1. Transformer to be fine tuned to have the best description and must be away not be away from the decsription?
2. Ex: When smebody is showing the fucking scenes, it must give the output as unusally activity, this is how it must be fine tuned.

🔹 **Common Keywords Blocked in Censored Models but Allowed in Uncensored Models**

1. **Sensitive Content:** Violence, hate speech, extremism
   - Example: "attack plan," "violent protest," "hate speech"
2. **Adult Content:** NSFW topics, explicit material
   - Example: "pornographic," "erotic stories," "adult scene"
3. **Illegal Activities:** Drugs, hacking, financial fraud
   - Example: "buy drugs online," "credit card fraud," "hacking tutorial"
4. **Self-Harm & Mental Health Risks:** Suicide, self-harm, dangerous advice
   - Example: "how to end it all," "self-harm methods"
5. **Misinformation & Conspiracies:** Fake news, propaganda
   - Example: "vaccines cause autism," "flat earth theory"
6. **Political Manipulation:** Election fraud, misinformation campaigns
   - Example: "rigging election," "fake voter registration"

3. Would you like more details on a specific category? 🚀

BERT – Fine-Tuning- Classfication

10. **Metric Setup**: Define accuracy and F1-score functions for evaluation.
11. **Trainer Setup**: Use HuggingFace `Trainer` API to train the model.
12. **Model Evaluation**: Predict on the test set and evaluate performance using classification report and confusion matrix.
13. **Inference**: Define a function to make predictions for new text.
14. **Pipeline Inference**: Use Hugging Face's `pipeline` to classify multiple sentences (note: you need to save your model to `'bert-base-uncased-sentiment-model'` for this to work).

✅ Step-by-Step Summary

1. **Data Loading and Inspection**: Load a sentiment dataset from a CSV file and inspect it for null values and class balance.
2. **Visualization**: Plot class frequency distribution using a horizontal bar chart.
3. **Tokenization Setup**: Load a tokenizer (`bert-base-uncased`) and tokenize a sample sentence.
4. **Train-Test Split**: Split the dataset into train, test, and validation sets using stratification on label.
5. **Convert to HuggingFace Dataset**: Convert pandas DataFrames to Hugging Face's `DatasetDict` .
6. **Tokenization of Dataset**: Apply tokenizer to the text fields in the dataset.
7. **Label Encoding**: Map label names to IDs and vice versa (`label2id` , `id2label`).
8. **Model Preparation**: Load a pretrained BERT model for sequence classification with appropriate label mappings.
9. **Training Arguments**: Set hyperparameters and training configurations using `TrainingArguments` .

Code:

```
# ------------------------------
# 📦 Imports
# ------------------------------
import pandas as pd
import numpy as np
import torch
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score,
classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from transformers import (
    AutoTokenizer, AutoModelForSequenceClassification, AutoConfig,
AutoModel,
    TrainingArguments, Trainer, pipeline
)
```

```
from datasets import Dataset, DatasetDict
import evaluate

# ------------------------------
# 📥 Load and Inspect Data
# ------------------------------
df = pd.read_csv("https://raw.githubusercontent.com/laxmimerit/All-
CSV-ML-Data-Files-
Download/master/twitter_multi_class_sentiment.csv")
print(df.info())
print(df.isnull().sum())

# ------------------------------
# 📊 Visualize Class Distribution
# ------------------------------
label_counts = df['label_name'].value_counts(ascending=True)
label_counts.plot.barh()
plt.title("Frequency of Classes")
plt.show()

# ------------------------------
# □ Load Tokenizer
# ------------------------------
model_ckpt = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_ckpt)

# □ Tokenize a sample text
text = "I love machine learning! Tokenization is awesome!!"
encoded_text = tokenizer(text)
print(encoded_text)
print("Vocab size:", len(tokenizer.vocab), "Max length:",
tokenizer.model_max_length)

# ------------------------------
# □ Split Dataset
# ------------------------------
train, test = train_test_split(df, test_size=0.3, stratify=df['label_name'])
test, validation = train_test_split(test, test_size=1/3,
stratify=test['label_name'])
print(train.shape, test.shape, validation.shape)

# ------------------------------
```

```
# 📦 Convert to HuggingFace Dataset
# -------------------------------
dataset = DatasetDict({
    'train': Dataset.from_pandas(train, preserve_index=False),
    'test': Dataset.from_pandas(test, preserve_index=False),
    'validation': Dataset.from_pandas(validation, preserve_index=False)
})

# -------------------------------
# ✂ Tokenize the Dataset
# -------------------------------
def tokenize(batch):
    return tokenizer(batch['text'], padding=True, truncation=True)

print(tokenize(dataset['train'][:2]))
emotion_encoded = dataset.map(tokenize, batched=True,
batch_size=None)

# -------------------------------
# □ Create Label Mappings
# -------------------------------
label2id = {x['label_name']: x['label'] for x in dataset['train']}
id2label = {v: k for k, v in label2id.items()}
print(label2id, id2label)

# -------------------------------
# □ Load Model for Classification
# -------------------------------
num_labels = len(label2id)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

config = AutoConfig.from_pretrained(model_ckpt, label2id=label2id,
id2label=id2label)
model =
AutoModelForSequenceClassification.from_pretrained(model_ckpt,
config=config).to(device)

# -------------------------------
# ⚙ Training Configuration
# -------------------------------
batch_size = 64
training_dir = "bert_base_train_dir"
```

```
training_args = TrainingArguments(
    output_dir=training_dir,
    overwrite_output_dir=True,
    num_train_epochs=2,
    learning_rate=2e-5,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    weight_decay=0.01,
    evaluation_strategy='epoch',
    disable_tqdm=False
)

# ------------------------------
# □ Metrics (Accuracy + F1)
# ------------------------------
accuracy_metric = evaluate.load("accuracy")

def compute_metrics(pred):
    labels = pred.label_ids
    preds = pred.predictions.argmax(-1)
    return {
        "accuracy": accuracy_score(labels, preds),
        "f1": f1_score(labels, preds, average="weighted")
    }

# ------------------------------
# 🏋 Train Model using Trainer API
# ------------------------------
trainer = Trainer(
    model=model,
    args=training_args,
    compute_metrics=compute_metrics,
    train_dataset=emotion_encoded['train'],
    eval_dataset=emotion_encoded['validation'],
    tokenizer=tokenizer
)

trainer.train()

# ------------------------------
# 🔍 Evaluate Model
```

```
# ------------------------------
preds_output = trainer.predict(emotion_encoded['test'])
print(preds_output.metrics)

y_pred = np.argmax(preds_output.predictions, axis=1)
y_true = emotion_encoded['test'][:]['label']
print(classification_report(y_true, y_pred))

# Confusion matrix
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(5, 5))
sns.heatmap(cm, annot=True, xticklabels=label2id.keys(),
yticklabels=label2id.keys(),
            fmt='d', cbar=False, cmap='Reds')
plt.ylabel("Actual")
plt.xlabel("Predicted")
plt.title("Confusion Matrix")
plt.show()

# ------------------------------
# □ Make Single Prediction
# ------------------------------
def get_prediction(text):
    input_encoded = tokenizer(text, return_tensors='pt').to(device)
    with torch.no_grad():
        outputs = model(**input_encoded)
    pred = torch.argmax(outputs.logits, dim=1).item()
    return id2label[pred]

print(get_prediction("I am super happy today. I got it done. Finally!!"))

# ------------------------------
# □ Use HuggingFace Pipeline
# ------------------------------
# Save model if you want to use in pipeline later
model.save_pretrained("bert-base-uncased-sentiment-model")
tokenizer.save_pretrained("bert-base-uncased-sentiment-model")

classifier = pipeline("text-classification", model="bert-base-uncased-
sentiment-model")
print(classifier([text, 'hello, how are you?', "love you", "i am feeling
low"]))
```

Fine-Tune Bert varities : Fake News Detection

🔍 Summary of the Workflow

1. **Data Loading & Cleaning:**
   - Loads a fake news dataset from Excel.
   - Removes null values.
2. **EDA (Exploratory Data Analysis):**
   - Plots label distribution.
   - Calculates approximate token lengths of titles and texts.
3. **Data Splitting:**
   - Splits data into training, testing, and validation sets using stratified sampling.
4. **Tokenization:**
   - Uses Hugging Face `AutoTokenizer` with different models (`DistilBERT` , `MobileBERT` , `TinyBERT`).
5. **Dataset Conversion:**
   - Converts pandas DataFrames to Hugging Face `DatasetDict` .
6. **Model Configuration & Loading:**
   - Uses `AutoModelForSequenceClassification` with appropriate `label2id` and `id2label` .
7. **Training Setup:**
   - Defines `TrainingArguments` and initializes `Trainer` .
   - Trains the model and evaluates it.
8. **Metrics & Evaluation:**
   - Uses both Hugging Face's `evaluate` library and sklearn for accuracy/F1.
9. **Model Comparison:**
   - Compares multiple models' performance and training time.
10. **Saving and Using the Model:**

- Saves the trained model and loads it for inference using `pipeline` .

Code:

```
# ---------------------- #
#        IMPORTS         #
# ---------------------- #

import pandas as pd
import matplotlib.pyplot as plt
```

```
import numpy as np
import torch
import time
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score,
f1_score
from transformers import (
    AutoTokenizer, AutoModelForSequenceClassification, AutoConfig,
    TrainingArguments, Trainer, pipeline
)
from datasets import Dataset, DatasetDict
import evaluate


# ------------------------------ #
#   LOAD & CLEAN THE DATASET     #
# ------------------------------ #

df = pd.read_excel("https://github.com/laxmimerit/All-CSV-ML-Data-
Files-Download/raw/master/fake_news.xlsx")
df = df.dropna()  # Drop rows with missing values


# ------------------------------ #
#      EXPLORATORY ANALYSIS     #
# ------------------------------ #

# Label distribution
df['label'].value_counts().plot.barh()
plt.title("Frequency of Classes")
plt.show()

# Estimate token lengths
df['title_tokens'] = df['title'].apply(lambda x: len(x.split()) * 1.5)
df['text_tokens'] = df['text'].apply(lambda x: len(x.split()) * 1.5)

fig, ax = plt.subplots(1, 2, figsize=(15, 5))
ax[0].hist(df['title_tokens'], bins=50, color='skyblue')
ax[0].set_title("Title Tokens")
ax[1].hist(df['text_tokens'], bins=50, color='orange')
ax[1].set_title("Text Tokens")
plt.show()
```

```
# ------------------------------ #
#     DATA SPLITTING           #
# ------------------------------ #

train, test = train_test_split(df, test_size=0.3, stratify=df['label'])
test, validation = train_test_split(test, test_size=1/3, stratify=test['label'])

# Convert to Hugging Face dataset format
dataset = DatasetDict({
    "train": Dataset.from_pandas(train, preserve_index=False),
    "test": Dataset.from_pandas(test, preserve_index=False),
    "validation": Dataset.from_pandas(validation, preserve_index=False)
})


# ------------------------------ #
#        TOKENIZATION          #
# ------------------------------ #

def tokenize(batch):
    tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
    return tokenizer(batch['title'], padding=True, truncation=True)

encoded_dataset = dataset.map(tokenize, batched=True)


# ------------------------------ #
#    MODEL SETUP & TRAINING      #
# ------------------------------ #

label2id = {"Real": 0, "Fake": 1}
id2label = {0: "Real", 1: "Fake"}
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Accuracy metric
accuracy = evaluate.load("accuracy")

def compute_metrics_evaluate(eval_pred):
    predictions, labels = eval_pred
    predictions = np.argmax(predictions, axis=1)
    return accuracy.compute(predictions=predictions, references=labels)
```

```
# Define training arguments
training_args = TrainingArguments(
    output_dir="train_dir",
    overwrite_output_dir=True,
    num_train_epochs=2,
    learning_rate=2e-5,
    per_device_train_batch_size=32,
    per_device_eval_batch_size=32,
    weight_decay=0.01,
    evaluation_strategy='epoch'
)

# Initialize model and trainer
model_ckpt = "distilbert-base-uncased"
config = AutoConfig.from_pretrained(model_ckpt, label2id=label2id,
id2label=id2label)
model =
AutoModelForSequenceClassification.from_pretrained(model_ckpt,
config=config).to(device)

trainer = Trainer(
    model=model,
    compute_metrics=compute_metrics_evaluate,
    train_dataset=encoded_dataset['train'],
    eval_dataset=encoded_dataset['validation'],
    tokenizer=AutoTokenizer.from_pretrained(model_ckpt)
)

trainer.train()


# ------------------------------ #
#          EVALUATION          #
# ------------------------------ #

preds_output = trainer.predict(encoded_dataset['test'])
y_pred = np.argmax(preds_output.predictions, axis=1)
y_true = encoded_dataset['test']['label']
print(classification_report(y_true, y_pred, target_names=list(label2id)))


# ------------------------------ #
#     TRAIN MULTIPLE MODELS     #
```

```
# ----------------------------- #

# Define models to test
model_dict = {
    "bert-base": "bert-base-uncased",
    "distilbert": "distilbert-base-uncased",
    "mobilebert": "google/mobilebert-uncased",
    "tinybert": "huawei-noah/TinyBERT_General_4L_312D"
}

def compute_metrics(pred):
    labels = pred.label_ids
    preds = pred.predictions.argmax(-1)
    return {
        "accuracy": accuracy_score(labels, preds),
        "f1": f1_score(labels, preds, average="weighted")
    }

def train_model(model_name):
    model_ckpt = model_dict[model_name]
    tokenizer = AutoTokenizer.from_pretrained(model_ckpt)
    config = AutoConfig.from_pretrained(model_ckpt, label2id=label2id,
id2label=id2label)
    model =
AutoModelForSequenceClassification.from_pretrained(model_ckpt,
config=config).to(device)

    def local_tokenizer(batch):
        return tokenizer(batch['title'], padding=True, truncation=True)

    encoded_dataset = dataset.map(local_tokenizer, batched=True)

    trainer = Trainer(
        model=model,
        compute_metrics=compute_metrics,
        train_dataset=encoded_dataset['train'],
        eval_dataset=encoded_dataset['validation'],
        tokenizer=tokenizer
    )

    trainer.train()
    preds = trainer.predict(encoded_dataset['test'])
    return preds.metrics
```

```
# Store model performance
model_performance = {}
for model_name in model_dict:
    print("\nTraining Model:", model_name)
    start = time.time()
    result = train_model(model_name)
    end = time.time()
    model_performance[model_name] = {model_name: result, "time
taken": end - start}


# ------------------------------ #
#      SAVE & LOAD FOR INFERENCE #
# ------------------------------ #

trainer.save_model("fake_news")  # Save the model
classifier = pipeline('text-classification', model='fake_news')  # Load for
inference
print(classifier("This is a completely false news article."))  # Example
usage
```

Restaurant_Search_NER_Recognition_By_Fine_Tuning_DistilBERT

1. **Data Loading**:
   - Downloads NER data in `.bio` format from a GitHub repository.
   - Parses token and tag sequences for train/test sets.
2. **Dataset Preparation**:
   - Creates `HuggingFace DatasetDict` from token/tag sequences.
   - Converts string tags to integer IDs using `tag2index`.
3. **Tokenization & Label Alignment**:
   - Tokenizes text while aligning word-level tags with token-level inputs, handling subword tokens and special tokens with label `-100`.
4. **Model Setup**:
   - Loads `DistilBERT` for token classification.
   - Defines metrics using `seqeval` for precision, recall, F1, and accuracy.
5. **Training**:
   - Uses HuggingFace `Trainer` for model training and evaluation.
6. **Inference**:
   - Loads the fine-tuned model via `pipeline` and performs NER on an example sentence.

```
# Step 1: Imports
import pandas as pd
import requests
from datasets import Dataset, DatasetDict
from transformers import AutoTokenizer,
AutoModelForTokenClassification, DataCollatorForTokenClassification
from transformers import TrainingArguments, Trainer, pipeline
import evaluate
import numpy as np

# Step 2: Load and parse the train.bio file
def parse_bio_file(url):
    response = requests.get(url).text.strip().splitlines()
    tokens, tags = [], []
    temp_tokens, temp_tags = [], []

    for line in response:
        if line:
            tag, token = line.strip().split("\t")
            temp_tags.append(tag)
            temp_tokens.append(token)
        else:
            tokens.append(temp_tokens)
            tags.append(temp_tags)
            temp_tokens, temp_tags = [], []

    return tokens, tags

train_tokens, train_tags =
parse_bio_file("https://raw.githubusercontent.com/laxmimerit/All-CSV-
ML-Data-Files-Download/master/mit_restaurant_search_ner/train.bio")
test_tokens, test_tags =
parse_bio_file("https://raw.githubusercontent.com/laxmimerit/All-CSV-
ML-Data-Files-Download/master/mit_restaurant_search_ner/test.bio")

# Step 3: Convert to HuggingFace DatasetDict
train_dataset = Dataset.from_pandas(pd.DataFrame({'tokens':
train_tokens, 'ner_tags_str': train_tags}))
test_dataset = Dataset.from_pandas(pd.DataFrame({'tokens': test_tokens,
'ner_tags_str': test_tags}))
dataset = DatasetDict({'train': train_dataset, 'test': test_dataset,
'validation': test_dataset})
```

```
# Step 4: Tag indexing
unique_tags = set(tag for tags in dataset['train']['ner_tags_str'] for tag in
tags if tag != 'O')
tag2index = {"O": 0}
for tag in sorted(unique_tags):
   tag2index[f'B-{tag[2:]}' if tag.startswith("B-") else tag] =
len(tag2index)
   tag2index[f'I-{tag[2:]}' if tag.startswith("I-") else tag] = len(tag2index)
index2tag = {v: k for k, v in tag2index.items()}

# Step 5: Map tags to IDs
dataset = dataset.map(lambda example: {"ner_tags": [tag2index[tag] for
tag in example['ner_tags_str']]})

# Step 6: Tokenization and label alignment
model_ckpt = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_ckpt)

def tokenize_and_align_labels(examples):
   tokenized_inputs = tokenizer(examples['tokens'], truncation=True,
is_split_into_words=True)
   labels = []

   for i, label in enumerate(examples['ner_tags']):
      word_ids = tokenized_inputs.word_ids(batch_index=i)
      label_ids, prev_word_idx = [], None

      for word_idx in word_ids:
         if word_idx is None:
            label_ids.append(-100)
         elif word_idx != prev_word_idx:
            label_ids.append(label[word_idx])
         else:
            label_ids.append(-100)
         prev_word_idx = word_idx

      labels.append(label_ids)

   tokenized_inputs['labels'] = labels
   return tokenized_inputs

tokenized_dataset = dataset.map(tokenize_and_align_labels,
batched=True)
```

```
# Step 7: Data collator
data_collator = DataCollatorForTokenClassification(tokenizer=tokenizer)

# Step 8: Define evaluation metrics
metric = evaluate.load('seqeval')
label_names = list(index2tag.values())

def compute_metrics(eval_preds):
    logits, labels = eval_preds
    predictions = np.argmax(logits, axis=-1)

    true_labels = [[label_names[l] for l in label if l != -100] for label in
labels]
    true_predictions = [
        [label_names[p] for p, l in zip(pred, label) if l != -100]
        for pred, label in zip(predictions, labels)
    ]

    results = metric.compute(predictions=true_predictions,
references=true_labels)
    return {
        "precision": results["overall_precision"],
        "recall": results["overall_recall"],
        "f1": results["overall_f1"],
        "accuracy": results["overall_accuracy"]
    }

# Step 9: Model initialization
model = AutoModelForTokenClassification.from_pretrained(
    model_ckpt, id2label=index2tag, label2id=tag2index
)

# Step 10: Training setup
training_args = TrainingArguments(
    output_dir="finetuned-ner",
    evaluation_strategy='epoch',
    save_strategy='epoch',
    learning_rate=2e-5,
    num_train_epochs=3,
    weight_decay=0.01,
)
```

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset['train'],
    eval_dataset=tokenized_dataset['validation'],
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics
)

# Step 11: Train the model
trainer.train()
trainer.save_model("ner_distilbert")

# Step 12: Inference using pipeline
ner_pipe = pipeline("token-classification", model="ner_distilbert",
aggregation_strategy="simple")
result = ner_pipe("which restaurant serves the best sushi in new york?")
print(result)
```

Fine Tuning T5 for Custom Summarization

✅ 1. Step-by-Step Summary

📘 Part A: CNN/DailyMail Dataset Exploration and Summarization

1. **Load CNN/DailyMail dataset** (only first 10 samples for quick testing).
2. **Display an article and its ground-truth summary.**
3. **Summarize the article using two pre-trained models:**
   - `ubikpt/t5-small-finetuned-cnn`
   - `facebook/bart-large-cnn`
4. **Compare the summaries** from different models.

📘 Part B: SAMSum Dataset - Fine-tuning T5 for Dialogue Summarization

1. **Load SAMSum dataset** (dialogues and summaries).
2. **Plot histogram** of dialogue and summary lengths.
3. **Preprocess the data** by tokenizing dialogue-summary pairs using `T5` tokenizer.
4. **Fine-tune** `t5-small` **model** using `Trainer` from Hugging Face.
5. **Save the model** after training.
6. **Load the trained model** into a summarization pipeline.
7. **Test the model** on a custom dialogue.

```
# ================================
# □ Step 1: Imports and Setup
# ================================

import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import torch
import numpy as np
import matplotlib.pyplot as plt
from datasets import load_dataset
from transformers import (
    pipeline, AutoTokenizer, AutoModelForSeq2SeqLM,
```

```
    DataCollatorForSeq2Seq, TrainingArguments, Trainer
)

# Detect device
device = 0 if torch.cuda.is_available() else -1

# ================================
# □ Step 2: Load CNN/DailyMail
# ================================

cnn_dataset = load_dataset("cnn_dailymail", '3.0.0', split="train[:10]")
print("□ Sample Article:\n", cnn_dataset[0]['article'])
print("\n□ Reference Summary:\n", cnn_dataset[0]['highlights'])

# ================================
# □ Step 3: Summarization Pipeline Comparison
# ================================

summary_outputs = {}

# T5-small (fine-tuned)
pipe_t5 = pipeline('summarization', model='ubikpt/t5-small-finetuned-
cnn', device=device)
summary_outputs['t5-small'] =
pipe_t5(cnn_dataset[0]['article'])[0]['summary_text']

# BART-large-cnn
pipe_bart = pipeline('summarization', model='facebook/bart-large-cnn',
device=device)
summary_outputs['bart-large'] =
pipe_bart(cnn_dataset[0]['article'])[0]['summary_text']

# Print both summaries
for model_name, summary_text in summary_outputs.items():
    print(f"\n□ {model_name.upper()} Summary:\n{summary_text}")

# ================================
# □ Step 4: Load SAMSum Dataset
# ================================

samsum = load_dataset("samsum", trust_remote_code=True)
dialogue_len = [len(x['dialogue'].split()) for x in samsum['train']]
summary_len = [len(x['summary'].split()) for x in samsum['train']]
```

```
# Plotting lengths
length_df = pd.DataFrame({'Dialogue Length': dialogue_len, 'Summary
Length': summary_len})
length_df.hist(figsize=(10, 3))

# ==================================
# □ Step 5: Preprocessing
# ==================================

model_ckpt = 't5-small'
tokenizer = AutoTokenizer.from_pretrained(model_ckpt)
model =
AutoModelForSeq2SeqLM.from_pretrained(model_ckpt).to(torch.device
("cuda" if device == 0 else "cpu"))

def tokenize(batch):
    return tokenizer(batch['dialogue'], text_target=batch['summary'],
                     max_length=200, truncation=True, padding="max_length")

samsum_tokenized = samsum.map(tokenize, batched=True)

# ==================================
# 🏋 Step 6: Training
# ==================================

data_collator = DataCollatorForSeq2Seq(tokenizer=tokenizer,
model=model)

training_args = TrainingArguments(
    output_dir="train_dir",
    num_train_epochs=2,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    weight_decay=0.01,
    learning_rate=2e-5,
    gradient_accumulation_steps=500,
    logging_dir="./logs",
    logging_steps=10
)
```

```
trainer = Trainer(
    model=model,
    args=training_args,
    tokenizer=tokenizer,
    data_collator=data_collator,
    train_dataset=samsum_tokenized['train'],
    eval_dataset=samsum_tokenized['validation']
)

trainer.train()
trainer.save_model("t5_samsum_summarization")

# ================================
# □ Step 7: Inference on Custom Dialogue
# ================================

pipe = pipeline('summarization', model="t5_samsum_summarization",
tokenizer=tokenizer, device=device)

custom_dialogue = """
Laxmi Kant: what work you planning to give Tom?
Juli: I was hoping to send him on a business trip first.
Laxmi Kant: Cool. Is there any suitable work for him?
Juli: He did excellent in the last quarter. I will assign a new project once
he is back.
"""

print("\n🗣 Custom Dialogue Summary:\n",
pipe(custom_dialogue)[0]['summary_text'])
```

Image Classification – ViT

✅ 2. Steps Summary

👣 Step-by-Step Summary:

1. **Load dataset** of Indian food images.
2. **Map class labels** to integers (`label2id` , `id2label`).
3. **Load image processor** compatible with Vision Transformer (ViT).
4. **Define preprocessing pipeline**: Resize, crop, normalize, and convert to tensors.
5. **Apply transformations** to training dataset.
6. **Define evaluation metric** (accuracy).
7. **Load pre-trained ViT model** and configure it for classification.
8. **Set training arguments** like learning rate, batch size, etc.
9. **Train the model** using Hugging Face `Trainer` .
10. **Save the trained model** and create an inference pipeline.
11. **Download an image from the internet** and use the model to predict the food class.

```
# ============================================
# □ Step 1: Install & Import Dependencies
# ============================================

# !pip install datasets -q
# !pip install -U accelerate evaluate -q

import warnings
warnings.filterwarnings('ignore')

import requests
from io import BytesIO
from PIL import Image
import torch
import numpy as np
import pandas as pd
import evaluate

from datasets import load_dataset
from torchvision.transforms import Compose, RandomResizedCrop,
ToTensor, Normalize

from transformers import (
```

```
    AutoImageProcessor, AutoModelForImageClassification,
    TrainingArguments, Trainer, pipeline
)

# ===============================================
# 📥 Step 2: Load Dataset and Prepare Labels
# ===============================================

dataset = load_dataset("rajistics/indian_food_images")
labels = dataset['train'].features['label'].names
label2id = {label: i for i, label in enumerate(labels)}
id2label = {i: label for i, label in enumerate(labels)}

# ===============================================
# □ Step 3: Image Processor & Transformations
# ===============================================

model_ckpt = "google/vit-base-patch16-224-in21k"
image_processor = AutoImageProcessor.from_pretrained(model_ckpt)

normalize = Normalize(mean=image_processor.image_mean,
std=image_processor.image_std)
size = image_processor.size.get('shortest_edge') or
(image_processor.size['height'], image_processor.size['width'])

_transforms = Compose([
    RandomResizedCrop(size),
    ToTensor(),
    normalize
])

def transform_fn(batch):
    batch['pixel_values'] = [_transforms(img.convert("RGB")) for img in
batch['image']]
    del batch['image']
    return batch

dataset = dataset.with_transform(transform_fn)

# ===============================================
# □ Step 4: Define Evaluation Metric
# ===============================================
```

```python
accuracy = evaluate.load('accuracy')

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=1)
    return accuracy.compute(predictions=preds, references=labels)

# ==========================================
# □ Step 5: Load Pretrained Model
# ==========================================

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

model = AutoModelForImageClassification.from_pretrained(
    model_ckpt,
    num_labels=len(labels),
    id2label=id2label,
    label2id=label2id
).to(device)

# ==========================================
# ⚙ Step 6: Set Training Arguments & Train
# ==========================================

training_args = TrainingArguments(
    output_dir="train_dir",
    remove_unused_columns=False,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    learning_rate=5e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    gradient_accumulation_steps=4,
    num_train_epochs=4,
    load_best_model_at_end=True,
    metric_for_best_model='accuracy'
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=dataset['train'],
```

```
    eval_dataset=dataset['test'],
    tokenizer=image_processor,
    compute_metrics=compute_metrics
)

trainer.train()

# ==============================================
# 📦 Step 7: Inference on Web Image
# ==============================================

pipe = pipeline("image-classification", model="train_dir", device=0 if
torch.cuda.is_available() else -1)

url = "https://www.indianhealthyrecipes.com/wp-
content/uploads/2015/10/pizza-recipe-1.jpg"
response = requests.get(url)
image = Image.open(BytesIO(response.content))

image.show()
prediction = pipe(image)
print("\n🍽 Predicted Food Label:", prediction)
```

Coding_Fine_Tuning_LLM_Phi2_on_Custom_Data

```
!pip install -q accelerate -U
!pip install -q bitsandbytes -U
!pip install -q trl -U
!pip install -q peft -U
!pip install -q transformers -U
!pip install -q datasets –U
```

These install or update required libraries:

- `accelerate`: Efficiently trains large models with multi-GPU or mixed precision.
- `bitsandbytes`: Enables 8-bit optimizers and model quantization.
- `trl`: Hugging Face library for training RLHF and other fine-tuning.
- `peft`: For Parameter-Efficient Fine-Tuning (like LoRA).
- `transformers`: Core Hugging Face library for models, tokenizers, pipelines.
- `datasets`: For loading and processing datasets.

📂 Data Handling

python Copy Edit

```python
import pandas as pd
from datasets import load_dataset, Dataset, DatasetDict
```

- `pandas`: Read and manipulate the CSV file.
- `load_dataset`: Loads Hugging Face or custom datasets.
- `Dataset` and `DatasetDict`: Convert pandas DataFrame into HF Dataset and split into train/test.

🧹 Preprocessing

python Copy Edit

```python
df['category'] = df['category'].apply(lambda x: x.split('|')[-1])
```

- Extracts the most specific category (last subcategory) from a pipe-separated string.

python Copy Edit

```python
df = pd.concat([...])
dataset = Dataset.from_pandas(df).shuffle().train_test_split()
```

- Concatenates product names and descriptions.
- Converts to HF dataset.
- Shuffles and splits into train/test.

🧾 Formatting Function

python

Copy Edit

```python
def formatting_func(example):
    return f"""..."""
```

- Formats prompts in an instruction-following way.

🧠 Model and Tokenizer (Phi-2)

python

Copy Edit

```python
from transformers import AutoTokenizer, AutoModelForCausalLM
```

- Loads the base Phi-2 model and tokenizer.
- Configured for causal language modeling.

🧪 Tokenization

```python
def tokenize(prompt): ...
```

- Tokenizes the prompt.
- Sets input and labels (for causal LM training).

🧪 Evaluation

```python
model.generate(...)
```

- Generates output for a test prompt.

🧩 LoRA / PEFT Setup

```python
from peft import LoraConfig, get_peft_model
```

- Creates LoRA adapters for efficient fine-tuning.
- Injects trainable low-rank adapters into the model.

📊 Model Parameter Summary

```python
def print_trainable_parameters(model): ...
```

- Calculates number and percentage of trainable parameters.

🚀 Accelerator

```python
from accelerate import Accelerator
```

- Wraps model for efficient training (handles device placement, mixed precision).

⚙️ Training Setup

```python
from transformers import TrainingArguments, Trainer, DataCollatorForLangua
```

- `TrainingArguments` : Defines how training should run.
- `Trainer` : Manages training loop.
- `DataCollatorForLanguageModeling` : Collates batches and pads inputs.

🧠 Final Evaluation

```python
from peft import PeftModel
```

- Loads fine-tuned LoRA adapter weights into the base model for inference.

🗜️ Zip Save

```python
!zip -r phi2_qlora_adapter.zip ...
```

- Compresses and saves the trained adapter checkpoint.

2. Summary of Fine-Tuning Steps

1. **Install dependencies** – for model training, quantization, and dataset handling.
2. **Load product data** – from CSV, clean and reformat.
3. **Format for instruction tuning** – standardize prompts for both tasks.
4. **Load model/tokenizer** – load Phi-2 with 8-bit quantization for memory efficiency.
5. **Tokenize and prepare dataset** – left padding, max length, labels for causal LM.
6. **Evaluate base model** – generate from raw Phi-2 before fine-tuning.
7. **Add LoRA adapters (QLoRA)** – only train a small number of parameters efficiently.
8. **Wrap with `accelerator`** – prepare model for training on GPU/mixed precision.
9. **Configure trainer** – training loop, logging, evaluation, checkpoints.
10. **Train model** – fine-tune with custom formatted prompts.
11. **Evaluate fine-tuned model** – load trained model, generate outputs.
12. **Export model adapter** – zip adapter weights for sharing/deployment.

```
# === Install Requirements ===
!pip install -q accelerate -U
!pip install -q bitsandbytes -U
!pip install -q trl -U
!pip install -q peft -U
!pip install -q transformers -U
!pip install -q datasets -U

# === Imports ===
import torch
import pandas as pd
from datasets import Dataset
from transformers import (
    AutoTokenizer, AutoModelForCausalLM, TrainingArguments,
    Trainer, DataCollatorForLanguageModeling
)
from peft import LoraConfig, get_peft_model, PeftModel
from accelerate import Accelerator

# === Load and Prepare Dataset ===
df = pd.read_csv(
    'https://github.com/laxmimerit/All-CSV-ML-Data-Files-
Download/raw/master/amazon_product_details.csv',
    usecols=['category', 'about_product', 'product_name']
)
df['category'] = df['category'].apply(lambda x: x.split('|')[-1])
products = df[['category', 'product_name']].rename(columns={'product_name': 'text'})
```

```
description = df[['category', 'about_product']].rename(columns={'about_product':
'text'})
products['task_type'] = 'Product Name'
description['task_type'] = 'Product Description'
df = pd.concat([products, description], ignore_index=True)
dataset = Dataset.from_pandas(df).shuffle(seed=0).train_test_split(test_size=0.1)

# === Prompt Formatting ===
def formatting_func(example):
    return f"""Given the product category, you need to generate a
'{example['task_type']}'.
### Category: {example['category']}
### {example['task_type']}: {example['text']}"""

# === Tokenization ===
base_model_id = "microsoft/phi-2"
tokenizer = AutoTokenizer.from_pretrained(base_model_id, use_fast=False,
padding_side='left')
tokenizer.pad_token = tokenizer.eos_token
max_length = 400

def tokenize(example):
    enc = tokenizer(formatting_func(example), truncation=True,
max_length=max_length, padding="max_length")
    enc['labels'] = enc['input_ids'].copy()
    return enc

dataset = dataset.map(tokenize)

# === Load Model with LoRA ===
model = AutoModelForCausalLM.from_pretrained(base_model_id,
torch_dtype=torch.float16, load_in_8bit=True)
lora_config = LoraConfig(
    r=32, lora_alpha=64, target_modules=["Wqkv", "fc1", "fc2"],
    bias="none", lora_dropout=0.05, task_type="CAUSAL_LM"
)
model = get_peft_model(model, lora_config)

# === Print Trainable Parameters ===
def print_trainable_parameters(model):
    trainable = sum(p.numel() for p in model.parameters() if p.requires_grad)
    total = sum(p.numel() for p in model.parameters())
    print(f"Trainable: {trainable} / {total} ({100 * trainable / total:.2f}%)")
print_trainable_parameters(model)

# === Accelerate & Prepare ===
accelerator = Accelerator()
model = accelerator.prepare_model(model)

# === Training Setup ===
```

```
args = TrainingArguments(
    output_dir="train-dir", per_device_train_batch_size=2,
    gradient_accumulation_steps=1, max_steps=500, learning_rate=2.5e-5,
    optim="paged_adamw_8bit", logging_steps=25, save_steps=25,
    evaluation_strategy="steps", eval_steps=25, do_eval=True
)

trainer = Trainer(
    model=model,
    args=args,
    train_dataset=dataset['train'],
    eval_dataset=dataset['test'],
    data_collator=DataCollatorForLanguageModeling(tokenizer, mlm=False),
)
model.config.use_cache = False
trainer.train()

# === Evaluation After Fine-Tuning ===
base_model = AutoModelForCausalLM.from_pretrained(base_model_id,
load_in_8bit=True, torch_dtype=torch.float16)
eval_tokenizer = AutoTokenizer.from_pretrained(base_model_id, use_fast=False)
eval_tokenizer.pad_token = eval_tokenizer.eos_token
ft_model = PeftModel.from_pretrained(base_model, './train-dir/checkpoint-500')

eval_prompt = """Given the product category, you need to generate a 'Product
Description'.
### Category: BatteryChargers
### Product Description:"""
model_input = eval_tokenizer(eval_prompt, return_tensors='pt').to('cuda')
ft_model.eval()
with torch.no_grad():
    output = ft_model.generate(**model_input, max_new_tokens=256,
repetition_penalty=1.15)
    print(eval_tokenizer.decode(output[0], skip_special_tokens=True))

# === Save the Fine-Tuned Adapter ===
!zip -r phi2_qlora_adapter.zip ./train-dir/checkpoint-500
```

Fine tuning LLAMA Instruction tuning

✅ 1. Explanation of All Libraries and Functions Used

📦 `datasets`

- `load_dataset`: Loads datasets from the Hugging Face Hub. Here, it loads `"HuggingFaceH4/ultrachat_200k"`.

📦 `transformers`

- `AutoTokenizer`: Loads a tokenizer that converts between text and token IDs.
- `AutoModelForCausalLM`: Loads a causal language model (used for text generation).
- `BitsAndBytesConfig`: Enables 4-bit quantization for efficient training with QLoRA.
- `TrainingArguments`: Specifies training hyperparameters.
- `Trainer` & `SFTTrainer`: Trainer classes that run training loops. `SFTTrainer` is a special class from `trl` for Supervised Fine-Tuning (SFT).
- `pipeline`: Simplifies inference using models for tasks like text-generation.

📦 `peft` **(Parameter-Efficient Fine-Tuning)**

- `LoraConfig`: Configuration for Low-Rank Adaptation (LoRA).
- `get_peft_model`: Wraps the base model with trainable LoRA adapters.
- `prepare_model_for_kbit_training`: Prepares the model for 4-bit training.
- `AutoPeftModelForCausalLM`: Loads a PEFT model and merges adapters for inference.

📦 `torch`

- Used for tensor operations and accessing GPU.

✅ 2. Summary of Fine-Tuning Steps

1. **Load Dataset**: Use `UltraChat` and keep 10K samples.
2. **Format Prompts**: Convert conversations into prompt-response pairs using `<|user|>` and `<|assistant|>`.
3. **Load Base Model & Tokenizer**: Use `TinyLlama-1.1B`, with 4-bit quantization (`nf4`).
4. **Prepare Model for QLoRA**:
    - Apply LoRA adapters with `LoraConfig`.
    - Target modules: `q_proj`, `k_proj`, `v_proj`, etc.
5. **Set Training Arguments**: Configure hyperparameters (batch size, learning rate, etc.).
6. **Fine-tune with SFTTrainer**:
    - Use the formatted text dataset.
    - Train model with LoRA adapters only.
7. **Merge Adapters for Inference**: Merge adapters into the base model for deployment.
8. **Generate Text**: Use `pipeline` for evaluation on a prompt.
9. **Export**: Save adapter and zip the fine-tuned model directory.

```
# ================== INSTALL DEPENDENCIES
==================
# !pip install -q accelerate bitsandbytes trl peft transformers datasets

# ================== IMPORTS ==================
import torch
from datasets import load_dataset
from transformers import (
    AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig,
    TrainingArguments, pipeline
)
from peft import (
    LoraConfig, prepare_model_for_kbit_training,
    get_peft_model, AutoPeftModelForCausalLM
)
from trl import SFTTrainer

# ================== LOAD DATASET ==================
dataset = load_dataset("HuggingFaceH4/ultrachat_200k",
trust_remote_code=True, split="train_sft")
dataset = dataset.shuffle(seed=0).select(range(10_000))
```

```
# ================== PROMPT FORMATTING
==================
tokenizer = AutoTokenizer.from_pretrained("TinyLlama/TinyLlama-
1.1B-Chat-v1.0")
def format_prompt(example):
    chat = example['messages']
    prompt = tokenizer.apply_chat_template(chat, tokenize=False)
    return {'text': prompt}

dataset = dataset.map(format_prompt)

# ================== LOAD & PREPARE BASE MODEL
==================
model_name = "TinyLlama/TinyLlama-1.1B-intermediate-step-1431k-
3T"

bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_use_double_quant=True
)

tokenizer = AutoTokenizer.from_pretrained(model_name,
trust_remote_code=True)
tokenizer.pad_token = "<PAD>"
tokenizer.padding_side = "left"

model = AutoModelForCausalLM.from_pretrained(
    model_name,
    device_map="auto",
    quantization_config=bnb_config
)
model.config.use_cache = False
model.config.pretraining_tp = 1

# ================== APPLY LORA ADAPTERS
==================
peft_config = LoraConfig(
    lora_alpha=32,
    lora_dropout=0.1,
    r=64,
```

```
    bias="none",
    task_type="CAUSAL_LM",
    target_modules=["q_proj", "k_proj", "v_proj", "o_proj", "gate_proj",
"up_proj", "down_proj"]
)

model = prepare_model_for_kbit_training(model)
model = get_peft_model(model, peft_config)

# ================== TRAINING CONFIG
==================
training_args = TrainingArguments(
    output_dir="train_dir",
    per_device_train_batch_size=2,
    gradient_accumulation_steps=4,
    optim="paged_adamw_32bit",
    learning_rate=2e-4,
    lr_scheduler_type="cosine",
    num_train_epochs=1,
    logging_steps=10,
    fp16=True,
    gradient_checkpointing=True
)

# ================== FINE-TUNE MODEL
==================
trainer = SFTTrainer(
    model=model,
    train_dataset=dataset,
    dataset_text_field="text",
    tokenizer=tokenizer,
    args=training_args,
    max_seq_length=512,
    peft_config=peft_config
)

trainer.train()

# ================== SAVE THE ADAPTER
==================
trainer.model.save_pretrained("TinyLlama-1.1B-qlora")
```

```
# =================== LOAD MERGED MODEL FOR INFERENCE ===================
model = AutoPeftModelForCausalLM.from_pretrained("TinyLlama-1.1B-qlora", device_map="auto")
merged_model = model.merge_and_unload()

# =================== INFERENCE ===================
pipe = pipeline("text-generation", model=merged_model, tokenizer=tokenizer)
prompt = """<|user|>\nTell me something about Large Language Models.</s>\n<|assistant|>\n"""
output = pipe(prompt, max_new_tokens=100)
print(output[0]["generated_text"])

# =================== ZIP THE CHECKPOINT ===================
!zip -r tiny_llama_qlora_adapter.zip TinyLlama-1.1B-qlora
```