling:** We used the CrossingMinds "shopping-queries-image" dataset, which provides a Parquet file of product IDs and image URLs. We loaded the entire table, dropped any rows missing `product_id` or `image_url`, and then randomly sampled 1 000 rows (`random_state=42`) to ensure reproducibility.
2. **Image download:** For each sampled row, we fetched the image URL with a 10 s timeout. Invalid or unreachable URLs were skipped and counted as failures.

5.2 Curating the Dataset

We then applied the same curation procedure described in the previous section. Note that here the Iteratively Improved version of Prompt Engineering was used to curate the test data.

The result is a fully self-contained test set of 1000 images ready for inference with both baseline and fine-tuned VQA models.

6 Metrics Used

To comprehensively evaluate our one-word VQA benchmark, we employ four complementary metrics:

6.1 Exact Matchness

Exact Match (EM) measures the fraction of predictions that exactly match the ground-truth answer (case-insensitive string equality). For N examples:

$$\text{EM} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}(\hat{a}_i = a_i).$$

EM provides a strict measure of correctness, directly reflecting whether the model produces the exact expected one-word answer.

6.2 BERT F1

BERT F1 adapts the BERTScore framework to compute token-level precision, recall, and F1 by aligning contextual embeddings of predicted and reference answers. It rewards semantic similarity even if the surface form differs (e.g., synonyms):

$$\text{BERT F1} = 2 \times \frac{P \times R}{P + R},$$

where P and R are the embedding-based precision and recall scores. This semantic metric complements EM by capturing near-correct or synonymous predictions.

6.3 Percentage of One-Word Answers (Self-thought)

To enforce our single-word answer constraint, we compute:

$$\text{OneWord\%} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}\Big(\text{tokens}(\hat{a}_i) = 1\Big).$$

A high OneWord% indicates consistent compliance with the dataset format, ensuring the model adheres to the one-word requirement.

6.4 BERT F1 for One-Word Answers (Self-thought)

We further isolate semantic quality within valid one-word outputs by restricting the BERT F1 calculation to those predictions that contain exactly one token:

$$\text{BERT F1}_{\text{1-word}} = \frac{1}{N'} \sum_{i:\,\text{tokens}(\hat{a}_i)=1} \text{BERT F1}(\hat{a}_i, a_i),$$

where N' is the number of one-word predictions. This focused metric decouples semantic evaluation from length adherence.

By combining a strict exact-match measure, a semantic embedding-based F1, and two one-word-specific metrics, we gain a holistic view of model performance—spanning raw correctness, semantic similarity, and format compliance.“‘

7 Choosing the Models

7.1 Off-the-Shelf Models

To establish a performance baseline, we selected a diverse set of both parameter-efficient “small” models (tens to hundreds of millions of parameters) and high-capacity “large” models (several billions of parameters).

The models we considered are:

- **vilt-b32-finetuned-vqa (117M)** A ViLT model with 32×32 patch embeddings and approximately 86 M parameters. ViLT eliminates a convolutional backbone by embedding raw image patches directly, then jointly encoding them with text tokens in a unified transformer for efficient multimodal fusion.
- **Salesforce/blip-vqa-base(385M)** This Blip based VQA model uses a frozen vision encoder paired with a lightweight language decoder, fine-tuned on standard VQA benchmarks to generate concise, context-aware answers.
- **Qwen/Qwen2-VL-2B-Instruct(2.21B)** A vision–language transformer from the Qwen2 family, instruction-tuned for VQA tasks. It couples large-scale image–text pretraining with guided fine-tuning, enabling robust comprehension of diverse visual questions.

- **Qwen/Qwen2.5-VL-3B-Instruct(3.75B)** The next-generation Qwen2.5 model scaled to 3 B parameters. With deeper transformer layers and wider attention heads, it achieves richer visual reasoning and more detailed answer generation.

We tested and chose the Qwen model family because they can natively handle images at varying resolutions without resizing the images internally.

- **paligemma-3b-pt-224(2.92B)** A PaliGemma model trained on mixed-resolution (224×224) inputs. It blends Pali's language capabilities with Gemma's multimodal pretraining to balance efficiency and accuracy across VQA scenarios.This model was chosen for it's extremely fast inference time which will be shown later.

- **gemma-3-4b-it(4.3B)** A Gemma-based multimodal transformer in the 3–4 B parameter range, instruction-tuned for interactive VQA. It supports multilingual queries and adaptive reasoning, making it well-suited for complex e-commerce image questions.

The Gemma models internally resize images to a fixed resolution—potentially distorting key features—but in return deliver significantly faster inference times.

By comparing these off-the-shelf models on our curated single-word VQA dataset, we can quantify the trade-off between model size and baseline accuracy prior to any parameter-efficient fine-tuning.

7.2 Baseline Inference Results

We evaluated each off-the-shelf model on our test data and measured both speed and accuracy according to our four metrics:

7.2.1 vilt-b32-finetuned-vqa (117M parameters)

The ViLT-B32 model completed inference in **213.5 s**, or **35.6 ms** per image. Its performance metrics were:

- **Exact-Match Accuracy:** 0.46
- **BERTScore F1 (all predictions):** 0.83
- **One-Word Compliance:** 98.70%
- **BERTScore F1 (one-word only):** 0.84

As we can see, the results on the baseline itself with the vilt model was quite good despite being a smaller model. So, we thought it would perform very nicely upon fine-tuning. But, we found an interesting insight about it when we were trying to finetune it. It will be covered in the finetuning section.

7.2.2 Salesforce/blip-vqa-base (385M parameters)

This model processed the entire test set in **538.9 s**, averaging **89.9 ms** per image. Its performance metrics were:

- **Exact-Match Accuracy:** 0.58
- **BERTScore F1 (all predictions):** 0.81
- **One-Word Compliance:** 95.51%
- **BERTScore F1 (one-word only):** 0.85

7.2.3 Qwen2-VL-2B-Instruct (2.21B parameters)

Iterative Improvements

1. **Iteration 1 (Without one-word-answer constraint):** The Qwen2-VL-2B-Instruct without one-word-answer constraint completed inference in **2059.5 s**, or **343.6 ms** per image. Its performance metrics were:

   - **Exact-Match Accuracy:** 0.00
   - **BERTScore F1 (all predictions):** –0.08
   - **One-Word Compliance:** 0.00%
   - **BERTScore F1 (one-word only):** NaN

   As we can see, that the baseline model gave abysmally poor results which we never expected. When we looked back again to see what the problem could be, we found that Qwen2-VL-2B-Instruct is an image text to text description model not a VQA model. So, whenever an image and a question was going for input into the model, the model generated a full sentence instead of generating a single word answer that we wanted. Hence, that is why, the BERT score and exact match scores were so poor (basically 0) because it was matching the values of ground truth (which was a single word) to a prediction which was a full sentence. We countered this issue and ran the model again in the 2nd iteration.

2. **Iteration 2 (With one-word-answer constraint):** The Qwen2-VL-2B-Instruct with one-word-answer constraint completed inference in **1208.9 s**, or **201.7 ms** per image. Its performance metrics were:

   - **Exact-Match Accuracy:** 0.57
   - **BERTScore F1 (all predictions):** 0.78
   - **One-Word Compliance:** 89.22%
   - **BERTScore F1 (one-word only):** 0.86

   To counter the issue that we faced in the 1st iteration, we added a prompt - "Answer in exactly one word" before passing each question as the input to the model so as to forcibly make the model give one word answers to the questions. This had a great improvement as compared to the 1st iteration as you can see from the scores shown here.
   We can also see that the inference time has also been reduced because it takes less computation to give a single word answer as compared to a whole sentence.

7.2.4 Qwen2.5-VL-3B-Instruct (3.75B parameters)

Iterative Improvements

1. **Iteration 1 (Without one-word-answer constraint):** The Qwen2.5-VL-3B-Instruct without one-word-answer constraint completed inference in **2563.8 s**, or **427.7 ms** per image. Its performance metrics were:

   - **Exact-Match Accuracy:** 0.00
   - **BERTScore F1 (all predictions):** –0.13
   - **One-Word Compliance:** 0.00%

- **BERTScore F1 (one-word only):** NaN

Again, we can see that the same problem arises here (and thus, we get abysmal scores) that challenged us in the previous model for the 1st iteration. We perform similar step in the 2nd iteration to counter it.

2. **Iteration 2 (With one-word-answer constraint):** The Qwen2.5-VL-3B-Instruct with one-word-answer constraint completed inference in **1832.6 s**, or **305.7 ms** per image. Its performance metrics were:

   - **Exact-Match Accuracy:** 0.43
   - **BERTScore F1 (all predictions):** 0.74
   - **One-Word Compliance:** 89.52%
   - **BERTScore F1 (one-word only):** 0.81

   We again send the prompt - "Answer in exactly one word" before passing each question as the input to the model so as to forcibly make the model give one word answers to the questions. This gave us much better results as you can see.
   Here also, we can see that the inference time has also been reduced because it takes less computation to give a single word answer as compared to a whole sentence.

7.2.5 google/paligemma-3b-mix-224 (2.92B parameters)

The paligemma-3b-mix-224 model completed inference in **4418.2 s**, or **737.1 ms** per image. Its performance metrics were:

- **Exact-Match Accuracy:** 0.63
- **BERTScore F1 (all predictions):** 0.79
- **One-Word Compliance:** 93.03%
- **BERTScore F1 (one-word only):** 0.84

We used the knowledge learned from the iterations in Qwen 2.0 and Qwen 2.5 and gave the model the constraint of "Answering in one word" knowing that it would perform better as it is also an image text to text model.

7.2.6 google/gemma-3-4b-it (4.3 B parameters)

The gemma-3-4b-it model completed inference in **27189.4 s**, or **4536.1 ms** per image. Its performance metrics were:

- **Exact-Match Accuracy:** 0.41
- **BERTScore F1 (all predictions):** 0.77
- **One-Word Compliance:** 100.00%
- **BERTScore F1 (one-word only):** 0.77

We once again used the knowledge learned from the iterations in Qwen 2.0 and Qwen 2.5 and gave the constraint of "Answering in one word".
This was the largest model that we tested and tried to finetune. As it is evident, because of the size of this model, the inference time is quite slow. But, the ability of the model to only give one word answers is exceptional. Sadly, we faced some problems while fine-tuning it which are focused later in the finetuning section.

Metric vs Baseline Models

1. Exact Match vs Baseline Models:

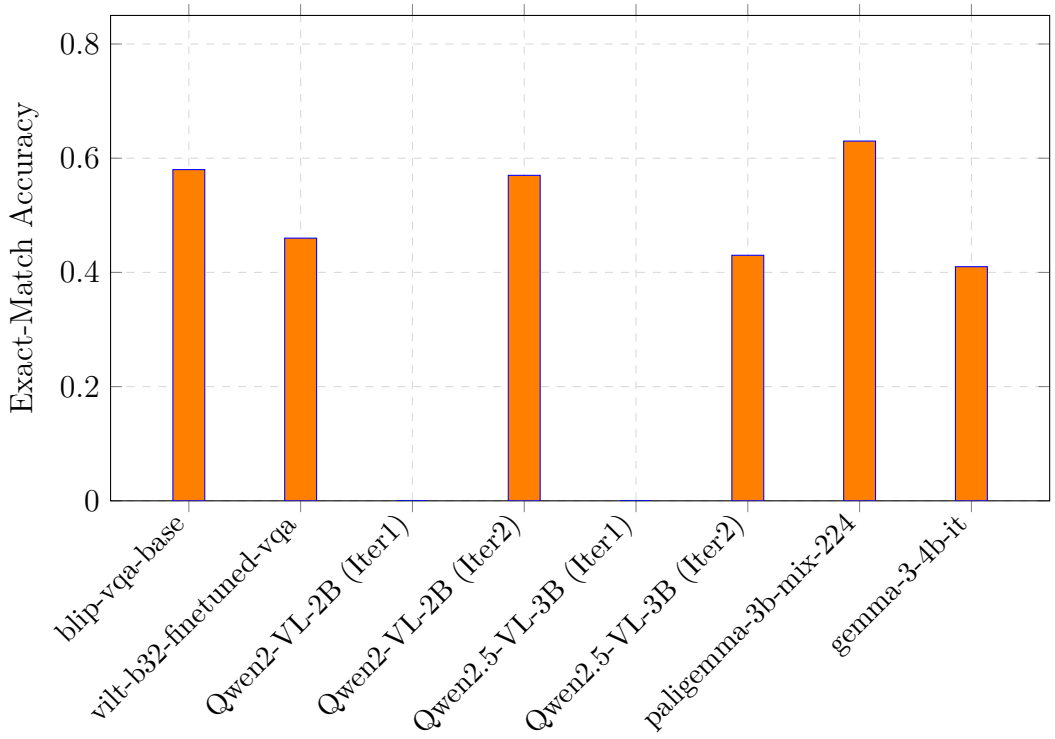

Figure 2: Exact-Match Accuracy across all baseline models and iterations.

2. BERTScore F1 vs Baseline Models:

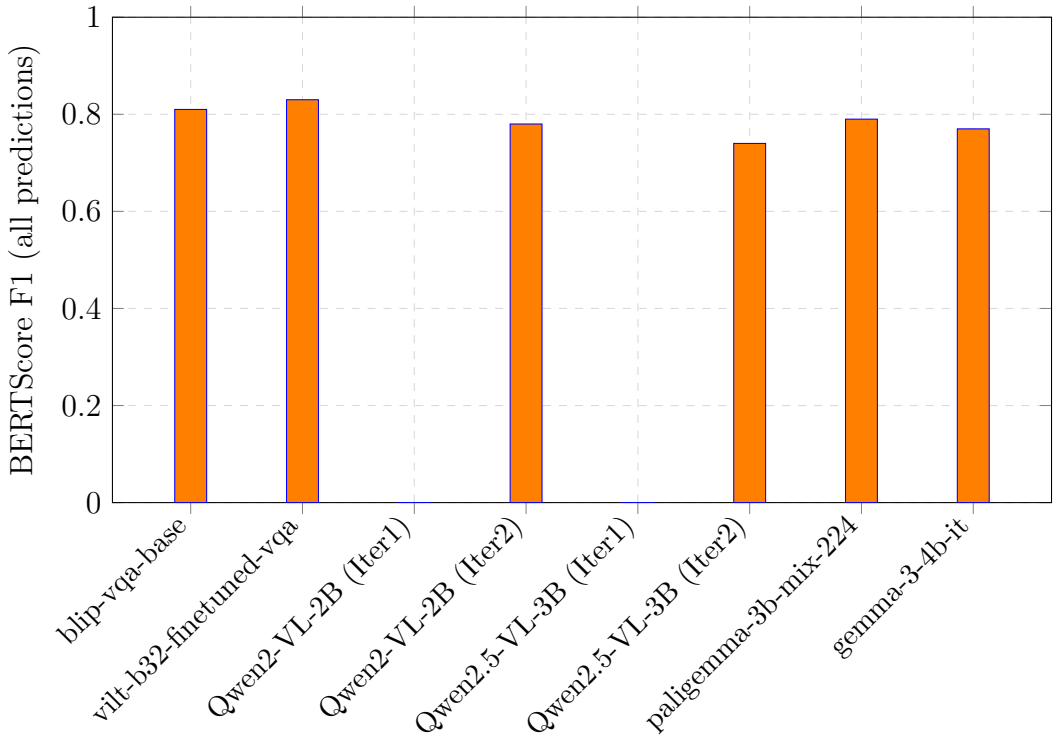

Figure 3: BERTScore F1 (all predictions) across all baseline models and iterations.

3. One-Word Compliance vs Baseline Models:

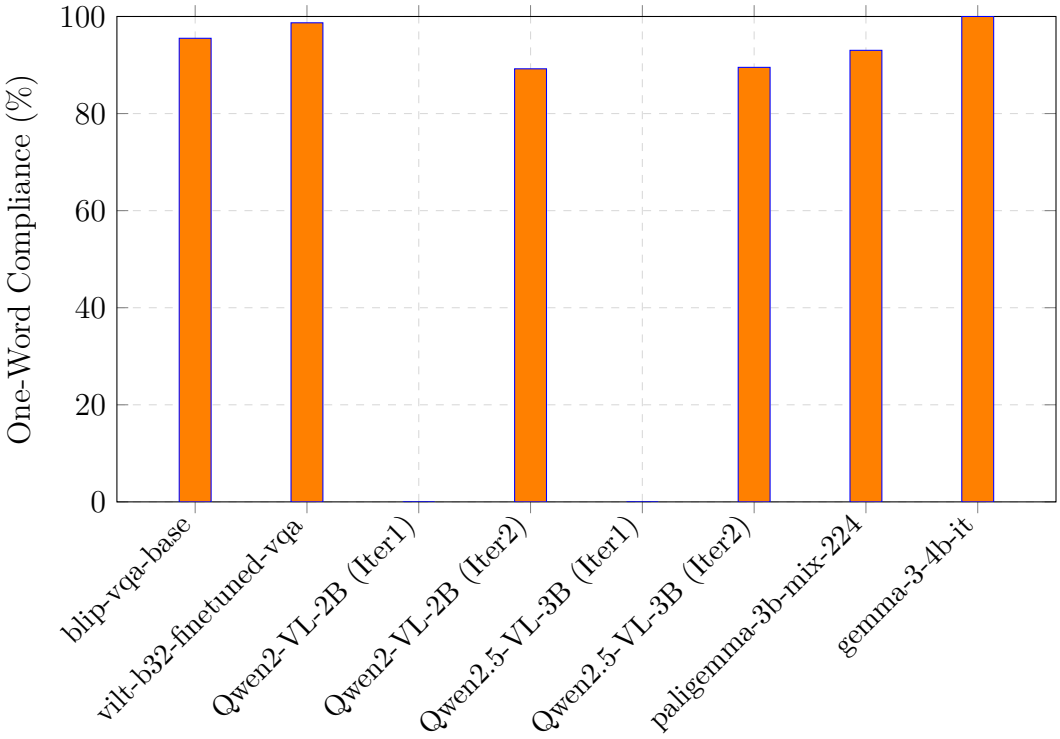

Figure 4: Percentage of one-word-only predictions across all baseline models and iterations.

4. BERTScore F1 for One Word vs Baseline Models:

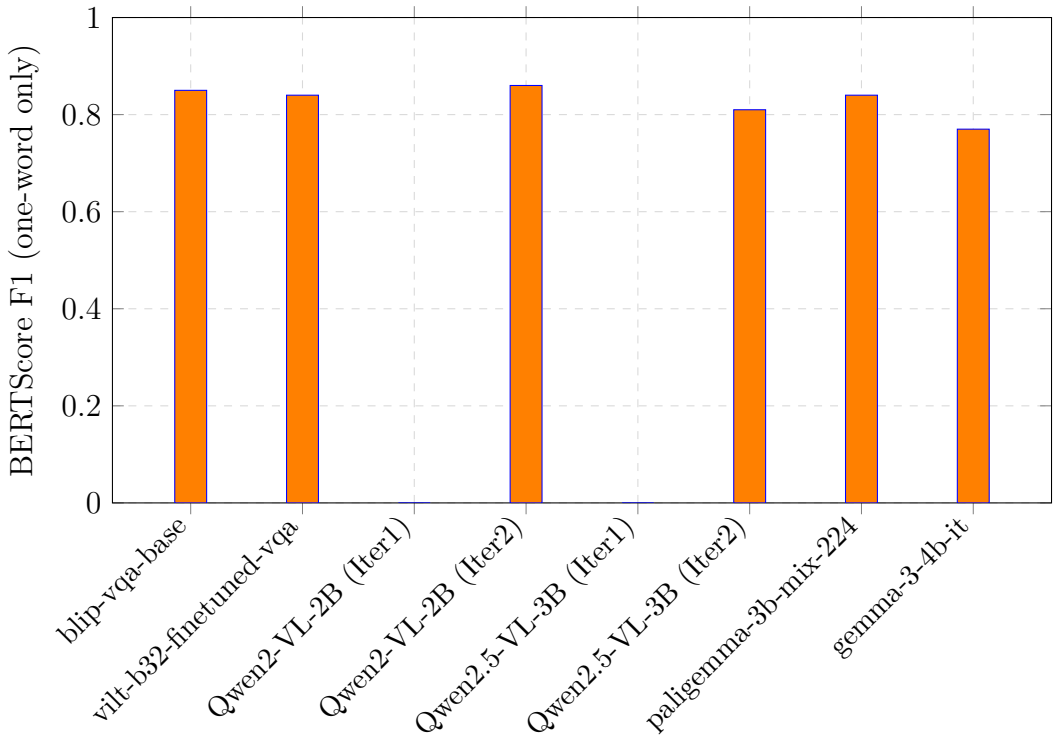

Figure 5: BERTScore F1 computed only on one-word-only predictions across all baseline models and iterations.

8 Fine-Tuning with LoRA

- **LoRA setup:**
  - Rank of 16 for all models.
  - Fine-tuned both attention and MLP layers.
- **Batching and optimization:**
  - Batch sizes chosen to fit available GPU VRAM.
  - Gradient accumulation set to 1 (doubling it would double step time, so we preferred more steps). Only in Blip was it set as 2 but Blip Early Stopped
  - Cosine learning-rate scheduler used throughout for non-linear decay.
- **Data splits and scheduling:**
  - 90–10 train/validation split on the ABO dataset.
  - Training steps (or epochs) capped to respect Kaggle's 12-hour limit.
- **Early stopping:**
  - Patience set to 2 to allow minor rises in validation loss.
  - Ensures $\geq$ 3–4 validation checks and retains the best checkpoint if training stops.
  - In Blip the patience was set as 4 because it allowed many validation runs owing to it's small size.

8.1 dandelin/vilt-b32-finetuned-vqa (117M) - Couldn't be Finetuned

8.1.1 Parameters and GPU Used

- per_device_batch_size = 32
- epochs = 1
- GPU: 2*T4

8.1.2 Optimizations Used

No optimizations were needed as the model was sufficiently small to fit in full precision on Kaggle VRAM and was fast during inference as well.

8.1.3 Problems Encountered

- While inspecting training for one epoch, we saw the `train_loss` drop to 0 by step 900, with `val_loss` likewise nearing 0.
- A manual review revealed most outputs were simply "unavailable" or random tokens.
- We discovered this stems from ViLT's fixed classifier over a small vocablury rather than a generative decoder.(see [2]).
- **Not Trainable:** This vocabulary bottleneck means that ViLT cannot be meaningfully fine-tuned to produce answers outside its predefined vocabulary.

8.1.4 Results

- No metrics were computed in this model since all answers observed visually were wrong and it did not have the capacity to learn the correct answers because of the model's limited vocabulary.

8.2 Salesforce/blip-vqa-base(385M)

8.2.1 Parameters and GPU Used

- per_device_batch_size = 32
- epochs = 10
- GPU: 2*T4

8.2.2 Optimizations Used

No optimizations were needed as the model was sufficiently small to fit in full precision on Kaggle VRAM and was fast during inference as well.

8.2.3 Problems Encountered

No Special Problems were encountered during finetuning this model.

8.2.4 Results

The Salesforce/blip-vqa-base finetuned model completed inference in **552.5 s**, or **92.2 ms** per image. Its performance metrics were:

- **Exact-Match Accuracy:** 0.71
- **BERTScore F1 (all predictions):** 0.84
- **One-Word Compliance:** 99.17%
- **BERTScore F1 (one-word only):** 0.85

8.2.5 Comparison With Baseline

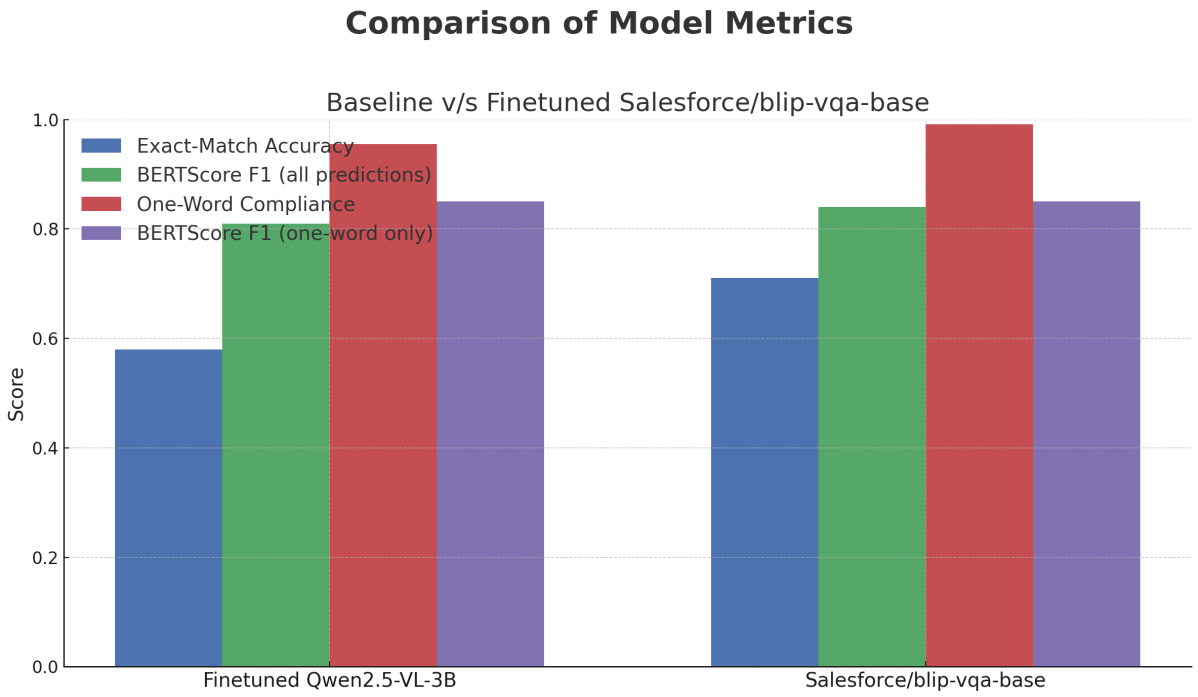

Figure 6: Blip model Baseline v/s finetuned performance

Change in model metrics after Finetuning:

- **Exact-Match Accuracy:** + 0.13
- **BERTScore F1 (all predictions):** + 0.03
- **One-Word Compliance:** + 3.66 %
- **BERTScore F1 (one-word only):** 0.00

1. We believe the modest gains are due to BLIP's smaller parameter count, which limits its capacity to learn.

2. Note: training was halted by early stopping at epoch 9.

8.3 Qwen/Qwen2-VL-2B-Instruct(2.21B)

This model was chosen for it's capacity to deal with images of different resolutions without needing to resize them.

8.3.1 Parameters and GPU Used

- per_device_batch_size = 8
- steps = 16000/8500
- GPU: 2*T4/P100

8.3.2 Optimizations Used

- **16 bit quantization:** The model was loaded in FP16 precision to fit within Kaggle's GPU VRAM limits
- **Validation subset:** After observing steadily decreasing validation loss in one iteration, we limited the validation set to 50% of the original in the next iteration—reducing validation from $\approx$ 1.5 hrs to under 45 mins..

8.3.3 Problems Encountered

- **Image size:** Qwen2-VL requires each image dimension to be at least 28 px, so we filtered out any images smaller than this threshold.
- **GPU utilization:** Qwen's built-in multi-resolution support prevents multi-GPU training on Kaggle. This issue has been extensively discussed on Github Forums and is still Open (see [3]), so we could either train on a single GPU or preprocess by resizing all images to a fixed resolution.We have tried both!

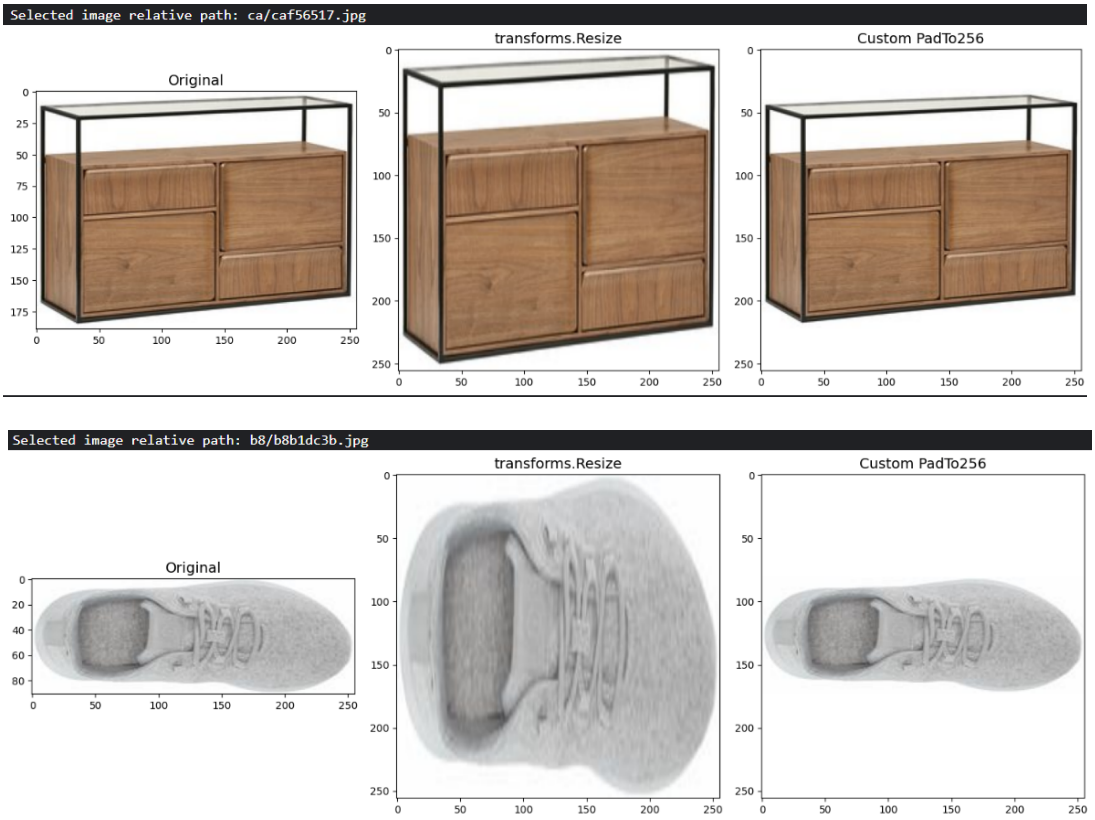

Figure 7: Comparison of Resizing Techniques - a)Original Image b) Resized using transforms.resize() c)Custom resize Logic(Padto256)

- **Aspect ratio:** Applying the standard `Resize` distorts shapes, which can break shape-focused questions (e.g., "What is the shape?").(Refer Fig.7) To address this, we implement a custom logic that resizes the images while maintaining aspect ration and then makes all remaining pixels white.

- **Custom Trainer subclass with batch-dropping:** We defined *CustomTrainer* as a subclass of *Seq2SeqTrainer* and set *drop_last=True* in both the training arguments and the overridden *get_eval_dataloader* to skip the final smaller-than-batch-size minibatch and avoid tensor-stacking errors.

8.3.4 Iterative Improvements

1. **Iteration 1: Without One Word Answer Prompt** The Qwen2-VL-2B-Instruct finetuned model without one-word-answer constraint completed inference in **1069.9 s**, or **178.5 ms** per image. Its performance metrics were:

   - **Exact-Match Accuracy:** 0.15
   - **BERTScore F1 (all predictions):** 0.49
   - **One-Word Compliance:** 73.54%
   - **BERTScore F1 (one-word only):** 0.64

2. **Iteration 2: With One Word Answer Prompt + No External Resize (P100)** In this iteration, we leveraged Qwen2-VL-2B-Instruct's native support for multi-resolution images. Due to the GPU utilization constraints discussed earlier,

all inferences were performed on a single P100 rather than a 2×T4 setup.
The fine-tuned model with one-word answer prompt and no external resizing completed inference in **6,274.2 s**—approximately **1,046.8 ms** per image. Its performance metrics were:

- **Exact-Match Accuracy:** 0.50
- **BERTScore F1 (all predictions):** 0.66
- **One-Word Compliance:** 74.96%
- **BERTScore F1 (one-word only):** 0.85

3. **Iteration 3: With One Word Answer Prompt + External Resize** In this iteration we introduced our custom resize logic as discussed above.
The Qwen2-VL-2B-Instruct finetuned model with one-word-answer and external resizing constraint completed inference in **619.2 s**, or **103.3 ms** per image. Its performance metrics were:

- **Exact-Match Accuracy:** 0.66
- **BERTScore F1 (all predictions):** 0.84
- **One-Word Compliance:** 96.28%
- **BERTScore F1 (one-word only):** 0.87

8.3.5 Results

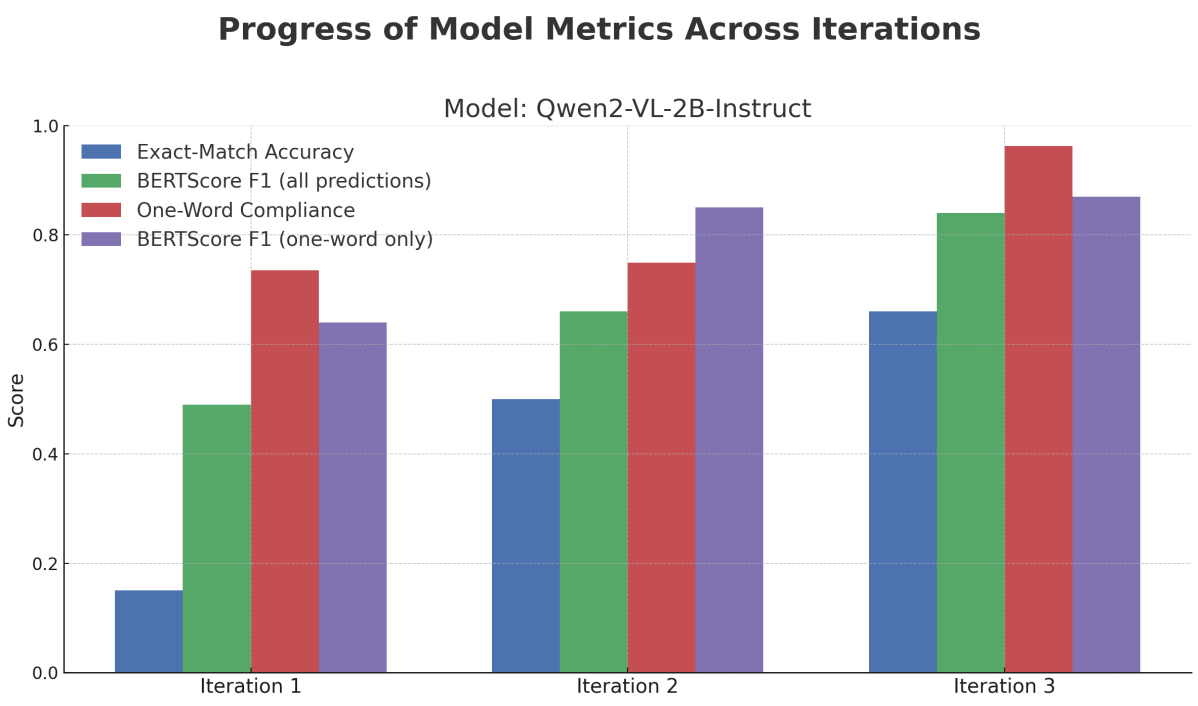

Figure 8: Iterative improvements in the Qwen2 finetuned model.

1. As shown in Figure 8, each iteration produced consistent improvements across all metrics.

2. The Iteration 3 model (the final iteration) was used as the benchmark for the final results.

3. The huge increase in performance between iterations 2 and 3 can also be attributed to the fact that after implementing our custom resizing we were able to fit double the number of training steps. (From 8500 to 16000)

8.3.6 Comparison With Baseline

- Iteration 1 for both the baseline and the fine-tuned model did not have one-word constraint, so we compare those two directly.
- Iteration 2 of the baseline and Iterations 2 and 3 of the fine-tuned model all used the one-word prompt; hence, we group those three for comparison.Also for plotting purposes we only plot the baseline iteration 2 with fine-tuned iteration 2 since it gave the best results.

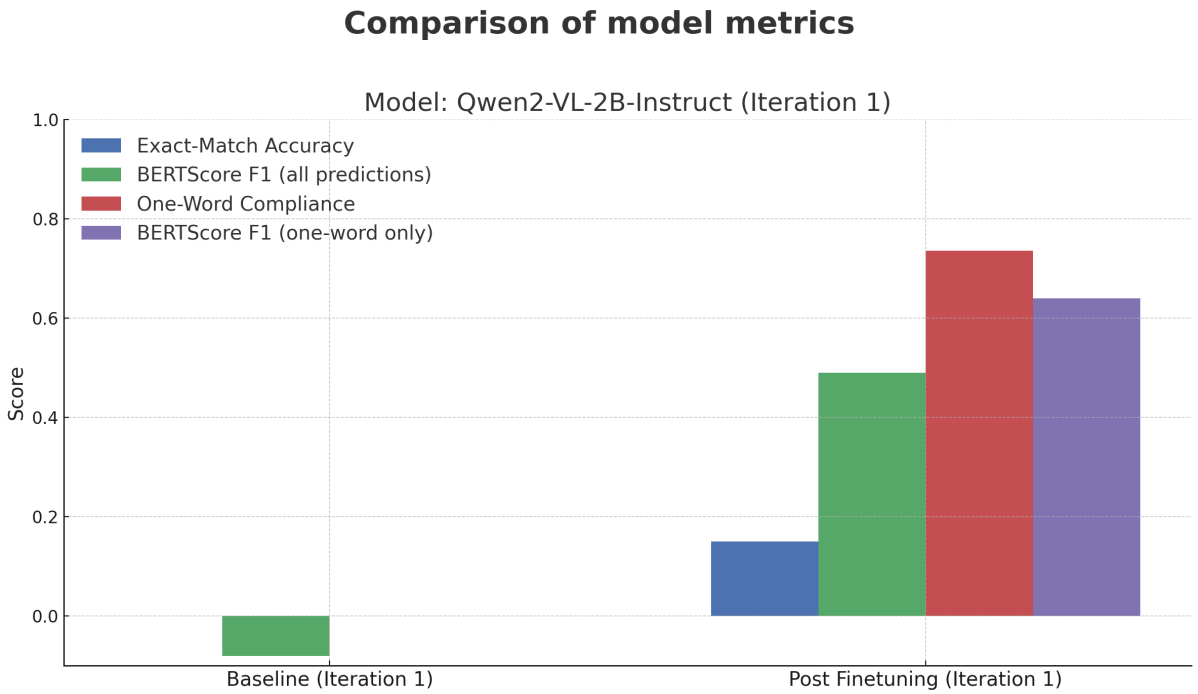

Figure 9: Qwen2 model Baseline v/s finetuned performance without prompt.

Change in model metrics after Finetuning:

- **Exact-Match Accuracy:** + 0.15
- **BERTScore F1 (all predictions):** + 0.57
- **One-Word Compliance:** + 73.54 %
- **BERTScore F1 (one-word only):** + 0.64

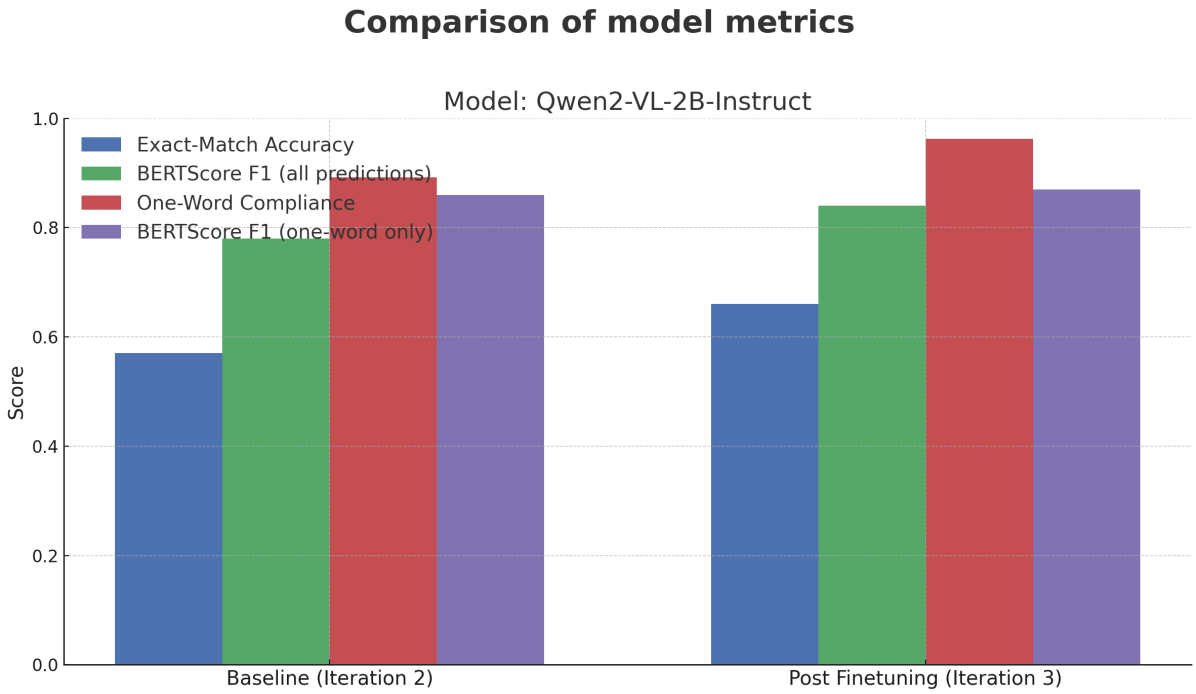

Figure 10: Qwen2 model Baseline v/s finetuned performance with prompt.

Change in model metrics after Finetuning:

- **Exact-Match Accuracy:** + 0.09
- **BERTScore F1 (all predictions):** + 0.06
- **One-Word Compliance:** + 7.54 %
- **BERTScore F1 (one-word only):** + 0.01

1. In both cases we see huge improvement gains thanks to the fact that we were able to tune the model for a decent number of steps.

8.4 Qwen/Qwen2.5-VL-2B-Instruct(3.75B)

8.4.1 Parameters Used

- per_device_batch_size = 16
- steps = 4800
- GPU : P100

8.4.2 Optimizations Used

- **4-bit quantization:** We used BitsAndBytes to load the model in 4-bit precision on Kaggle, otherwise it would not fit.
- **Validation subset:** After observing steadily decreasing validation loss in one iteration, we limited the validation set to 2,000 examples in the next iteration—reducing validation time from 1.5–2 hrs to under 10 mins in favour of increasing number of steps.

8.4.3 Problems Encountered

- **Image size:** Qwen2.5-VL demands at least 28 px in each dimension, so we filtered out any smaller images.
- **GPU utilization:** Due to a BitsAndBytes multi-GPU placement bug, we were limited to a single P100 GPU.This is an Open issue extensively discussed on Github Forums(see [4]).

8.4.4 Iterative Improvements

1. **Iteration 1: Without One Word Answer Prompt + No Resize** The Qwen2.5.-VL-3B-Instruct finetuned model without one-word-answer prompt completed inference in **3214.3 s**, or **536.2 ms** per image. Its performance metrics were:
   - **Exact-Match Accuracy:** 0.00
   - **BERTScore F1 (all predictions):** -0.10
   - **One-Word Compliance:** 0.08%
   - **BERTScore F1 (one-word only):** 0.79
2. **Iteration 2: With One Word Answer Prompt + External Resize** The Qwen2.5.-VL-3B-Instruct finetuned model with one-word-answer prompt and external resizing using custom code completed inference in **1943.8 s**, or **324.3 ms** per image. Its performance metrics were:
   - **Exact-Match Accuracy:** 0.27
   - **BERTScore F1 (all predictions):** 0.71
   - **One-Word Compliance:** 93.93%
   - **BERTScore F1 (one-word only):** 0.75

8.4.5 Results

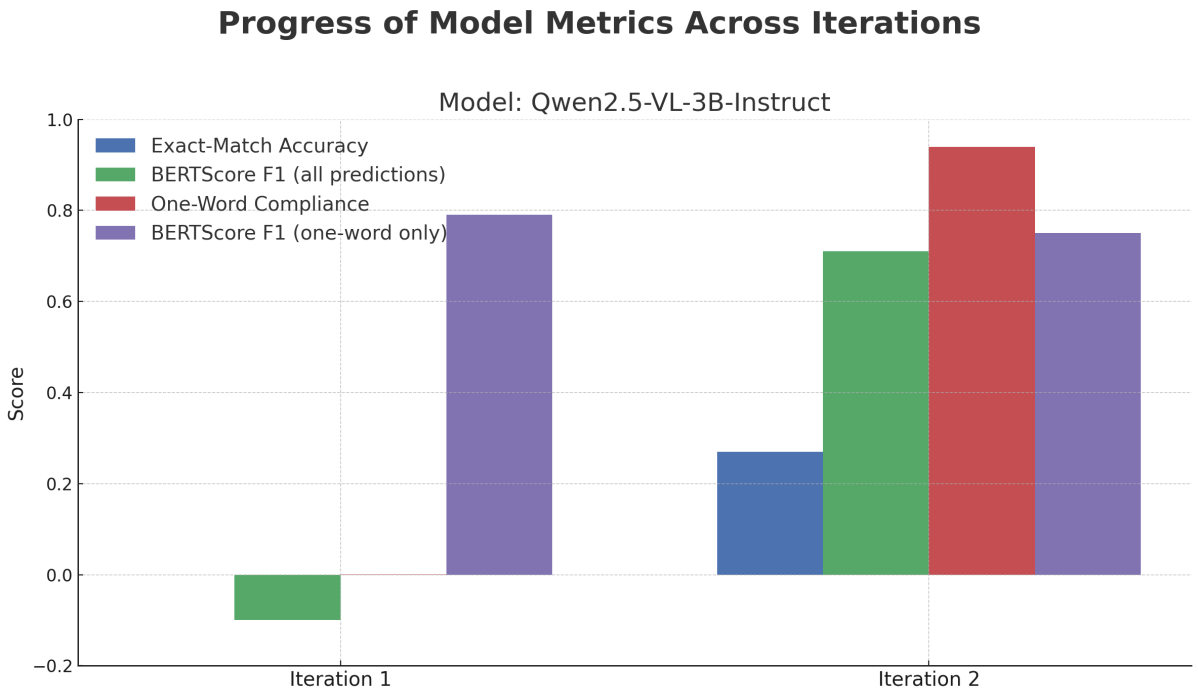

Figure 11: Iterative improvements in the Qwen2.5 finetuned model.

1. As shown in Figure 11, each iteration produced consistent improvements across all metrics.

2. The Iteration 2 model (the final iteration) was used as the benchmark for the final results.

3. We believe that the metrics for iteration 1 are this low because of the fact that were only able to train this model for 2400 steps which is not even 1 epoch because of the resizing problem and also because of higher inference time.

8.4.6 Comparison With Baseline

- Iteration 1 for both the baseline and the fine-tuned model did not one-word constraint, so we compare those two directly.

- Iteration 2 of the baseline and Iterations 2 the fine-tuned model all used the one-word prompt; hence, we group those two for comparison.

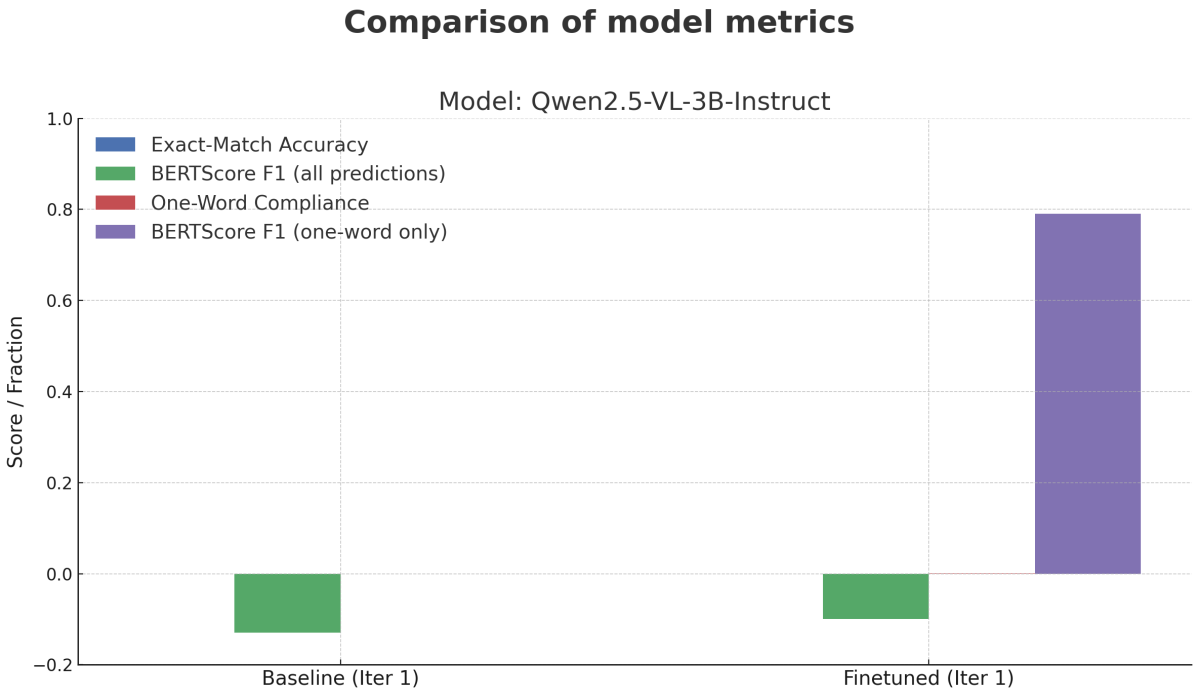

Figure 12: Qwen2.5 model Baseline v/s finetuned performance without prompt.

Change in model metrics after Finetuning:

- **Exact-Match Accuracy:** 0.00
- **BERTScore F1 (all predictions):** + 0.03
- **One-Word Compliance:** + 0.08 %
- **BERTScore F1 (one-word only):** + 0.79

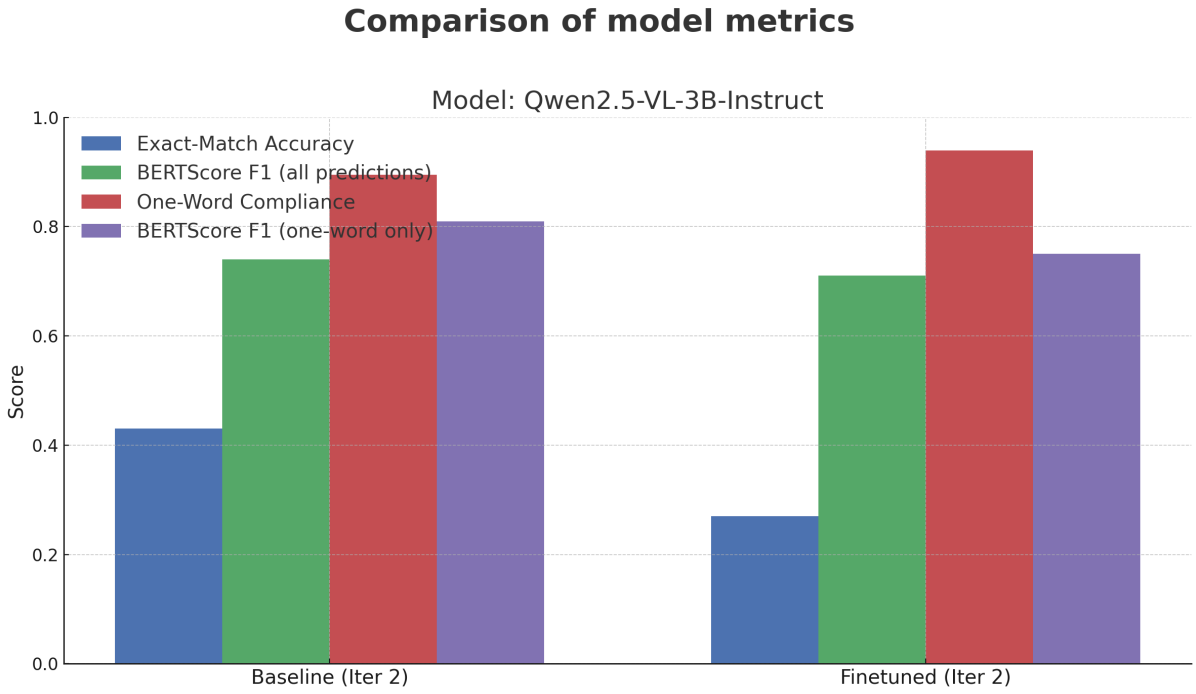

Figure 13: Qwen2.5 model Baseline v/s finetuned performance with prompt.

Change in model metrics after Finetuning:

- **Exact-Match Accuracy:** - 0.16
- **BERTScore F1 (all predictions):** - 0.03
- **One-Word Compliance:** + 4.41 %
- **BERTScore F1 (one-word only):** - 0.06

Why Fine-tuning Didn't Yield Large Gains Despite our optimizations, the fine-tuned Qwen2.5-VL-3B model showed minimal or even negative changes in some metrics compared to the baseline. Possible explanations include:

- **Insufficient Training Steps:** With only 4,800 steps (under one full epoch), the model may not have had enough opportunity to adjust its lora weights meaningfully, especially given its 3.75 B parameters.
- **Quantization Noise:** The 4-bit quantization required to fit within GPU memory introduces approximation error that can degrade the effectiveness of fine-tuning updates.
- **Baseline Precision Advantage:** The baseline inference ran in 16-bit precision, whereas fine-tuning had to use 4-bit—this extra reduction in precision likely added noise and hindered improvements.
- **Baseline Strength:** The base model's pre-trained multimodal alignment may already be well-tuned for ABO-style VQA, leaving little headroom for further improvement without larger or more diverse fine-tuning data.

8.5 google/paligemma-3b-mix-224(2.92B)

8.5.1 Parameters Used

- per_device_batch_size = 1
- epochs = 1
- GPU : 2*T4

8.5.2 Optimizations Used

- **16 bit quantization:** The model was loaded in FP16 precision to fit within Kaggle's GPU VRAM limits

8.5.3 Problems Encountered

- **GPU utilization:** Due to a BitsAndBytes multi-GPU placement bug, we would have been limited to a single P100 GPU had we used 4 bit quantization.This is an Open issue extensively discussed on Github Forums(see [4]).

Since we were able to fit this model in half precision with batch size 1 on 2 gpus we didn't do ahead with 4 bit quantization.

8.5.4 Iterative Improvements

- Learning from Qwen 2 and 2.5 we trained the model with the prompt *'Answer in exactly one word'* in the first iteration itself.
- Since this model internally automatically resizes images to 224*224 we didn't face any problems here.

8.5.5 Results:

The paligemma-3b-mix-224 finetuned model with one-word-answer completed inference in **1328.6 s**, or **221.7 ms** per image. Its performance metrics were:

- **Exact-Match Accuracy:** 0.78
- **BERTScore F1 (all predictions):** 0.91
- **One-Word Compliance:** 99.93%
- **BERTScore F1 (one-word only):** 0.91

8.5.6 Comparison With Baseline

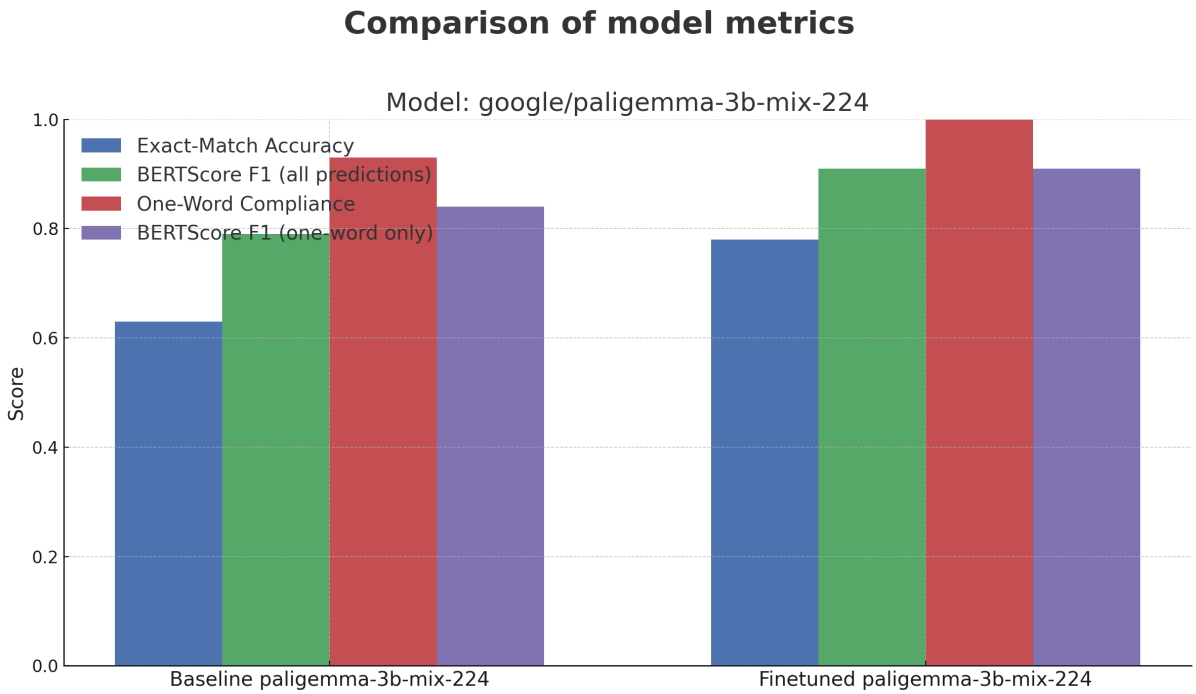

Figure 14: Paligemma model Baseline v/s finetuned performance.

Change in model metrics after Finetuning:

- **Exact-Match Accuracy:** + 0.15
- **BERTScore F1 (all predictions):** + 0.12
- **One-Word Compliance:** + 6.9 %
- **BERTScore F1 (one-word only):** + 0.07

1. The huge improvements in this model after finetuing can be attributed to the fact that the model was able to run for 1 full epoch and with batch_size 1, it was able to undergo close to 60,000 updates.

8.6 google/gemma-3-4b-it(4.3B) - Could Not Finetune

8.6.1 Optimizations Used

- **4-bit quantization:** We used BitsAndBytes to load the model in 4-bit precision on Kaggle, greatly lowering memory footprint.

8.6.2 Problems Encountered

- **GPU utilization:** Due to a BitsAndBytes multi-GPU placement bug, we were limited to a single P100 GPU while using 4 bit quantization.This is an Open issue extensively discussed on Github Forums(see [4]).
- **OOM:** Even with 4-bit quantization and a per-device batch size of 1, the model still exceeded the P100's VRAM capacity.Therefore we were unable to train it.

8.6.3 Other Trials

- **Unsloth:** We attempted to fine-tune this model with Unsloth but found no dedicated notebook and encountered numerous errors in existing examples; additionally, Unsloth's internal use of 4-bit BitsAndBytes restricts training to a single GPU as well.This is an Open issue on Github Forums(see [5]).

9 FINAL Results

The final comparison is done between the following models:

1. Salesforce/blip-vqa-base
2. Qwen/Qwen2-VL-2B-Instruct(Iteration 3)
3. Qwen/Qwen2.5-VL-2B-Instruct(Iteration 2)
4. google/paligemma-3b-mix-224

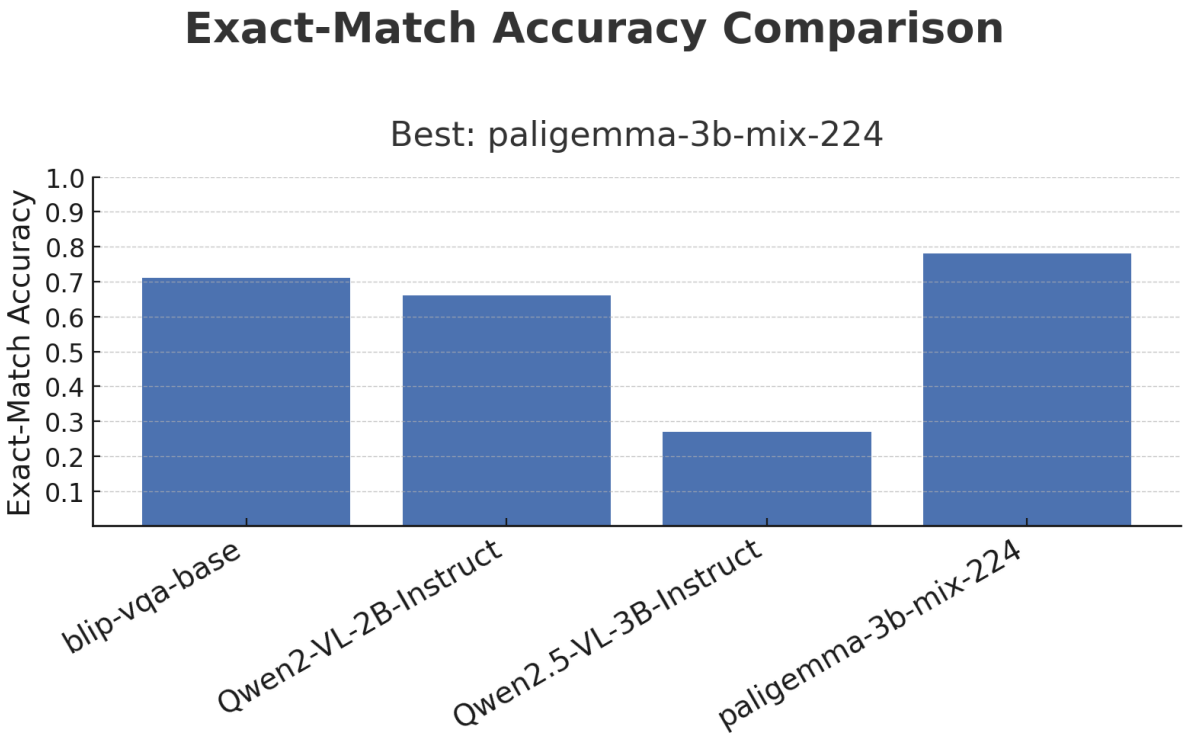

Figure 15: **Exact-Match Accuracy Comparison.**
Best: paligemma-3b-mix-224

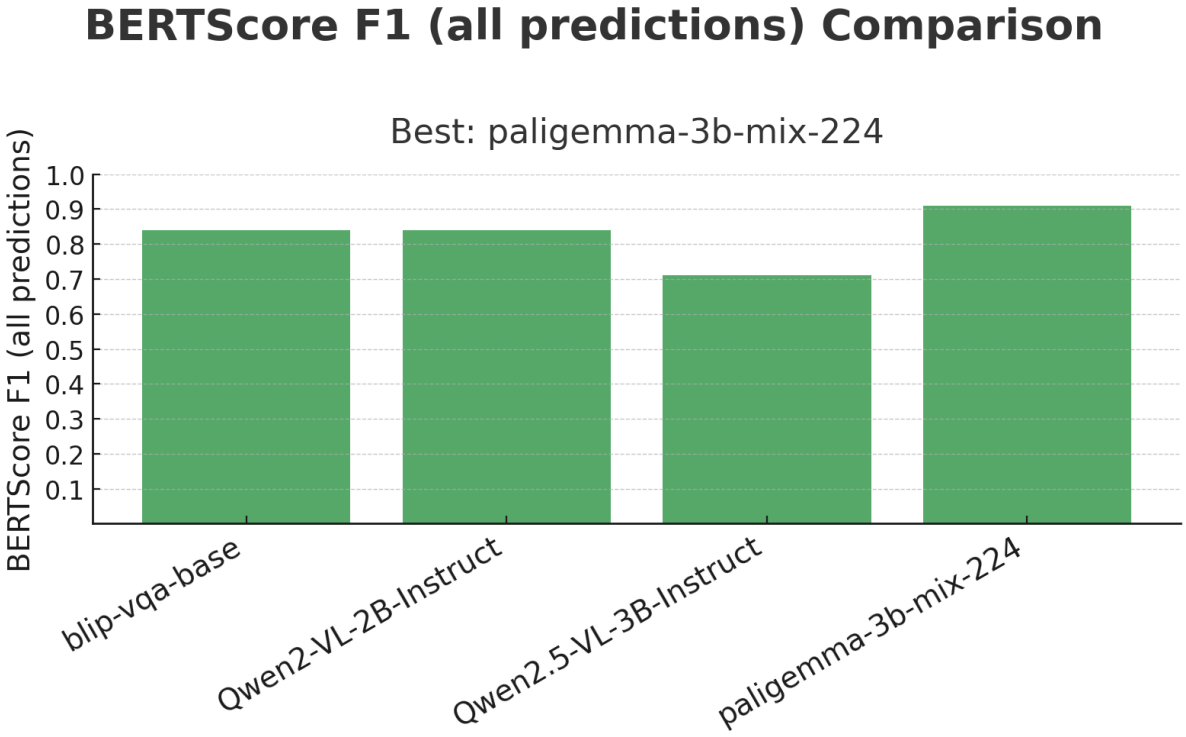

Figure 16: **BERTScore F1 (all predictions) Comparison.**
Best: paligemma-3b-mix-224

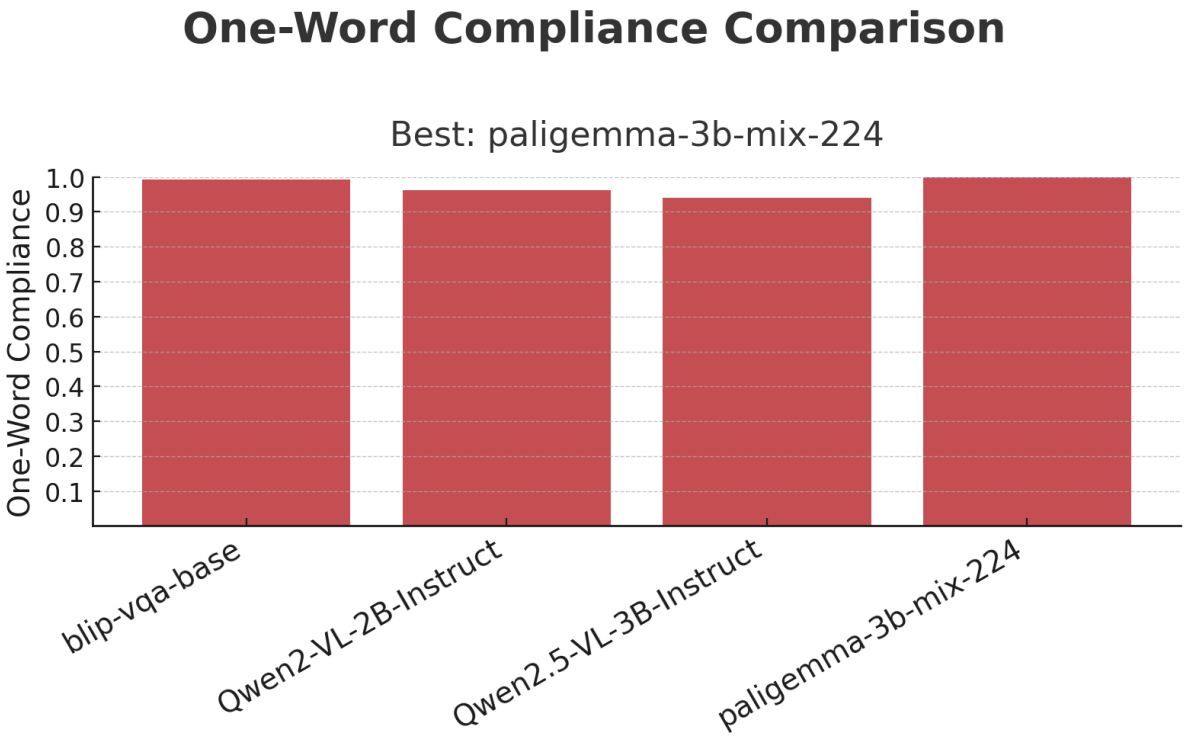

Figure 17: **One-Word Compliance Comparison.**
Best: paligemma-3b-mix-224

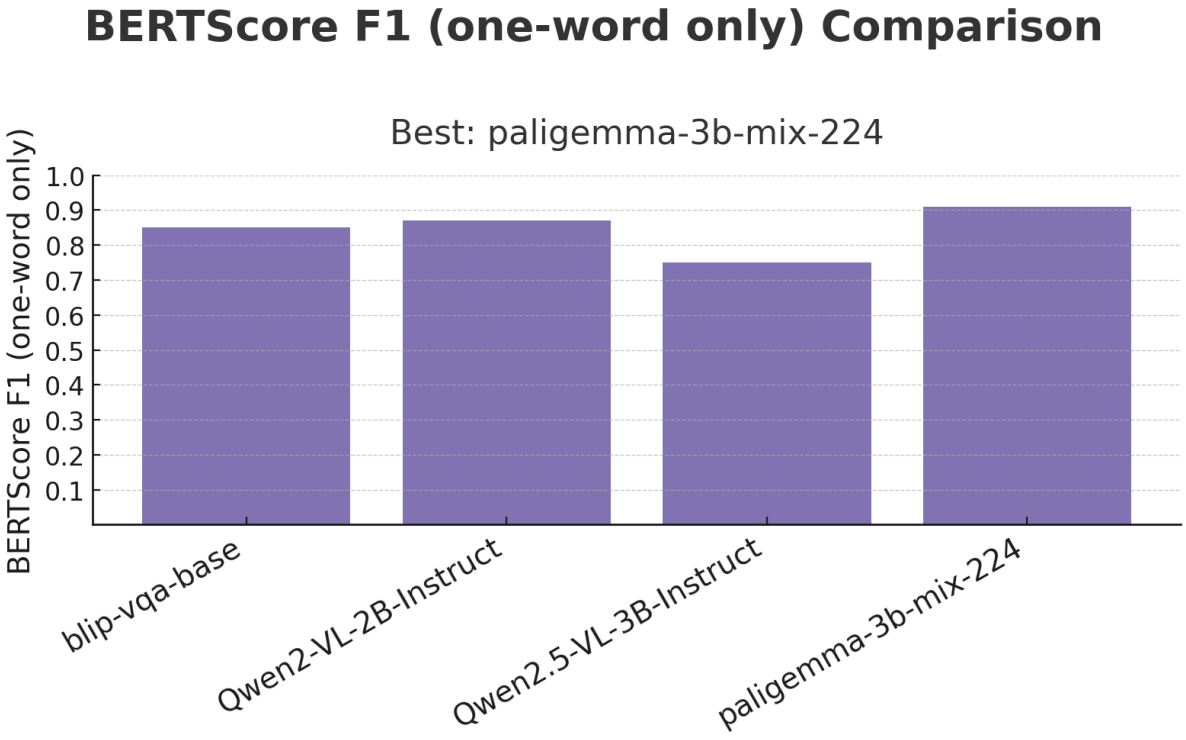

Figure 18: **BERTScore F1 (one-word only) Comparison.**
Best: paligemma-3b-mix-224

- As can be seen from the above graphs,*paligemma-3b-mix-224* performs the best

across all metrics. Hence this is the model that is finally submitted for evaluations.

- The lora adapter for this model was uploaded to Hugging Face and used in the final inference script.

10 Conclusion and Learnings

- **Off-the-shelf vs. fine-tuning:** Smaller, off-the-shelf VQA models deliver strong baseline performance but offer limited headroom for further improvement through fine-tuning.
- **Prompt importance:** Careful prompt design is critical—mis-specified or misleading or incomplete prompts can drastically undermine both training and inference quality.
- **Resolution trade-offs:** Although the Qwen family natively handles multi-resolution inputs, its slower inference makes it challenging to run enough updates under Kaggle's time constraints.
- **Pre-resize for speed:** Resizing images before feeding them to Qwen significantly reduces per-step computation time, enabling more training steps within the same time budget and thereby boosting performance.
- **Gemma's speed advantage:** Gemma models resize internally (risking some distortion) but achieve very fast inference, allowing many more training or evaluation steps within limited-time environments.
- **Quantization challenges:** 4-bit quantization helps fit large models into VRAM, but BitsAndBytes' multi-GPU issues(see [4]) and Kaggle's 12-hour limit make fine-tuning difficult. In our Qwen2.5 runs, the combination of aggressive quantization and few update steps led to underfitting, and even Qwen2 could likely have improved with more iterations.

Deliverables

- Git repository with code and notebooks.
- Curated dataset CSV.
- Inference script.
- Full report PDF.
- Readme

References

[1] TechAI Flow. *Prompt Engineering for AI Workflows: Google TechAI Flow Whitepaper*, 2025. [Online]. Available: `https://www.gptaiflow.tech/assets/files/2025-01-18-pdf-1-TechAI-Goolge-whitepaper_Prompt%20Engineering_v4-af36dcc7a49bb7269a58b1c9b89a8ae1.pdf`

[2] Dataloop AI, "Dandelin/ViLT-B32-Fine-tuned-VQA-Limitations," Dataloop AI Library, 2025. [Online]. Available: `https://dataloop.ai/library/model/dandelin_vilt-b32-finetuned-vqa/#limitations`.

[3] Hugging Face Transformers Maintainers, "Issue #33666: Runtime error when training Qwen2-VL on multiple GPUs with variable image resolutions," GitHub, 2024. [Online]. Available: https://github.com/huggingface/transformers/issues/33666.

[4] rlanday, "TypeError: Input tensors need to be on the same GPU, but found the following tensor and device combinations," Issue #1179, bitsandbytes-foundation/bitsandbytes, GitHub, Apr. 12, 2024. Available: https://github.com/bitsandbytes-foundation/bitsandbytes/issues/1179.

[5] brand17, "Issue #450: Use of the second GPU," unslothai/unsloth, GitHub, May 11, 2024. Available: https://github.com/unslothai/unsloth/issues/450.