```
%%capture
%pip install -U bitsandbytes
%pip install -U transformers
%pip install -U accelerate
%pip install -U peft
%pip install -U trl


import os
os.environ["PYTORCH_CUDA_ALLOC_CONF"] = "expandable_segments:True,max_split_size_mb:64"
import numpy as np
import pandas as pd
import re
from tqdm import tqdm
import bitsandbytes as bnb
import torch
import torch.nn as nn
import transformers
from datasets import Dataset
from peft import LoraConfig, PeftConfig, PeftModel
from trl import SFTTrainer
from trl import setup_chat_format
from transformers import (AutoModelForCausalLM,
                          AutoTokenizer,
                          BitsAndBytesConfig,
                          TrainingArguments,
                          pipeline,
                          logging)
from sklearn.metrics import (accuracy_score,
                             classification_report,
                             confusion_matrix)
from sklearn.model_selection import train_test_split


from google.colab import drive
drive.mount('/content/drive')
```

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

```
#Preparing the dataset
data_dir = '/content/drive/MyDrive/Dataset'
data_records = []

def normalize_citations_consistent(text):
    """
    Normalize citations by converting different citation formats
    into a standardized placeholder "[Citation]".
    Detects numeric citations like "[12]" or "[12, 34]" and
    author-year citations like "(Smith et al., 2020)" or "Smith et al. (2020)".
    """
    # Replace numeric citations in square brackets.
    text = re.sub(r'\[\s*\d+(?:\s*,\s*\d+)*\s*\]', '[Citation]', text)
    # Replace citations in the form "(Smith et al., 2020)" or "(Smith, 2020)".
    text = re.sub(r'\([A-Z][a-zA-Z]+(?: et al\.)?,?\s*\d{4}\)', '[Citation]', text)
    # Also handle cases like "Smith et al. (2020)" without outer parentheses.
    text = re.sub(r'[A-Z][a-zA-Z]+(?: et al\.)?\s+\(\s*\d{4}\s*\)', '[Citation]', text)
    # Normalize excessive whitespace.
    text = re.sub(r'\s+', ' ', text).strip()
    return text

def extract_title_and_abstract(text):
    """
    Extracts the title and abstract from text using 'Title:' and 'Abstract:' keywords.
    Falls back to treating the whole text as abstract if title is not found.

    Returns:
        (title, abstract): Tuple of extracted title and abstract
    """
    title = ""
    abstract = text.strip()

    # Normalize line endings
    text = text.replace('\r\n', '\n').replace('\r', '\n')

    # Extract title (case-insensitive, from first "Title:")
    title_match = re.search(r'\bTitle:\s*(.*)', text, re.IGNORECASE)
    if title_match:
        title = title_match.group(1).strip()
        # Remove title line from text
        text = re.sub(r'\bTitle:\s*.*\n?', '', text, flags=re.IGNORECASE, count=1).strip()

    # Extract abstract (case-insensitive, from first "Abstract:")
```

```
        # Extract abstract (case-insensitive, from first "Abstract:")
        abstract_match = re.search(r'\bAbstract:\s*(.*)', text, re.IGNORECASE | re.DOTALL)
        if abstract_match:
            abstract = abstract_match.group(1).strip()

        return title, abstract

    # Iterate through each subfolder (e.g. folders named "cancer" or "non-cancer").
    for folder in os.listdir(data_dir):
        folder_path = os.path.join(data_dir, folder)
        if os.path.isdir(folder_path):
            # Determine the label based on folder name.
            label = "Noncancer" if "non-cancer" in folder.lower() else "Cancer"
            for file in os.listdir(folder_path):
                if file.endswith('.txt'):
                    file_path = os.path.join(folder_path, file)
                    with open(file_path, 'r', encoding='utf-8') as f:
                        text = f.read().strip()
                    # Normalize citations.
                    text = normalize_citations_consistent(text)
                    # Extract title and abstract.
                    title, abstract_text = extract_title_and_abstract(text)
                    abstract_id = os.path.splitext(file)[0]
                    data_records.append({
                        "text": re.sub(r"<ID:[^>]+>\s*", "", abstract_text),
                        "label": label
                    })

    # Create a Pandas DataFrame with the loaded data:
    df = pd.DataFrame(data_records)
    df.head()


    # Shuffle the DataFrame
    df = df.sample(frac=1, random_state=85).reset_index(drop=True)

    # Split the DataFrame
    train_size = 0.8
    eval_size = 0.1

    # Calculate sizes
    train_end = int(train_size * len(df))
    eval_end = train_end + int(eval_size * len(df))

    # Split the data
    X_train = df[:train_end]
    X_eval = df[train_end:eval_end]
    X_test = df[eval_end:]

    # Define the prompt generation functions
    def generate_prompt(data_point):
        return f"""
                Classify the text into Cancer, Noncancer and return the answer as the corresponding research paper label.
    text: {data_point["text"]}
    label: {data_point["label"]}""".strip()

    def generate_test_prompt(data_point):
        return f"""
                Classify the text into Cancer, Noncancer and return the answer as the corresponding research paper label.
    text: {data_point["text"]}
    label: """.strip()

    # Generate prompts for training and evaluation data
    X_train.loc[:,'text'] = X_train.apply(generate_prompt, axis=1)
    X_eval.loc[:,'text'] = X_eval.apply(generate_prompt, axis=1)

    # Generate test prompts and extract true labels
    y_true = X_test.loc[:,'label']
    X_test = pd.DataFrame(X_test.apply(generate_test_prompt, axis=1), columns=["text"])


    X_train.label.value_counts()
```

label	count
Noncancer	410
Cancer	390

dtype: int64

```
y_true.value_counts()
```

label	count
Cancer	55
Noncancer	45

dtype: int64

```
# Convert to datasets
train_data = Dataset.from_pandas(X_train[["text"]])
eval_data = Dataset.from_pandas(X_eval[["text"]])
test_data = Dataset.from_pandas(X_test[["text"]])


base_model_name = "microsoft/phi-4"

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

model = AutoModelForCausalLM.from_pretrained(
    base_model_name,
    device_map="auto",
    torch_dtype="float16",
    quantization_config=bnb_config,
)

model.config.use_cache = False
model.config.pretraining_tp = 1
```

```
config.json: 100% 802/802 [00:00<00:00, 40.7kB/s]
model.safetensors.index.json: 100% 20.4k/20.4k [00:00<00:00, 1.45MB/s]
Fetching 6 files: 100% 6/6 [03:59<00:00, 98.70s/it]
model-00005-of-00006.safetensors: 100% 4.77G/4.77G [03:58<00:00, 26.0MB/s]
model-00001-of-00006.safetensors: 100% 4.93G/4.93G [03:57<00:00, 16.9MB/s]
model-00003-of-00006.safetensors: 100% 4.90G/4.90G [03:58<00:00, 38.9MB/s]
model-00004-of-00006.safetensors: 100% 4.77G/4.77G [03:54<00:00, 36.8MB/s]
model-00002-of-00006.safetensors: 100% 4.95G/4.95G [03:58<00:00, 159MB/s]
model-00006-of-00006.safetensors: 100% 4.99G/4.99G [03:58<00:00, 102MB/s]
Loading checkpoint shards: 100% 6/6 [02:27<00:00, 24.36s/it]
generation_config.json: 100% 156/156 [00:00<00:00, 17.0kB/s]
```

```
tokenizer = AutoTokenizer.from_pretrained(base_model_name)

tokenizer.pad_token_id = tokenizer.eos_token_id
```

```
tokenizer_config.json: 100% 17.7k/17.7k [00:00<00:00, 2.02MB/s]
vocab.json: 100% 1.61M/1.61M [00:00<00:00, 5.84MB/s]
merges.txt: 100% 917k/917k [00:00<00:00, 4.42MB/s]
tokenizer.json: 100% 4.25M/4.25M [00:00<00:00, 28.9MB/s]
added_tokens.json: 100% 2.50k/2.50k [00:00<00:00, 185kB/s]
special_tokens_map.json: 100% 95.0/95.0 [00:00<00:00, 9.23kB/s]
```

```
match_labels = re.compile(r'\b(Non[-\s]?Cancer|Cancer)\b', flags=re.IGNORECASE)

def predict(test, model, tokenizer):
    y_pred = []

    for i in tqdm(range(len(test))):
        prompt = test.iloc[i]["text"]
        pipe = pipeline(task="text-generation",
                        model=model,
                        tokenizer=tokenizer,
```

```
                    max_new_tokens=5,
                    temperature=0.1)

        result = pipe(prompt)
        answer = result[0]['generated_text'].split("label:")[-1].strip()
        generated_text = result[0]['generated_text']
        # Apply regex to extract the label
        matches = match_labels.findall(answer)
        if matches:
            # Normalize and pick the first match
            label = matches[0].replace("-", "").replace(" ", "").capitalize()
            if label == "Noncancer":
                y_pred.append("Noncancer")
            else:
                y_pred.append("Cancer")
        else:
            y_pred.append("none")

    return y_pred


y_pred = predict(X_test, model, tokenizer)
```

```
  0%|          | 0/100 [00:00<?, ?it/s]Device set to use cuda:0
/usr/local/lib/python3.11/dist-packages/transformers/generation/configuration_utils.py:631: UserWarning: `do_sample` i
  warnings.warn(
  1%|          | 1/100 [00:01<03:08,  1.90s/it]Device set to use cuda:0
  2%|▏         | 2/100 [00:02<01:58,  1.21s/it]Device set to use cuda:0
  3%|▎         | 3/100 [00:03<01:45,  1.09s/it]Device set to use cuda:0
  4%|▍         | 4/100 [00:04<01:34,  1.02it/s]Device set to use cuda:0
  5%|▌         | 5/100 [00:05<01:31,  1.04it/s]Device set to use cuda:0
  6%|▌         | 6/100 [00:06<01:37,  1.04s/it]Device set to use cuda:0
  7%|▋         | 7/100 [00:07<01:38,  1.06s/it]Device set to use cuda:0
  8%|▊         | 8/100 [00:08<01:34,  1.03s/it]Device set to use cuda:0
  9%|▉         | 9/100 [00:09<01:27,  1.04it/s]Device set to use cuda:0
 10%|▉         | 10/100 [00:10<01:30,  1.01s/it]Device set to use cuda:0
 11%|█         | 11/100 [00:11<01:28,  1.01it/s]Device set to use cuda:0
 12%|█▏        | 12/100 [00:12<01:36,  1.10s/it]Device set to use cuda:0
 13%|█▎        | 13/100 [00:13<01:29,  1.03s/it]Device set to use cuda:0
 14%|█▍        | 14/100 [00:14<01:30,  1.06s/it]Device set to use cuda:0
 15%|█▌        | 15/100 [00:15<01:32,  1.08s/it]Device set to use cuda:0
 16%|█▌        | 16/100 [00:17<01:38,  1.17s/it]Device set to use cuda:0
 17%|█▋        | 17/100 [00:17<01:22,  1.01it/s]Device set to use cuda:0
 18%|█▊        | 18/100 [00:18<01:23,  1.02s/it]Device set to use cuda:0
 19%|█▉        | 19/100 [00:19<01:20,  1.01it/s]Device set to use cuda:0
 20%|██        | 20/100 [00:21<01:29,  1.12s/it]Device set to use cuda:0
 21%|██        | 21/100 [00:22<01:26,  1.10s/it]Device set to use cuda:0
 22%|██▏       | 22/100 [00:23<01:36,  1.24s/it]Device set to use cuda:0
 23%|██▎       | 23/100 [00:25<01:32,  1.20s/it]Device set to use cuda:0
 24%|██▍       | 24/100 [00:25<01:18,  1.03s/it]Device set to use cuda:0
 25%|██▌       | 25/100 [00:26<01:06,  1.13it/s]Device set to use cuda:0
 26%|██▌       | 26/100 [00:27<01:09,  1.06it/s]Device set to use cuda:0
 27%|██▋       | 27/100 [00:28<01:09,  1.06it/s]Device set to use cuda:0
 28%|██▊       | 28/100 [00:29<01:12,  1.01s/it]Device set to use cuda:0
 29%|██▉       | 29/100 [00:29<00:59,  1.19it/s]Device set to use cuda:0
 30%|███       | 30/100 [00:30<01:04,  1.09it/s]Device set to use cuda:0
 31%|███       | 31/100 [00:31<01:02,  1.10it/s]Device set to use cuda:0
 32%|███▏      | 32/100 [00:32<01:06,  1.02it/s]Device set to use cuda:0
 33%|███▎      | 33/100 [00:33<01:04,  1.03it/s]Device set to use cuda:0
 34%|███▍      | 34/100 [00:34<01:01,  1.07it/s]Device set to use cuda:0
 35%|███▌      | 35/100 [00:35<01:00,  1.07it/s]Device set to use cuda:0
 36%|███▌      | 36/100 [00:36<00:52,  1.23it/s]Device set to use cuda:0
 37%|███▋      | 37/100 [00:37<00:56,  1.12it/s]Device set to use cuda:0
 38%|███▊      | 38/100 [00:38<00:56,  1.10it/s]Device set to use cuda:0
 39%|███▉      | 39/100 [00:39<01:01,  1.01s/it]Device set to use cuda:0
 40%|████      | 40/100 [00:40<01:06,  1.11s/it]Device set to use cuda:0
 41%|████      | 41/100 [00:41<00:55,  1.06it/s]Device set to use cuda:0
 42%|████▏     | 42/100 [00:42<01:02,  1.08s/it]Device set to use cuda:0
 43%|████▎     | 43/100 [00:44<01:04,  1.13s/it]Device set to use cuda:0
 44%|████▍     | 44/100 [00:45<01:09,  1.25s/it]Device set to use cuda:0
 45%|████▌     | 45/100 [00:47<01:12,  1.33s/it]Device set to use cuda:0
 46%|████▌     | 46/100 [00:48<01:04,  1.20s/it]Device set to use cuda:0
 47%|████▋     | 47/100 [00:48<00:58,  1.11s/it]Device set to use cuda:0
 48%|████▊     | 48/100 [00:50<00:57,  1.11s/it]Device set to use cuda:0
 49%|████▉     | 49/100 [00:51<00:56,  1.11s/it]Device set to use cuda:0
 50%|█████     | 50/100 [00:52<00:56,  1.12s/it]Device set to use cuda:0
 51%|█████     | 51/100 [00:53<00:51,  1.06s/it]Device set to use cuda:0
 52%|█████▏    | 52/100 [00:54<00:51,  1.07s/it]Device set to use cuda:0
 53%|█████▎    | 53/100 [00:55<00:47,  1.02s/it]Device set to use cuda:0
 54%|█████▍    | 54/100 [00:56<00:44,  1.03it/s]Device set to use cuda:0
```

```
def evaluate(y_true, y_pred):
    labels = ["Noncancer","Cancer"]
    mapping = {label: idx for idx, label in enumerate(labels)}

    def map_func(x):
```

```
        return mapping.get(x, -1)  # Map to -1 if not found, but should not occur with correct data

    y_true_mapped = np.vectorize(map_func)(y_true)
    y_pred_mapped = np.vectorize(map_func)(y_pred)

    # Calculate accuracy
    accuracy = accuracy_score(y_true=y_true_mapped, y_pred=y_pred_mapped)
    print(f'Accuracy: {accuracy:.3f}')

    # Generate accuracy report
    unique_labels = set(y_true_mapped)  # Get unique labels

    for label in unique_labels:
        label_indices = [i for i in range(len(y_true_mapped)) if y_true_mapped[i] == label]
        label_y_true = [y_true_mapped[i] for i in label_indices]
        label_y_pred = [y_pred_mapped[i] for i in label_indices]
        label_accuracy = accuracy_score(label_y_true, label_y_pred)
        print(f'Accuracy for label {labels[label]}: {label_accuracy:.3f}')

    # Generate classification report
    class_report = classification_report(y_true=y_true_mapped, y_pred=y_pred_mapped, target_names=labels, labels=list(range(
    print('\nClassification Report:')
    print(class_report)

    # Generate confusion matrix
    conf_matrix = confusion_matrix(y_true=y_true_mapped, y_pred=y_pred_mapped, labels=list(range(len(labels))))
    print('\nConfusion Matrix:')
    print(conf_matrix)


#Not FineTuned
evaluate(y_true, y_pred)
```

```
Accuracy: 0.800
Accuracy for label Noncancer: 0.822
Accuracy for label Cancer: 0.782

Classification Report:
              precision    recall  f1-score   support

   Noncancer       0.90      0.82      0.86        45
      Cancer       0.88      0.78      0.83        55

   micro avg       0.89      0.80      0.84       100
   macro avg       0.89      0.80      0.84       100
weighted avg       0.89      0.80      0.84       100


Confusion Matrix:
[[37  6]
 [ 4 43]]
```

```
import bitsandbytes as bnb
#Extract All the linear modules name
def find_all_linear_names(model):
    cls = bnb.nn.Linear4bit
    lora_module_names = set()
    for name, module in model.named_modules():
        if isinstance(module, cls):
            names = name.split('.')
            lora_module_names.add(names[0] if len(names) == 1 else names[-1])
    if 'lm_head' in lora_module_names:  # needed for 16 bit
        lora_module_names.remove('lm_head')
    return list(lora_module_names)


modules = find_all_linear_names(model)
modules
```

```
['down_proj', 'gate_up_proj', 'qkv_proj', 'o_proj']
```

```
#Model Set up
from trl import SFTTrainer, SFTConfig
output_dir="phi-fine-tuned-model"

peft_config = LoraConfig(
    lora_alpha=8,
    lora_dropout=0.1,
    r=4,
    bias="none",
    task_type="CAUSAL_LM",
    target_modules=modules,
)
```

```
training_arguments = SFTConfig(
    output_dir="logs",
    num_train_epochs=1,
    gradient_checkpointing=True,
    per_device_train_batch_size=1,
    gradient_accumulation_steps=4,
    optim="paged_adamw_32bit",  # Use fused AdamW optimizer
    save_steps=100,
    load_best_model_at_end=True,
    logging_steps=25,
    learning_rate=2e-4,
    weight_decay=0.001,
    fp16=True,
    bf16=False,
    max_grad_norm=0.3,
    max_steps=100,
    warmup_ratio=0.03,
    group_by_length=False,
    save_strategy="steps",
    eval_steps=100,
    eval_accumulation_steps=1,
    lr_scheduler_type="cosine",
    report_to="tensorboard",
    eval_strategy="steps",              # save checkpoint every epoch
    max_seq_length=512,
    packing=False,
    dataset_kwargs={
        "add_special_tokens": False,  # Template with special tokens
        "append_concat_token": False,  # Add EOS token as separator token
    }
)


trainer = SFTTrainer(
    model=model,
    train_dataset=train_data,
    eval_dataset=eval_data,
    peft_config=peft_config,
    processing_class=tokenizer,
    args=training_arguments,
)
```

```
/usr/local/lib/python3.11/dist-packages/peft/mapping_func.py:73: UserWarning: You are trying to modify a model with PEFT
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/peft/tuners/tuners_utils.py:167: UserWarning: Already found a `peft_config` attr
  warnings.warn(
Converting train dataset to ChatML: 100%  800/800 [00:00<00:00, 12857.83 examples/s]
Applying chat template to train dataset: 100%  800/800 [00:00<00:00, 13854.19 examples/s]
Tokenizing train dataset: 100%  800/800 [00:03<00:00, 156.91 examples/s]
Truncating train dataset: 100%  800/800 [00:00<00:00, 19463.92 examples/s]
Converting eval dataset to ChatML: 100%  100/100 [00:00<00:00, 2801.92 examples/s]
Applying chat template to eval dataset: 100%  100/100 [00:00<00:00, 2843.56 examples/s]
Tokenizing eval dataset: 100%  100/100 [00:00<00:00, 364.11 examples/s]
Truncating eval dataset: 100%  100/100 [00:00<00:00, 3316.07 examples/s]
No label_names provided for model class `PeftModelForCausalLM`. Since `PeftModel` hides base models input arguments, if
```

```
# Train model
torch.cuda.empty_cache()
trainer.train()
```

[100/100 22:00, Epoch 0/1]

Step	Training Loss
25	1.722300
50	1.558900
75	1.520500
100	1.556800

```
TrainOutput(global_step=100, training_loss=1.5896308135986328, metrics={'train_runtime': 1331.211,
'train_samples_per_second': 0.3, 'train_steps_per_second': 0.075, 'total_flos': 1.258496707571712e+16, 'train_loss':
1.5896308135986328})
```

```
torch.cuda.empty_cache()
```

```
# Move model to CPU to free GPU memory
trainer.model.cpu()

torch.cuda.empty_cache()

# Save the fine-tuned model
trainer.save_model("phi-fine-tuned-model")
tokenizer.save_pretrained("phi-fine-tuned-model")
```

```
!cp -r /content/logs/checkpoint-100  /content/drive/MyDrive/phiLLMNONREASONCheckPoint/
```