MOUNT GOOGLE DRIVE

```
# Mount Google Drive
from google.colab import drive
import os

# Mount Google Drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

1. EDA and Data Processing

```
import os
import json
import pickle
import torch
from PIL import Image
from torch.utils.data import Dataset, DataLoader
from transformers import BertTokenizer, BertModel
from torchvision import transforms
import torch.nn as nn
import matplotlib.pyplot as plt

# ✅ Paths
base_path = '/content/drive/MyDrive/VQA_Project'
image_folder = os.path.join(base_path, 'Images/AbstractScenes_v1')

# ✅ Load questions & annotations
with open(os.path.join(base_path, 'Questions/MultipleChoice_Questions_train_abstract_v1
    train_questions = json.load(f)['questions']
with open(os.path.join(base_path, 'Questions/MultipleChoice_Questions_val_abstract_v1.j
    val_questions = json.load(f)['questions']
with open(os.path.join(base_path, 'Annotations/Abstract_Scenes_Annotations_Train.json')
    train_annotations = json.load(f)['annotations']
with open(os.path.join(base_path, 'Annotations/Abstract_Scenes_Annotations_Val.json'))
    val_annotations = json.load(f)['annotations']

# ✅ Prepare Data
train_ans_map = {ann['question_id']: ann['multiple_choice_answer'] for ann in train_anr
val_ans_map = {ann['question_id']: ann['multiple_choice_answer'] for ann in val_annotat

def prepare_data(questions, ans_map, split='train'):
    data = []
    for q in questions:
```

```
        q_id = q['question_id']
        image_id = q['image_id']
        img_filename = f"abstract_v002_{split}2015_{image_id:012d}.png"
        if ans_map.get(q_id) in q['multiple_choices']:
            data.append({
                'image': img_filename,
                'question': q['question'],
                'choices': q['multiple_choices'],
                'answer': ans_map[q_id]
            })
    return data

train_data = prepare_data(train_questions, train_ans_map, split='train')
val_data = prepare_data(val_questions, val_ans_map, split='val')

# ✅ Image transform
image_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])

# ✅ Tokenizer
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

# ✅ Dataset class with safety patch
class VQADataset(Dataset):
    def __init__(self, data, image_folder, tokenizer, transform=None, max_length=32):
        self.data = data
        self.image_folder = image_folder
        self.tokenizer = tokenizer
        self.transform = transform
        self.max_length = max_length

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        item = self.data[idx]
        image_path = os.path.join(self.image_folder, item["image"])
        try:
            image = Image.open(image_path).convert("RGB")
        except:
            image = Image.new("RGB", (224, 224), color=(0, 0, 0))  # Black placeholder

        if self.transform:
            image = self.transform(image)

        encoded = self.tokenizer(
            [f"{item['question']} {choice}" for choice in item["choices"]],
            padding="max_length", truncation=True, max_length=self.max_length, return_t
        )
```

```
        label = item["choices"].index(item["answer"])

        return {
            "image": image,
            "input_ids": encoded["input_ids"],
            "attention_mask": encoded["attention_mask"],
            "label": torch.tensor(label),
            "question": item["question"],  # ✅ Include question
            "choices": item["choices"],    # ✅ Include choices
            "answer": item["answer"]       # ✅ Include correct answer
        }

train_dataset = VQADataset(train_data, image_folder, tokenizer, transform=image_transfo
val_dataset = VQADataset(val_data, image_folder, tokenizer, transform=image_transform)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, num_workers=0)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False, num_workers=0)
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public model
  warnings.warn(
tokenizer_config.json: 100%  48.0/48.0 [00:00<00:00, 5.70kB/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 4.59MB/s]
tokenizer.json: 100%  466k/466k [00:00<00:00, 23.1MB/s]
config.json: 100%  570/570 [00:00<00:00, 72.7kB/s]
```

Check Last Saved Model & Accuracy

```
import os
import pickle
import datetime

# ✅ Set paths
base_path = "/content/drive/MyDrive/VQA_Project"
model_path = os.path.join(base_path, "vqa_finetuned_best.pt")
history_path = os.path.join(base_path, "train_history.pkl")

# ✅ Check last modified time
if os.path.exists(model_path):
    last_modified = os.path.getmtime(model_path)
    last_save_time = datetime.datetime.fromtimestamp(last_modified).strftime('%Y-%m-%d %
    print(f"📁 Model Last Saved: {last_save_time}")
else:
    print("❌ Model file not found.")
```

```
17
18 # ✅ Load last saved validation accuracy
19 if os.path.exists(history_path):
20     with open(history_path, "rb") as f:
21         history = pickle.load(f)
22
23     # Extract last epoch data
24     if "val_accuracies" in history and history["val_accuracies"]:
25         last_epoch = len(history["val_accuracies"])
26         last_val_acc = history["val_accuracies"][-1] * 100  # Convert to percentage
27         last_train_acc = history["train_accuracies"][-1] * 100  # Convert to percentage
28
29         print(f"📊 Last Saved Epoch: {last_epoch}")
30         print(f"✅ Last Train Accuracy: {last_train_acc:.2f}%")
31         print(f"✅ Last Validation Accuracy: {last_val_acc:.2f}%")
32     else:
33         print("⚠️ No accuracy data found in history file.")
34 else:
35     print("❌ Training history file not found.")
36
```

```
📁 Model Last Saved: 2025-03-24 13:53:03
📊 Last Saved Epoch: 8
✅ Last Train Accuracy: 75.67%
✅ Last Validation Accuracy: 63.72%
```

Load the Model

```
1 import torch
2 import torch.nn as nn
3 from transformers import BertModel
4 from torchvision.models import resnet18
5
6 class VQAModel(nn.Module):
7     def __init__(self, hidden_dim=512, num_choices=4):  # Ensure num_choices is set corr
8         super(VQAModel, self).__init__()
9
10        # ✅ Image encoder (ResNet18 without final classifier)
11        self.cnn = resnet18(pretrained=True)
12        self.cnn.fc = nn.Identity()  # Remove classification layer
13        self.cnn_out_dim = 512  # ResNet18 output size
14
15        # ✅ Text encoder (BERT, frozen)
16        self.bert = BertModel.from_pretrained('bert-base-uncased')
17        for param in self.bert.parameters():
18            param.requires_grad = False  # Freeze BERT during training
19
20        self.bert_out_dim = self.bert.config.hidden_size  # 768
21
22        # ✅ Fusion layer (Image + Text features)
```

```
        self.fusion = nn.Sequential(
            nn.Linear(self.cnn_out_dim + self.bert_out_dim, hidden_dim),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(hidden_dim, 1)  # One score per choice
        )

    def forward(self, images, input_ids, attention_mask):
        batch_size, num_choices, seq_len = input_ids.shape

        # ✅ Encode images
        img_feat = self.cnn(images)  # [batch, 512]
        img_feat = img_feat.unsqueeze(1).repeat(1, num_choices, 1)  # [batch, num_choice

        # ✅ Encode text
        input_ids = input_ids.view(-1, seq_len)  # Flatten for BERT batch processing
        attention_mask = attention_mask.view(-1, seq_len)
        text_outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask)
        text_feat = text_outputs.pooler_output  # [batch*num_choices, 768]
        text_feat = text_feat.view(batch_size, num_choices, -1)  # [batch, num_choices,

        # ✅ Fuse image + text
        fused = torch.cat([img_feat, text_feat], dim=-1)  # [batch, num_choices, 1280]
        scores = self.fusion(fused).squeeze(-1)  # [batch, num_choices]

        return scores  # Logits

```

2 Load Model Checkpoint

```
import torch
import os
import torchvision.models as models
from torchvision.models import ResNet18_Weights

resnet18(weights=ResNet18_Weights.DEFAULT)  # For most up-to-date weights
backbone = resnet18(weights=ResNet18_Weights.DEFAULT)
backbone.fc = nn.Identity()

# ✅ Define device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# ✅ Initialize model & move to device
model = VQAModel().to(device)

# ✅ Load the checkpoint with correct architecture
checkpoint_path = os.path.join(base_path, "vqa_finetuned_best.pt")
state_dict = torch.load(checkpoint_path, map_location=device, weights_only=True)

# ✅ Load state dict with `strict=False` to avoid minor mismatches
```

```
missing_keys, unexpected_keys = model.load_state_dict(state_dict, strict=False)

print(f"⚠️ Missing keys: {missing_keys}")  # Should be empty
print(f"⚠️ Unexpected keys: {unexpected_keys}")  # Should be minimal

# ✅ Set model to evaluation mode
model.eval()
print("✅ Model successfully loaded and ready for evaluation.")

```

```
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.cache
100%|██████████| 44.7M/44.7M [00:00<00:00, 60.2MB/s]
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: UserWarning: T
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: UserWarning: A
  warnings.warn(msg)
model.safetensors: 100% 440M/440M [00:02<00:00, 227MB/s]
⚠️ Missing keys: []
⚠️ Unexpected keys: []
✅ Model successfully loaded and ready for evaluation.
```

Reload Data Sample Prediction

```
# ✅ Ibitialize dataset & dataloader
train_dataset = VQADataset(train_data, image_folder, tokenizer, transform=image_transfor
val_dataset = VQADataset(val_data, image_folder, tokenizer, transform=image_transform)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, num_workers=0)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=True, num_workers=0)

```

Sample Predictions on Validation Data

```
import matplotlib.font_manager as fm
print([f.name for f in fm.fontManager.ttflist])
```

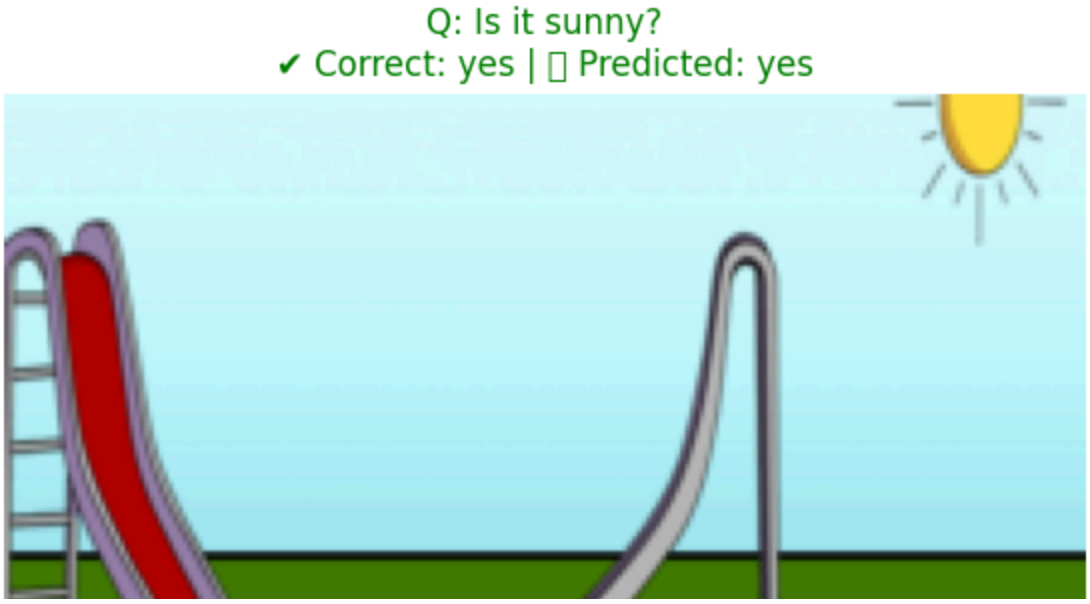
```
['STIXSizeTwoSym', 'DejaVu Serif', 'STIXGeneral', 'cmr10', 'STIXNonUnicode', 'STIXSizeOr
```

```
import random
import matplotlib.pyplot as plt
from PIL import Image
import torch

```

```
# ✅ Configure Matplotlib to use fonts available on your system
from matplotlib import rcParams
rcParams['font.family'] = ['DejaVu Sans', 'Liberation Sans', 'STIXGeneral']  # Based on
rcParams['axes.unicode_minus'] = False  # Ensure proper rendering of minus signs and Uni

# ✅ Define function to randomly pick images from the dataset for sample predictions
def show_random_predictions(model, dataset, num_samples=10):
    model.eval()
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    fig, axes = plt.subplots(num_samples, 1, figsize=(8, 6 * num_samples))  # Increased
    if num_samples == 1:
        axes = [axes]  # Ensure iterable format

    # 🔀 Randomly select samples from the dataset
    sample_indices = random.sample(range(len(dataset)), num_samples)

    with torch.no_grad():
        for i, idx in enumerate(sample_indices):
            batch = dataset[idx]  # Directly fetch a sample from dataset
            image = batch['image'].unsqueeze(0).to(device)  # Add batch dimension
            input_ids = batch['input_ids'].unsqueeze(0).to(device)
            attention_mask = batch['attention_mask'].unsqueeze(0).to(device)
            label = batch['label'].item()

            # 🔍 Make prediction
            logits = model(image, input_ids, attention_mask)
            pred_idx = torch.argmax(logits, dim=1).item()

            # 🖼️ Get question & choices
            img = image[0].cpu().permute(1, 2, 0).numpy()
            question = batch['question']
            choices = batch['choices']

            # 🔄 Ensure index stays within bounds
            predicted_index = min(pred_idx, len(choices) - 1)
            correct_index = min(label, len(choices) - 1)

            predicted_choice = choices[predicted_index]
            correct_choice = choices[correct_index]

            # ✅ Text-based alternatives for Emojis
            correct_emoji = "✔"  # Unicode check mark
            predicted_emoji = "🔮"  # Unicode crystal ball

            # 🎨 Plot
            axes[i].imshow(img)
            axes[i].axis("off")
            axes[i].set_title(
                f"Q: {question}\n{correct_emoji} Correct: {correct_choice} | {predicted_
                color="green" if predicted_choice == correct_choice else "red"
```

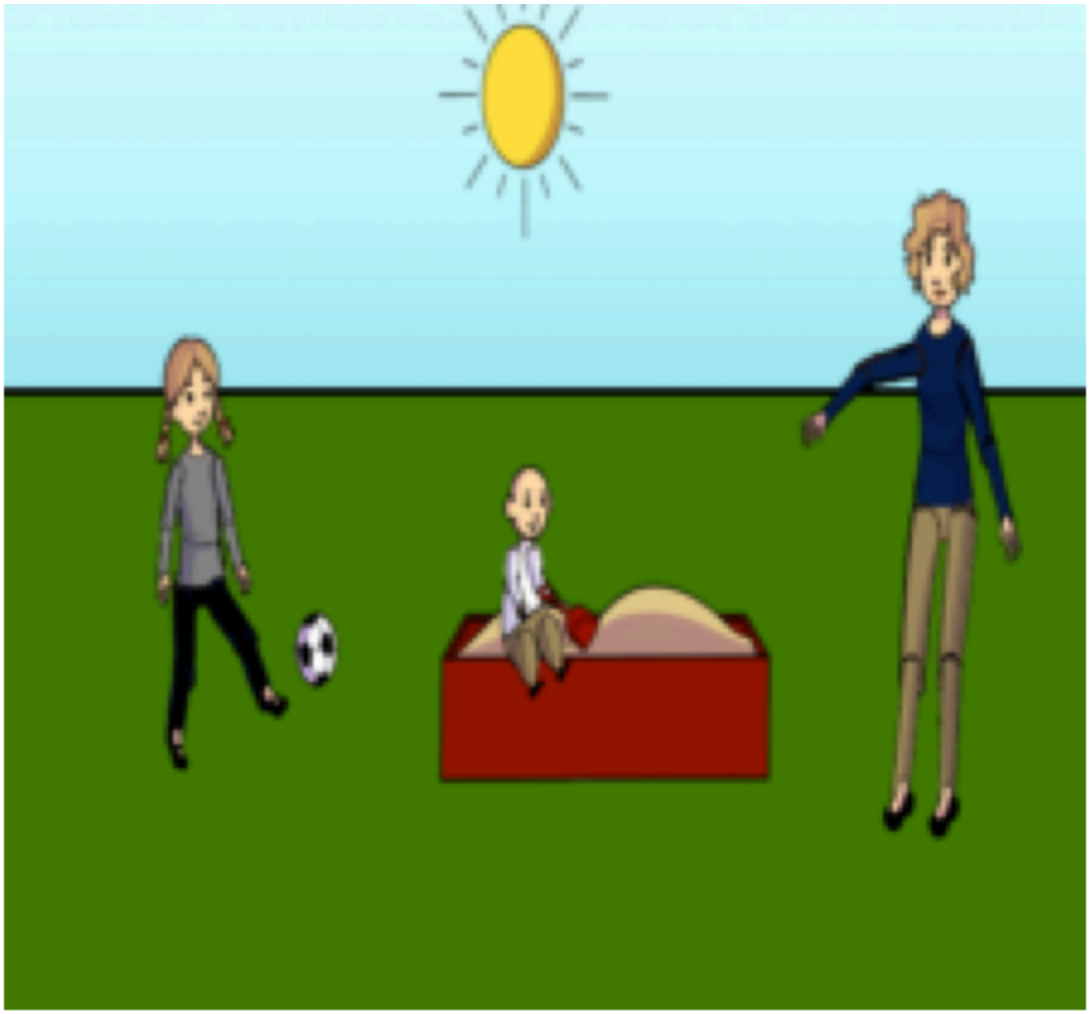
```
            )

    plt.tight_layout()
    plt.show()

# 🔥 Run with randomized selection from the dataset
```

```
<ipython-input-8-e9719dd27217>:59: UserWarning: Glyph 128302 (\N{CRYSTAL BALL}) missing
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Gly
  fig.canvas.print_figure(bytes_io, **kw)
```



Q: Where is the game control at?
✔ Correct: floor | □ Predicted: floor

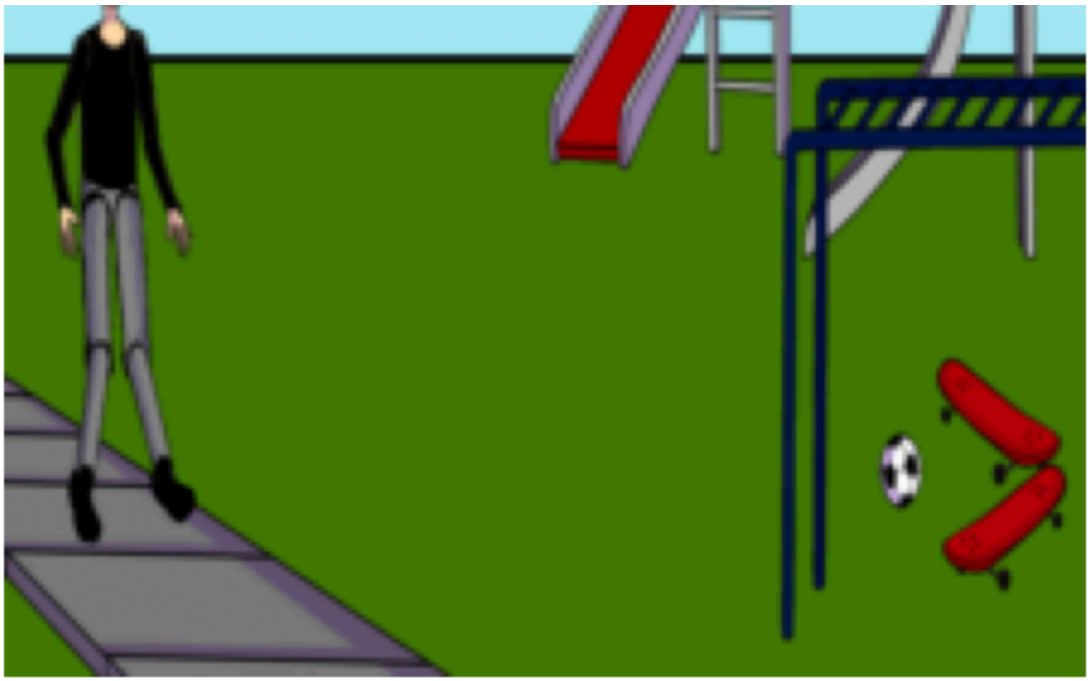
Q: What is being kicked?
✔ Correct: soccer ball | □ Predicted: football



Q: Is the woman going to eat alone?
✔ Correct: yes | □ Predicted: yes

Q: Is there a TV?
✔ Correct: no | □ Predicted: no

Q: Is that a basketball on the ground?
✔ Correct: no | □ Predicted: no



Q: Are they women?
✔ Correct: yes | ✘ Predicted: no

Q: Does the dog belong to the woman?
✔ Correct: yes | ✘ Predicted: yes

✔ Correct: yes | □ Predicted: yes

Q: Are the 2 people dancing?
✔ Correct: yes | □ Predicted: yes

✅ Step 2: Generate a Classification Report

```
from sklearn.metrics import classification_report

# ✅ Function to evaluate and generate classification report
def generate_classification_report(model, dataloader):
    model.eval()
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    all_preds, all_labels = [], []

    with torch.no_grad():
        for batch in dataloader:
            images = batch['image'].to(device)
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['label'].cpu().numpy()  # Convert to NumPy

            logits = model(images, input_ids, attention_mask)
            preds = torch.argmax(logits, dim=1).cpu().numpy()

            all_preds.extend(preds)
            all_labels.extend(labels)

    # 📊 Print classification report
    print("🧪 Classification Report:")
    print(classification_report(all_labels, all_preds))

# 🔥 Run classification report
generate_classification_report(model, val_loader)

```

```
🧪 Classification Report:
              precision    recall  f1-score   support

           0       0.64      0.63      0.63      1715
           1       0.65      0.66      0.65      1671
           2       0.62      0.64      0.63      1652
           3       0.65      0.64      0.65      1681
           4       0.63      0.63      0.63      1623
           5       0.63      0.64      0.63      1657
           6       0.65      0.65      0.65      1694
           7       0.64      0.64      0.64      1687
           8       0.64      0.64      0.64      1685
           9       0.65      0.63      0.64      1641
          10       0.64      0.64      0.64      1668
          11       0.64      0.64      0.64      1673
          12       0.65      0.64      0.64      1669
          13       0.64      0.64      0.64      1656
          14       0.65      0.65      0.65      1646
          15       0.64      0.63      0.63      1641
          16       0.64      0.63      0.63      1640
```

```
              17       0.62      0.63      0.63      1701

        accuracy                           0.64     30000
       macro avg       0.64      0.64      0.64     30000
    weighted avg       0.64      0.64      0.64     30000
```

```
Start coding or generate with AI.
```

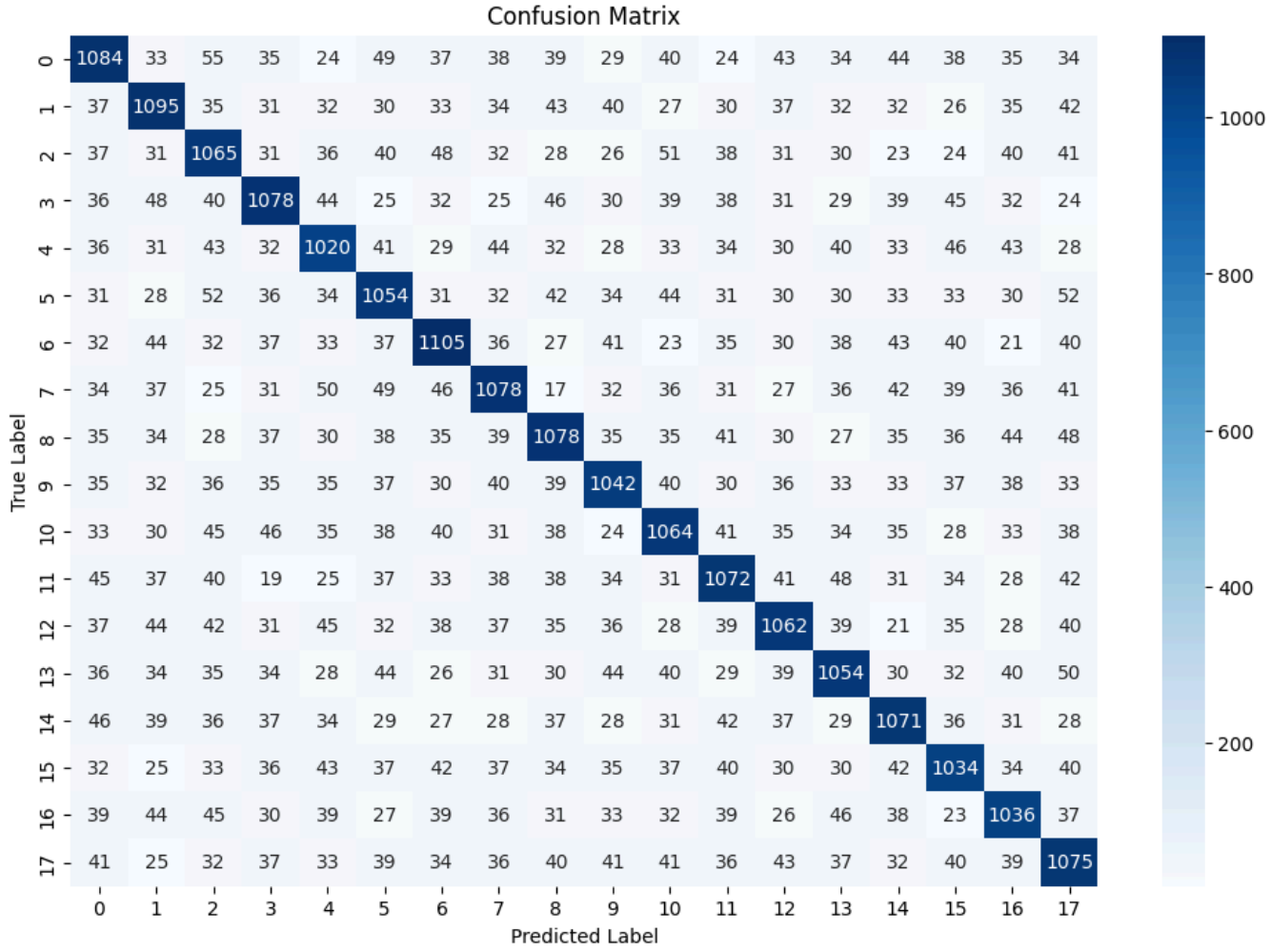
```
Start coding or generate with AI.
```

✅ Step 1: Extract True Labels (y_true) & Predictions (y_pred)

```
import torch
from torch.utils.data import DataLoader

val_dataloader = DataLoader(val_dataset, batch_size=32, shuffle=False)  # Adjust batch_s

y_true = []
y_pred = []

# Ensure model is in evaluation mode
model.eval()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Loop through validation data
with torch.no_grad():
    for batch in val_dataloader:  # Replace with your actual DataLoader
        images = batch['image'].to(device)
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['label'].cpu().numpy()  # Extract true labels

        # Get predictions
        logits = model(images, input_ids, attention_mask)
        predictions = torch.argmax(logits, dim=1).cpu().numpy()  # Convert to numpy

        # Store results
        y_true.extend(labels)
        y_pred.extend(predictions)

```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Compute confusion matrix
cm = confusion_matrix(y_true, y_pred)
```

```
 7
 8 # Get class labels (if available)
 9 class_labels = sorted(set(y_true))  # Extract unique class labels
10
11 # Plot confusion matrix
12 plt.figure(figsize=(12, 8))
13 sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels, yticklabels
14 plt.xlabel("Predicted Label")
15 plt.ylabel("True Label")
16 plt.title("Confusion Matrix")
17 plt.show()
18
```



```
from collections import Counter

# Initialize counters
question_type_counter = Counter()
answer_counter = Counter()

# Single iteration over the dataset
for sample in train_dataset:
    question_type_counter.update([sample['question'][:10].lower()])  # Count question ty
    answer_counter.update([sample['answer']])  # Count answers

# Display results
print("Top question types:", question_type_counter.most_common(10))
print("Answer distribution:", answer_counter.most_common(10))
```

```
import pickle
import matplotlib.pyplot as plt

# ✅ Load training history
with open("/content/drive/MyDrive/VQA_Project/train_history.pkl", "rb") as f:
    history = pickle.load(f)

train_losses = history["train_losses"]
val_losses = history["val_losses"]
train_accuracies = history["train_accuracies"]
val_accuracies = history["val_accuracies"]
```