# Visual Question Answering on VQAv2 dataset using VIT family and BERT family of models– VR Major Project Report

**Samarpita Bhaumik**
MT2023053

**Sunnidhya Roy**
MT2023079

## INTRODUCTION

Visual Question Answering (VQA) is a challenging task that requires models to comprehend both images and text to answer questions accurately. In this study, we explore the performance of models from the Vision Transformer (VIT) family and the Bidirectional Encoder Representations from Transformers (BERT) family on the VQAv2 dataset. The VIT models, known for their ability to process images directly, and the BERT models, renowned for their text understanding capabilities, are compared in terms of their effectiveness in handling the multimodal nature of VQA tasks. Through extensive experimentation and evaluation on the VQAv2 dataset, we analyze the strengths and weaknesses of both model families in addressing the complexities of visual question answering. The findings of this study provide valuable insights into the suitability of VIT and BERT models for VQA tasks, shedding light on their respective contributions to advancing the field of multimodal AI research.

LORA (Low Rank Adaptation) is also used on top of the models to fine-tune the given task, it helps to reduce the number of trainable parameters thus improving training time while maintaining similar level of accuracy.

## 1. OUR DATASET:

```
Generating train split: 443757/0 [01:49<00:00, 4890.30 examples/s]

Generating validation split: 214354/0 [00:49<00:00, 4864.92 examples/s]

Generating testdev split: 107394/0 [00:15<00:00, 7394.69 examples/s]

Generating test split: 447793/0 [01:02<00:00, 5676.88 examples/s]

Out[2]:
[Dataset({
    features: ['question_type', 'multiple_choice_answer', 'answers', 'image_id', 'answer_type', 'question_id', 'question', 'image'],
    num_rows: 110939
}),
 Dataset({
    features: ['question_type', 'multiple_choice_answer', 'answers', 'image_id', 'answer_type', 'question_id', 'question', 'image'],
    num_rows: 53588
})]
```

**Fig 1. Dataset (Train and Validation)**

```
In [3]:
# Access the train split
train_dataset = dataset[0]

# Print the first row
print(train_dataset[0])

{'question_type': 'what is this', 'multiple_choice_answer': 'net', 'answers': [{'answer': 'net', 'answer_confidence': 'maybe', 'answer_id': 1}, {'answer': 'net', 'answer_confidence': 'yes', 'answer_id': 2}, {'answer': 'net', 'answer_confidence': 'yes', 'answer_id': 3}, {'answer': 'netting', 'answer_confidence': 'yes', 'answer_id': 4}, {'answer': 'net', 'answer_confidence': 'yes', 'answer_id': 5}, {'answer': 'net', 'answer_confidence': 'yes', 'answer_id': 6}, {'answer': 'mesh', 'answer_confidence': 'maybe', 'answer_id': 7}, {'answer': 'net', 'answer_confidence': 'yes', 'answer_id': 8}, {'answer': 'net', 'answer_confidence': 'yes', 'answer_id': 9}, {'answer': 'net', 'answer_confidence': 'yes', 'answer_id': 10}], 'image_id': 458752, 'answer_type': 'other', 'question_id': 458752000, 'question': 'What is this photo taken looking through?', 'image': <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=640x480 at 0x7B14C3C3A7D0>}
```

**Fig 2. Dataset content**

```
In [8]:
print(config.id2label)

{0: 'net', 1: 'pitcher', 2: 'orange', 3: 'yes', 4: 'white', 5: 'skiing', 6: 'red', 7: 'frisbee', 8: 'brushing teeth', 9: 'no', 10: 'black and white', 11: 'skateboard', 12: '1', 13: 'blue', 14: 'green', 15: 'motorcycle', 16: 'gray', 17: '2', 18: 'purse', 19: 'skis', 20: 'poles', 21: 'surfboard', 22: 'dog', 23: 'on', 24: 'office', 25: 'large', 26: 'very big', 27: 'laptop', 28: 'vent', 29: 'computer', 30: 'black', 31: 'bear', 32: '3', 33: 'wii', 34: 'glasses', 35: 'tree', 36: 'eating', 37: 'log', 38: '5', 39: 'raft', 40: 'left', 41: 'living room', 42: 'pink', 43: 'right', 44: 'railing', 45: 'grass', 46: 'wire', 47: '10 years', 48: 'knife', 49: 'cake', 50: 'banana', 51: 'chef', 52: 'vanilla', 53: '4', 54: 'outdoor', 55: 'mustard', 56: 'bun', 57: 'clouds', 58: 'dock', 59: 'brown', 60: 'silver', 61: 'refrigerator', 62: 'square', 63: 'teddy', 64: 'elm', 65: 'stripes', 66: 'baseball', 67: 'catcher', 68: 'beer', 69: 'bottom', 70: 'north', 71: 'nike', 72: 'yellow and white', 73: 'morning', 74: 'elephant', 75: 'red and white', 76: 'propeller', 77: 'tan', 78: 'wall', 79: 'rolex', 80: 'clock', 81: 'table', 82: '0', 83: 'wood', 84: 'christmas', 85: 'spinach', 86: 'thick', 87: 'bag', 88: 'leaves', 89: 'necklace', 90: '6', 91: 'bathroom', 92: 'shower', 93: 'towel', 94: 'solid', 95: 'referee', 96: 'wilson', 97: '8:00', 98: 'e', 99: '24', 100: 'hat', 101: 'grazing', 102: 'sheep', 103: '10', 104: 'tag', 105: 'spanish', 106: 'hot dog', 107: 'plate', 108: 'lunch', 109: 'butter', 110: 'peppers', 111: 'onions', 112: 'very', 113: 'mayonnaise', 114: 'mayo', 115: 'sweet potato', 116: 'pig', 117: 'sweet', 118: 'flowers', 119: 'floral', 120: 'yellow', 121: 'window', 122: '7', 123: 'pizza', 124: 'car', 125: 'None', 126: 'cargo', 127: 'stairs', 128: 'abstract', 129: 'rug', 130: 'baseball cap', 131: 'texting', 132: 'pole', 133: 'crosswalk', 134: 'nothing', 135: 'urban', 136: 'bus', 137: 'light',
```

**Fig 3. Id2label (mapping every string label value to integer id value)**

## 2. STRUCTURE OF OUR ALGORITHM COMMON TO ALL THE ARCHITECTURES.

### 2.1 Without using LORA

**[1]** Initially we import the **load_dataset function** from the datasets library to load a specific dataset from the **HuggingFace** repository. It loads 25% of both the training and validation sets of the VQAv2 dataset, specified by the split parameter. The commented-out line shows how to load the entire dataset without any splits. Finally, the dataset variable contains the loaded portions, and the final line outputs its contents.

**[2]** Then we import the **ViltConfig** class from the transformers library and uses it to load a pre-trained configuration for the ViLT (Vision-and-Language Transformer) model. Specifically, it loads the configuration for the **"dandelin/vilt-b32-finetuned-vqa"** model, which is a version of ViLT fine-tuned for the Visual Question Answering (VQA) task. The loaded configuration is stored in the config variable. The **id2label** attribute is a dictionary mapping numerical class IDs to their corresponding label names. This is often used in classification tasks to interpret the model's output by converting numerical predictions back into human-readable labels.

**[3]** We then import the **tqdm progress bar** utility and define two functions: **get_score** and add_**labels_scores**. The **get_score** function calculates a score based on a given count, capping the score at 1.0. The **add_labels_scores** function processes an annotation dictionary by counting the occurrences of each answer, mapping valid answers to their corresponding label IDs using **config.label2id**, and calculating scores for each label using the get_score function. It then updates the annotation with these labels and scores before returning it.

**[4]** We then define a **custom PyTorch dataset class VQADataset for the Visual Question Answering (VQA) v2 dataset**. The class initializes with questions, annotations, a preprocessor, and a tokenizer. It implements the **__len__** method to return the number of annotations and the **__getitem__** method to retrieve items by index. The **__getitem__** method processes an image and question pair by converting the image to RGB and using the preprocessor to encode it. It also tokenizes the question text and combines these encodings. Labels and scores from the annotations are converted into target tensors, which are integrated into the encoding dictionary. The method returns this dictionary containing input IDs, token type IDs, attention masks, and label targets for use in training models.

**[5]** We then randomly select **15,000 indices** from a training subset (**subset_train**) and creates a subset containing only the questions and their corresponding annotations (with labels and scores added, images, answers etc.) using the sampled indices. **(This is done as we are not able to run with more than 15000 data as Kaggle RAM is going out of memory).**

**[6]** We then initialize a tokenizer and a feature extractor using pre-trained models for processing text and images, respectively.

**[7]** Using the custom VQADataset we then form 3 datasets for train, validation and test.

**[8]** We then define a collate function **collate_fn** to preprocess batches of data for training and validation. It extracts input IDs, pixel values, attention masks, token type IDs, and labels from each item in the batch and stacks them into tensors. The commented-out section indicates a potential operation for creating padded pixel values and pixel masks, which is currently not in use. The function then constructs a new batch dictionary containing the processed tensors for pixel values, input IDs, token type IDs, attention masks, and labels. Two data loaders (**train_dataloader** and **val_dataloader**) are created using this collate function for the VQA2 dataset and **its validation subset, specifying batch sizes of 4 and employing 4 worker processes for data loading.**

**[9]** We then define a custom PyTorch module **MultimodalVQAModel** for a multimodal question answering model. It initializes with parameters such as the number of labels, intermediate dimension size, and names of pre-trained models for text and image processing. The model consists of text and image encoders loaded from pre-trained models, followed by a fusion layer that combines their outputs. **The fusion layer is a linear transformation followed by a ReLU activation function and dropout**. Subsequently, a linear layer is used for classification. The forward method takes input pixel values, input IDs, token type IDs, attention masks, and labels. It processes text and image inputs separately through their respective encoders and fuses their representations. The fused output is passed through the classification layer to generate logits. If labels are provided (in case of validation and train), the method calculates the cross-entropy loss and returns both the logits and the loss.

**[10]** We then define a custom function **set_seed** to ensure reproducibility by setting the random seed for Python's random module, NumPy, and PyTorch. It checks if a CUDA-enabled GPU is available and, if so, sets the random seed for GPU operations as well. **The seed value used is 42, a commonly used value for reproducibility (This was done mainly because the dataset could not be downloaded due to its size and so to ensure consistence of data while testing this is done)**. Additionally, it configures PyTorch to ensure deterministic behavior on GPU by disabling cuDNN's nondeterministic algorithms and enabling deterministic mode. Finally, it determines the device to be used for computation, either CPU or GPU, and prints the selected device. **This setup aims to provide deterministic behavior across different runs of the code, facilitating reproducibility of results.**

**[11]** We then define a **LightningMultimodalVQAModel** class using PyTorch Lightning, which wraps a multimodal VQA model to facilitate training, validation, and testing. The forward method passes inputs through the model. The **training_step**, **validation_step**, and **test_step** methods perform the respective phases, logging loss, accuracy, and F1 score at each step and epoch. These metrics are computed by comparing predicted and actual labels. The **configure_optimizers** method sets up the **AdamW optimizer with a learning rate of 5e-5 for training the model.** This setup leverages PyTorch Lightning's streamlined interface to manage training loops and logging, enhancing code readability and maintainability.

**[12]** We then initialize a multimodal VQA model and wraps it in a LightningMultimodalVQAModel class for use with PyTorch Lightning. It sets up a model checkpoint callback to save the best model based on validation accuracy. A PyTorch Lightning trainer is then initialized to run for a maximum of n epochs with the checkpoint callback. Finally, the trainer fits the model using the training and validation data loaders, handling the training loop and checkpointing automatically. **No. of trainable parameters: 214,898,489.**

**[13]** Post this step we perform validation and inference using the trained model where we provide an image and question as input and the model gives us an array of predicted answers.

This is the general architecture of our pipeline. We have used many different combination of models and optimization.

## 2.2 WITH LORA

**Steps [1] to [10**] same as **section 2.1**

**[11]** We then configure and apply the **Low-Rank Adaptation (LoRA)** technique to a model using the **peft** library. It creates a **LoraConfig** with specified parameters: rank (r) of the low-rank matrices**(=16)**, scaling factor (**lora_alpha=16**), target modules (query and value layers), and dropout rate (**lora_dropout=0.1**). The **get_peft_model** function then applies this LoRA configuration to the existing model, enhancing its efficiency and adaptability by modifying only specific parts of the model.

**[12]** We then initialize a multimodal VQA model and wraps it in a LightningMultimodalVQAModel class for use with PyTorch Lightning where we use the peft model which we initialized in step [11]. It sets up a model checkpoint callback to save the best model based on validation accuracy. A PyTorch Lightning trainer is then initialized to run for a maximum of n epochs with the checkpoint callback. Finally, the trainer fits the model using the training and validation data loaders, handling the training loop and checkpointing automatically. **No. of trainable parameters: 1,179,648 which is 54% of the total no. of parameters.**

**[13]** Post this step we perform validation and inference using the trained model where we provide an image and question as input and the model gives us an array of predicted answers.

This is the general architecture of our pipeline using LORA. We have used many different combinations of models and optimization.

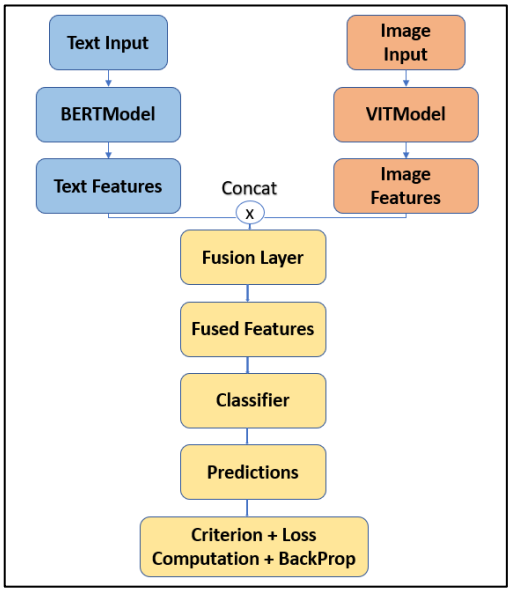
## 2.3 Model Selection

1) **BERT (Bidirectional Encoder Representations from Transformers)**
   - **a.** Contextual Understanding
   - **b.** State of the art performance, very heavy though.
   - **c.** Pretrained on Large corpus
2) **ALBERT (A lite BERT)**
   - **a.** Parameter Efficiency (useful when we have limited resources)
   - **b.** Scalability
   - **c.** Performance (Though light it performs as good as BERT)
3) **ROBERTa (Robustly optimized BERT)**
   - **a.** Training enhancement by training on larger corpus.
   - **b.** Dynamic masking which means masked token change at each epoch providing a wholesome training of the model.
   - **c.** Performance gains especially when trained on huge amount of data.
4) **VIT (Vision Transformer)**
   - **a.** Transformer Architecture hence improved contextual understanding.
   - **b.** Scalability.
   - **c.** Simpler Preprocessing.
   - **d.** State of the art performance
5) **BEIT (Bidirectional Encoder Representations from Image Transformers)**
   - **a.** Masked Modelling which helps in self-supervision by predicting missing patches of the same image given as input.
   - **b.** Enhanced Data efficiency due to masking as it can be trained on unlabeled data.
   - **c.** Transfer Learning by fine tuning to downstream tasks.

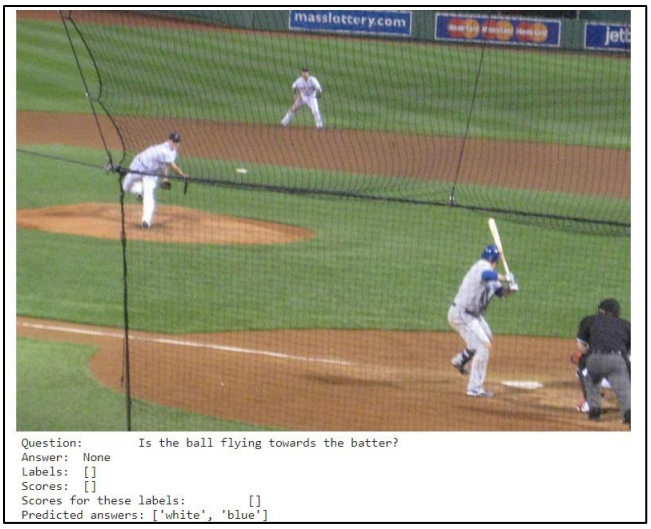
# 3. OUR PROPOSED SOLUTIONS:

## 3.1 Approach 1 – BERT + VIT + 15epochs + 15000 datapoints

**Model Architecture:**

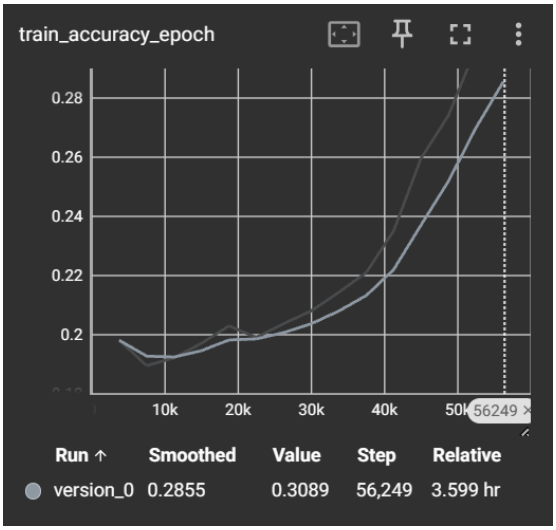



**Results:**

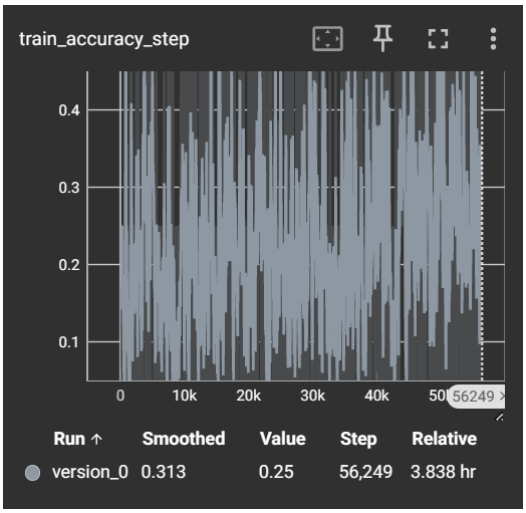




**Train Accuracy Epoch:**

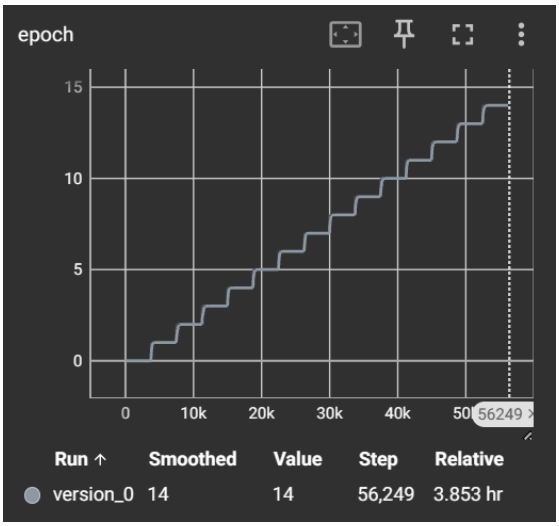



**Train Accuracy Step:**

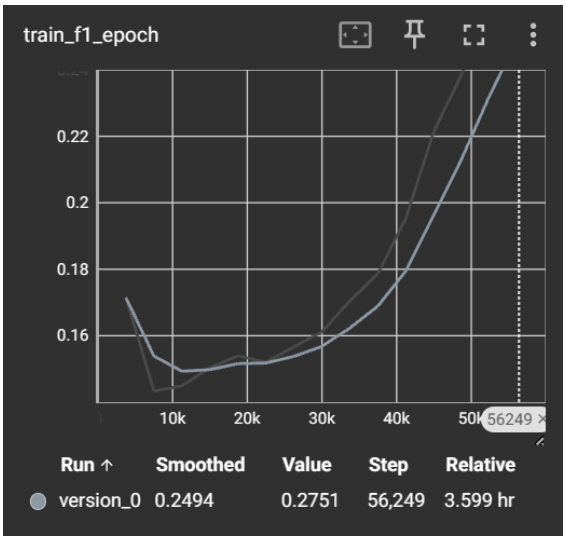



**Metrics:**

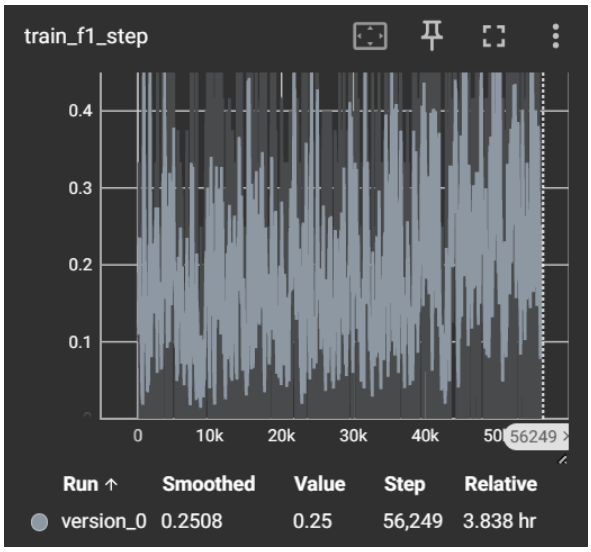
**Epoch:**



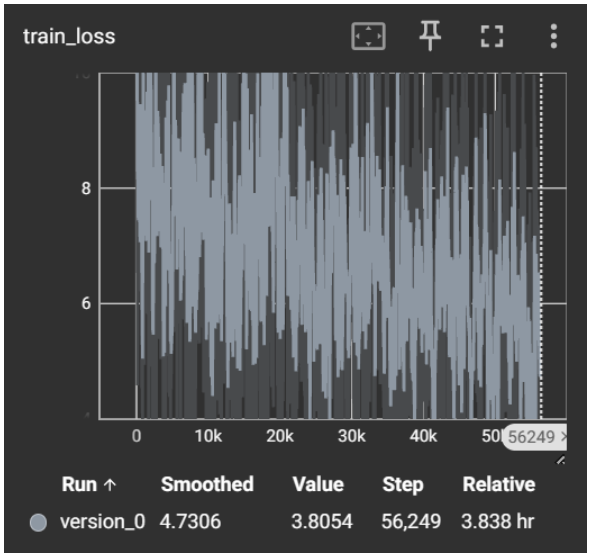

**Train F1 epoch:**



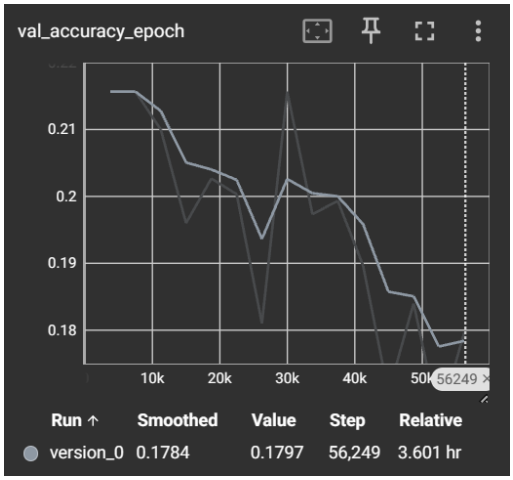
**Train F1 step:**



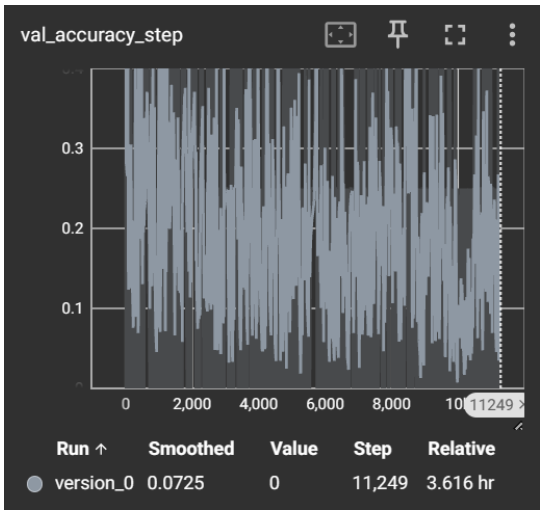

**Train Loss:**



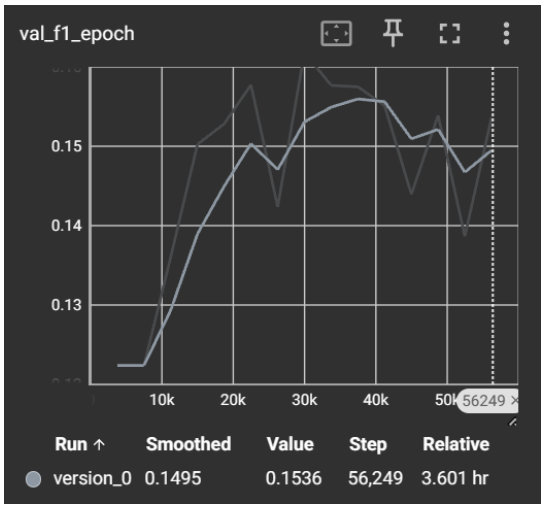

**Validation Accuracy epoch:**



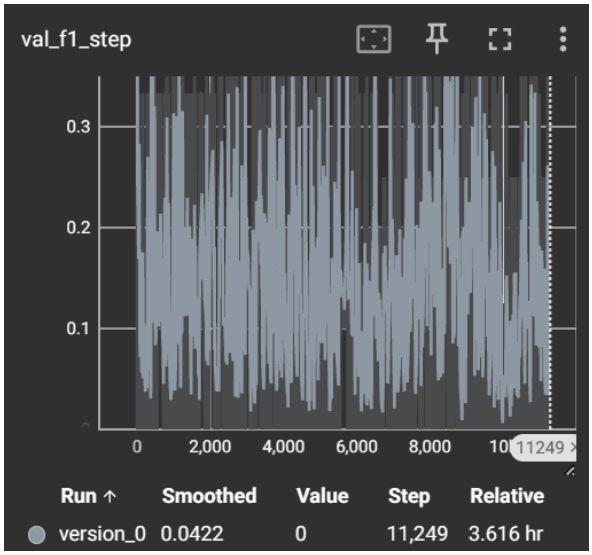

**Validation Accuracy Step:**



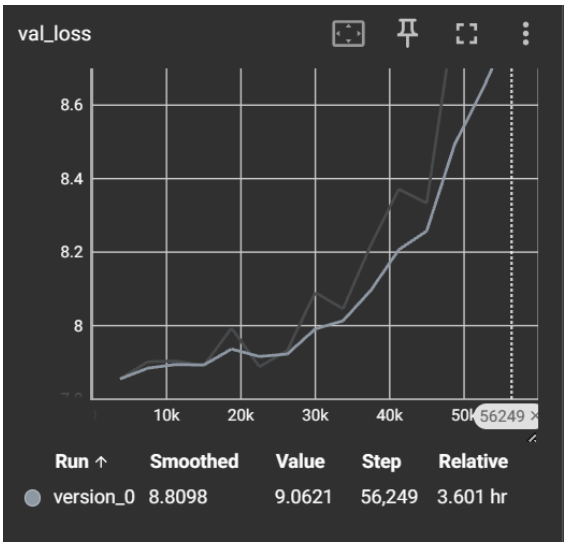

**Validation F1 epoch:**



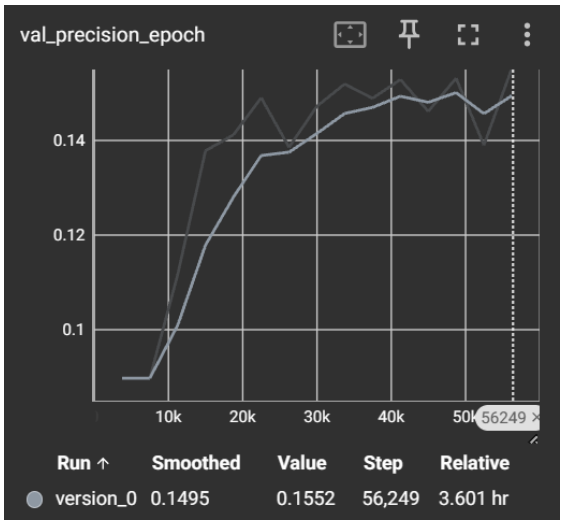

**Validation F1 step**



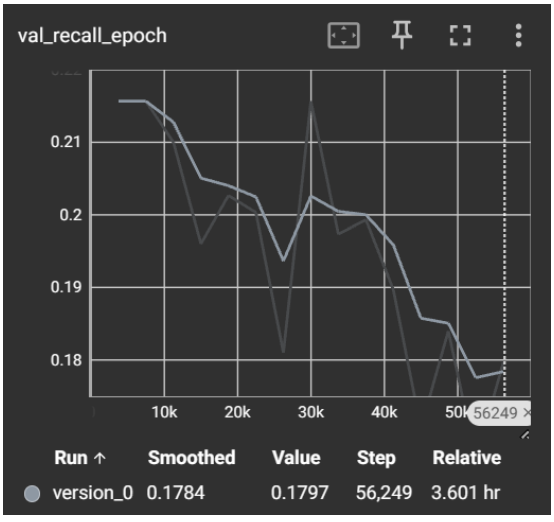
**Validation Loss:**




**Precision:**

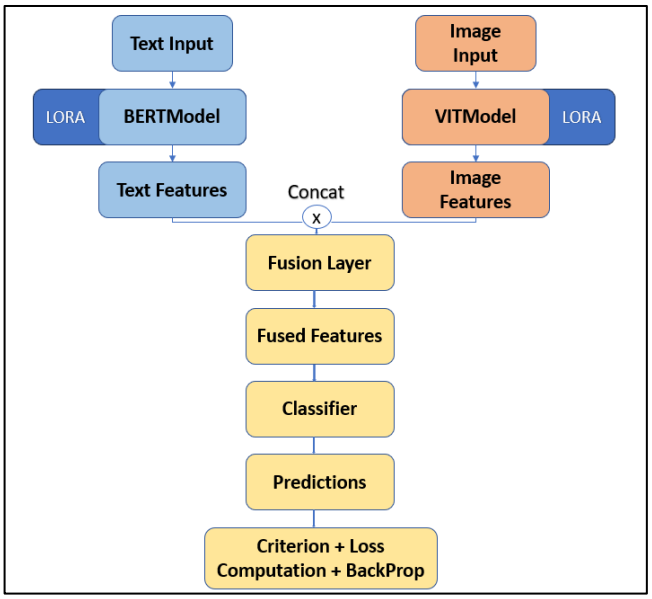



**Recall:**

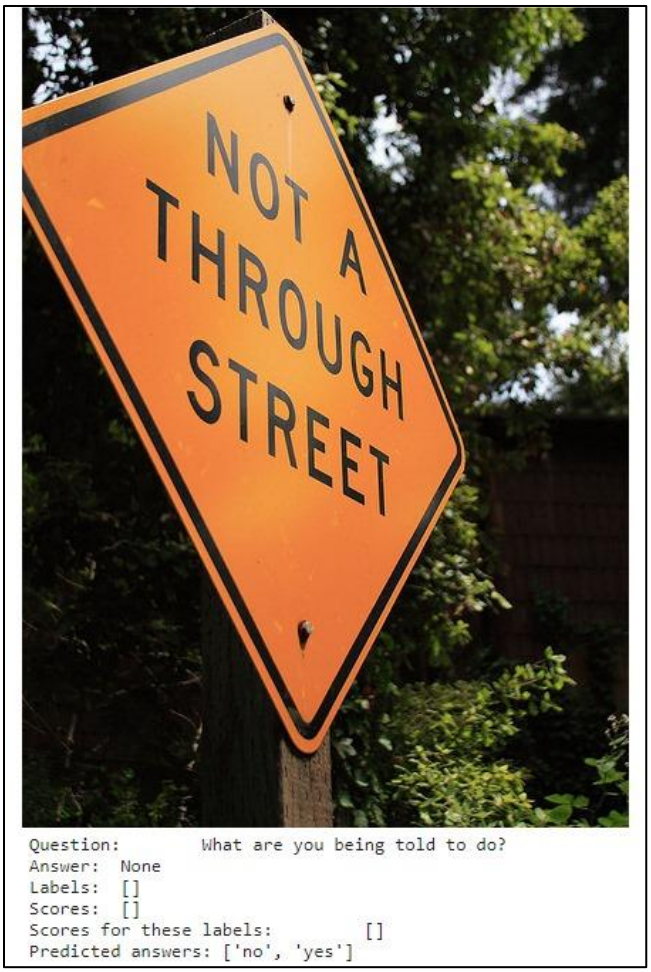



## 3.2 Approach 2 – BERT + VIT + LORA + 15epochs + 15000 datapoints

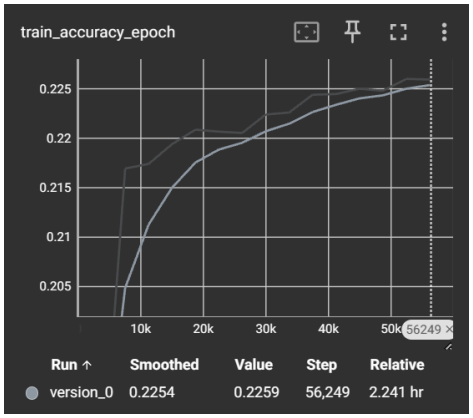
**Model Architecture:**



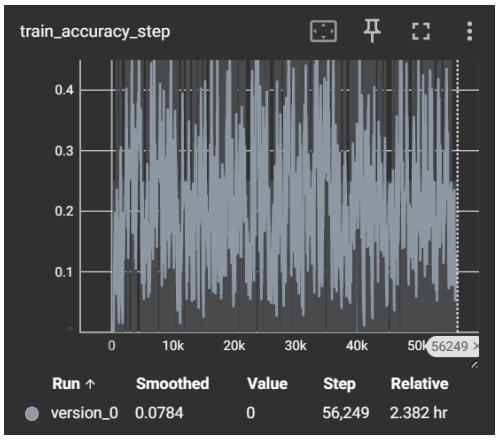

**Results:**


**Train Accuracy Epoch:**

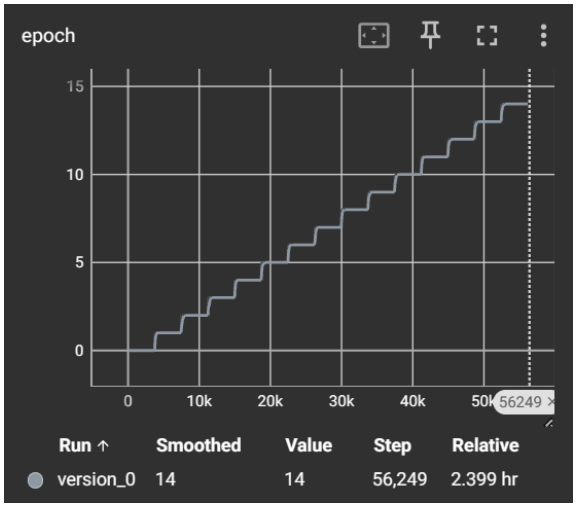



**Train Accuracy Step:**

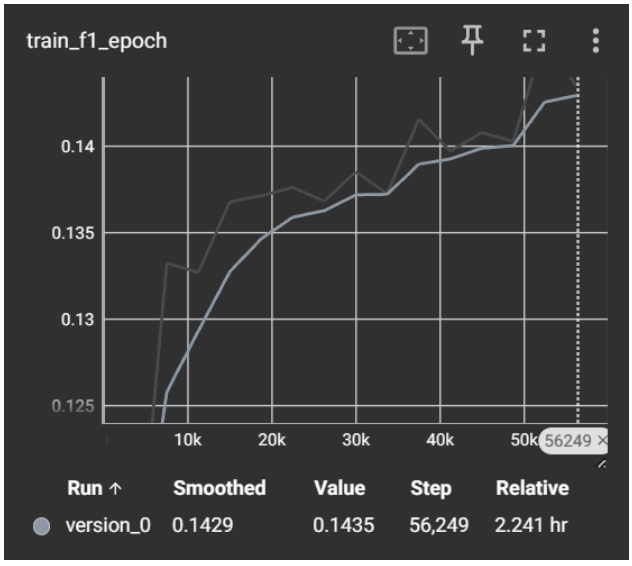



**Metrics:**

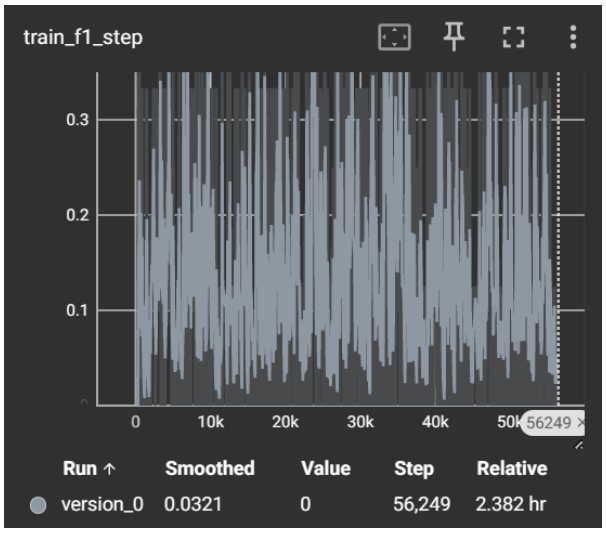
**Epoch:**



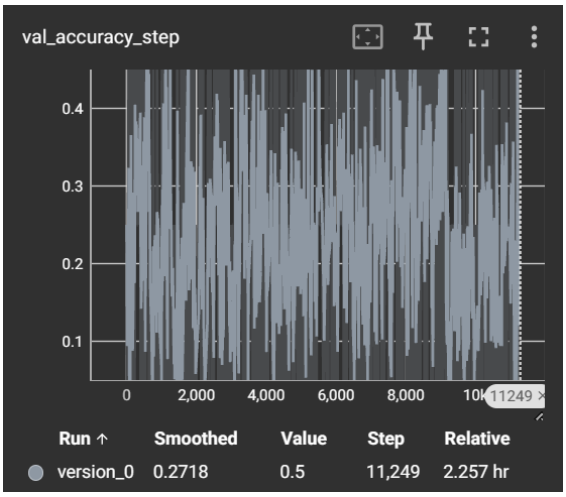

**Train F1 epoch:**



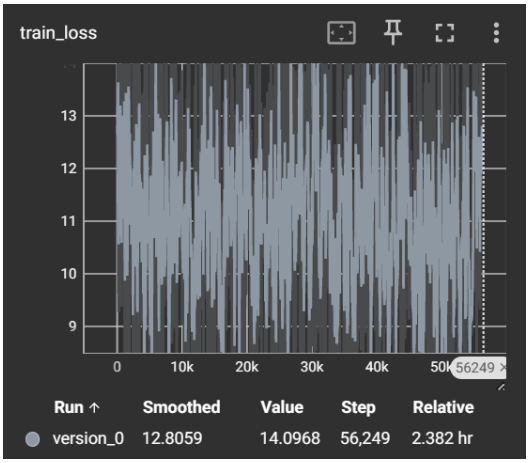
**Train F1 step:**



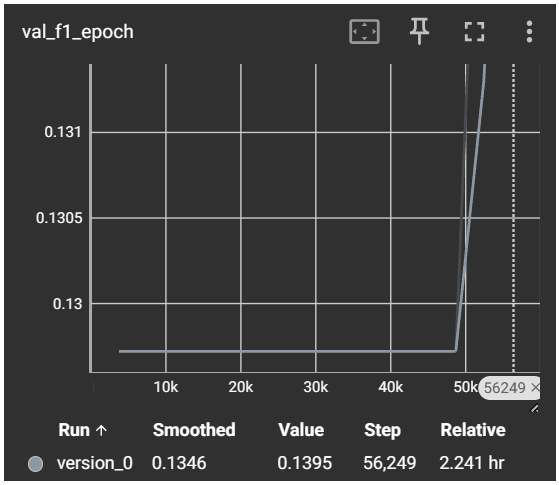

**Validation Accuracy Step:**



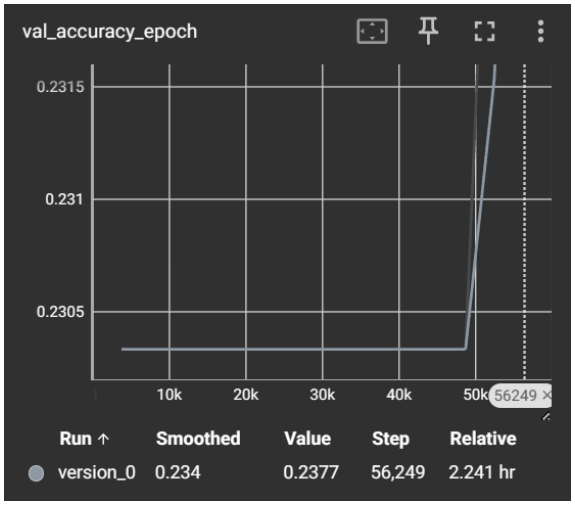

**Train Loss:**



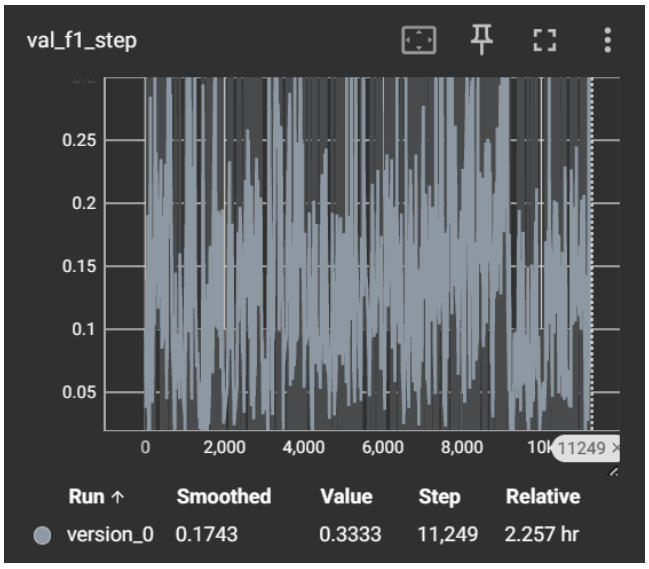

**Validation F1 epoch:**



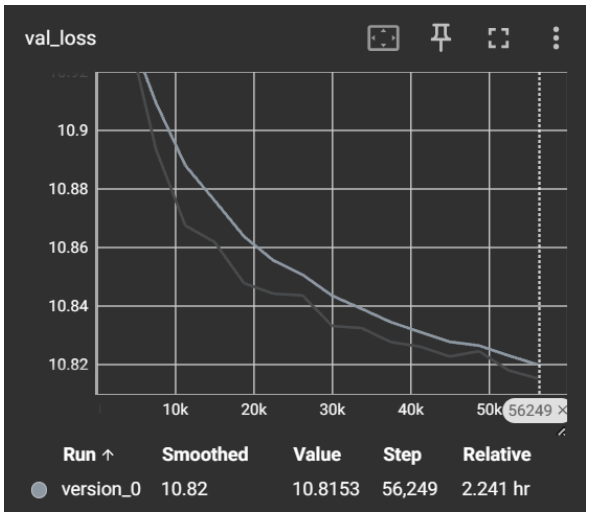

**Validation Accuracy:**



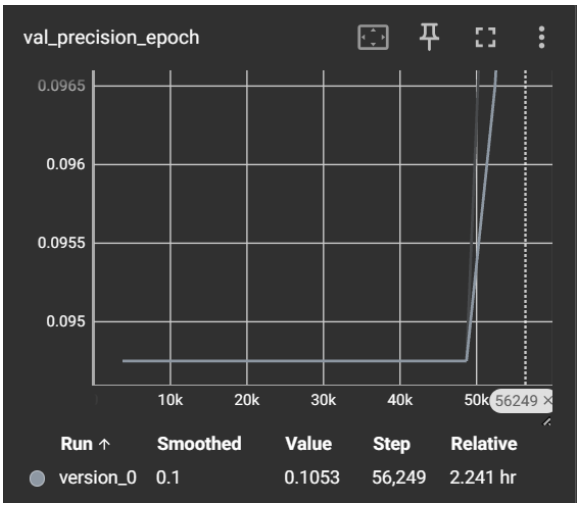

**Validation F1 step**



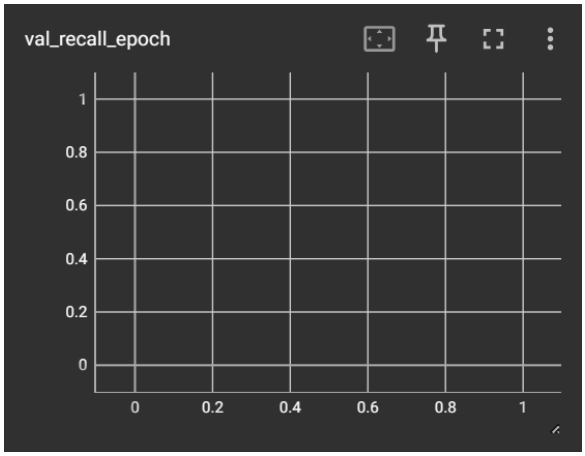
**Validation Loss:**




**Precision:**

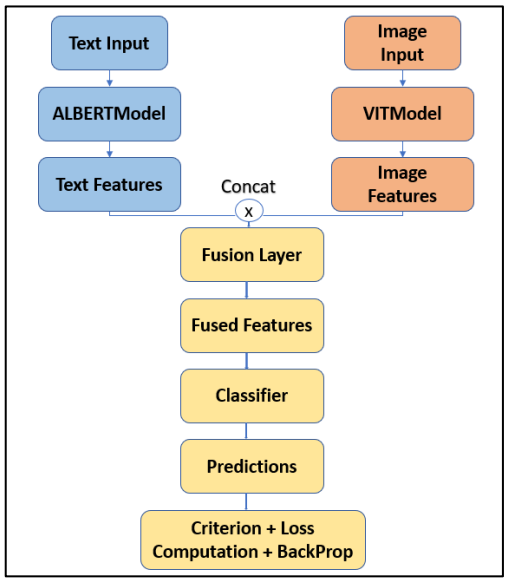



**Recall: (0)**

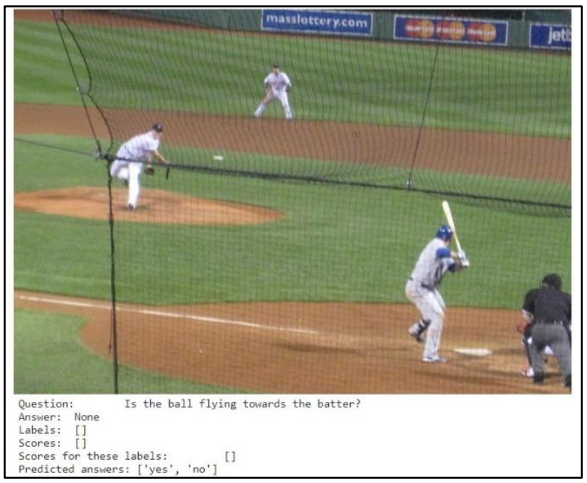



## 3.3 Approach 3 – ALBERT + VIT + 15epochs + 15000 datapoints

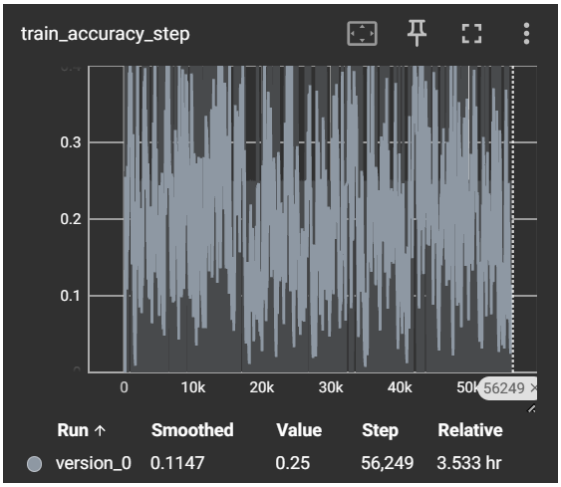
**Model Architecture:**




**Results**

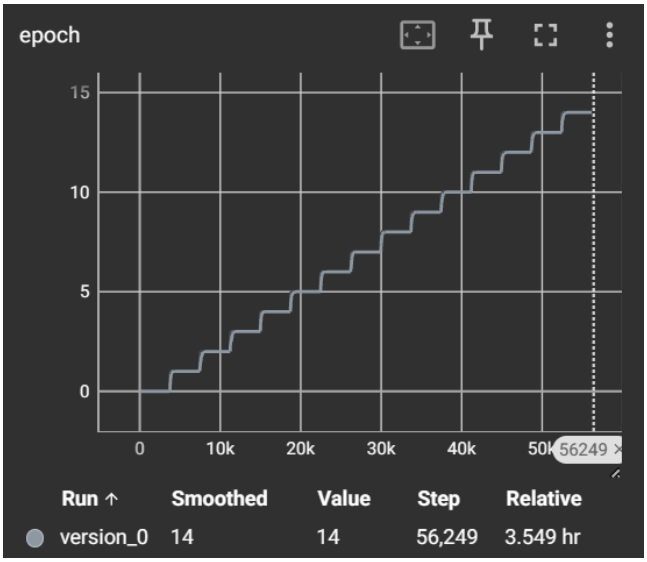




**Train Accuracy Step:**

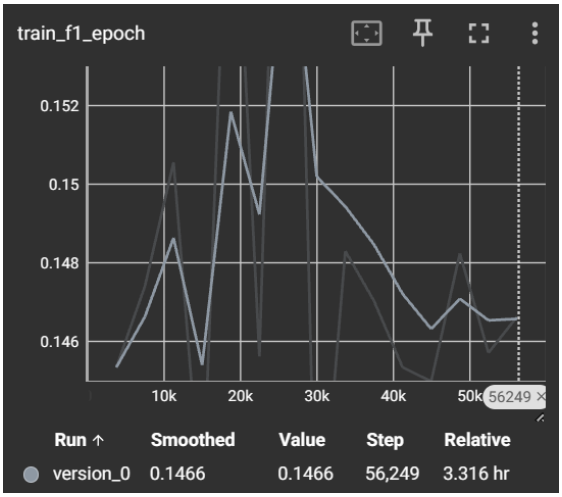



**Metrics:**

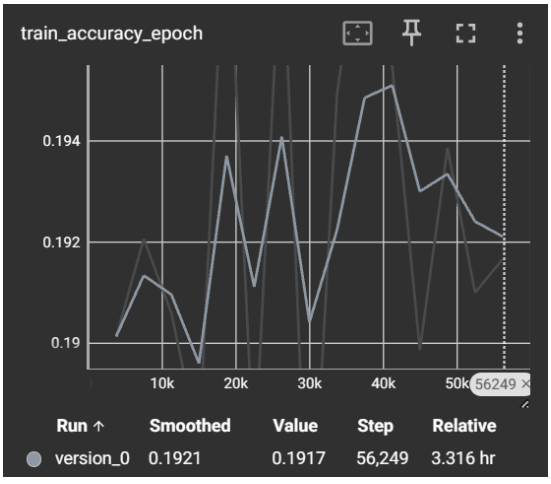
**Epoch:**



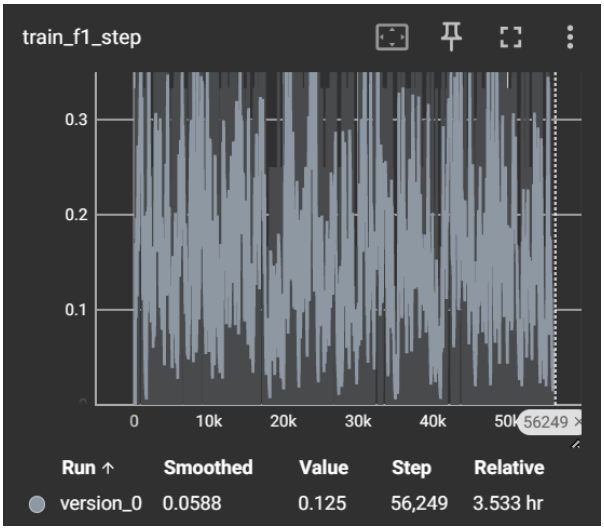

**Train F1 epoch:**



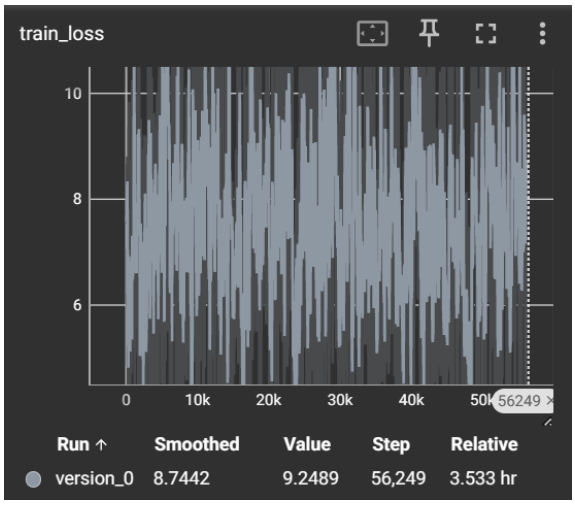

**Train Accuracy Epoch:**



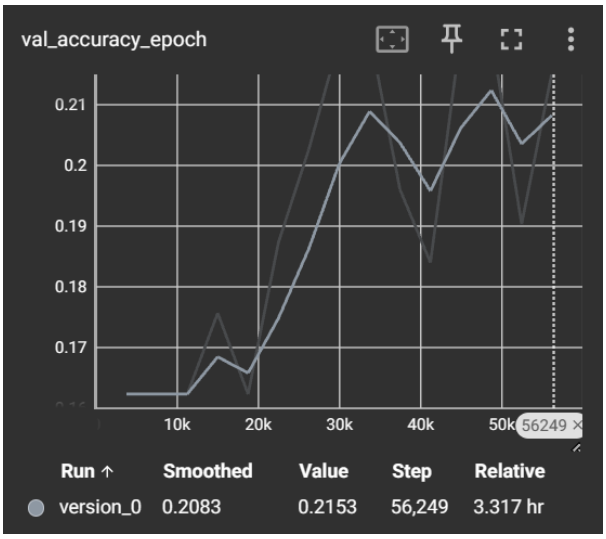

**Train F1 step:**



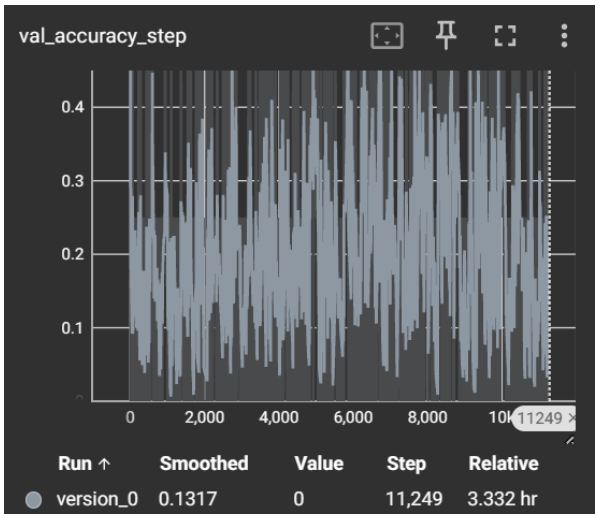
**Train Loss:**



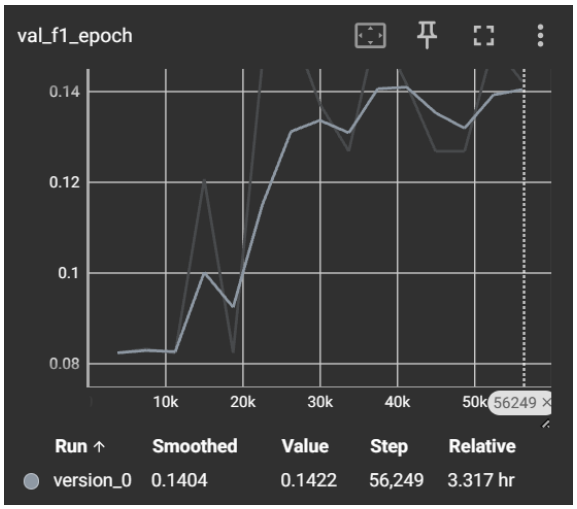

**Validation Accuracy:**



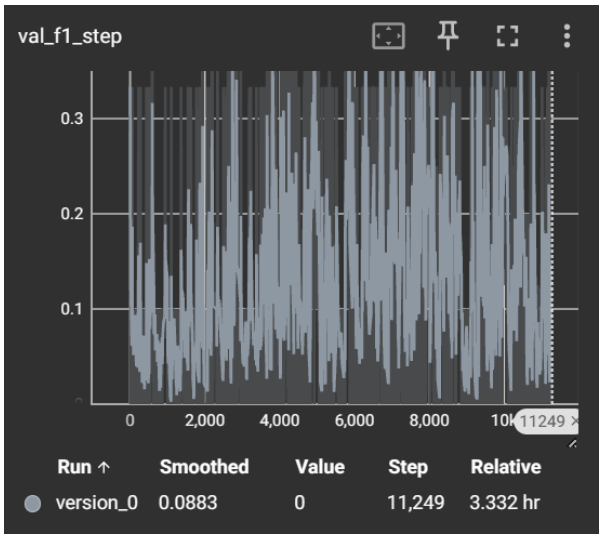

**Validation Accuracy Step:**




**Validation F1 epoch:**



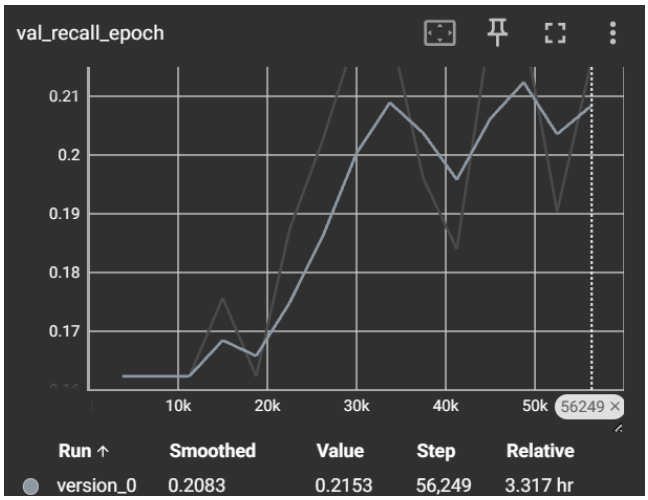

**Validation F1 step**




**Validation Loss:**

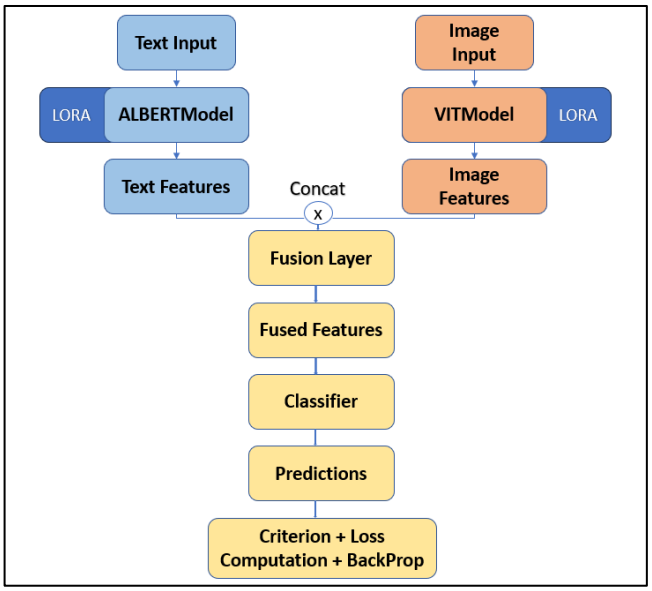
**Precision:**


**Recall:**

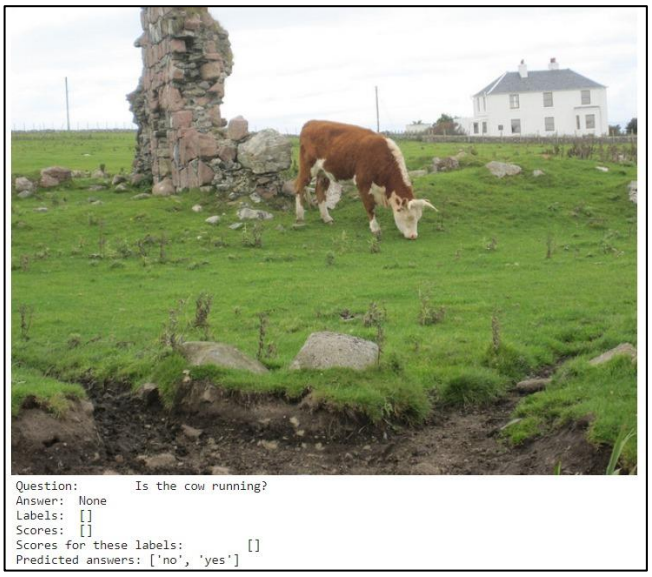


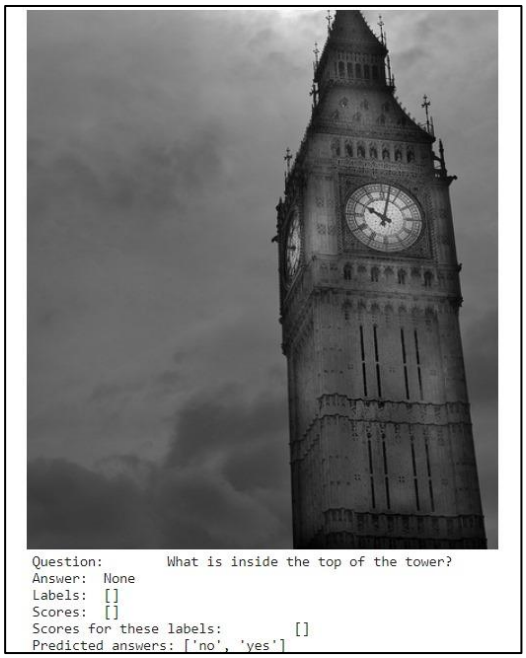

## 3.4 Approach 4 – ALBERT + VIT + LORA + 15epochs + 15000 datapoints

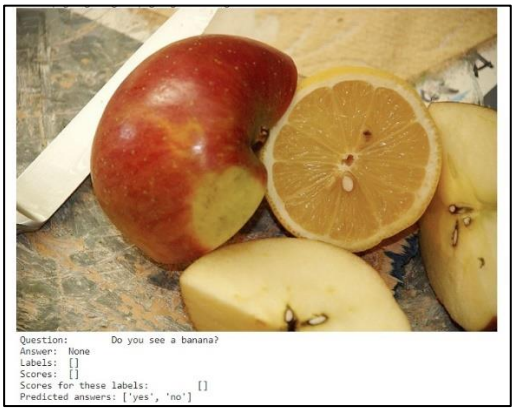
**Model Arc\hitecture:**



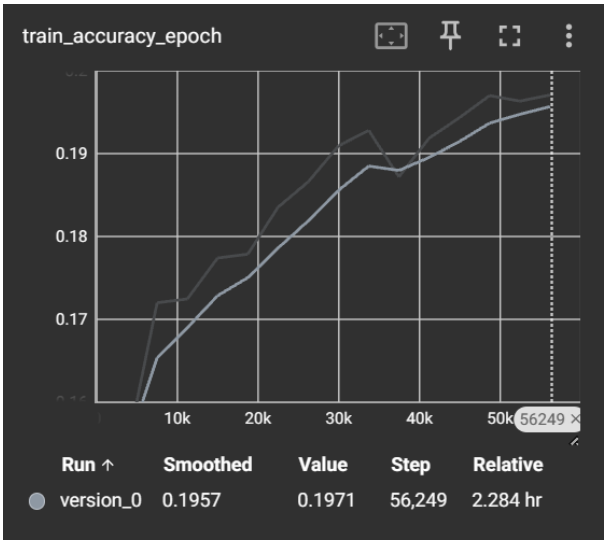

**Results**



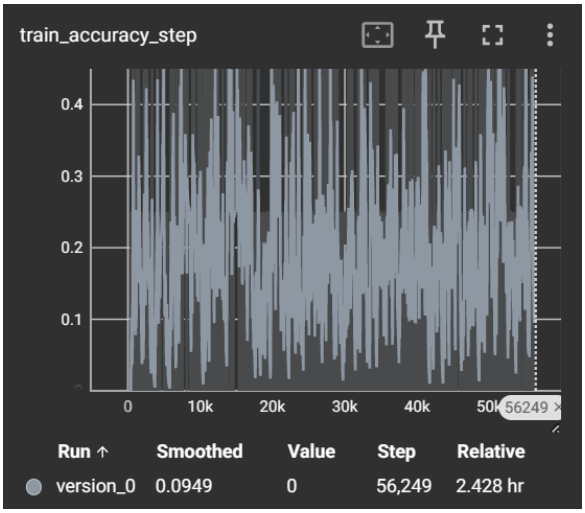



**Train Accuracy Epoch:**

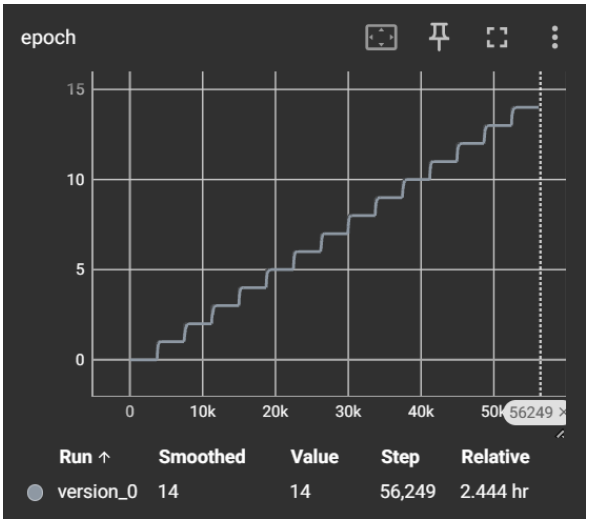



**Train Accuracy Step:**

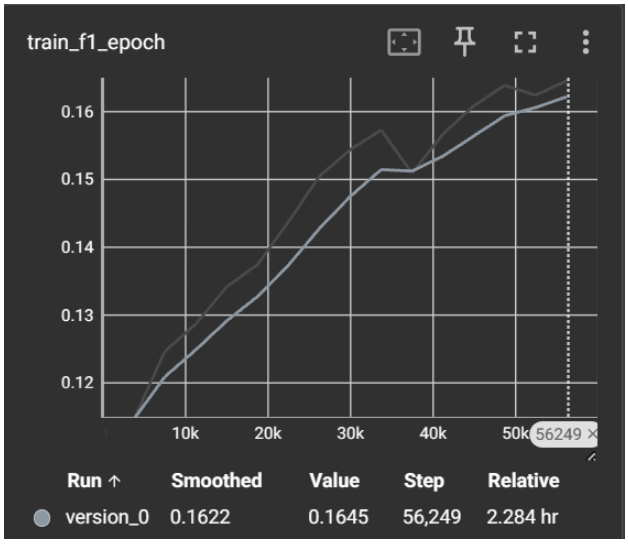



**<u>Metrics:</u>**

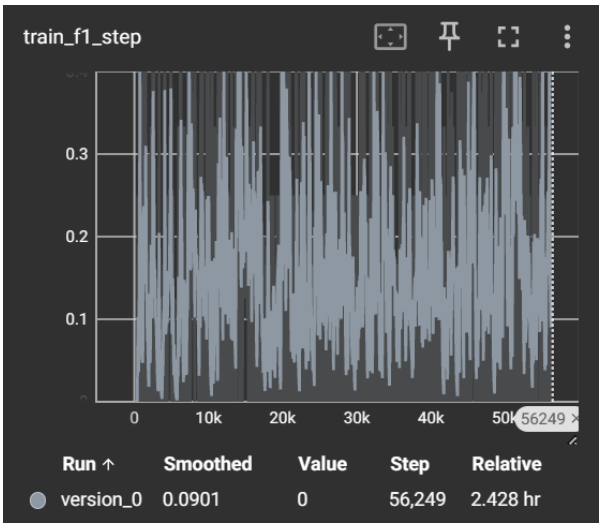
**Epoch:**



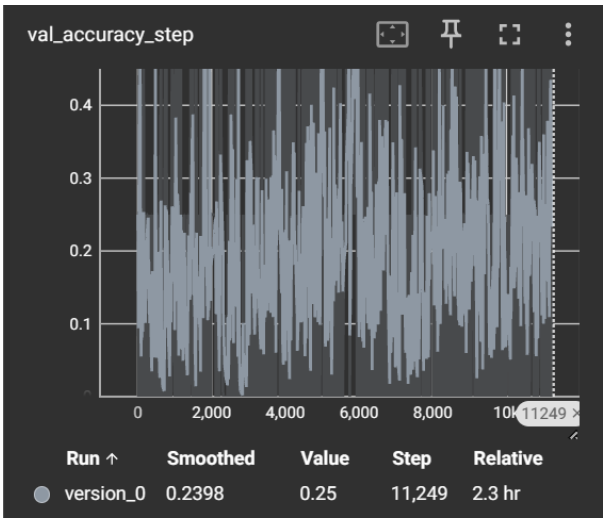

**Train F1 epoch:**



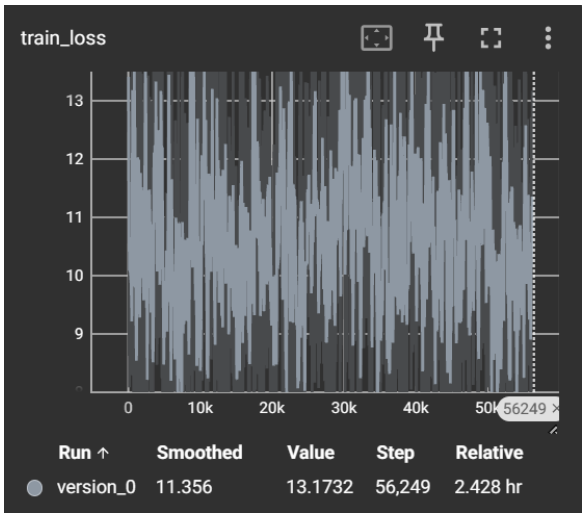
**Train F1 step:**



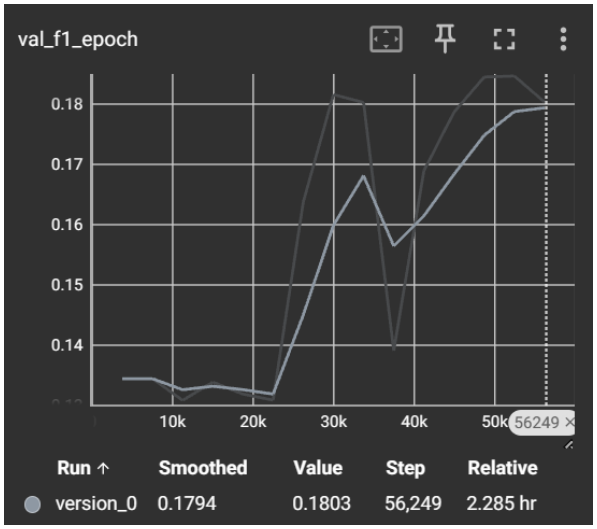

**Validation Accuracy Step:**



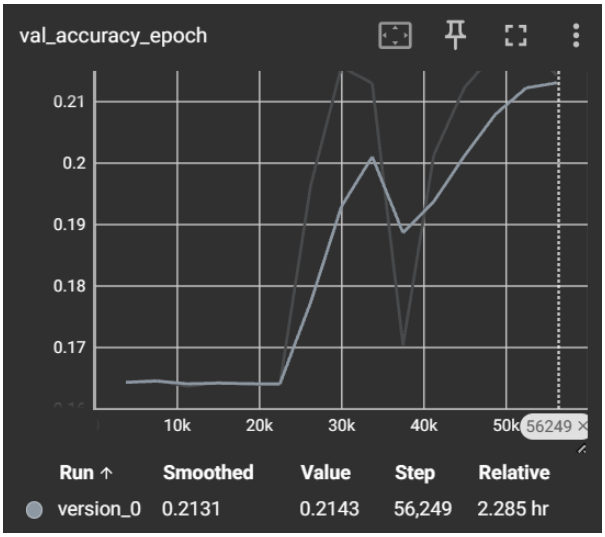

**Train Loss:**



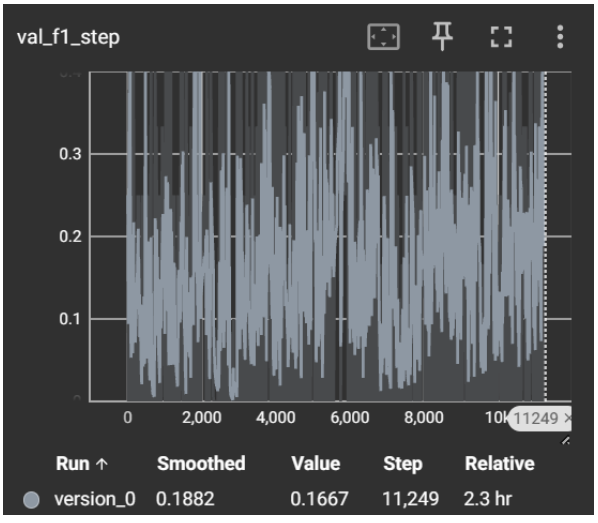

**Validation F1 epoch:**



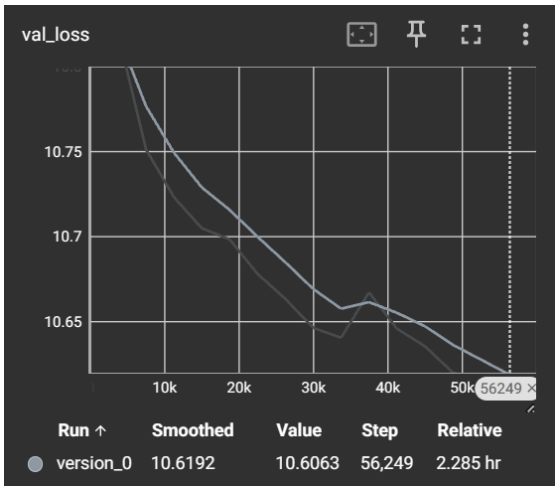

**Validation Accuracy:**



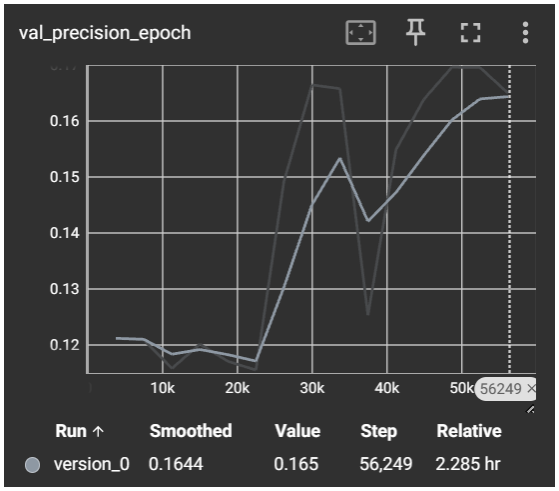

**Validation F1 step**



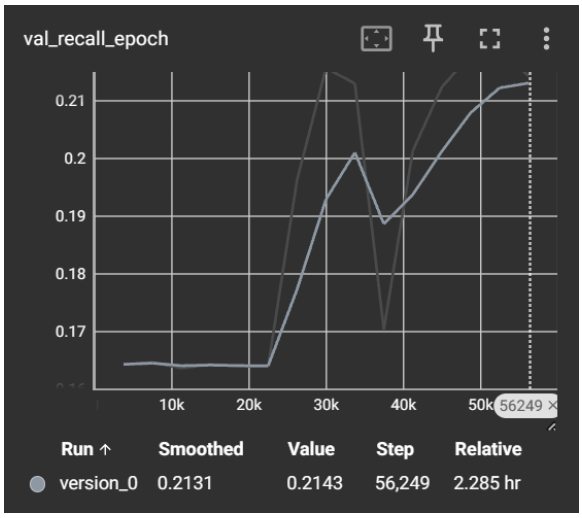
**Validation Loss:**




**Precision:**

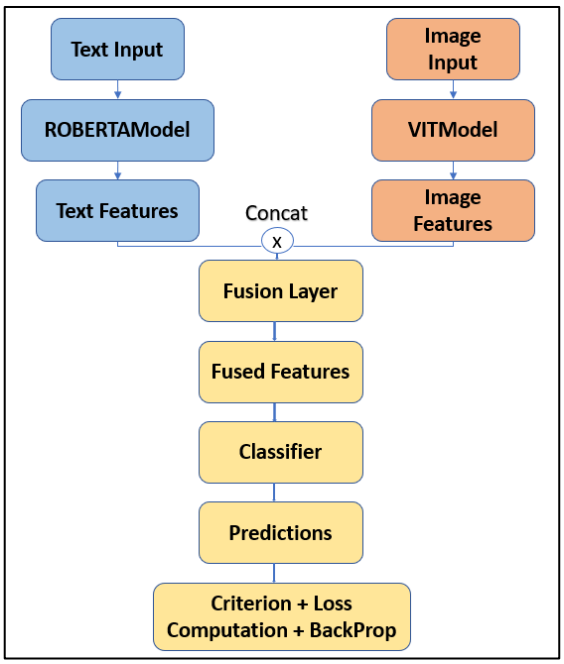



**Recall:**




## 3.5 Approach 5 – ROBERTA + VIT + 15epochs + 15000 datapoints

**Model Architecture:**

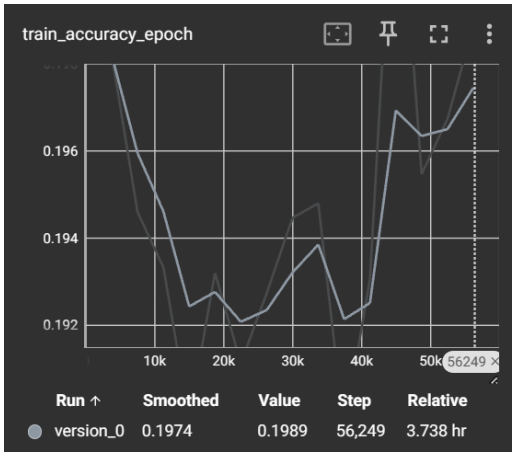



**Results**

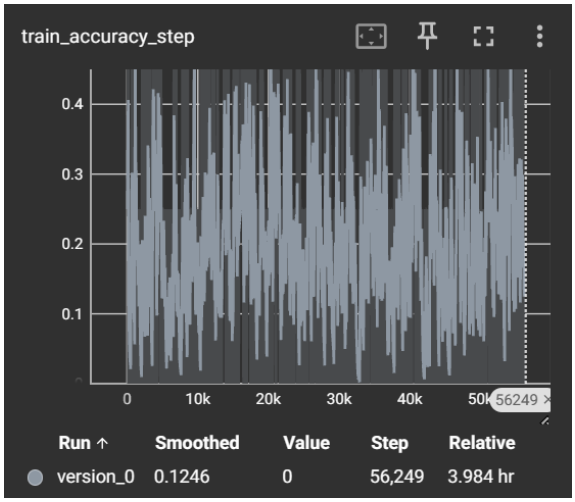

```
Question:        What type of beverage is sold here?
Answer:  None
Labels:  []
Scores:  []
Scores for these labels:         []
Predicted answers: ['yes', 'no']
```

```
Question:        Will wine be served?
Answer:  None
Labels:  []
Scores:  []
Scores for these labels:         []
Predicted answers: ['yes', 'no']
```

**Train Accuracy Epoch:**

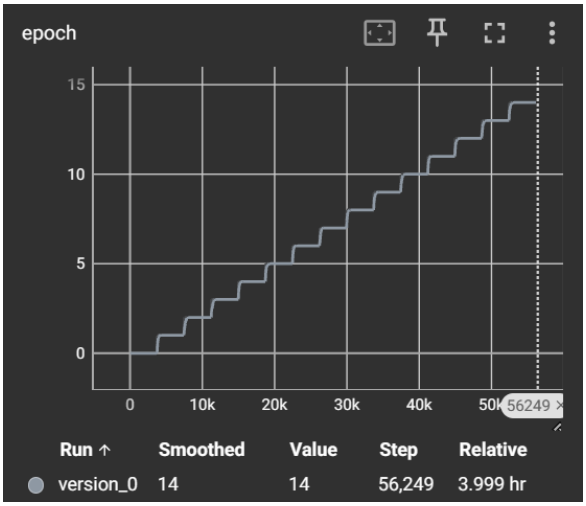



**Train Accuracy Step:**

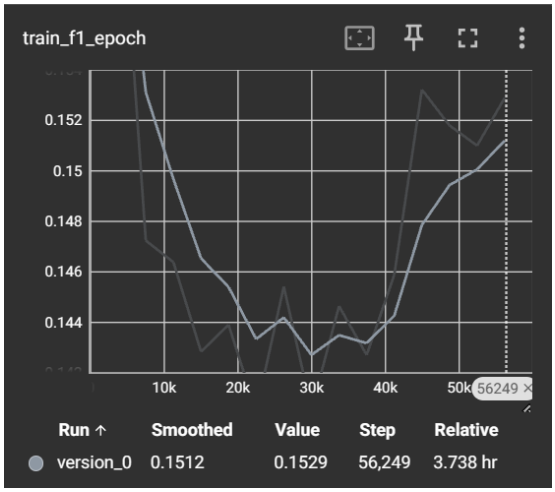



**Metrics:**

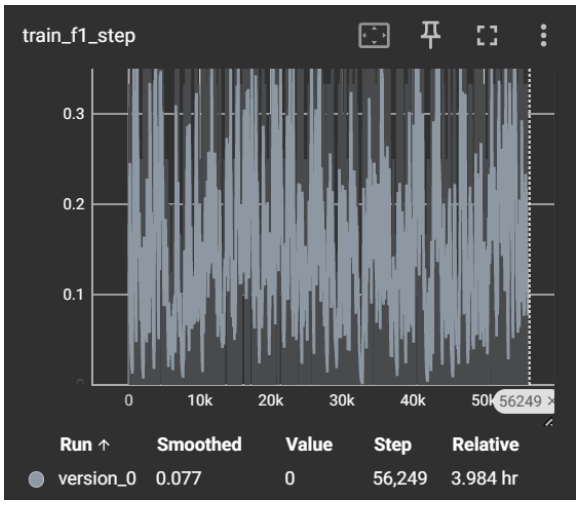
**Epoch:**



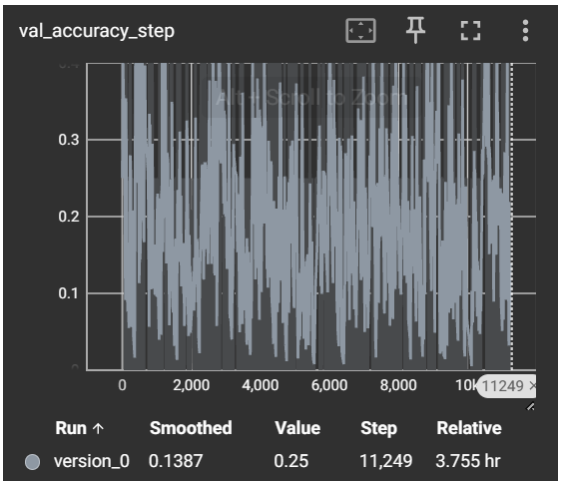

**Train F1 epoch:**



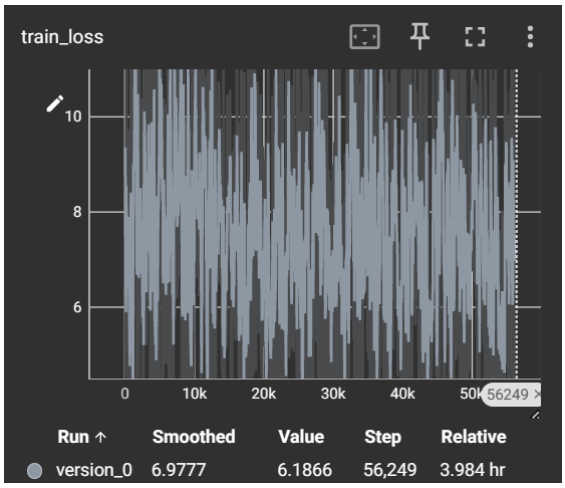
**Train F1 step:**



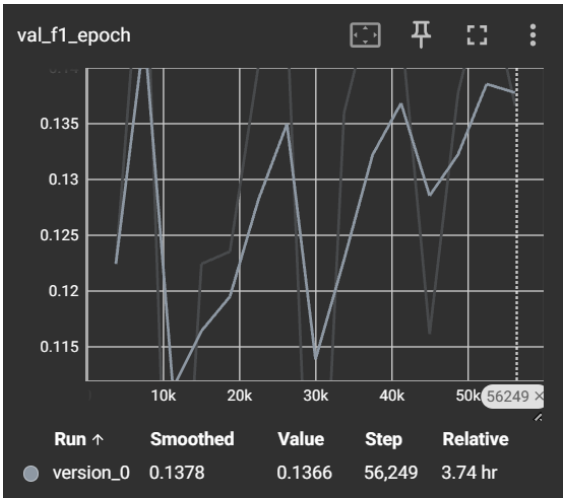

**Validation Accuracy Step:**



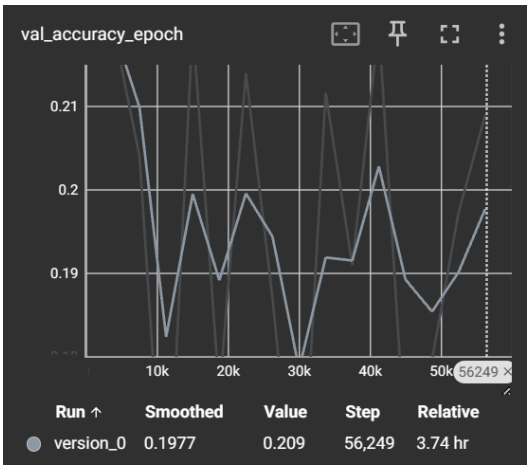

**Train Loss:**



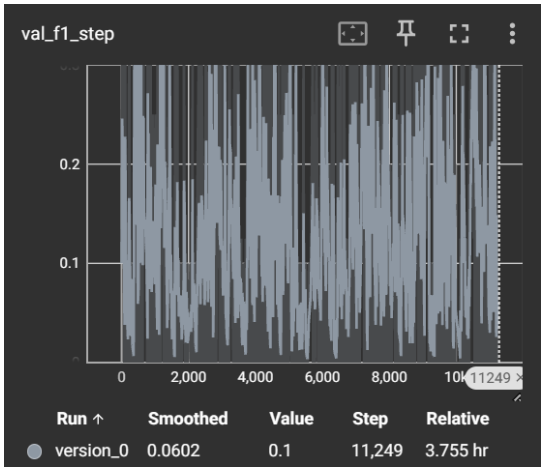

**Validation F1 epoch:**



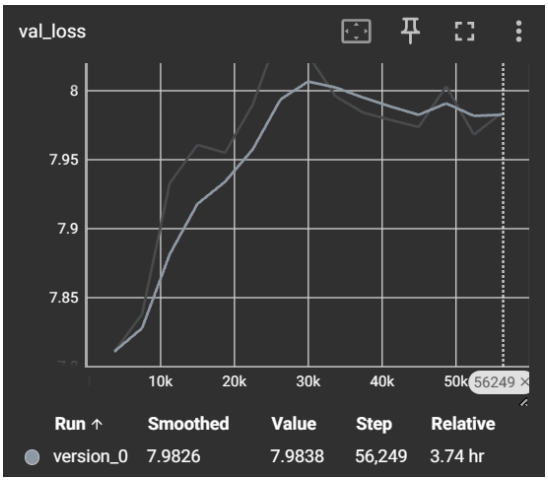

**Validation Accuracy:**



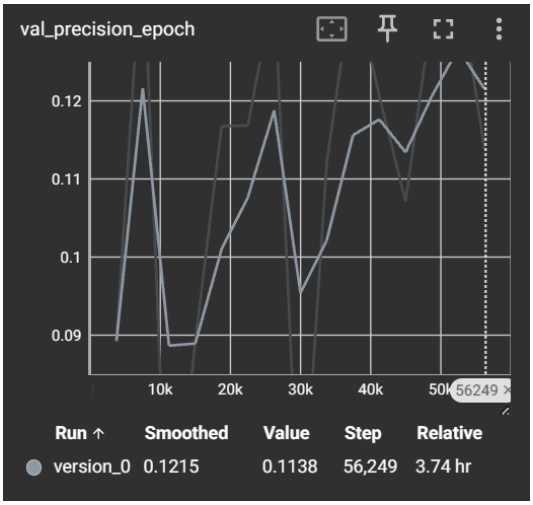

**Validation F1 step**



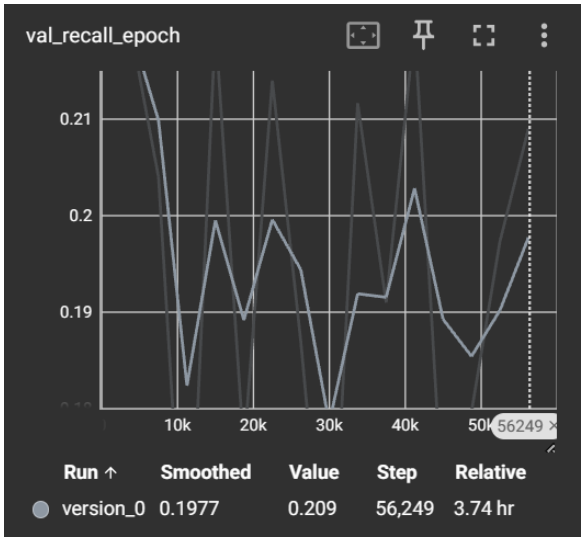
**Validation Loss:**




**Precision**

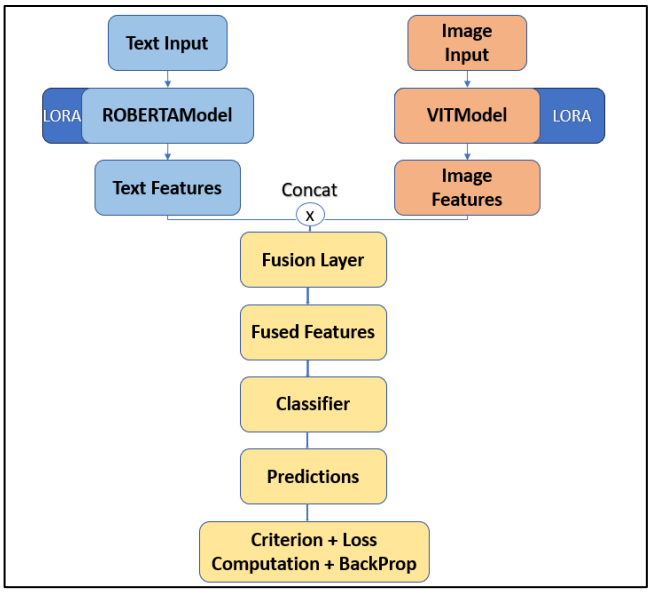



**Recall**

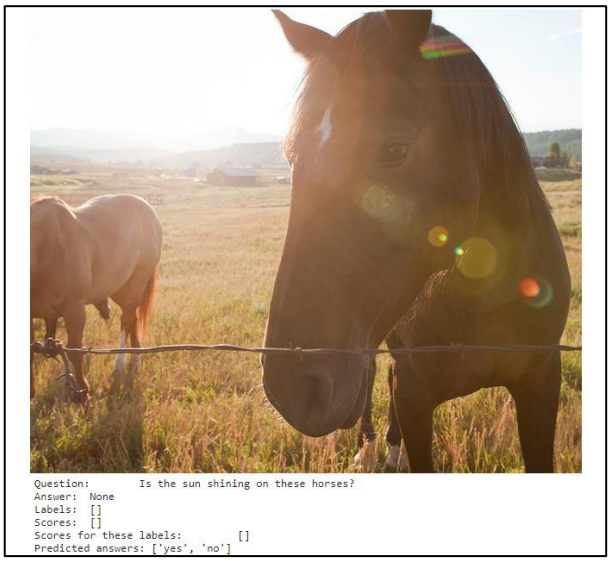



## 3.6 Approach 6 – ROBERTA + VIT + LORA + 15epochs + 15000 datapoints

**Model Architecture:**



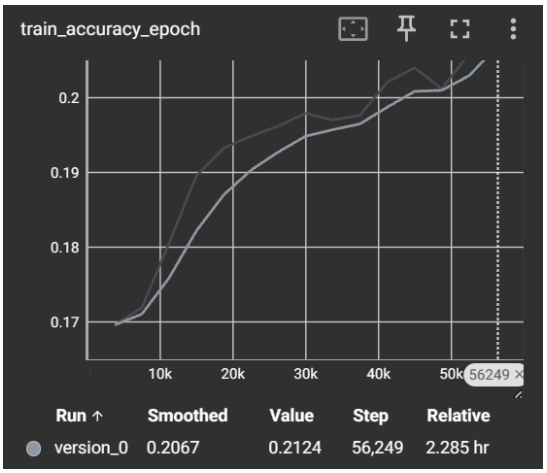

**Results:**



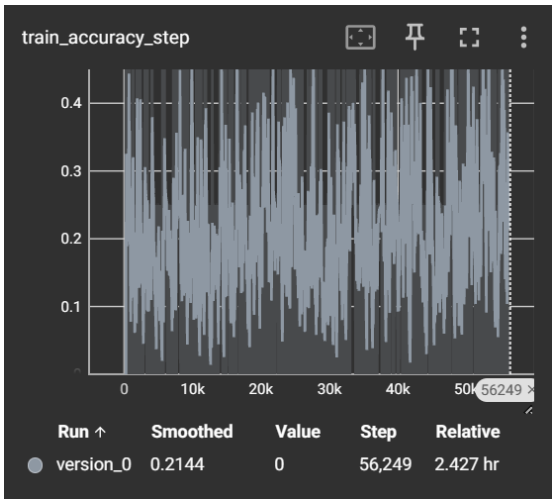
```
Question:        What color is the mouse?
Answer:  None
Labels:  []
Scores:  []
Scores for these labels:         []
Predicted answers: ['no', 'yes']
```

```
Question:        Is there water in the photo?
Answer:  None
Labels:  []
Scores:  []
Scores for these labels:         []
Predicted answers: ['yes', 'no']
```

**Train Accuracy Epoch:**

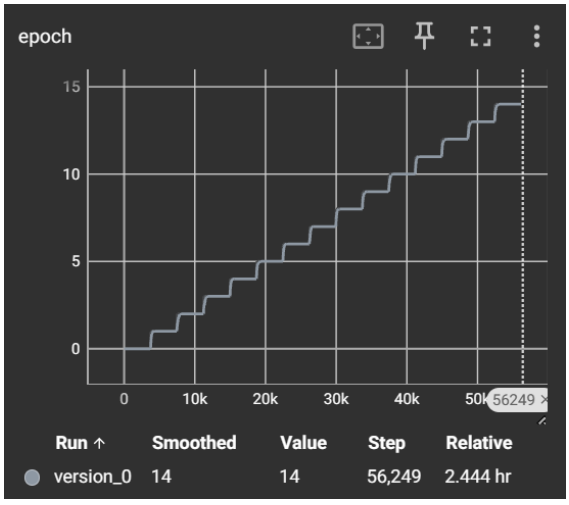



**Train Accuracy Step:**

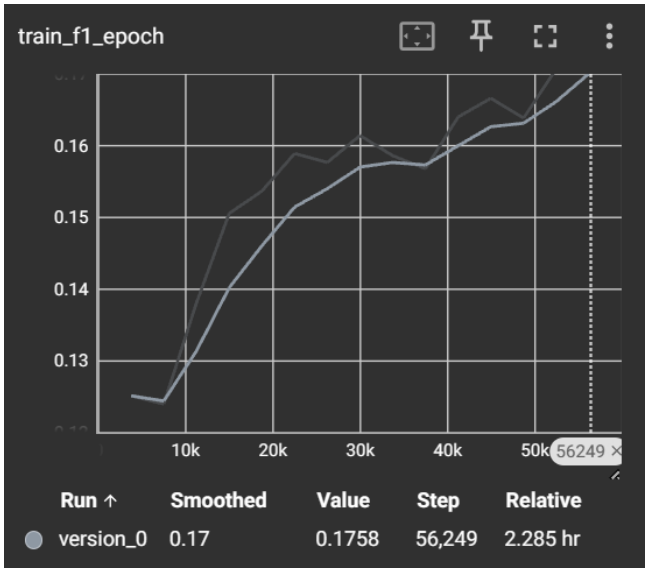



**<u>Metrics:</u>**

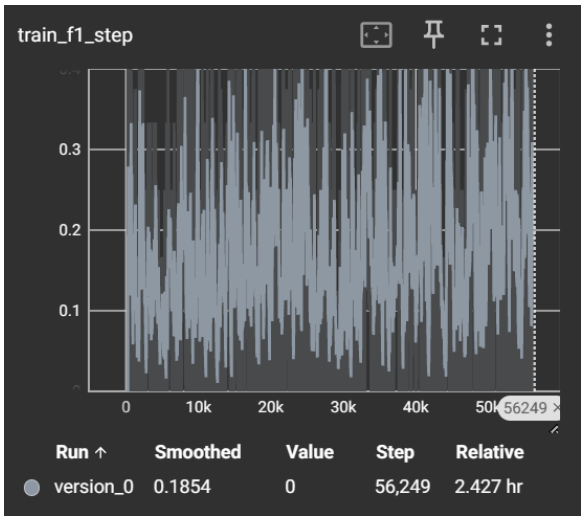
**Epoch:**



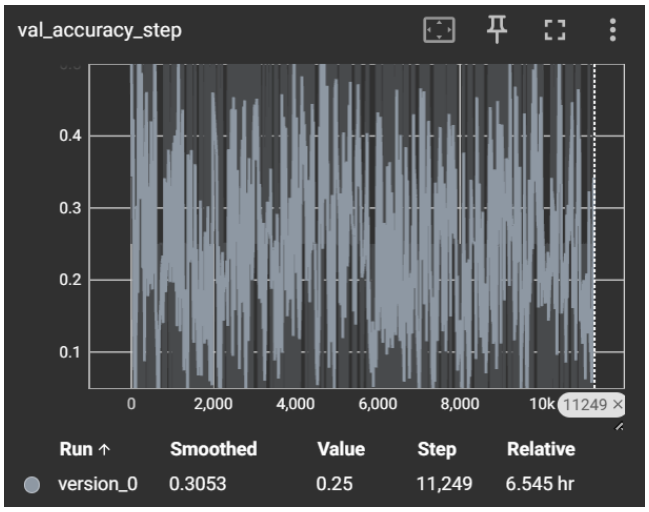

**Train F1 epoch:**



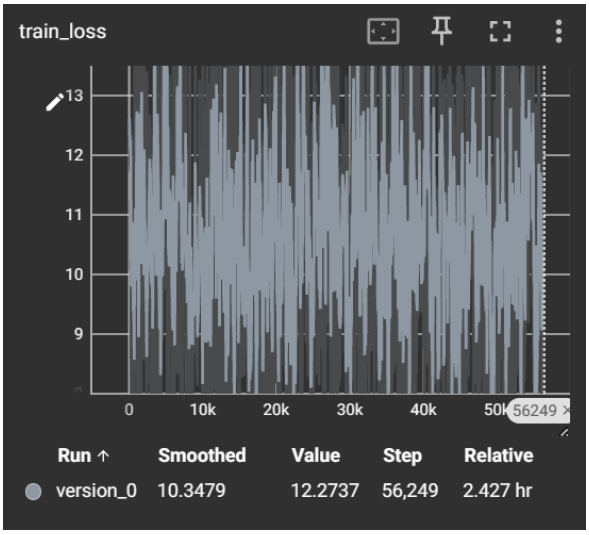
**Train F1 step:**



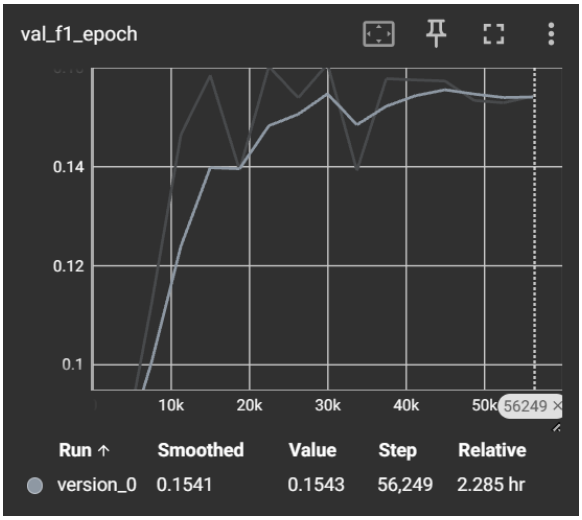

**Validation Accuracy Step:**



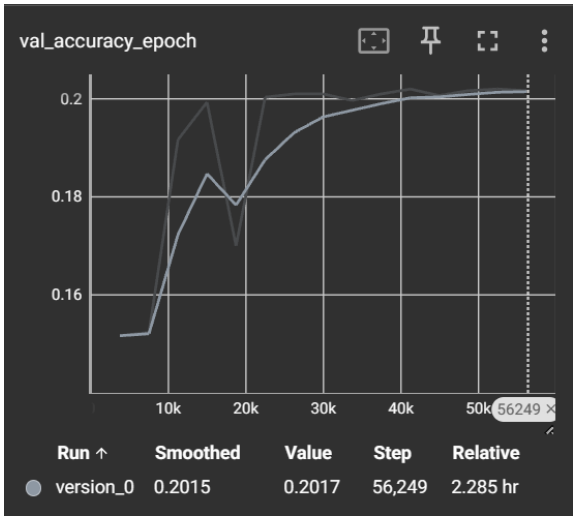

**Train Loss:**



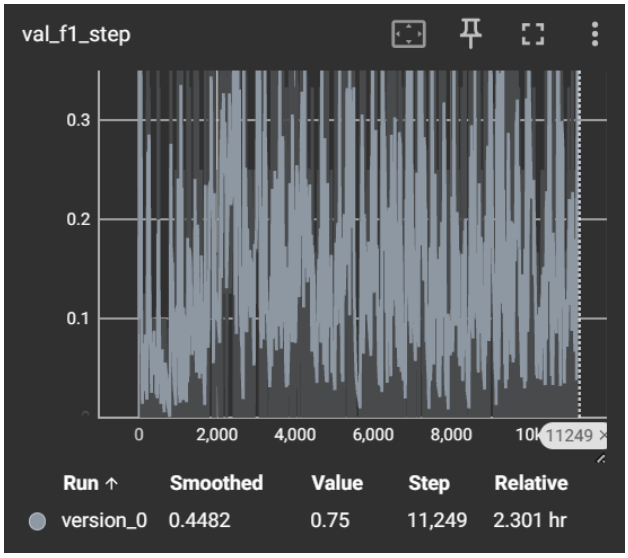

**Validation F1 epoch:**



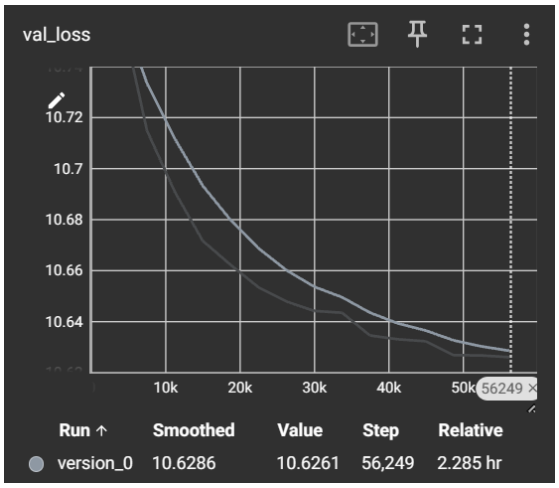

**Validation Accuracy:**



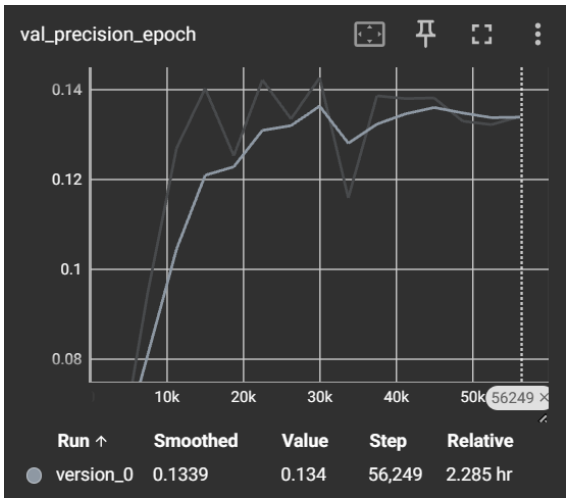

**Validation F1 step**



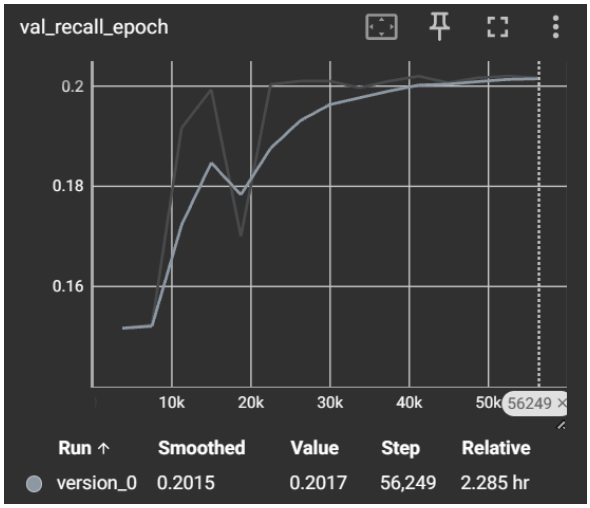
**Validation Loss:**




**Precision:**

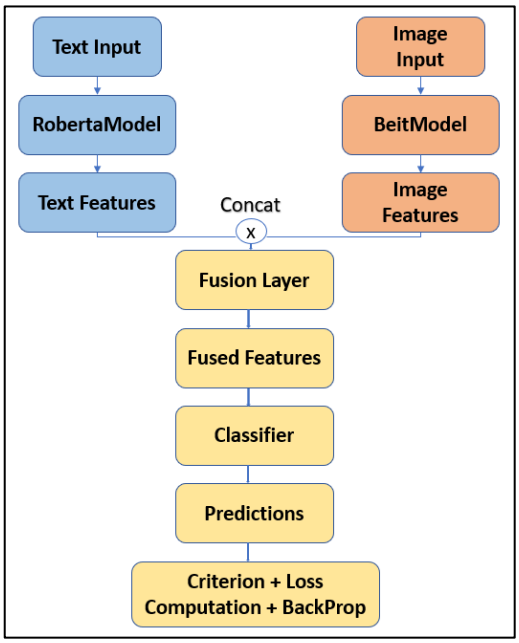



**Recall:**



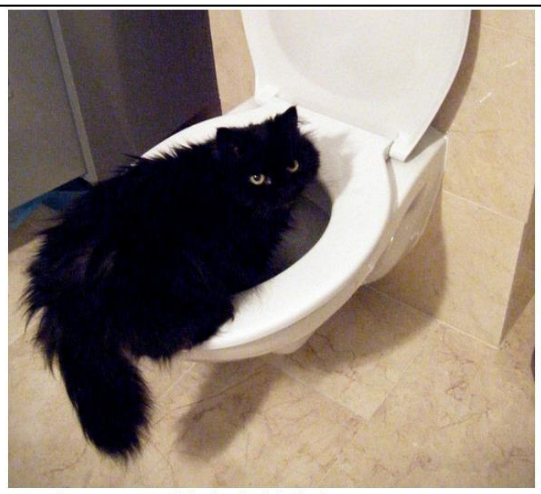

## 3.7 Approach 7 – ROBERTA + BEIT + 15epochs + 15000 datapoints

**Model Architecture:**




**Results:**

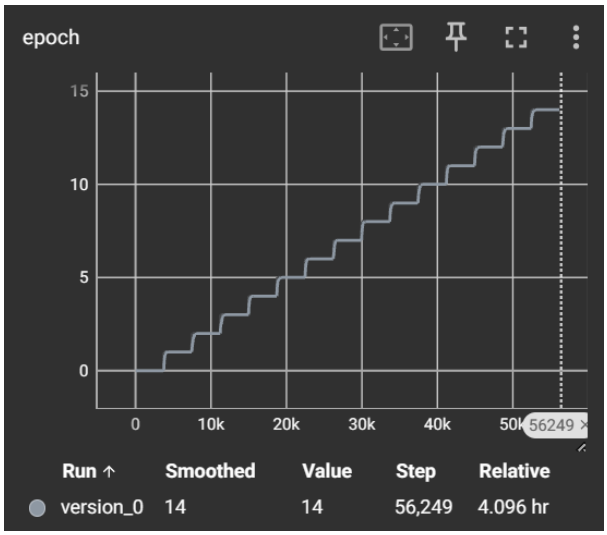
```
Question:         Is this a bedroom?
Answer:  None
Labels:  []
Scores:  []
Scores for these labels:          []
Predicted answers: ['yes', 'no']
```



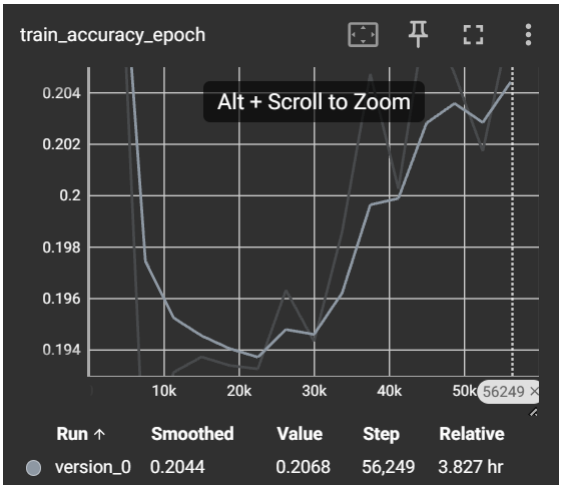
```
Question:         Is this in the kitchen?
Answer:  None
Labels:  []
Scores:  []
Scores for these labels:          []
Predicted answers: ['yes', 'no']
```

**Metrics:**

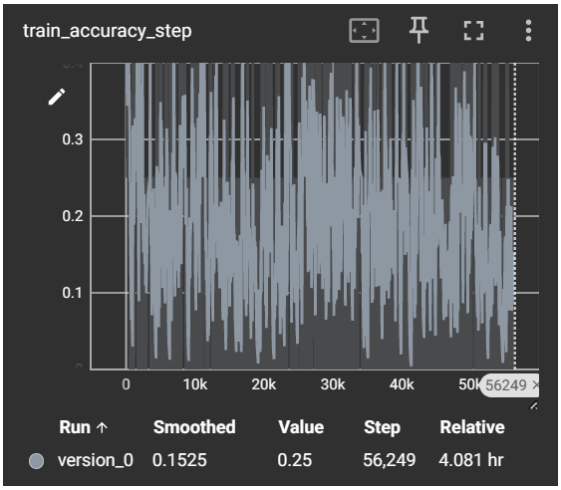
**Epoch**



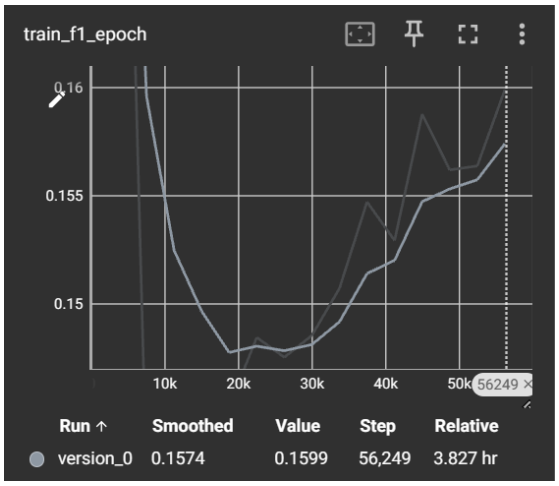

**Train Accuracy Epoch:**



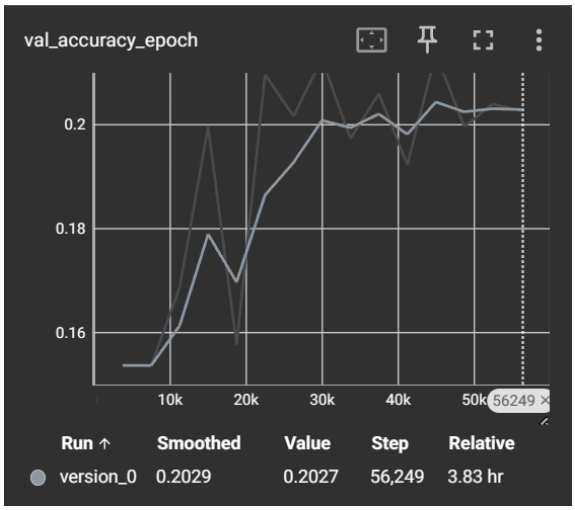

**Train Accuracy Step:**



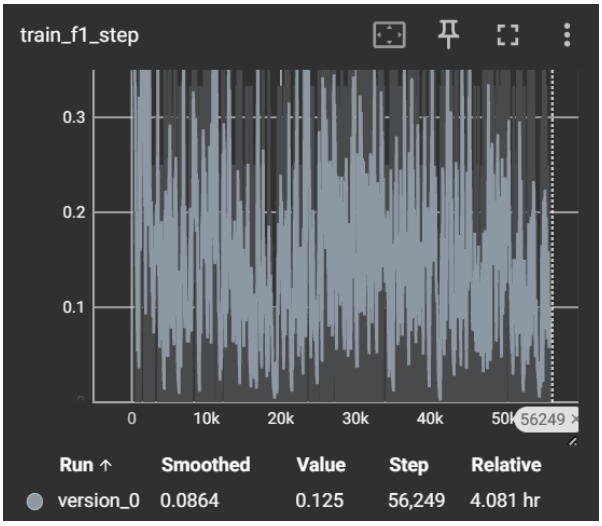
**Train F1 epoch:**



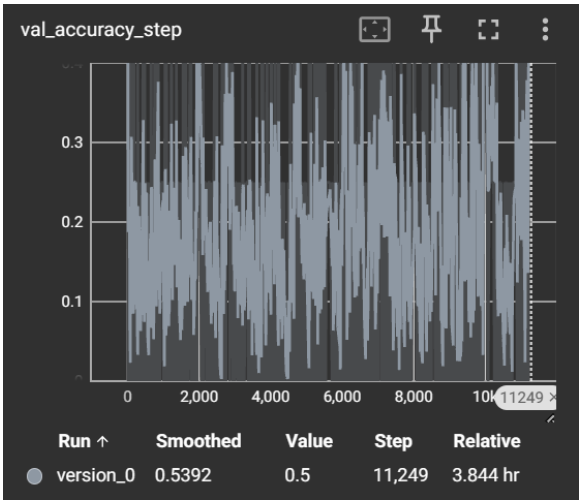

**Validation Accuracy:**



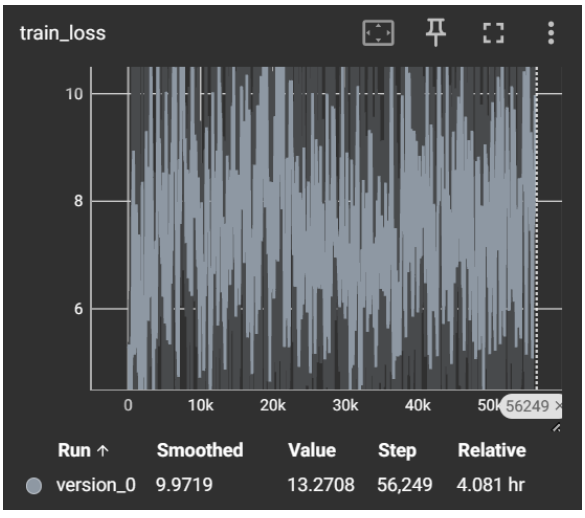

**Train F1 step:**



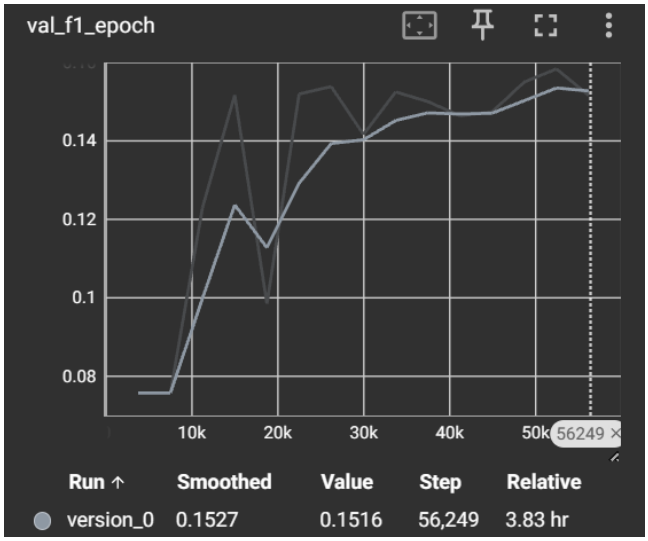

**Validation Accuracy Step:**



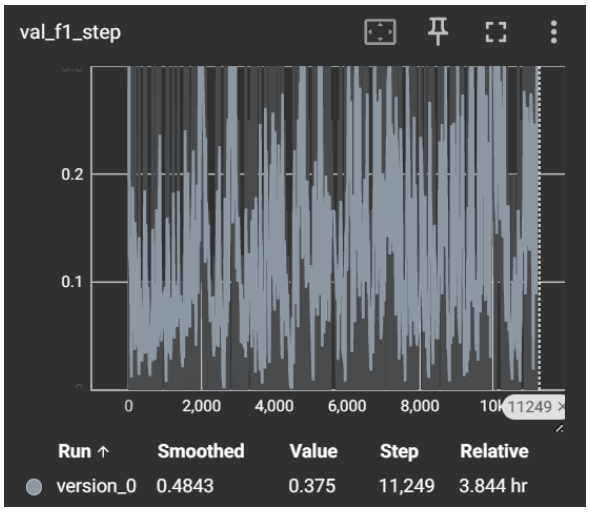

**Train Loss:**



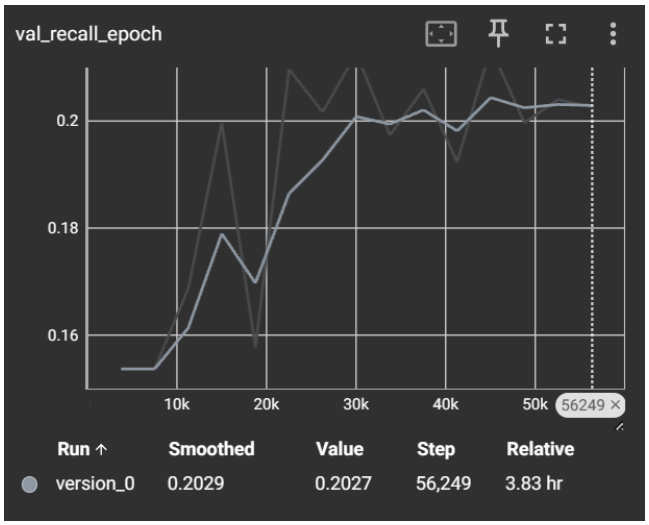

**Validation F1 epoch:**



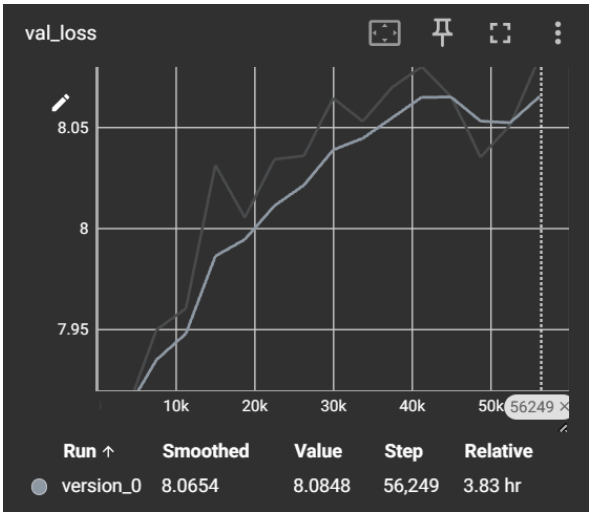
**Validation F1 step**




**Recall**



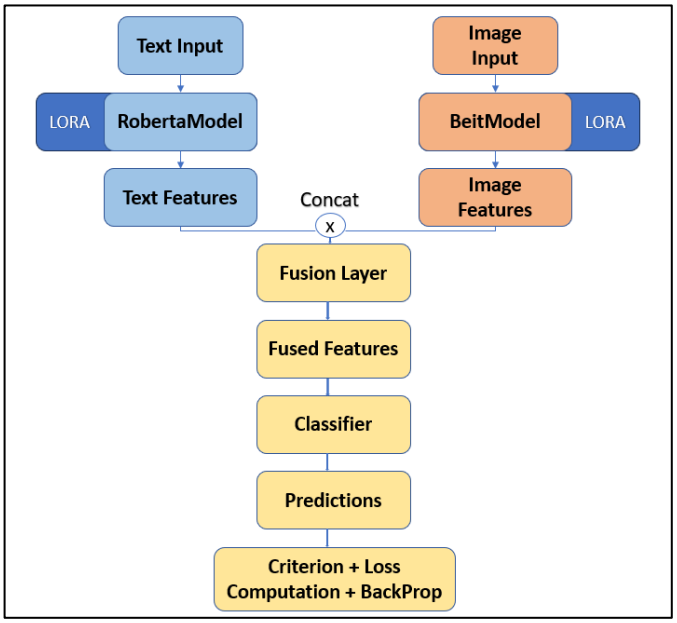

**Validation Loss:**




**Precision**


## 3.8 Approach 8 – ROBERTA + BEIT + LORA + 15epochs + 15000 datapoints

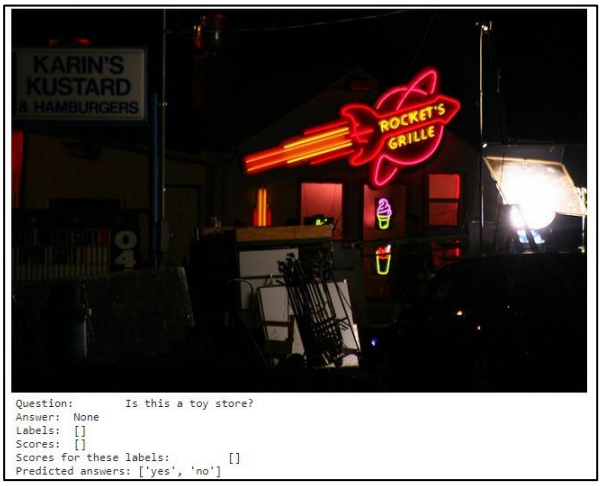
**Model Architecture:**



**Results:**

```
Question:       Is this in a restaurant?
Answer:  None
Labels:  []
Scores:  []
Scores for these labels:        []
Predicted answers: ['yes', 'no']
```

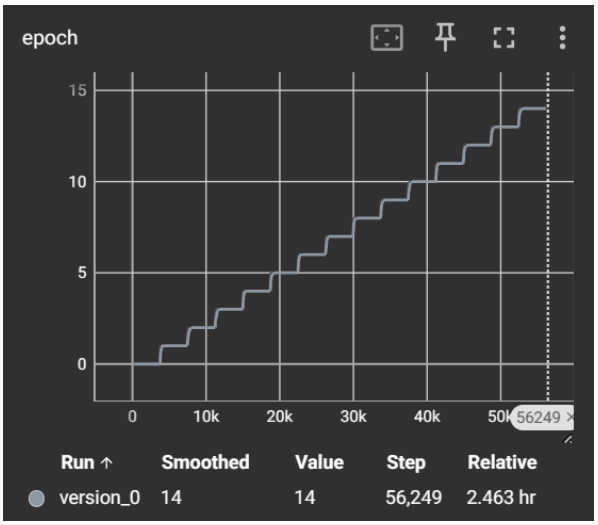
```
Question:       How many motorcycles are on the street?
Answer:  None
Labels:  []
Scores:  []
Scores for these labels:        []
Predicted answers: ['yes', 'no']
```



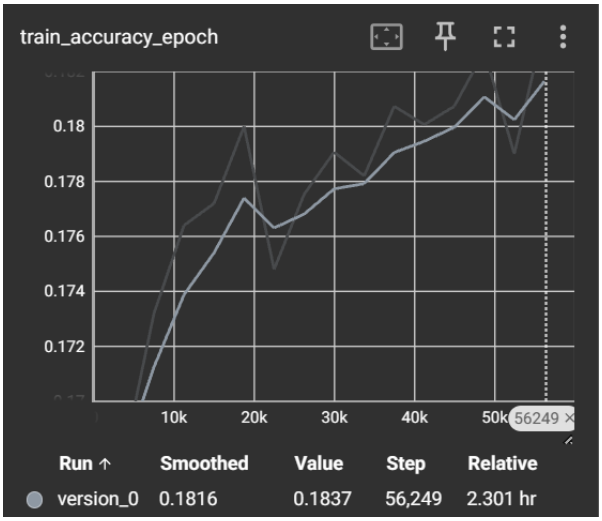

```
Question:       Is this a toy store?
Answer:  None
Labels:  []
Scores:  []
Scores for these labels:        []
Predicted answers: ['yes', 'no']
```

**Metrics:**

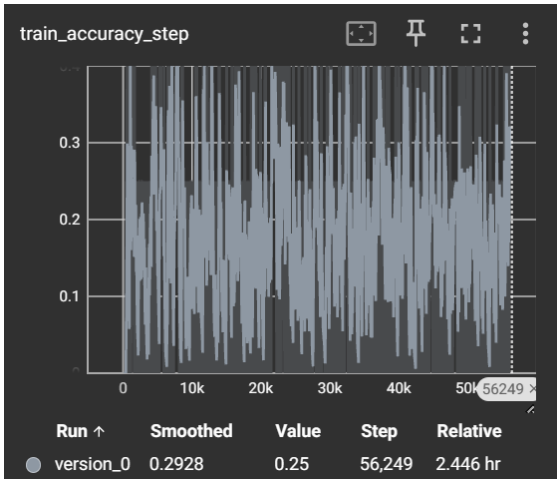
**Epoch:**



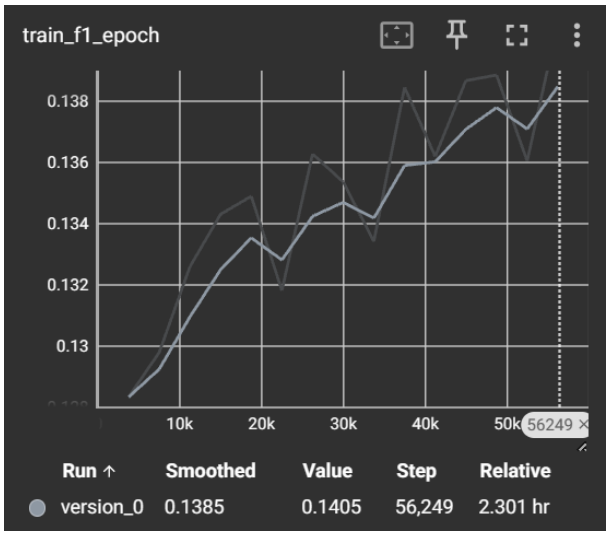

**Train Accuracy Epoch:**



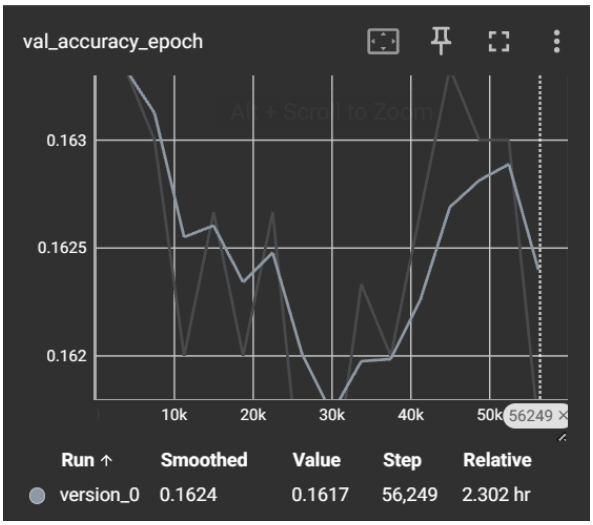

**Train Accuracy Step:**



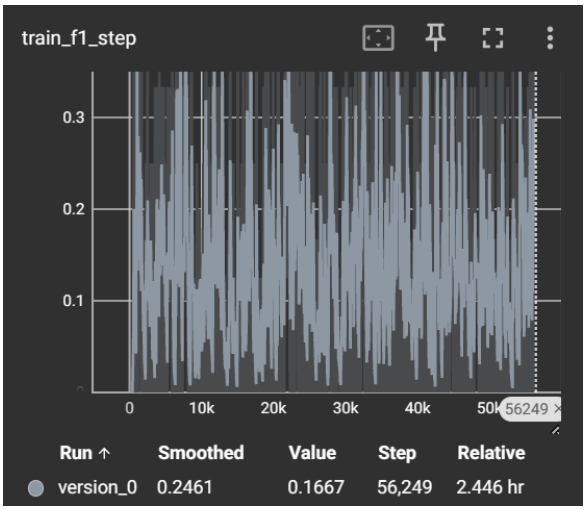
**Train F1 epoch:**



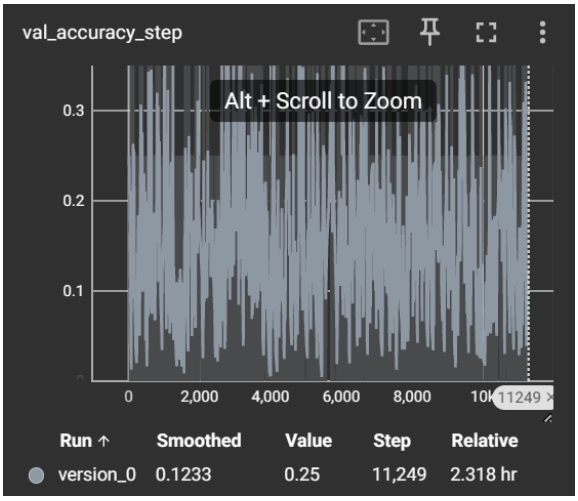

**Validation Accuracy:**



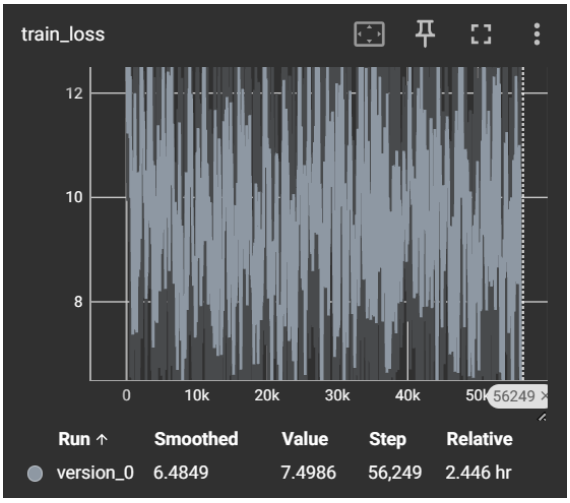

**Train F1 step:**



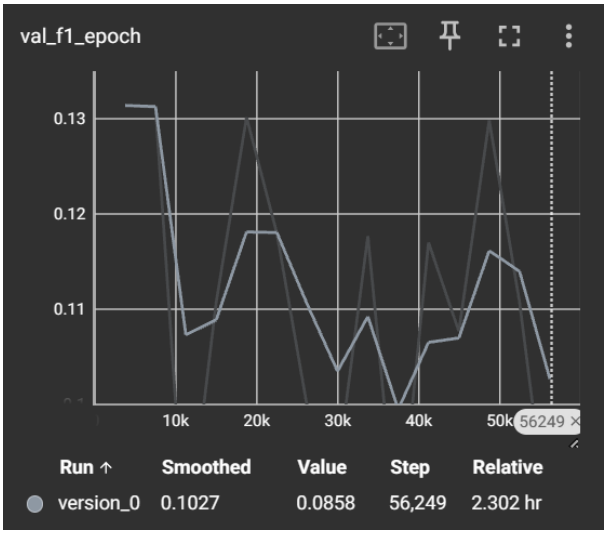

**Validation Accuracy Step:**



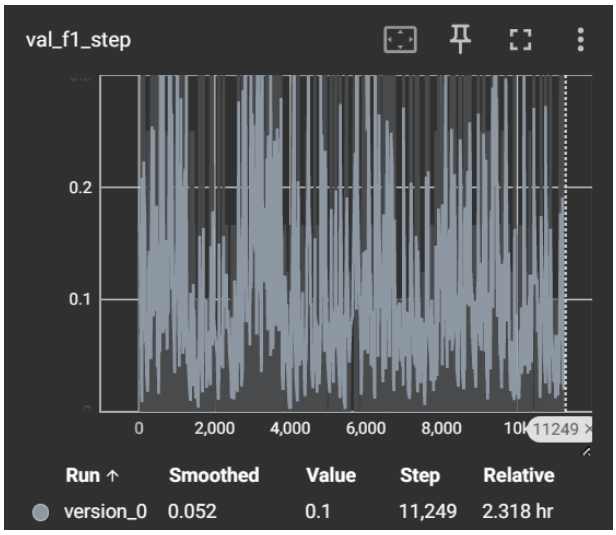

**Train Loss:**



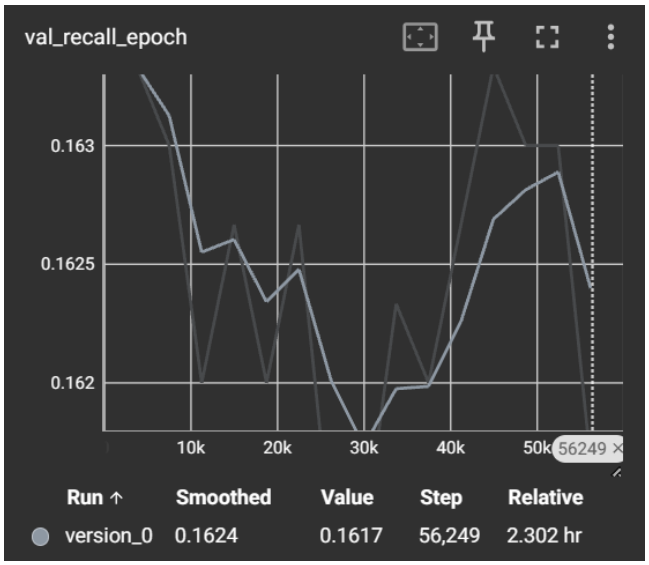

**Validation F1 epoch:**



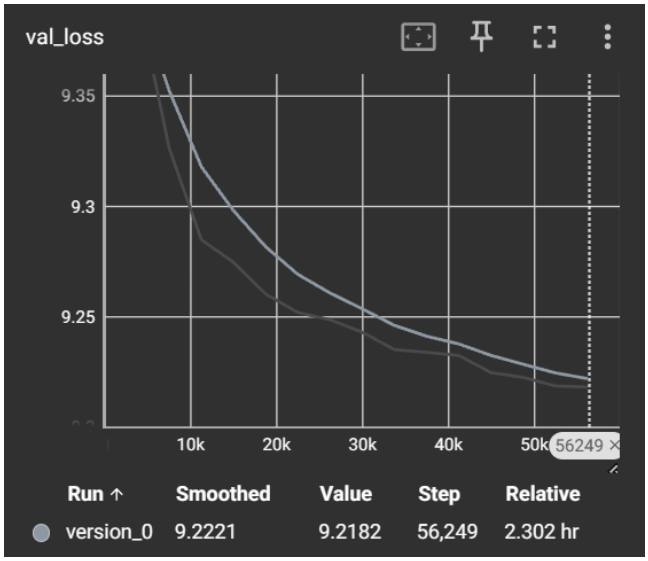
**Validation F1 step**



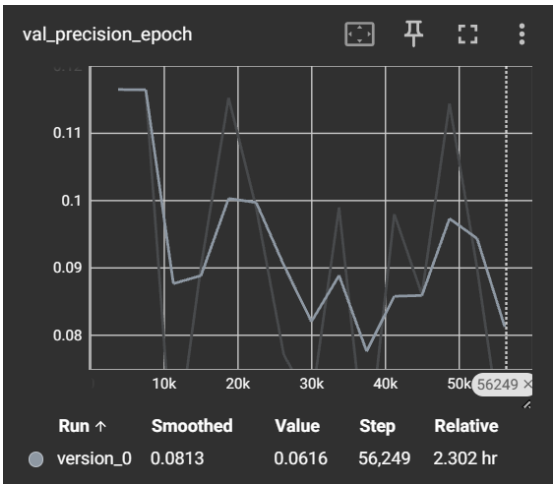

**Recall:**




**Validation Loss:**




**Precision:**



| Model (15000data + 15 epochs) | Time Taken | Accuracy | F1 Score | Precision | Recall |
|---|---|---|---|---|---|
| **BERT + VIT** | ~4hrs (It even took 9hrs once) | 0.1785 | 0.1495 | 0.1495 | 0.1784 |
| **BERT + VIT + LORA** | ~2.4hrs | 0.234 (**max value** got to 0.3324) | 0.1346 | 0.1 | 0 |
| **ALBERT + VIT** | ~3.6hrs | 0.2083 | 0.1404 | 0.1217 | 0.2083 |
| **ALBERT + VIT + LORA** | ~2.5hrs | 0.2131 | 0.1794 | 0.1645 | 0.2131 |
| **ROBERTA + VIT** | ~4hrs | 0.1977 | 0.1378 | 0.1215 | 0.1977 |
| **ROBERTA + VIT + LORA** | ~2.5hrs | 0.215 | 0.1541 | 0.1339 | 0.2015 |
| **ROBERTA + BEIT** | ~4.1hrs | 0.2029 (**max** went to 0.375) | 0.1527 | 0.1383 | 0.2029 |
| **ROBERTA + BEIT + LORA** | ~2.5hrs | 0.1624 | 0.1027 | 0.0813 | 0.1624 |

**Important Notes:**

1) The **max values** were obtained when we used dropout as 0.5. Thus, we can change around the hyperoperators to get better results. But according to our observation BERT+VIT+LORA and ROBERTA+BEIT seem to give better performace than the rest. But if data and time to train is increased with suitable hyperparameters these models can perform better.
2) Note: We have trained the model only on 15000 data for 15 epochs hence the results are somewhat good mostly for yes/no kind of questions and some more, if we can train this model on more data with more epochs ~100 with suitable hyperparameters we will surely get improved accuracy.
3) The model is able to properly answer yes no question and if it is trained longer on larger data this will work robustly as a Visual Question Answering Service.

**How to use our Code?**

1) Model-name ipnyb file (example – vit-lora-albert-precision) is used for Data Preprocessing, Training and Testing
2) The ipnyb file with name as output (ex: vit-lora-albert-output) is used for Inference.
3) The model weights and the lighting logs for Tensorboard are also present either inside a folder inside each models' folder.