```
# IMPORTANT: SOME KAGGLE DATA SOURCES ARE PRIVATE
# RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES.
import kagglehub
kagglehub.login()

```

```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

aditya0506_dataset_zip_path = kagglehub.dataset_download('aditya0506/dataset-zip')
metaresearch_llama_3_2_transformers_3b_instruct_1_path = kagglehub.model_download('metaresearch/llama-3.2/Transformers/3

print('Data source import complete.')

```

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

```

```
import numpy as np
import pandas as pd
import re
import os
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
```

```
2025-04-16 09:25:23.367876: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to register cuFFT fact
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1744795523.541222      31 cuda_dnn.cc:8310] Unable to register cuDNN factory: Attempting to register factory
E0000 00:00:1744795523.597096      31 cuda_blas.cc:1418] Unable to register cuBLAS factory: Attempting to register facto
```

```
#Preparing the dataset
data_dir = '/kaggle/input/dataset-zip/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:")
    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()
```

	text	label
0	Title: Clear Cell Variant of Papillary Thyroid...	Cancer
1	Title: [Hereditary Breast and Ovarian Cancer S...	Cancer
2	Title: [Hepatic epithelioid angiomyolipoma/PEC...	Cancer
3	Title: Giant Lumbar Polypoid Tumor with Bullae...	Cancer
4	Title: Multidisciplinary Challenges in Mastocy...	Cancer

```
# 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
Cancer       410
Noncancer    390
Name: count, dtype: int64
```

```
y_true.value_counts()
```

```
label
Noncancer    55
Cancer       45
Name: count, dtype: int64
```

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

```
train_data['text'][3]
```

```
"Classify the text into Cancer, Noncancer and return the answer as the corresponding research paper label.\ntext: Title: Neurological disease in xeroderma pigmentosum: prospective cohort study of its features and progression. Abstract: Xeroderma pigmentosum (XP) results from biallelic mutations in any of eight genes involved in DNA repair systems, thus defining eight different genotypes (XPA, XPB, XPC, XPD, XPE, XPF, XPG and XP variant or XPV). In addition to cutaneous and ophthalmological features, some patients present with XP neurological disease. It is unknown whether the different neurological signs and their progression differ among groups. Therefore, we aim to characterize the XP neurological disease and its evolution in the heterogeneous UK XP cohort. Patients with XP were followed in the UK National XP Service, from 2009 to 2021. Age of onset for different events was recorded. Cerebellar ataxia and additional neurological signs and symptoms were rated with the Scale for the Assessment and Rating of Ataxia (SARA), the Inventory of Non-Ataxia Signs (INAS) and the Activities of Daily Living questionnaire (ADL). Patients' mutations received scores based on their predicted effects. Data from available ancillary tests were collected. Ninety-three XP patients were recruited. Thirty-six (38.7%) reported neurological symptoms, especially in the XPA, XPD and XPG groups, with early-onset and late-onset forms, and typically appearing after cutaneous and ophthalmological symptoms. XPA, XPD and XPG patients showed higher SARA scores compared to XPC, XPE and XPV. SARA total scores significantly increased over time in XPD (0.91 points/year, 95% confidence interval: 0.61, 1.21) and XPA (0.63 points/year, 95% confidence interval: 0.38, 0.89). Hyporeflexia, hypopallesthaesia, upper motor neuron signs, chorea, dystonia, oculomotor signs and cognitive impairment were frequent findings in XPA, XPD and XPG. Cerebellar and global brain atrophy, axonal sensory and sensorimotor neuropathies, and sensorineural hearing loss were common findings in patients. Some XPC, XPE and XPV cases presented with abnormalities on examination and/or ancillary tests, suggesting underlying neurological involvement. More severe mutations were associated with a faster progression in SARA total score in XPA (0.40 points/year per 1-unit increase in severity score) and XPD (0.60 points/year per 1-unit increase), and in ADL total score in XPA (0.35 points/year per 1-unit increase). Symptomatic and asymptomatic forms of neurological disease are frequent in XP patients, and neurological symptoms can be an important cause of disability. Typically, the neurological disease will be preceded by cutaneous and ophthalmological features, and these should be actively searched in patients with idiopathic late-onset neurological syndromes. Scales assessing cerebellar function, especially walking and speech, and disability can show progression in some of the groups. Mutation severity can be used as a prognostic biomarker for stratification purposes in clinical trials.\nlabel: Noncancer"
```

Loading the model and tokenizer

```
base_model_name = "/kaggle/input/llama-3.2/transformers/3b-instruct/1"

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
```

```
Loading checkpoint shards:   0%|          | 0/2 [00:00<?, ?it/s]
```

```
tokenizer = AutoTokenizer.from_pretrained(base_model_name)

tokenizer.pad_token_id = tokenizer.eos_token_id
```

Model evalution before fine-tuning

```
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)
```

```
 81%|████████  | 81/100 [00:46<00:09,  1.92it/s]Device set to use cuda:0
 82%|████████  | 82/100 [00:47<00:09,  1.93it/s]Device set to use cuda:0
 83%|████████  | 83/100 [00:47<00:09,  1.78it/s]Device set to use cuda:0
 84%|████████  | 84/100 [00:48<00:09,  1.70it/s]Device set to use cuda:0
 85%|████████  | 85/100 [00:48<00:08,  1.77it/s]Device set to use cuda:0
 86%|████████  | 86/100 [00:49<00:08,  1.72it/s]Device set to use cuda:0
 87%|████████  | 87/100 [00:49<00:06,  1.90it/s]Device set to use cuda:0
 88%|████████  | 88/100 [00:50<00:06,  1.73it/s]Device set to use cuda:0
 89%|████████  | 89/100 [00:51<00:06,  1.72it/s]Device set to use cuda:0
 90%|████████  | 90/100 [00:51<00:05,  1.67it/s]Device set to use cuda:0
 91%|████████  | 91/100 [00:52<00:05,  1.73it/s]Device set to use cuda:0
 92%|████████  | 92/100 [00:54<00:07,  1.09it/s]Device set to use cuda:0
 93%|████████  | 93/100 [00:54<00:05,  1.20it/s]Device set to use cuda:0
 94%|████████  | 94/100 [00:55<00:04,  1.25it/s]Device set to use cuda:0
 95%|████████  | 95/100 [00:55<00:03,  1.36it/s]Device set to use cuda:0
 96%|████████  | 96/100 [00:56<00:02,  1.48it/s]Device set to use cuda:0
 97%|████████  | 97/100 [00:57<00:01,  1.53it/s]Device set to use cuda:0
 98%|████████  | 98/100 [00:57<00:01,  1.45it/s]Device set to use cuda:0
 99%|████████  | 99/100 [00:58<00:00,  1.50it/s]Device set to use cuda:0
100%|████████  | 100/100 [00:58<00:00,  1.70it/s]
```

```
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(ra
    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: 0.390
Accuracy for label Noncancer: 0.255
Accuracy for label Cancer: 0.556

Classification Report:
              precision    recall  f1-score   support

   Noncancer       0.45      0.25      0.33        55
      Cancer       0.38      0.56      0.45        45

   micro avg       0.40      0.39      0.40       100
   macro avg       0.42      0.41      0.39       100
weighted avg       0.42      0.39      0.38       100


Confusion Matrix:
[[14 41]
 [17 25]]
```

```
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
```

```
['gate_proj', 'v_proj', 'o_proj', 'up_proj', 'down_proj', 'k_proj', 'q_proj']
```

```
#Model Set up
from trl import SFTTrainer, SFTConfig
output_dir="llama-3.2-fine-tuned-model"

peft_config = LoraConfig(
    lora_alpha=16,
    lora_dropout=0.1,
    r=8,
    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=8,
    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=-1,
    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,
)
```

```
Converting train dataset to ChatML:   0%|          | 0/800 [00:00<?, ? examples/s]
Applying chat template to train dataset:   0%|          | 0/800 [00:00<?, ? examples/s]
Tokenizing train dataset:   0%|          | 0/800 [00:00<?, ? examples/s]
Truncating train dataset:   0%|          | 0/800 [00:00<?, ? examples/s]
Converting eval dataset to ChatML:   0%|          | 0/100 [00:00<?, ? examples/s]
Applying chat template to eval dataset:   0%|          | 0/100 [00:00<?, ? examples/s]
Tokenizing eval dataset:   0%|          | 0/100 [00:00<?, ? examples/s]
Truncating eval dataset:   0%|          | 0/100 [00:00<?, ? examples/s]
No label_names provided for model class `PeftModelForCausalLM`. Since `PeftModel` hides base models input arguments, if
```

```
# Train model
trainer.train()
```

[100/100 21:28, Epoch 1/1]

Step	Training Loss	Validation Loss
100	1.833500	1.768290

```
TrainOutput(global_step=100, training_loss=1.867441635131836, metrics={'train_runtime': 1301.2147,
'train_samples_per_second': 0.615, 'train_steps_per_second': 0.077, 'total_flos': 5103678867886080.0, 'train_loss':
```

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

```
 61%|██████    | 61/100 [00:39<00:28,  1.36it/s]Device set to use cuda:0
 62%|██████    | 62/100 [00:40<00:26,  1.43it/s]Device set to use cuda:0
 63%|██████    | 63/100 [00:41<00:26,  1.39it/s]Device set to use cuda:0
 64%|██████    | 64/100 [00:41<00:25,  1.44it/s]Device set to use cuda:0
 65%|██████    | 65/100 [00:42<00:23,  1.51it/s]Device set to use cuda:0
 66%|██████    | 66/100 [00:43<00:24,  1.37it/s]Device set to use cuda:0
 67%|██████    | 67/100 [00:43<00:25,  1.32it/s]Device set to use cuda:0
 68%|██████    | 68/100 [00:44<00:22,  1.40it/s]Device set to use cuda:0
 69%|██████    | 69/100 [00:45<00:21,  1.43it/s]Device set to use cuda:0
 70%|██████    | 70/100 [00:45<00:20,  1.45it/s]Device set to use cuda:0
 71%|██████    | 71/100 [00:46<00:19,  1.49it/s]Device set to use cuda:0
 72%|██████    | 72/100 [00:47<00:18,  1.53it/s]Device set to use cuda:0
 73%|██████    | 73/100 [00:47<00:18,  1.44it/s]Device set to use cuda:0
 74%|██████    | 74/100 [00:48<00:20,  1.26it/s]Device set to use cuda:0
 75%|██████    | 75/100 [00:49<00:19,  1.26it/s]Device set to use cuda:0
 76%|██████    | 76/100 [00:50<00:17,  1.36it/s]Device set to use cuda:0
 77%|██████    | 77/100 [00:51<00:17,  1.35it/s]Device set to use cuda:0
 78%|██████    | 78/100 [00:51<00:14,  1.49it/s]Device set to use cuda:0
 79%|██████    | 79/100 [00:52<00:15,  1.39it/s]Device set to use cuda:0
 80%|██████    | 80/100 [00:52<00:12,  1.58it/s]Device set to use cuda:0
 81%|██████    | 81/100 [00:53<00:11,  1.60it/s]Device set to use cuda:0
 82%|██████    | 82/100 [00:53<00:10,  1.72it/s]Device set to use cuda:0
 83%|██████    | 83/100 [00:54<00:10,  1.55it/s]Device set to use cuda:0
 84%|██████    | 84/100 [00:55<00:11,  1.45it/s]Device set to use cuda:0
 85%|██████    | 85/100 [00:56<00:09,  1.60it/s]Device set to use cuda:0
 86%|██████    | 86/100 [00:56<00:09,  1.50it/s]Device set to use cuda:0
 87%|██████    | 87/100 [00:57<00:08,  1.61it/s]Device set to use cuda:0
 88%|██████    | 88/100 [00:57<00:07,  1.57it/s]Device set to use cuda:0
 89%|██████    | 89/100 [00:58<00:06,  1.62it/s]Device set to use cuda:0
 90%|██████    | 90/100 [00:59<00:06,  1.51it/s]Device set to use cuda:0
 91%|██████    | 91/100 [00:59<00:05,  1.50it/s]Device set to use cuda:0
 92%|██████    | 92/100 [01:01<00:08,  1.04s/it]Device set to use cuda:0
 93%|██████    | 93/100 [01:02<00:06,  1.04it/s]Device set to use cuda:0
 94%|██████    | 94/100 [01:03<00:05,  1.07it/s]Device set to use cuda:0
 95%|██████    | 95/100 [01:04<00:04,  1.15it/s]Device set to use cuda:0
 96%|██████    | 96/100 [01:04<00:03,  1.23it/s]Device set to use cuda:0
 97%|██████    | 97/100 [01:05<00:02,  1.27it/s]Device set to use cuda:0
 98%|██████    | 98/100 [01:06<00:01,  1.20it/s]Device set to use cuda:0
 99%|██████    | 99/100 [01:07<00:00,  1.23it/s]Device set to use cuda:0
100%|██████    | 100/100 [01:07<00:00,  1.47it/s]Accuracy: 0.740
Accuracy for label Noncancer: 0.909
Accuracy for label Cancer: 0.533

Classification Report:
              precision    recall  f1-score   support

   Noncancer       0.70      0.91      0.79        55
      Cancer       0.83      0.53      0.65        45

    accuracy                           0.74       100
   macro avg       0.77      0.72      0.72       100
weighted avg       0.76      0.74      0.73       100


Confusion Matrix:
[[50  5]
 [21 24]]
```

```
trainer.save_model(output_dir)
tokenizer.save_pretrained(output_dir)
```

```
('llama-3.2-fine-tuned-model/tokenizer_config.json',
 'llama-3.2-fine-tuned-model/special_tokens_map.json',
 'llama-3.2-fine-tuned-model/tokenizer.json')
```

```
# Load Model base model
model = AutoModelForCausalLM.from_pretrained(
    base_model_name,
    device_map="auto",
    torch_dtype="float16",
    quantization_config=bnb_config,
)

# Merge LoRA and base model and save
peft_model = PeftModel.from_pretrained(model, output_dir)
```

```
peft_model = PeftModel.from_pretrained(model, output_dir)
merged_model = peft_model.merge_and_unload()
merged_model.save_pretrained("merged-LoRA-llama-model",
                             safe_serialization=True)

tokenizer = AutoTokenizer.from_pretrained(output_dir)
tokenizer.save_pretrained("merged-LoRA-llama-model-tokenizer")
```

```
Loading checkpoint shards:   0%|          | 0/2 [00:00<?, ?it/s]
/usr/local/lib/python3.11/dist-packages/peft/tuners/lora/bnb.py:351: UserWarning: Merge lora module to 4-bit linear may
  warnings.warn(
('merged-LoRA-llama-model-tokenizer/tokenizer_config.json',
 'merged-LoRA-llama-model-tokenizer/special_tokens_map.json',
 'merged-LoRA-llama-model-tokenizer/tokenizer.json')
```