

Model Evaluation

Please run with high RAM and on the A100 in Google Colab. It will not run with a less well-performing GPU.

This notebook fine-tunes a pretrained LLaVA model using LoRA on our curated dataset, a subset of the Chart-To-Text dataset.

Additionally, the notebook will evaluate metrics for the baseline LLaVA model, our model, and the state-of-the-art ChartInsight model.

The estimated time to run the entire notebook on the above settings is roughly 80 mins.

```
!pip -q uninstall -y transformers accelerate tokenizers huggingface-hub safetensors pillow peft  
!pip -q install -U transformers accelerate tokenizers huggingface-hub safetensors "pillow<12.0,>=11.0.0" peft
```

```
import transformers, peft
print("Transformers:", transformers.__version__)
print("PEFT:", peft.__version__)
```

```
44.0/44.0 kB 3.7 MB/s eta 0:00:00  
12.0/12.0 MB 148.7 MB/s eta 0:00:00  
380.9/380.9 kB 35.5 MB/s eta 0:00:00  
3.3/3.3 MB 129.3 MB/s eta 0:00:00  
566.1/566.1 kB 44.7 MB/s eta 0:00:00  
507.2/507.2 kB 43.6 MB/s eta 0:00:00  
6.6/6.6 MB 146.7 MB/s eta 0:00:00  
556.4/556.4 kB 49.8 MB/s eta 0:00:00
```

Transformers: 4.57.2
PEFT: 0 18 0

```
# !pip -q uninstall -y spacy
# !pip -q install -U spacy==3.8.0 --no-deps
# !python -q -m spacy download en_core_web_sm
# !pip -q install -U --no-deps nltk

!pip -q uninstall -y spacy
!pip -q install "spacy<4" # let pip pick a stable 3.x version with all deps
!python -m spacy download en_core_web_sm
```

```
Collecting en-core-web-sm==3.8.0
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-3.8.0/en_core_web_sm-3.8.0-p
                                                               12.8/12.8 MB 147.4 MB/s eta 0:00:00
✓ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
△ Restart to reload dependencies
If you are in a Jupyter or Colab notebook, you may need to restart Python in
order to load all the package's dependencies. You can do this by selecting the
'Restart kernel' or 'Restart runtime' option.
```

```
# clone helper function repo
!git clone -q https://github.com/salaniz/pycocoevalcap.git
%cd pycocoevalcap
!python -q setup.py install
!pip -q install --upgrade --no-deps langcodes rouge-score
```

```
/content/pycocoevalcap  
running install  
/usr/local/lib/python3.12/dist-packages/setuptools/_distutils/cmd.py:66: SetuptoolsDeprecationWarning: setup.py ins:  
!!
```

```
*****
Please avoid running ``setup.py`` directly.
Instead, use pypa/build, pypa/installer or other
standards-based tools.
```

See https://blog.ganssle.io/articles/2021/10/setup_py-deprecated.html for details.

```
!!  
    self.initialize_options()  
/usr/local/lib/python3.12/dist-packages/setuptools/_distutils/cmd.py:66: EasyInstallDeprecationWarning: easy_instal'  
!!
```

```
*****
Please avoid running ``setup.py`` and ``easy_install``.
Instead, use pypa/build, pypa/installer or other
standards-based tools.

See https://github.com/pypa/setuptools/issues/917 for details.
*****
```

```
!!
self.initialize_options()
running bdist_egg
running egg_info
creating pycocoevalcap.egg-info
writing pycocoevalcap.egg-info/PKG-INFO
writing dependency_links to pycocoevalcap.egg-info/dependency_links.txt
writing requirements to pycocoevalcap.egg-info/requirements.txt
writing top-level names to pycocoevalcap.egg-info/top_level.txt
writing manifest file 'pycocoevalcap.egg-info/SOURCES.txt'
reading manifest file 'pycocoevalcap.egg-info/SOURCES.txt'
writing manifest file 'pycocoevalcap.egg-info/SOURCES.txt'
installing library code to build/bdist.linux-x86_64/egg
running install_lib
running build_py
creating build/lib/pycocoevalcap
copying ./eval.py -> build/lib/pycocoevalcap
creating build/lib/pycocoevalcap/cider
copying ./cider/cider.py -> build/lib/pycocoevalcap/cider
copying ./cider/cider_scorer.py -> build/lib/pycocoevalcap/cider
creating build/lib/pycocoevalcap/meteor
copying ./meteor/meteor.py -> build/lib/pycocoevalcap/meteor
creating build/lib/pycocoevalcap/tokenizer
copying ./tokenizer/ptbtokenizer.py -> build/lib/pycocoevalcap/tokenizer
creating build/lib/pycocoevalcap/example
copying ./example/coco_eval_example.py -> build/lib/pycocoevalcap/example
creating build/lib/pycocoevalcap/rouge
copying ./rouge/rouge.py -> build/lib/pycocoevalcap/rouge
creating build/lib/pycocoevalcap/bleu
copying ./bleu/bleu_scorer.py -> build/lib/pycocoevalcap/bleu
copying ./bleu/bleu.py -> build/lib/pycocoevalcap/bleu
```

```
# clone the dataset locally
%cd /content
!git clone -q --no-checkout --depth=1 --filter=tree:0 https://github.com/vis-nlp/Chart-to-text.git repo
%cd repo
!git sparse-checkout init --cone >/dev/null 2>&1
!git sparse-checkout set statista_dataset/dataset >/dev/null 2>&1
!git checkout -q
!ls statista_dataset/dataset
```

```
/content
/content/repo
remote: Enumerating objects: 128, done.
remote: Counting objects: 100% (128/128), done.
remote: Compressing objects: 100% (114/114), done.
remote: Total 128 (delta 5), reused 127 (delta 5), pack-reused 0 (from 0)
Receiving objects: 100% (128/128), 4.43 MiB | 19.62 MiB/s, done.
Resolving deltas: 100% (5/5), done.
remote: Enumerating objects: 139146, done.
remote: Counting objects: 100% (139146/139146), done.
remote: Compressing objects: 100% (136524/136524), done.
remote: Total 139146 (delta 1046), reused 139145 (delta 1046), pack-reused 0 (from 0)
Receiving objects: 100% (139146/139146), 1.23 GiB | 17.26 MiB/s, done.
Resolving deltas: 100% (1046/1046), done.
captions data dataset_split imgs metadata.csv multiColumn sta.txt titles
```

```
# clone ChartInsighter Benchmark locally
%cd /content/repo
!git clone https://github.com/wangfen01/ChartInsighter.git
!ls /ChartInsighter
```

```
/content/repo
Cloning into 'ChartInsighter'...
remote: Enumerating objects: 579, done.
remote: Counting objects: 100% (579/579), done.
remote: Compressing objects: 100% (515/515), done.
remote: Total 579 (delta 64), reused 563 (delta 61), pack-reused 0 (from 0)
Receiving objects: 100% (579/579), 23.32 MiB | 16.85 MiB/s, done.
Resolving deltas: 100% (64/64), done.
ls: cannot access '/ChartInsighter': No such file or directory
```

```

import csv
import inspect
import json
import os
import re
from pathlib import Path
import requests

# --- Third-Party Libraries ---
import numpy as np
import pandas as pd
import requests
import spacy
import torch
from nltk.translate.bleu_score import SmoothingFunction, corpus_bleu
from PIL import Image
from torch import amp
from rouge_score import rouge_scorer
from torch.optim import AdamW
from torch.utils.data import DataLoader, Dataset
from torch import amp
from torch.nn.utils import clip_grad_norm_
from tqdm.auto import tqdm
from tqdm import tqdm
from transformers import LlavaForConditionalGeneration, LlavaProcessor
from peft import LoraConfig, get_peft_model

```

```

MODEL_ID = "llava-hf/llava-1.5-7b-hf"

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
#device = torch.device("mps" if torch.backends.mps.is_available() else "cpu")
dtype = torch.bfloat16 # MPS prefers fp32

DEFAULT_DATASET_DIR = "/content/repo/statista_dataset/dataset"
nlp = spacy.load("en_core_web_sm")

captions = np.array(os.listdir(DEFAULT_DATASET_DIR+"/captions"))
images = np.array(os.listdir(DEFAULT_DATASET_DIR+"/imgs"))

CHARTINSIGHTER_BENCHMARK_DIR = "/content/repo/ChartInsighter"

captions_chart_complex = np.array(os.listdir(CHARTINSIGHTER_BENCHMARK_DIR+"/complex_chart/gold_summary"))
captions_chart_moderate = np.array(os.listdir(CHARTINSIGHTER_BENCHMARK_DIR+"/moderate_chart/gold_summary"))
captions_chart_simple = np.array(os.listdir(CHARTINSIGHTER_BENCHMARK_DIR+"/simple_chart/gold_summary"))

images_chart_complex = np.array(os.listdir(CHARTINSIGHTER_BENCHMARK_DIR+"/complex_chart/chart"))
images_chart_moderate = np.array(os.listdir(CHARTINSIGHTER_BENCHMARK_DIR+"/moderate_chart/chart"))
images_chart_simple = np.array(os.listdir(CHARTINSIGHTER_BENCHMARK_DIR+"/simple_chart/chart"))

```

```

filtered_dataset_url = "https://raw.githubusercontent.com/xingmingxu/ECS289A-final-project/main/filtered_dataset.json"
response = requests.get(filtered_dataset_url)

with open("filtered_dataset.json", "wb") as f:
    f.write(response.content)

print("File downloaded!")

# Load it
with open("filtered_dataset.json", "r") as f:
    data = json.load(f)

print("Number of items:", len(data))
print("First item:", data[0])

```

File downloaded!
Number of items: 976
First item: {'id': '1000', 'caption': 'This graph depicts the average ticket price in the NFL (National Football Conference) for the 2023 season. The x-axis represents the team, and the y-axis represents the average ticket price in dollars. The chart shows that the average ticket price varies significantly between teams, with some teams like the New England Patriots and the Green Bay Packers having higher average ticket prices compared to others.'}

```

# From a list of sample names
# unique_names = np.array(list(unique_caption_names))
unique_names = np.array([d["id"] for d in data])

```

```
np.random.seed(42)
indices = np.random.permutation(len(unique_names))

idx = len(unique_names) // 5

tst_idx = indices[:idx]#.tolist()
val_idx = indices[idx:idx*2]#.tolist()
trn_idx = indices[idx*2:]#.tolist()
```

▼ Training Loop

```
MODEL_ID = "llava-hf/llava-1.5-7b-hf"

processor = LlavaProcessor.from_pretrained(
    MODEL_ID,
    trust_remote_code=True,
    use_fast=False,
)

model = LlavaForConditionalGeneration.from_pretrained(
    MODEL_ID,
    torch_dtype=torch.bfloat16,
    low_cpu_mem_usage=True,
    device_map="cuda",
    trust_remote_code=True,
)

model.gradient_checkpointing_enable()
model.config.use_cache = False

BATCH_SIZE=1

# --- LoRA config ---
lora_config = LoraConfig(
    r=8,
    lora_alpha=16,
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM",
    # standard LLaMA-style target modules
    target_modules=[
        "q_proj", "k_proj", "v_proj", "o_proj",
        "gate_proj", "up_proj", "down_proj",
    ],
)
model = get_peft_model(model, lora_config)
model.print_trainable_parameters()
```

```
/usr/local/lib/python3.12/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://huggingface.co/settings/token)
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 models or datasets.
  warnings.warn(
preprocessor_config.json: 100%                                         505/505 [00:00<00:00, 44.1kB/s]
processor_config.json: 100%                                         173/173 [00:00<00:00, 20.6kB/s]
tokenizer_config.json: 1.45k/? [00:00<00:00, 182kB/s]
tokenizer.model: 100%                                         500k/500k [00:01<00:00, 389kB/s]
added_tokens.json: 100%                                         41.0/41.0 [00:00<00:00, 5.13kB/s]
special_tokens_map.json: 100%                                         552/552 [00:00<00:00, 73.4kB/s]
tokenizer.json: 3.62M/? [00:00<00:00, 126MB/s]
chat_template.jinja: 100%                                         674/674 [00:00<00:00, 93.5kB/s]
chat_template.json: 100%                                         701/701 [00:00<00:00, 94.8kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
config.json: 100%                                         950/950 [00:00<00:00, 134kB/s]
model.safetensors.index.json: 70.1k/? [00:00<00:00, 8.05MB/s]
Fetching 3 files: 100%                                         3/3 [00:41<00:00, 41.57s/it]
model-00003-of-00003.safetensors: 100%                                         4.18G/4.18G [00:19<00:00, 44.7MB/s]
model-00001-of-00003.safetensors: 100%                                         4.99G/4.99G [00:41<00:00, 274MB/s]
model-00002-of-00003.safetensors: 100%                                         4.96G/4.96G [00:36<00:00, 41.2MB/s]
Loading checkpoint shards: 100%                                         3/3 [00:08<00:00, 2.72s/it]
generation_config.json: 100%                                         141/141 [00:00<00:00, 20.3kB/s]
trainable params: 21,168,128 || all params: 7,084,595,200 || trainable%: 0.2988
```

```
class StatistaChartToText(Dataset):
    def __init__(self, fnames, split_idx,
                 base_dir=DEFAULT_DATASET_DIR,
                 # image_transform=None, text_transform=None
                 ):
        # Load indices from dataset
        self.fnames = fnames[split_idx]
        self.base = base_dir

        self.image_dir = os.path.join(self.base, "imgs")
        self.caption_dir = os.path.join(self.base, "captions")

        # self.image_transform = image_transform
        # self.text_transform = text_transform

        if len(self.fnames) == 1:
            fname = self.fnames[0]
            image_path = os.path.join(self.image_dir, f"{fname}.png")
            caption_path = os.path.join(self.caption_dir, f"{fname}.txt")
            print("Example paths:")
            print(" image:", image_path)
            print(" caption:", caption_path)

    def __len__(self):
        # We're supposed to implement __len__
        return len(self.fnames)

    def __getitem__(self, index):
        # Loads caption, image pair ON DEMAND
        fname = self.fnames[index]
        image_path = os.path.join(self.image_dir, f"{fname}.png")
        caption_path = os.path.join(self.caption_dir, f"{fname}.txt")
```

```

image = Image.open(image_path).convert("RGB")
with open(caption_path, "r", encoding="utf-8") as f:
    caption = f.read().strip()

return {
    "image": image,
    "caption": caption,
    "id": fname
}

from torch.utils.data import Dataset
from PIL import Image
import os
import numpy as np

class ChartInsighterDataset(Dataset):

    def __init__(self, difficulty="complex", fnames=None, base_dir=CHARTINSIGHTER_BENCHMARK_DIR):
        """
        difficulty: "complex", "moderate", or "simple"
        fnames: list of file IDs (without extension), optional; if None, load all
        base_dir: root directory of ChartInsighter benchmark
        """
        self.base_dir = base_dir
        self.difficulty = difficulty

        self.image_dir = os.path.join(base_dir, f"{difficulty}_chart", "chart")
        self.caption_dir = os.path.join(base_dir, f"{difficulty}_chart", "gold_summary")

        if fnames is None:
            self.fnames = [os.path.splitext(f)[0] for f in os.listdir(self.caption_dir)]
        else:
            self.fnames = fnames

        if len(self.fnames) > 0:
            print(f"Example paths for {difficulty} charts:")
            print(" image:", os.path.join(self.image_dir, f"{self.fnames[0]}.png"))
            print(" caption:", os.path.join(self.caption_dir, f"{self.fnames[0]}.txt"))

    def __len__(self):
        # We're supposed to implement __len__
        return len(self.fnames)

    def __getitem__(self, index):

        # Loads caption, image pair ON DEMAND

        fname = self.fnames[index]
        image_path = os.path.join(self.image_dir, f"{fname}.png")
        caption_path = os.path.join(self.caption_dir, f"{fname}.txt")

        image = Image.open(image_path).convert("RGB")
        with open(caption_path, "r", encoding="utf-8") as f:
            caption = f.read().strip()

        return {
            "image": image,
            "caption": caption,
            "id": fname
        }

def collate_fn(examples):
    examples = [ex for ex in examples if ex is not None]
    if len(examples) == 0:
        return None

    images = [ex["image"] for ex in examples]
    captions = [ex["caption"] for ex in examples]

    # Build conversations: user (image + prompt) + assistant (caption)
    conversations = []
    for cap in captions:
        conversations.append([

```

```

    {
        "role": "user",
        "content": [
            {"type": "image"},
            {
                "type": "text",
                "text": (
                    "You are a helpful data analyst. Carefully examine the "
                    "chart and provide a clear, detailed natural language "
                    "description, including axes, units, main trends, and any "
                    "notable patterns or comparisons."
                ),
            },
        ],
    },
    {
        "role": "assistant",
        "content": [{"type": "text", "text": cap}],
    },
)
])
```

1) Turn conversations into plain text with the correct chat template

```

texts = [
    processor.apply_chat_template(
        conv,
        add_generation_prompt=False, # full conversation = user + assistant
        tokenize=False,
    ).strip()
    for conv in conversations
]
```

2) Tokenize text + images

```

batch = processor(
    text=texts,
    images=images,
    padding=True,
    return_tensors="pt",
)
```

3) Build labels from input_ids

```

labels = batch["input_ids"].clone()
```

--- Mask special tokens exactly like the TRL example ---

(a) mask padding

```

pad_id = processor.tokenizer.pad_token_id
if pad_id is not None:
    labels[labels == pad_id] = -100
```

(b) mask image tokens (boi_token)

```

# boi_token is the "begin-of-image" special token used by LLaVA
boi_token = processor.tokenizer.special_tokens_map.get("boi_token", None)
if boi_token is not None:
    image_token_id = processor.tokenizer.convert_tokens_to_ids(boi_token)
    labels[labels == image_token_id] = -100
```

(c) mask image soft tokens (used in some VLMs; TRL example does this)

```

labels[labels == 262144] = -100 # safe even if not present
```

(d) OPTIONAL: mask user tokens (only learn to predict assistant side)

```

# Instead of looking for "<assistant>", search for the string "ASSISTANT:"
# in *text space* and then map to token indices. Easiest is: skip (d) for now.
```

```

batch["labels"] = labels
```

```

return batch
```

Set up datasets, data loaders, run the model

```

train_dataset = StatistaChartToText(unique_names, trn_idx, base_dir=DEFAULT_DATASET_DIR)
train_loader = DataLoader(
    train_dataset,
    batch_size=BATCH_SIZE,
    shuffle=True,
```

```

        num_workers=4,
        collate_fn=collate_fn, # important
    )

test_dataset = StatistaChartToText(unique_names, tst_idx, base_dir=DEFAULT_DATASET_DIR)
test_loader = DataLoader(
    test_dataset,
    batch_size=BATCH_SIZE,
    shuffle=True,
    num_workers=4,
    collate_fn=collate_fn, # important
)

val_dataset = StatistaChartToText(unique_names, val_idx, base_dir=DEFAULT_DATASET_DIR)
val_loader = DataLoader(
    val_dataset,
    batch_size=BATCH_SIZE,
    shuffle=True,
    num_workers=4,
    collate_fn=collate_fn, # important
)

val_dataset_char_complex = ChartInsighterDataset(difficulty="complex")
val_dataset_char_moderate = ChartInsighterDataset(difficulty="moderate")
val_dataset_char_simple = ChartInsighterDataset(difficulty="simple")
val_loader_complex = DataLoader(
    val_dataset_char_complex,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=2,
    collate_fn=collate_fn
)

val_loader_moderate = DataLoader(
    val_dataset_char_moderate,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=2,
    collate_fn=collate_fn
)

val_loader_simple = DataLoader(
    val_dataset_char_simple,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=2,
    collate_fn=collate_fn
)

```

Example paths for complex charts:
image: /content/repo/ChartInsighter/complex_chart/chart/11.png
caption: /content/repo/ChartInsighter/complex_chart/gold_summary/11.txt

Example paths for moderate charts:
image: /content/repo/ChartInsighter/moderate_chart/chart/11.png
caption: /content/repo/ChartInsighter/moderate_chart/gold_summary/11.txt

Example paths for simple charts:
image: /content/repo/ChartInsighter/simple_chart/chart/11.png
caption: /content/repo/ChartInsighter/simple_chart/gold_summary/11.txt

```

num_epochs = 5
max_grad_norm = 1.0
learning_rate = 1e-6

optimizer = AdamW(model.parameters(), lr=learning_rate)

for epoch in range(num_epochs):
    model.train()
    epoch_loss = 0.0
    num_steps = 0

    pbar = tqdm(train_loader, desc=f"Epoch {epoch+1}/{num_epochs}")

    for step, batch in enumerate(pbar):
        # collate_fn can return None if it drops a batch
        if batch is None:
            continue

        batch = {k: v.to(device, non_blocking=True) for k, v in batch.items()}

```

```

optimizer.zero_grad(set_to_none=True)

# to support bfloat16, helps w memory
with amp.autocast("cuda", dtype=torch.bfloat16):
    outputs = model(**batch)
    loss = outputs.loss

# print(f"[Epoch {epoch+1} Step {step}] loss = {loss.item():.4f}")

# backward in full precision
loss.backward()
clip_grad_norm_(model.parameters(), max_grad_norm)

optimizer.step()

# tracking
epoch_loss += loss.item()
num_steps += 1

pbar.set_postfix({"loss": f"{loss.item():.4f}"})

if num_steps > 0:
    avg_loss = epoch_loss / num_steps
else:
    avg_loss = float("nan")

print(f"Epoch {epoch+1}/{num_epochs} finished. Avg loss: {avg_loss:.4f}")

```

```

Epoch 1/5: 100%                                     586/586 [04:13<00:00, 2.38it/s, loss=5.6553]
/usr/local/lib/python3.12/dist-packages/torch/utils/checkpoint.py:85: UserWarning: None of the inputs have requires_g
  warnings.warn(
`use_cache=True` is incompatible with gradient checkpointing. Setting `use_cache=False`.
Epoch 1/5 finished. Avg loss: 9.1079

Epoch 2/5: 100%                                     586/586 [04:09<00:00, 2.38it/s, loss=3.7748]
Epoch 2/5 finished. Avg loss: 4.3222

Epoch 3/5: 100%                                     586/586 [04:52<00:00, 2.35it/s, loss=3.7191]
Epoch 3/5 finished. Avg loss: 3.8534

Epoch 4/5: 100%                                     586/586 [04:11<00:00, 2.38it/s, loss=3.5315]
Epoch 4/5 finished. Avg loss: 3.6288

Epoch 5/5: 100%                                     586/586 [04:50<00:00, 2.35it/s, loss=3.5433]
Epoch 5/5 finished. Avg loss: 3.5371

```

```

def simple_overlap_score(pred: str, gt: str):
    """
    Very cheap lexical F1-style overlap between prediction and ground truth.
    Not a "real" metric, but useful to see if training is doing anything.
    """
    pred_tokens = set(pred.lower().split())
    gt_tokens = set(gt.lower().split())

    if not pred_tokens or not gt_tokens:
        return 0.0, 0.0, 0.0

    inter = len(pred_tokens & gt_tokens)
    prec = inter / len(pred_tokens)
    rec = inter / len(gt_tokens)
    if prec + rec == 0:
        f1 = 0.0
    else:
        f1 = 2 * prec * rec / (prec + rec)
    return prec, rec, f1

def extract_key_info(text):
    """
    Input:
        text: summary of table
    Output:
        key_info: list of entities and numbers as strings
    """
    doc = nlp(text)
    # extract entities
    entities = [ent.text for ent in doc.ents]

```

```

# extract numbers
pattern = re.compile(r"\d+\.?\d*")
numbers = [match.group() for match in pattern.finditer(text)]
# generate the key information
key_info = list(set(entities + numbers))
return key_info

def cs_compare(pred: str, gt: str):
    """
    Input:
        pred: predicted output, string
        gt: ground truth, string
        generated summary
    Output:
        dict with precision and recall
    """
    # get the keywords list
    gt_list = extract_key_info(gt)
    pred_list = extract_key_info(pred)

    gt_set = set(gt_list)
    pred_set = set(pred_list)

    precision = len(gt_set & pred_set) / len(pred_set) if len(pred_set)>0 else 0.0
    recall = len(gt_set & pred_set) / len(gt_set) if len(gt_set)>0 else 0.0
    if precision + recall > 0:
        f1 = 2 * precision * recall / (precision + recall)
    else:
        f1 = 0

    return precision, recall, f1

def BLEU_compare(pred: str, gt: str):
    doc1 = nlp(gt)
    doc2 = nlp(pred)

    gt_sentences = [sent.text.lower().split() for sent in doc1.sents]
    pred_sentences = [sent.text.lower().split() for sent in doc2.sents]

    n = max(len(gt_sentences), len(pred_sentences))
    while len(gt_sentences) < n:
        gt_sentences.append(gt_sentences[-1])
    while len(pred_sentences) < n:
        pred_sentences.append([])

    list_of_references = [[ref] for ref in gt_sentences]

    # print(list_of_references)
    bleu_score = corpus_bleu(
        list_of_references,
        pred_sentences,
        smoothing_function=SmoothingFunction().method1
    )
    return bleu_score

def ROUGE_L_compare(pred: str, gt: str):
    scorer = rouge_scorer.RougeScorer(['rougeL'], use_stemmer=True)
    scores = scorer.score(gt, pred)
    return scores

def evaluate_chart_descriptions(
    model,
    dataset,
    processor,
    device,
    num_samples: int = 16,
    max_new_tokens: int = 128,
    print_samples: int = 5,
):
    """
    Evaluate the model on a (subset of) dataset by generating a description
    for each chart and comparing with the ground-truth caption.

    Returns:
        results: list of dicts with
            - id
    """

```

```

        - gt_caption
        - pred_caption
        - precision / recall / f1 (lexical overlap)
    ....
model.eval()

# Choose which indices to eval
n = len(dataset)
if num_samples is None or num_samples > n:
    indices = list(range(n))
else:
    indices = list(range(num_samples))

results = []

for i, idx in enumerate(indices):
    sample = dataset[idx]
    image = sample["image"]
    gt_caption = sample["caption"]
    ex_id = sample.get("id", idx)

    # Same style of USER message as training
    conversation = [
        {
            "role": "user",
            "content": [
                {"type": "image"},
                {
                    "type": "text",
                    "text": (
                        "You are a helpful data analyst. Carefully examine the "
                        "chart and provide a clear, detailed natural language "
                        "description, including axes, units, main trends, and any "
                        "notable patterns or comparisons."
                    ),
                },
            ],
        },
    ],
]

# Build prompt with generation slot for assistant
prompt = processor.apply_chat_template(
    conversation,
    add_generation_prompt=True, # let model continue as assistant
    tokenize=False,
)

# Tokenize + add image
inputs = processor(
    text=prompt,
    images=image,
    return_tensors="pt",
    padding=True,
).to(device)

input_ids = inputs["input_ids"]

with torch.no_grad():
    with amp.autocast("cuda", dtype=torch.bfloat16):
        gen_ids = model.generate(
            **inputs,
            max_new_tokens=max_new_tokens,
            do_sample=False, # deterministic for eval
            eos_token_id=processor.tokenizer.eos_token_id,
            pad_token_id=processor.tokenizer.eos_token_id,
        )

# ADD OTHER METRICS HERE

# Only keep newly generated tokens (strip the prompt part)
gen_only = gen_ids[0, input_ids.shape[1]:]

pred_text = processor.tokenizer.decode(
    gen_only,
    skip_special_tokens=True,
).strip()

```

```

# cheap cleanup
pred_text = pred_text.replace("<image>", "").strip()

# simple lexical overlap metric
prec, rec, f1 = simple_overlap_score(pred_text, gt_caption)

# cs_score
prec_cs, rec_cs, f1_cs = cs_compare(pred_text, gt_caption)

# BLEU
bleu = BLEU_compare(pred_text, gt_caption)

# ROUGE-L
rouge_scores = ROUGE_L_compare(pred_text, gt_caption)

result = {
    "id": ex_id,
    "gt_caption": gt_caption,
    "pred_caption": pred_text,
    "precision": prec,
    "recall": rec,
    "f1": f1,
    "precision_cs": prec_cs,
    "recall_cs": rec_cs,
    "f1_cs": f1_cs,
    "bleu_score": bleu,
    "rouge_score": rouge_scores,
    # stick the other results here
}
results.append(result)

# ADD OTHER METRICS ABOVE, SAVE THEM TO RESULT

# Optionally print a few examples
if i < print_samples:
    print("=" * 80)
    print(f"Example {i+1} / {len(indices)} | ID: {ex_id}")
    print("\n==== MODEL OUTPUT ===")
    print(pred_text if pred_text else "[NO NEW TEXT GENERATED]")
    print("\n==== GROUND TRUTH ===")
    print(gt_caption)
    print(f"\nOverlap: Precision={prec:.3f}, Recall={rec:.3f}, F1={f1:.3f}")
    print(f"\nCS score: Precision={prec_cs:.3f}, Recall={rec_cs:.3f}, F1={f1_cs:.3f}")
    print(f"\nBLEU: {bleu:.3f}")
    print(f"\nROUGE-L: {rouge_scores['rougel'].fmeasure:.3f}")
    print("=" * 80)

# Aggregate metric over all evaluated samples
if results:
    avg_f1 = sum(r["f1"] for r in results) / len(results)
    avg_f1_cs = sum(r["f1_cs"] for r in results) / len(results)
    avg_bleu = sum(r["bleu_score"] for r in results) / len(results)
    avg_rouge = sum(r["rouge_score"]["rougel"].fmeasure for r in results) / len(results)
    print(f"\n[Eval] Average F1 over {len(results)} samples: {avg_f1:.4f}")
    print(f"\n[Eval] Average F1 CS over {len(results)} samples: {avg_f1_cs:.4f}")
    print(f"\n[Eval] Average BLEU Score over {len(results)} samples: {avg_bleu:.4f}")
    print(f"\n[Eval] Average ROUGE-L Score over {len(results)} samples: {avg_rouge:.4f}")
else:
    print("[Eval] No samples evaluated.")

return results

def evaluate_charInsighter_on_metrics(diff):
    ci_dir = Path("/content/repo/ChartInsighter") / diff / "ChartInsighter"
    gold_dir = Path("/content/repo/ChartInsighter") / diff / "gold_summary"

    ci_files = sorted([f for f in ci_dir.glob("*.txt")])
    gold_files = sorted([f for f in gold_dir.glob("*.txt")])

    results = []

    for gold_file in gold_files:
        fname = gold_file.stem
        ci_file = ci_dir / f"{fname}.txt"

        if not ci_file.exists():

```

```

print(f"Missing ChartInsighter caption for {fname}")
continue

with open(gold_file, "r", encoding="utf-8") as f:
    gold_text = f.read().strip()

with open(ci_file, "r", encoding="utf-8") as f:
    ci_text = f.read().strip()

simple_prec, simple_rec, simple_f1 = simple_overlap_score(ci_text, gold_text)
cs_prec, cs_rec, cs_f1 = cs_compare(ci_text, gold_text)
bleu = BLEU_compare(ci_text, gold_text)
rouge = ROUGE_L_compare(ci_text, gold_text)['rougeL'].fmeasure

results.append({
    "file": fname,
    "f1": simple_f1,
    "f1_cs": cs_f1,
    "bleu_score": bleu,
    "rouge_score": rouge
})

if results:
    avg_f1 = sum(r["f1"] for r in results) / len(results)
    avg_f1_cs = sum(r["f1_cs"] for r in results) / len(results)
    avg_bleu = sum(r["bleu_score"] for r in results) / len(results)
    avg_rouge = sum(r["rouge_score"] for r in results) / len(results)
    print(f"\n[Eval] Average F1 over {len(results)} samples: {avg_f1:.4f}")
    print(f"\n[Eval] Average F1 CS over {len(results)} samples: {avg_f1_cs:.4f}")
    print(f"\n[Eval] Average BLEU Score over {len(results)} samples: {avg_bleu:.4f}")
    print(f"\n[Eval] Average ROUGE-L Score over {len(results)} samples: {avg_rouge:.4f}")
else:
    print("[Eval] No samples evaluated.")

return results

```

Evaluations

We evaluate the naive method, our method, and ChartInsighter below.

```

# Load the original model fresh (no finetune)
base_model = LlavaForConditionalGeneration.from_pretrained(
    MODEL_ID,
    torch_dtype=torch.bfloat16,
    low_cpu_mem_usage=True,
    device_map="cuda",
    trust_remote_code=True,
)

print("Evaluating BASE MODEL")
base_results = evaluate_chart_descriptions(
    base_model,
    val_dataset,
    processor,
    device,
    num_samples=len(test_dataset),
    max_new_tokens=128,
    print_samples=3,
)

```

```

Loading checkpoint shards: 100%                                         3/3 [00:08<00:00, 2.69s/it]
Evaluating BASE MODEL
=====
Example 1 / 195 | ID: 9849

==== MODEL OUTPUT ===
The image displays a bar chart with a list of states in the United States. The chart is divided into two main sections. The chart is organized in a way that allows for easy comparison between the states. The values are displayed in a clear and concise manner, making it easy to interpret the data.

==== GROUND TRUTH ===
This statistic depicts the number of liver transplant in the U.S. in 2019 , by state . According to the data , there were approximately 10,000 liver transplants performed in 2019. The chart shows the top 10 states with the highest number of transplants.

Overlap: Precision=0.088, Recall=0.185, F1=0.119
cs score: Precision=0.000, Recall=0.000, F1=0.000
BLEU: 0.007
ROUGE-L: 0.144
=====

Example 2 / 195 | ID: 27218

==== MODEL OUTPUT ===
The chart displays a variety of clothing and accessory items, with each item represented by a different color. The items are arranged in a grid-like pattern, allowing for easy comparison between the different categories.

The chart is divided into three sections: the left section, which includes clothing items such as shoes, cosmetics, and accessories; the middle section, which includes apparel items such as shirts, pants, and jackets; and the right section, which includes accessories such as hats, bags, and shoes.

==== GROUND TRUTH ===
This statistic depicts the net sales share of Dillard 's worldwide in 2019 . In 2019 , the net sales share of Dillard was approximately 5.5%. The chart shows the top 10 brands with the highest net sales share.

Overlap: Precision=0.055, Recall=0.150, F1=0.080
cs score: Precision=0.000, Recall=0.000, F1=0.000
BLEU: 0.003
ROUGE-L: 0.103
=====

Example 3 / 195 | ID: 25895

==== MODEL OUTPUT ===
The image displays a bar chart that shows the number of committed crimes in the United States. The chart is divided into several categories, including violent crimes, property crimes, and other crimes.

The bar chart shows that property crime is the most common type of crime, with 1,391,000 cases reported in 2019. Violent crimes account for the second largest number of cases, followed by other crimes.

==== GROUND TRUTH ===
This statistic shows data on crime in U.S. cities outside metropolitan areas by the type of crime committed in 2019 . The chart shows the top 10 types of crime committed in 2019, with property crime being the most common.

Overlap: Precision=0.191, Recall=0.321, F1=0.240
cs score: Precision=0.083, Recall=0.250, F1=0.125
BLEU: 0.003
ROUGE-L: 0.200

print("Evaluating LORA MODEL")
base_results = evaluate_chart_descriptions(
    model,
    val_dataset,
    processor,
    device,
    num_samples=len(test_dataset),
    max_new_tokens=128,
    print_samples=3,
)

==== MODEL OUTPUT ===
This statistic shows the number of deaths in the United States in 2017, by state . In 2017 , the state with the highest number of deaths was California, with approximately 100,000 deaths. The chart shows the top 10 states with the highest number of deaths.

==== GROUND TRUTH ===
This statistic depicts the number of liver transplant in the U.S. in 2019 , by state . According to the data , there were approximately 10,000 liver transplants performed in 2019. The chart shows the top 10 states with the highest number of transplants.

Overlap: Precision=0.385, Recall=0.370, F1=0.377

```

BLEU: 0.047

ROUGE-L: 0.282

=====

Example 2 / 195 | ID: 27218

==== MODEL OUTPUT ===

This statistic shows the most popular clothing items among women in the United States as of 2018 . The most popular

==== GROUND TRUTH ===

This statistic depicts the net sales share of Dillard 's worldwide in 2019 . In 2019 , the net sales share of Dillar

Overlap: Precision=0.270, Recall=0.500, F1=0.351

cs score: Precision=0.000, Recall=0.000, F1=0.000

BLEU: 0.012

ROUGE-L: 0.194

=====

Example 3 / 195 | ID: 25895

==== MODEL OUTPUT ===

This statistic shows the number of violent crime incidents in the United States in 2019, by type of crime . In 2019

==== GROUND TRUTH ===

This statistic shows data on crime in U.S. cities outside metropolitan areas by the type of crime committed in 2019

Overlap: Precision=0.424, Recall=0.500, F1=0.459

cs score: Precision=0.167, Recall=0.250, F1=0.200

BLEU: 0.050

ROUGE-L: 0.368

=====

[Eval] Average F1 over 195 samples: 0.3176

[Eval] Average F1 CS over 195 samples: 0.0799

[Eval] Average BLEU Score over 195 samples: 0.0406

[Eval] Average ROUGE-L Score over 195 samples: 0.2255

```

print("Evaluating BASE MODEL on ChartInsighter Benchmark")
print("Evaluating COMPLEX charts")
complex_results = evaluate_chart_descriptions(
    base_model,
    val_dataset_char_complex,
    processor,
    device,
    num_samples=len(val_dataset_char_complex),
    max_new_tokens=128,
    print_samples=3
)
print("Evaluating MODERATE charts")
moderate_results = evaluate_chart_descriptions(
    base_model,
    val_dataset_char_moderate,
    processor,
    device,
    num_samples=len(val_dataset_char_moderate),
    max_new_tokens=128,
    print_samples=3
)
print("Evaluating SIMPLE charts")
simple_results = evaluate_chart_descriptions(
    base_model,
    val_dataset_char_simple,
    processor,
    device,
    num_samples=len(val_dataset_char_simple),
    max_new_tokens=128,
    print_samples=3
)

```

[Show hidden output](#)

```

print("Evaluating LORA MODEL on ChartInsighter Benchmark")
print("Evaluating COMPLEX charts")
complex_results = evaluate_chart_descriptions(
    model,
    val_dataset_char_complex,
    processor,
    device,
    num_samples=len(val_dataset_char_complex),
    max_new_tokens=128,
    print_samples=3
)
print("Evaluating MODERATE charts")
moderate_results = evaluate_chart_descriptions(
    model,
    val_dataset_char_moderate,
    processor,
    device,
    num_samples=len(val_dataset_char_moderate),
    max_new_tokens=128,
    print_samples=3
)
print("Evaluating SIMPLE charts")
simple_results = evaluate_chart_descriptions(
    model,
    val_dataset_char_simple,
    processor,
    device,
    num_samples=len(val_dataset_char_simple),
    max_new_tokens=128,
    print_samples=3
)

```

[Show hidden output](#)

```

print("Apply Our Metrics on ChartInsighter Benchmark")
difficulties = ["complex_chart", "moderate_chart", "simple_chart"]

all_results = {}

for diff in difficulties:
    print(f"\n==== Evaluating ChartInsighter Benchmark: {diff.upper()} ===")
    results = evaluate_charInsighter_on_metrics(diff)
    all_results[diff] = results

print("\n==== Summary of All Difficulties ===")
for diff in difficulties:
    res = all_results[diff]
    if res:
        avg_f1 = sum(r["f1"] for r in res) / len(res)
        avg_f1_cs = sum(r["f1_cs"] for r in res) / len(res)
        avg_bleu = sum(r["bleu_score"] for r in res) / len(res)
        avg_rouge = sum(r["rouge_score"] for r in res) / len(res)
        print(f"\n{diff.upper()} ({len(res)} samples):")
        print(f" Average F1 for All : {avg_f1:.4f}")
        print(f" Average F1 CS for All : {avg_f1_cs:.4f}")
        print(f" Average BLEU Score for All : {avg_bleu:.4f}")
        print(f" Average ROUGE-L Score for All : {avg_rouge:.4f}")
    else:
        print(f"\n{diff.upper()}: No samples evaluated.")

```

Apply Our Metrics on ChartInsighter Benchmark

```

==== Evaluating ChartInsighter Benchmark: COMPLEX_CHART ===

[Eval] Average F1 over 25 samples: 0.3140
[Eval] Average F1 CS over 25 samples: 0.3186
[Eval] Average BLEU Score over 25 samples: 0.0220
[Eval] Average ROUGE-L Score over 25 samples: 0.2260

==== Evaluating ChartInsighter Benchmark: MODERATE_CHART ===

[Eval] Average F1 over 25 samples: 0.2906
[Eval] Average F1 CS over 25 samples: 0.3078

```

```
[Eval] Average BLEU Score over 25 samples: 0.0240
[Eval] Average ROUGE-L Score over 25 samples: 0.2189
== Evaluating ChartInsighter Benchmark: SIMPLE_CHART ==
[Eval] Average F1 over 25 samples: 0.2808
[Eval] Average F1 CS over 25 samples: 0.3383
[Eval] Average BLEU Score over 25 samples: 0.0157
[Eval] Average ROUGE-L Score over 25 samples: 0.2284
== Summary of All Difficulties ==
COMPLEX_CHART (25 samples):
  Average F1 for All : 0.3140
  Average F1 CS for All : 0.3186
  Average BLEU Score for All : 0.0220
  Average ROUGE-L Score for All : 0.2260
MODERATE_CHART (25 samples):
  Average F1 for All : 0.2906
  Average F1 CS for All : 0.3078
  Average BLEU Score for All : 0.0240
  Average ROUGE-L Score for All : 0.2189
```