Downloading the data set

```
from google.colab import files
files.upload()
```

Choose Files kaggle.json

- **kaggle.json**(application/json) - 71 bytes, last modified: 5/5/2025 - 100% done

```
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"motaweamohammed","key":"d5c7a32972baceb431733f84766c75c4"}'}
```

```
import os
import zipfile

# Make a hidden .kaggle folder
os.makedirs("/root/.kaggle", exist_ok=True)

# Move kaggle.json to that folder
!mv kaggle.json /root/.kaggle/

# Set permissions
!chmod 600 /root/.kaggle/kaggle.json
```

```
!kaggle datasets download -d bilal1907/mimic-iii-10k
```

```
Dataset URL: https://www.kaggle.com/datasets/bilal1907/mimic-iii-10k
License(s): MIT
mimic-iii-10k.zip: Skipping, found more recently modified local copy (use --force to force download)
```

```
with zipfile.ZipFile("/content/mimic-iii-10k.zip", 'r') as zip_ref:
    zip_ref.extractall("/content/mimic-iii-10k")
```

```
!pip install scrubadub
```

```
Requirement already satisfied: scrubadub in /usr/local/lib/python3.11/dist-packages (2.0.1)
Requirement already satisfied: textblob==0.15.3 in /usr/local/lib/python3.11/dist-packages (from scrubadub) (0.15.3)
Requirement already satisfied: phonenumbers in /usr/local/lib/python3.11/dist-packages (from scrubadub) (9.0.5)
Requirement already satisfied: python-stdnum in /usr/local/lib/python3.11/dist-packages (from scrubadub) (2.0)
Requirement already satisfied: dateparser in /usr/local/lib/python3.11/dist-packages (from scrubadub) (1.2.1)
Requirement already satisfied: catalogue in /usr/local/lib/python3.11/dist-packages (from scrubadub) (2.0.10)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (from scrubadub) (1.6.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.11/dist-packages (from scrubadub) (4.13.2)
Requirement already satisfied: faker in /usr/local/lib/python3.11/dist-packages (from scrubadub) (37.1.0)
Requirement already satisfied: nltk>=3.1 in /usr/local/lib/python3.11/dist-packages (from textblob==0.15.3->scrubadub) (3.9.1)
Requirement already satisfied: python-dateutil>=2.7.0 in /usr/local/lib/python3.11/dist-packages (from dateparser->scrubadub) (2.9.0.post0)
Requirement already satisfied: pytz>=2024.2 in /usr/local/lib/python3.11/dist-packages (from dateparser->scrubadub) (2025.2)
Requirement already satisfied: regex!=2019.02.19,!=2021.8.27,>=2015.06.24 in /usr/local/lib/python3.11/dist-packages (from dateparser->scrubadub) (2024.11.6)
Requirement already satisfied: tzlocal>=0.2 in /usr/local/lib/python3.11/dist-packages (from dateparser->scrubadub) (5.3.1)
Requirement already satisfied: tzdata in /usr/local/lib/python3.11/dist-packages (from faker->scrubadub) (2025.2)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.11/dist-packages (from scikit-learn->scrubadub) (2.0.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn->scrubadub) (1.15.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn->scrubadub) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn->scrubadub) (3.6.0)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages (from nltk>=3.1->textblob==0.15.3->scrubadub) (8.1.8)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from nltk>=3.1->textblob==0.15.3->scrubadub) (4.67.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7.0->dateparser->scrubadub) (1.17.0)
```

```
%%capture
!pip install --no-deps bitsandbytes accelerate xformers==0.0.29.post3  peft trl triton
!pip install --no-deps cut_cross_entropy unsloth_zoo
!pip install sentencepiece protobuf datasets huggingface_hub hf_transfer
!pip install --no-deps unsloth
```

Data preprocessing

```
import pandas as pd
import numpy as np
import re
import scrubadub
import spacy
from sklearn.model_selection import train_test_split
#from sklearn.metrics import accuracy_score, f1_score, precision_recall_f1_support, confusion_matrix
import torch
from unsloth import FastLanguageModel
from google.colab import userdata
from tqdm import tqdm
import json
```

```
from torch.utils.tensorboard import SummaryWriter

# Enable tqdm for pandas
tqdm.pandas()
```

```
🦥 Unsloth: Will patch your computer to enable 2x faster free finetuning.
🦥 Unsloth Zoo will now patch everything to make training faster!
```

```
# Enable tqdm for pandas
tqdm.pandas()
```

```
# Load tables
notes = pd.read_csv('/content/mimic-iii-10k/MIMIC -III (10000 patients)/NOTEEVENTS/NOTEEVENTS_sorted.csv')
diagnoses = pd.read_csv('/content/mimic-iii-10k/MIMIC -III (10000 patients)/DIAGNOSES_ICD/DIAGNOSES_ICD_sorted.csv')
diagnosis_names = pd.read_csv('/content/mimic-iii-10k/MIMIC -III (10000 patients)/D_ICD_DIAGNOSES/D_ICD_DIAGNOSES.csv')
```

```
# 1. Filter for discharge summaries
notes = notes[notes['CATEGORY'] == 'Discharge summary']
notes = notes[['SUBJECT_ID', 'HADM_ID', 'TEXT']].dropna()
# Check the first few rows to verify
print(notes.head())
```

```
      SUBJECT_ID   HADM_ID                                               TEXT
30            3  145834.0  Admission Date:  [**2101-10-20**]     Discharg...
102           4  185777.0  Admission Date:  [**2191-3-16**]     Discharge...
116           6  107064.0  Admission Date: [**2175-5-30**]        Dischar...
158           9  150750.0  Name:  [**Known lastname 10050**], [**Known fi...
166           9  150750.0  Admission Date:  [**2149-11-9**]       Dischar...
```

```
# Display basic info
print("NOTEEVENTS Shape:", notes.shape)
print("Sample Text:\n", notes['TEXT'].iloc[0][:500])  # First 500 characters
print("\nDIAGNOSES Sample:\n", diagnoses[['SUBJECT_ID', 'HADM_ID', 'ICD9_CODE']].head())
```

```
NOTEEVENTS Shape: (13063, 3)
Sample Text:
 Admission Date:  [**2101-10-20**]     Discharge Date:  [**2101-10-31**]

Date of Birth:   [**2025-4-11**]     Sex:  M

Service:  Medicine

CHIEF COMPLAINT:  Admitted from rehabilitation for
hypotension (systolic blood pressure to the 70s) and
decreased urine output.

HISTORY OF PRESENT ILLNESS:  The patient is a 76-year-old
male who had been hospitalized at the [**Hospital1 190**] from [**10-11**] through [**10-19**] of [**2101**]
after undergoing a left femoral-AT bypass graft and was
subsequen

DIAGNOSES Sample:
    SUBJECT_ID  HADM_ID ICD9_CODE
0           2   163353     V3001
1           2   163353      V053
2           2   163353      V290
3           3   145834      2639
4           3   145834      6826
```

```
# 2. To print 5 random samples:
print("=== 5 Random Samples ===")
for idx, txt in notes['TEXT'].sample(5, random_state=42).items():
    print(f"[{idx}]\n{txt}\n{'-'*40}\n")
```

```
=== 5 Random Samples ===
[102949]
Name:  [**Known lastname **],[**Known firstname **]                     Unit No:   [**Numeric Identifier 4680**]

Admission Date:  [**2141-8-12**]              Discharge Date:   [**2141-8-17**]

Date of Birth:  [**2090-10-12**]             Sex:   M

Service: MEDICINE

Allergies:
Patient recorded as having No Known Allergies to Drugs

Attending:[**First Name3 (LF) 3930**]
Addendum:
[**2141-8-17**] 06:16AM BLOOD WBC-8.4 RBC-3.09* Hgb-9.2* Hct-29.3*
MCV-95 MCH-29.8 MCHC-31.5 RDW-13.9 Plt Ct-331
[**2141-8-17**] 06:16AM BLOOD Glucose-131* UreaN-15 Creat-1.1 Na-137
K-4.5 Cl-104 HCO3-26 AnGap-12
[**2141-8-17**] 06:16AM BLOOD ALT-182* AST-214* LD(LDH)-228
AlkPhos-179* TotBili-0.4
```

```
Discharge Disposition:
Extended Care

Facility:
[**Location (un) 4681**] Hills

                              [**Name6 (MD) **] [**Name8 (MD) **] MD [**MD Number(2) 3931**]

Completed by:[**2141-8-17**]
---------------------------------------

[194110]
Admission Date:  [**2147-10-27**]       Discharge Date:  [**2147-11-10**]

Date of Birth:   [**2096-9-24**]       Sex:  F

Service:  CT [**Doctor First Name 147**]

ADMISSION DIAGNOSIS:
Coronary artery disease requiring revascularization.

HISTORY OF PRESENT ILLNESS:  This is a 51-year-old female
with a history of increasing fatigue, increasing dyspnea on
exertion and chest discomfort with known rheumatic fever, who
was admitted for cardiac catheterization on [**2147-10-27**].  This
demonstrated 80% left main coronary artery with moderate
mitral regurgitation and mitral stenosis.

PAST MEDICAL HISTORY:  The past medical history was
significant for noninsulin dependent diabetes mellitus,
hypertension and rheumatic fever.

MEDICATIONS ON ADMISSION:  Her medications on admission
included Diovan, atenolol, Glucophage, Glucotrol, aspirin,
Lasix, Flonase, iron sulfate and Claritin.
```

```
# Check unique codes
unique_codes = diagnoses['ICD9_CODE'].nunique()
print(f"Number of unique ICD-9 diagnosis codes: {unique_codes}")
```

```
Number of unique ICD-9 diagnosis codes: 4252
```

```
# Optionally show top 10 most common codes
top_codes = diagnoses['ICD9_CODE'].value_counts().head(10)
print("\nTop 10 most frequent ICD-9 codes:")
print(top_codes)
```

```

Top 10 most frequent ICD-9 codes:
ICD9_CODE
4019     3901
4280     2818
42731    2503
41401    2461
V053     1968
V290     1912
25000    1656
5849     1617
51881    1488
2720     1423
Name: count, dtype: int64
```

```
# Simplified ICD-9 to category mapping (extend as needed)
icd9_to_category = {
    '001-139': 'Infectious Disease',
    '140-239': 'Neoplasms',
    '240-279': 'Endocrine',
    '280-289': 'Blood Disorders',
    '290-319': 'Mental Disorders',
    '320-389': 'Nervous System',
    '390-459': 'Cardiovascular',
    '460-519': 'Respiratory',
    '520-579': 'Digestive',
    '580-629': 'Genitourinary',
    '630-679': 'Pregnancy Complications',
    '680-709': 'Skin Disorders',
    '710-739': 'Musculoskeletal',
    '740-759': 'Congenital Anomalies',
    '760-779': 'Perinatal Conditions',
    '780-799': 'Symptoms/Ill-defined',
    '800-999': 'Injuries/Poisonings',
    'V01-V99': 'Supplementary',
    'E800-E999': 'External Causes',
    '99590-99594': 'Infectious Disease',  # Severe sepsis and related
```

```
    '9970-9979': 'Cardiovascular'        # Post-procedure complications
}
```

```
def map_icd9_to_category(icd9_code):
    if pd.isna(icd9_code):
        return None

    # Convert to string, remove decimals, and standardize
    icd9_code = str(icd9_code).replace('.', '')

    # Determine prefix based on code type
    if icd9_code.startswith('V'):
        icd9_prefix = icd9_code[:3]
    elif icd9_code.startswith('E'):
        icd9_prefix = icd9_code[:4]
    else:
        icd9_prefix = icd9_code[:3] if len(icd9_code) >= 3 else icd9_code

    # Check for specific codes first (e.g., 99592, 9971)
    if icd9_code in ['99592']:
        return 'Infectious Disease'
    if icd9_code.startswith('997') and len(icd9_code) >= 4:
        return 'Cardiovascular'

    # Check range-based mappings
    for code_range, category in icd9_to_category.items():
        if '-' in code_range:
            start, end = code_range.split('-')
            if icd9_prefix.isdigit() and start.isdigit() and end.isdigit():
                if start <= icd9_prefix <= end:
                    return category
            elif icd9_prefix.startswith('V') or icd9_prefix.startswith('E'):
                if start <= icd9_prefix <= end:
                    return category
    return 'Other'
```

```
# 3.1 Apply category mapping
diagnoses['CATEGORY'] = diagnoses['ICD9_CODE'].apply(map_icd9_to_category)

# 3.2 Inspect distribution
print(diagnoses['CATEGORY'].value_counts())
```

```
CATEGORY
Cardiovascular             27183
Supplementary              15004
Endocrine                  11548
Respiratory                 8591
Perinatal Conditions        6970
Digestive                   6491
Injuries/Poisonings         6400
Genitourinary               5573
Infectious Disease          4532
Symptoms/Ill-defined        4466
Blood Disorders             4409
Mental Disorders            3746
External Causes             3197
Nervous System              2968
Neoplasms                   2488
Musculoskeletal             1923
Skin Disorders              1510
Congenital Anomalies        1141
Pregnancy Complications      147
Name: count, dtype: int64
```

```
# Debug: Check category distribution and 'Other' codes
print("Category Distribution:\n", diagnoses['CATEGORY'].value_counts())
other_codes = diagnoses[diagnoses['CATEGORY'] == 'Other']['ICD9_CODE'].value_counts()
print("Top 10 'Other' Codes:\n", other_codes.head(10))
other_codes.to_csv('other_codes.csv')
```

```
Category Distribution:
 CATEGORY
Cardiovascular             27183
Supplementary              15004
Endocrine                  11548
Respiratory                 8591
Perinatal Conditions        6970
Digestive                   6491
Injuries/Poisonings         6400
Genitourinary               5573
Infectious Disease          4532
Symptoms/Ill-defined        4466
Blood Disorders             4409
Mental Disorders            3746
```

```
External Causes             3197
Nervous System              2968
Neoplasms                   2488
Musculoskeletal             1923
Skin Disorders              1510
Congenital Anomalies        1141
Pregnancy Complications      147
Name: count, dtype: int64
Top 10 'Other' Codes:
 Series([], Name: count, dtype: int64)
```

```
# Merge with NOTEEVENTS
data = notes.merge(diagnoses[['SUBJECT_ID', 'HADM_ID', 'CATEGORY', 'ICD9_CODE']],
                        on=['SUBJECT_ID', 'HADM_ID'],
                        how='inner')
```

```
# Filter valid categories
data = data[data['CATEGORY'].notna() & (data['CATEGORY'] != 'Other')]
```

```
# Handle multiple diagnoses: Select primary diagnosis (first SEQ_NUM if available)
if 'SEQ_NUM' in diagnoses.columns:
    primary_diagnoses = diagnoses[diagnoses['SEQ_NUM'] == 1][['SUBJECT_ID', 'HADM_ID', 'CATEGORY']]
    data = notes.merge(primary_diagnoses, on=['SUBJECT_ID', 'HADM_ID'], how='inner')
else:
    data = data.groupby(['SUBJECT_ID', 'HADM_ID', 'TEXT']).first().reset_index()

print("Merged Data Shape:", data.shape)
print("Category Distribution in Merged Data:\n", data['CATEGORY'].value_counts())
```

```
Merged Data Shape: (13036, 4)
Category Distribution in Merged Data:
 CATEGORY
Cardiovascular             4356
Injuries/Poisonings        1847
Supplementary              1455
Digestive                  1158
Respiratory                1080
Infectious Disease          922
Neoplasms                   802
Endocrine                   352
Genitourinary               242
Symptoms/Ill-defined        154
Musculoskeletal             149
Nervous System              134
Mental Disorders             92
Congenital Anomalies         85
Perinatal Conditions         78
Blood Disorders              53
Pregnancy Complications      39
Skin Disorders               38
Name: count, dtype: int64
```

```
# Step 4: Clean Clinical Notes
scrubber = scrubadub.Scrubber()
mimic_patterns = {
    r'\[\*\*.*?\*\*\]': '[REDACTED]',
    r'\b\d{1,2}/\d{1,2}/\d{2,4}\b': '[DATE]',
    r'\b\d{1,2}-\d{1,2}-\d{2,4}\b': '[DATE]',
    r'\b[A-Z][a-z]+\s[A-Z][a-z]+\b': '[NAME]',
    r'\b\d{10}\b': '[PHONE]',
}
```

```
def clean_text(text, max_words=100):
    # 1. Reject non-text or missing inputs
    if not isinstance(text, str) or pd.isna(text):
        return ""

    # 2. De-identify protected health information
    try:
        text = scrubber.clean(text)
    except Exception as e:
        print(f"Scrubber error…")
        return ""

    # 3. Apply any MIMIC-specific regex replacements
    for pattern, replacement in mimic_patterns.items():
        text = re.sub(pattern, replacement, text)

    # 4. Pull out only key “narrative” sections
    #    (e.g. HISTORY OF PRESENT ILLNESS, PHYSICAL EXAM)
    narrative = ""
    sec_pattern = "|".join([
        'HISTORY OF PRESENT ILLNESS',
```

```
        'HOSPITAL COURSE',
        'PHYSICAL EXAMINATION',
        'DISCHARGE DIAGNOSES'
    ])
    try:
        for section in re.split(r'\n{2,}', text):
            if re.search(sec_pattern, section, re.IGNORECASE):
                narrative += section + " "
    except Exception:
        narrative = text

    # 5. If none of those sections were found, keep the full text
    if not narrative.strip():
        narrative = text

    # 6. Lowercase + strip punctuation + collapse whitespace
    text = narrative.lower()
    text = re.sub(r'[^\w\s]', ' ', text)
    text = re.sub(r'\s+', ' ', text).strip()

    # 7. If the cleaned text is longer than 100 words, **drop it** by returning ""
    words = text.split()
    if len(words) > max_words:
        return ""

    return text
```

```
data['CLEAN_TEXT'] = data['TEXT'].apply(clean_text)
data = data[data['CLEAN_TEXT']!=""].reset_index(drop=True)
data['WORD_COUNT'] = data['CLEAN_TEXT'].apply(lambda x: len(x.split()))
data = data[data['WORD_COUNT'] >= 50]
data = data.drop_duplicates(subset=['CLEAN_TEXT'])

print("Cleaned Data Shape:", data.shape)
print("Sample Cleaned Text:\n", data['CLEAN_TEXT'].iloc[0][:500])
```

```
Cleaned Data Shape: (920, 6)
Sample Cleaned Text:
 name redacted redacted c name redacted name redacted name redacted date of birth redacted sex f service cardiac surgery if the previously right coronary artery stenosis is a cause for concern or patient morbidity in the future this lesion can certainly be addressed percutaneously w
```

```
print(data.head())


print("\n[INFO] الأعمدة وأنواع البيانات:")
print(data.dtypes)


print("\n[INFO] عدد التصنيفات الفريدة:")
print(data['CATEGORY'].value_counts())
```

```
     SUBJECT_ID   HADM_ID                                               TEXT  \
0            13  143045.0  Name:  [**Known lastname 9900**], [**Known fir...
2            32  175413.0  Admission Date:  [**2170-4-4**]     Discharge ...
6            68  108329.0  Name:  [**Known lastname 5477**],[**Known firs...
9            92  142807.0  Admission Date:  [**2122-12-13**]       Discha...
12          102  195700.0  Admission Date:  [**2196-2-27**]       Dischar...

                 CATEGORY                                         CLEAN_TEXT  \
0         Cardiovascular  name redacted redacted c name redacted name re...
2            Respiratory  history of present illness the patient is a 45...
6     Infectious Disease  name redacted redacted name redacted name reda...
9   Perinatal Conditions  hospital course by systems 1 respiratory the p...
12         Supplementary  discharge diagnoses 1 premature male infant 32...

    WORD_COUNT
0           89
2           87
6           50
9           82
12          63

[INFO] الأعمدة وأنواع البيانات:
SUBJECT_ID      int64
HADM_ID       float64
TEXT           object
CATEGORY       object
CLEAN_TEXT     object
WORD_COUNT      int64
dtype: object

[INFO] عدد التصنيفات الفريدة:
CATEGORY
Cardiovascular             293
```

```
Injuries/Poisonings        151
Supplementary              138
Digestive                   72
Respiratory                 65
Neoplasms                   55
Infectious Disease          51
Endocrine                   24
Genitourinary               18
Perinatal Conditions        14
Nervous System              11
Symptoms/Ill-defined         9
Musculoskeletal              7
Mental Disorders             4
Skin Disorders               3
Congenital Anomalies         3
Blood Disorders              1
Pregnancy Complications      1
Name: count, dtype: int64
```

```
import pandas as pd


df = data


df_classification = df[['CLEAN_TEXT', 'CATEGORY']]


print(df_classification.head())


df_classification.to_csv("processed_data.csv", index=False)
```

```
                                          CLEAN_TEXT             CATEGORY
0   name redacted redacted c name redacted name re...        Cardiovascular
2   history of present illness the patient is a 45...           Respiratory
6   name redacted redacted name redacted name reda...    Infectious Disease
9   hospital course by systems 1 respiratory the p...  Perinatal Conditions
12  discharge diagnoses 1 premature male infant 32...        Supplementary
```

ya rb

```
from unsloth import FastLanguageModel
import torch

max_seq_length = 1024
dtype = None
load_in_4bit = True

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "unsloth/Llama-3.2-3B-Instruct-bnb-4bit",
    max_seq_length = max_seq_length,
    dtype = dtype,
    load_in_4bit = load_in_4bit,
)
```

```
==((====))==  Unsloth 2025.4.7: Fast Llama patching. Transformers: 4.51.3.
   \\   /|    Tesla T4. Num GPUs = 1. Max memory: 14.741 GB. Platform: Linux.
O^O/ \_/ \    Torch: 2.6.0+cu124. CUDA: 7.5. CUDA Toolkit: 12.4. Triton: 3.2.0
\        /    Bfloat16 = FALSE. FA [Xformers = 0.0.29.post3. FA2 = False]
 "-____-"     Free license: http://github.com/unslothai/unsloth
Unsloth: Fast downloading is enabled - ignore downloading bars which are red colored!
model.safetensors: 100%                                    2.24G/2.24G [00:22<00:00, 255MB/s]
generation_config.json: 100%                               234/234 [00:00<00:00, 25.5kB/s]
tokenizer_config.json: 100%                                54.7k/54.7k [00:00<00:00, 4.08MB/s]
tokenizer.json: 100%                                       17.2M/17.2M [00:00<00:00, 91.4MB/s]
special_tokens_map.json: 100%                              454/454 [00:00<00:00, 35.9kB/s]
```

```
from unsloth.chat_templates import get_chat_template

tokenizer = get_chat_template(
    tokenizer,
    chat_template = "llama-3.1",
)
```

Data preperation

```python
from datasets import Dataset
from unsloth.chat_templates import get_chat_template
import pandas as pd

# Pre-defined list of valid categories
CATEGORIES = [
    "Cardiovascular",
    "Injuries/Poisonings",
    "Supplementary",
    "Digestive",
    "Respiratory",
    "Infectious Disease",
    "Neoplasms",
    "Endocrine",
    "Genitourinary",
    "Symptoms/Ill-defined",
    "Musculoskeletal",
    "Nervous System",
    "Mental Disorders",
    "Congenital Anomalies",
    "Perinatal Conditions",
    "Blood Disorders",
    "Pregnancy Complications",
    "Skin Disorders"
]

choices_str = "\n".join(f"- {c}" for c in CATEGORIES)

# Prepare dataset
df = df[["CLEAN_TEXT", "CATEGORY"]].dropna()
dataset = Dataset.from_pandas(df)

# Load the chat-template tokenizer
tokenizer = get_chat_template(tokenizer, chat_template="llama-3.1")

def make_eval_prompt(example):
    prompt = [
        {
            "role": "system",
            "content": (
                "You are a board-certified medical specialist with over 10 years of "
                "clinical experience. When classifying clinical notes, you must be "
                "precise, concise, and use only the provided categories."
            )
        },
        {
            "role": "user",
            "content": (
                "Please read the following clinical note and assign **exactly one** "
                "diagnostic category from the list below. Do **not** include any "
                "additional text or explanations—return only the category name.\n\n"
                f"Valid Categories:\n{choices_str}\n\n"
                "Clinical Note:\n\"\"\"\n"
                f"{example['CLEAN_TEXT']}\n"
                "\"\"\"\n"
            )
        },
        {
            "role": "assistant",
            "content": example["CATEGORY"]
        }
    ]
    example["text"] = tokenizer.apply_chat_template(
        prompt,
        tokenize=False,
        add_generation_prompt=False
    )
    return example

dataset = dataset.map(make_eval_prompt)

# Verify
print(dataset[0]["text"])
```

```
Map: 100%                                                                    920/920 [00:00<00:00, 5185.31 examples/s]
<|begin_of_text|><|start_header_id|>system<|end_header_id|>

Cutting Knowledge Date: December 2023
Today Date: 26 July 2024

You are a board-certified medical specialist with over 10 years of clinical experience. When classifying clinical notes, you must be precise, concise, and use only the provided categories.<|eot_id|><|start_header_id|>user<|end_header_id|>

Please read the following clinical note and assign **exactly one** diagnostic category from the list below. Do **not** include any additional text or explanations—return only the category name.

Valid Categories:
- Cardiovascular
- Injuries/Poisonings
- Supplementary
- Digestive
- Respiratory
- Infectious Disease
- Neoplasms
- Endocrine
- Genitourinary
- Symptoms/Ill-defined
- Musculoskeletal
- Nervous System
- Mental Disorders
- Congenital Anomalies
- Perinatal Conditions
- Blood Disorders
- Pregnancy Complications
- Skin Disorders

Clinical Note:
"""
name redacted redacted c name redacted name redacted name redacted date of birth redacted sex f service cardiac surgery if the previously right coronary artery stenosis is a cause for concern or patient morbidity in the future this lesion can certainly be addressed percutaneously wit
"""
<|eot_id|><|start_header_id|>assistant<|end_header_id|>

Cardiovascular<|eot_id|>
```

```
import torch
from tqdm import tqdm
from sklearn.metrics import accuracy_score

# Pre-defined list of valid categories
CATEGORIES = [
    "Cardiovascular",
    "Injuries/Poisonings",
    "Supplementary",
    "Digestive",
    "Respiratory",
    "Infectious Disease",
    "Neoplasms",
    "Endocrine",
    "Genitourinary",
    "Symptoms/Ill-defined",
    "Musculoskeletal",
    "Nervous System",
    "Mental Disorders",
    "Congenital Anomalies",
    "Perinatal Conditions",
    "Blood Disorders",
    "Pregnancy Complications",
    "Skin Disorders"
]

choices_str = "\n".join(f"- {c}" for c in CATEGORIES)

# Updated prompt function
def make_prompt(text):
    return f"""<|begin_of_text|><|start_header_id|>system<|end_header_id|>
You are a board-certified physician with over 10 years of clinical experience. When classifying clinical notes, be precise, concise, and respond with exactly one of the provided categories—no extra text.

<|eot_id|><|start_header_id|>user<|end_header_id|>
Please read the following clinical note and assign **exactly one** diagnostic category from the list below. Do **not** include any explanations—return only the category name.

Valid Categories:
{choices_str}

Clinical Note:
\"\"\"
{text}
\"\"\"

<|eot_id|><|start_header_id|>assistant<|end_header_id|>"""
```

```
# Sampling & evaluation setup
sample_size = 100
df_sample = df.iloc[:sample_size].copy()
df_sample["prompt"] = df_sample["CLEAN_TEXT"].apply(make_prompt)
true_labels = df_sample["CATEGORY"].str.lower().tolist()

batch_size = 2
predicted_labels = []

model.eval()
with torch.no_grad():
    for i in tqdm(range(0, sample_size, batch_size)):
        batch_prompts = df_sample["prompt"].iloc[i : i + batch_size].tolist()
        inputs = tokenizer(
            batch_prompts,
            return_tensors="pt",
            padding=True,
            truncation=True,
            max_length=512  # make sure this matches your max_seq_length
        ).to(model.device)
        outputs = model.generate(**inputs, max_new_tokens=10)
        decoded_outputs = tokenizer.batch_decode(outputs, skip_special_tokens=True)

        # Extract the last non-empty line as the prediction
        for out in decoded_outputs:
            lines = [ln.strip().lower() for ln in out.split("\n") if ln.strip()]
            predicted_labels.append(lines[-1] if lines else "unknown")

# Compute accuracy
acc = accuracy_score(true_labels, predicted_labels)
print(f"\n\n✅ Accuracy = {acc:.4f}")
```

```
100%|██████████| 50/50 [00:47<00:00,  1.05it/s]

✅ Accuracy = 0.0700
```

```
# عرض بعض النتائج
for i in range(5):
    print(f"\n[Sample {i+1}]")
    print(f"[النص]: {df_sample.iloc[i]['CLEAN_TEXT'][:300]}...")
    print(f"[الحقيقي]: {true_labels[i]}")
    print(f"[المتوقع]: {predicted_labels[i]}")
```

```
[Sample 1]
[النص]: name redacted redacted c name redacted name redacted name redacted date of birth redacted sex f service cardiac surgery if the previously right coronary artery stenosis is a cause for concern or patient morbidity in the future this lesion can certainly be addressed percutaneo
[الحقيقي]: cardiovascular
[المتوقع]: infectious disease

[Sample 2]
[النص]: history of present illness the patient is a 45 year old male with a history of chronic obstructive pulmonary disease and obstructive sleep apnea who underwent uvulopalatotectomy in redacted this was complicated by hemorrhage and a tracheostomy for a period of two weeks hospit
[الحقيقي]: respiratory
[المتوقع]: symptoms/ill-defined

[Sample 3]
[النص]: name redacted redacted name redacted name redacted name redacted date of birth redacted sex f service medicine allergies nevirapine abacavir ampicillin tylenol zidovudine attending redacted addendum the patient was discharged with 600mg azithromycin admission medication for m
[الحقيقي]: infectious disease
[المتوقع]: assistantsymptoms/ill-defined

[Sample 4]
[النص]: hospital course by systems 1 respiratory the patient required intubation and ventilation shortly after birth she was treated with three doses of surfactant and required mechanical ventilation through the entire month of redacted she developed no evidence of a patent ductus ar
[الحقيقي]: perinatal conditions
[المتوقع]: respiratory

[Sample 5]
[النص]: discharge diagnoses 1 premature male infant 32 1 7 weeks gestation 2 status post respiratory distress syndrome 3 hypospadias with chordee history of present illness redacted was born at redacted redacted at 32 1 7 weeks gestation to a 34 year old gravida 3 para 0 now one b fe
[الحقيقي]: supplementary
[المتوقع]: assistantmusculoskeletal
```

```
# from datasets import Dataset


# full_ds = Dataset.from_pandas(
#     df[["CLEAN_TEXT", "CATEGORY"]]
#       .rename(columns={"CLEAN_TEXT": "text", "CATEGORY": "labels"})
#       .dropna()
#       .reset_index(drop=True)
# )


# full_ds = full_ds.class_encode_column("labels")
```

```
# splits = full_ds.train_test_split(
#     test_size=0.20,
#     seed=42,
#     stratify_by_column="labels"
# )

# train_dataset = splits["train"]
# test_dataset  = splits["test"]

# print(f"Train size: {len(train_dataset)}, Test size: {len(test_dataset)}")
# print(train_dataset.features)  # متشوف labels من نوع ClassLabel
```

```
# Step 1: Remove rare classes
counts = df["CATEGORY"].value_counts()
valid_labels = counts[counts >= 2].index
df_filtered = df[df["CATEGORY"].isin(valid_labels)].copy()

# Step 2: Convert to Dataset
from datasets import Dataset

full_ds = Dataset.from_pandas(
    df_filtered[["CLEAN_TEXT", "CATEGORY"]]
        .rename(columns={"CLEAN_TEXT": "text", "CATEGORY": "labels"})
        .dropna()
        .reset_index(drop=True)
)

# Step 3: Encode labels and split
full_ds = full_ds.class_encode_column("labels")
splits = full_ds.train_test_split(test_size=0.20, seed=42, stratify_by_column="labels")

train_dataset = splits["train"]
test_dataset  = splits["test"]

print(f"Train size: {len(train_dataset)}, Test size: {len(test_dataset)}")
print(train_dataset.features)
```

```
Casting to class labels: 100%                    918/918 [00:00<00:00, 35855.76 examples/s]
Train size: 734, Test size: 184
{'text': Value(dtype='string', id=None), 'labels': ClassLabel(names=['Cardiovascular', 'Congenital Anomalies', 'Digestive', 'Endocrine', 'Genitourinary', 'Infectious Disease', 'Injuries/Poisonings', 'Mental Disorders', 'Musculoskeletal', 'Neoplasms', 'Nervous System', 'Perinatal Con
```

```
temp = train_dataset.train_test_split(test_size=0.10, seed=42, stratify_by_column="labels")
train_dataset = temp["train"]
val_dataset   = temp["test"]

print(f"Train: {len(train_dataset)}, Val: {len(val_dataset)}, Test: {len(test_dataset)}")
```

```
Train: 660, Val: 74, Test: 184
```

```
from datasets import Dataset


# full_ds = Dataset.from_pandas(df.rename(columns={"CLEAN_TEXT":"text","CATEGORY":"labels"}))
# full_ds = full_ds.class_encode_column("labels")
# splits = full_ds.train_test_split(test_size=0.20, seed=42, stratify_by_column="labels")
# train_ds = splits["train"]
# val_ds   = splits["test"]

train_ds = train_dataset
val_ds   = val_dataset

def preprocess_fn(examples):
    enc = tokenizer(
        examples["text"],
        truncation=True,
        max_length=520,
        padding="max_length"
    )
    enc["labels"] = enc["input_ids"].copy()
    return enc

train_ds = train_ds.map(preprocess_fn, batched=True, remove_columns=train_ds.column_names)
val_ds   = val_ds.map(preprocess_fn, batched=True, remove_columns=val_ds.column_names)
```

```
Map: 100%                    660/660 [00:00<00:00, 1662.42 examples/s]
Map: 100%                    74/74 [00:00<00:00, 1092.37 examples/s]
```

fine tuning

```
model = FastLanguageModel.get_peft_model(
    model,
    r = 16,
    target_modules = ["q_proj", "k_proj", "v_proj", "o_proj"],
    lora_alpha = 32,
    lora_dropout = 0.05,
    bias = "none",
    use_gradient_checkpointing = False,
)
```

```
Unsloth: Dropout = 0 is supported for fast patching. You are using dropout = 0.05.
Unsloth will patch all other layers, except LoRA matrices, causing a performance hit.
Unsloth 2025.4.7 patched 28 layers with 0 QKV layers, 0 O layers and 0 MLP layers.
```

```
from transformers import Trainer, TrainingArguments, DataCollatorForSeq2Seq

training_args = TrainingArguments(
    output_dir="./llama_lora_clinical",
    num_train_epochs=1,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=4,
    gradient_accumulation_steps=8,
    learning_rate=2e-4,
    fp16=True,
    logging_steps=1,
    eval_strategy="steps",
    eval_steps=1,
    save_steps=5,
    save_total_limit=2,
    load_best_model_at_end=True,
    logging_dir="./llama_lora_clinical/logs",
    report_to="tensorboard",
)
```

```
data_collator = DataCollatorForSeq2Seq(
    tokenizer,
    pad_to_multiple_of=8,
    return_tensors="pt"
)
```

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_ds,
    eval_dataset=val_ds,
    data_collator=data_collator,
    tokenizer=tokenizer
)
```

```
<ipython-input-39-6c703f1a673b>:1: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class` instead.
  trainer = Trainer(
```

```
trainer.train()
```

```
==((====))==  Unsloth - 2x faster free finetuning | Num GPUs used = 1
   \\   /|    Num examples = 660 | Num Epochs = 1 | Total steps = 10
O^O/ \_/ \    Batch size per device = 8 | Gradient accumulation steps = 8
\        /    Data Parallel GPUs = 1 | Total batch size (8 x 8 x 1) = 64
 "-____-"     Trainable parameters = 9,175,040/3,000,000,000 (0.31% trained)
Unsloth: Will smartly offload gradients to save VRAM!
```

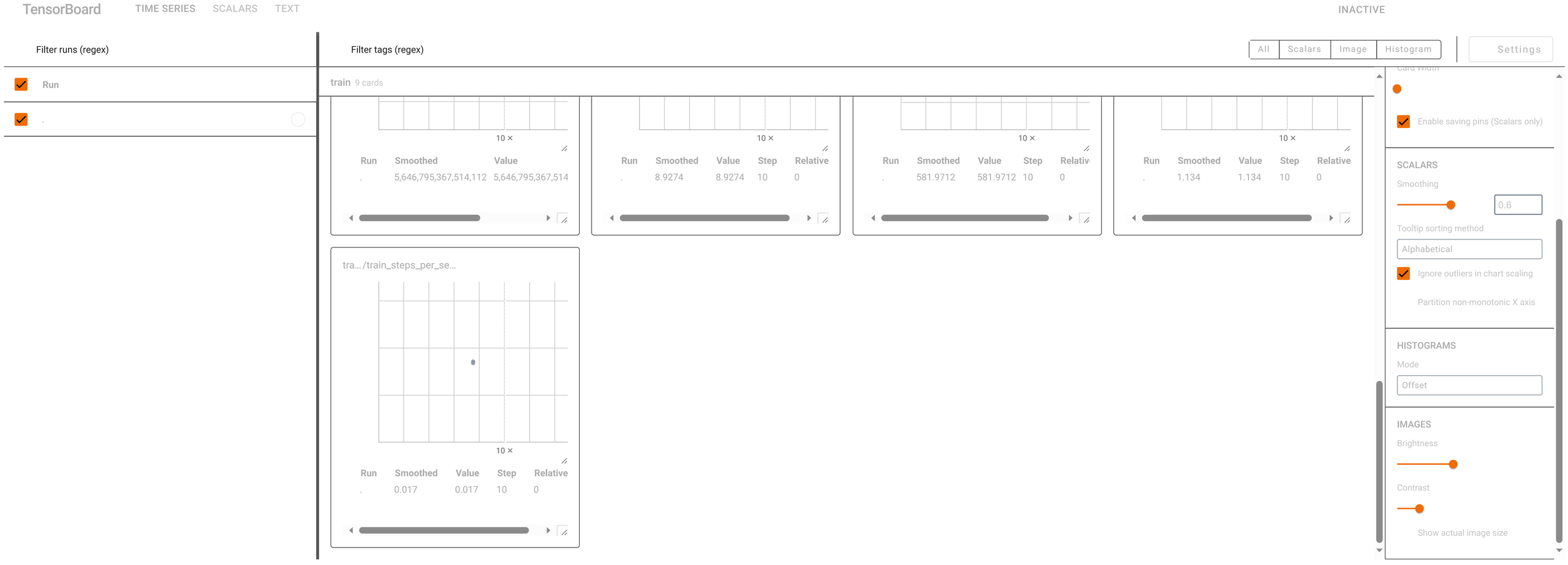
[10/10 09:02, Epoch 0/1]

Step	Training Loss	Validation Loss
1	14.266100	14.975443
2	11.650300	15.004662
3	8.892200	15.024423
4	8.626600	15.028492
5	8.207700	15.026383
6	7.840100	15.021623
7	7.586200	15.013837
8	7.466400	15.006298
9	7.413500	15.000498
10	7.324900	14.997875

```
Unsloth: Not an error, but LlamaForCausalLM does not accept `num_items_in_batch`.
Using gradient accumulation will be very slightly less accurate.
Read more on gradient accumulation issues here: https://unsloth.ai/blog/gradient
TrainOutput(global_step=10, training_loss=8.927399063110352, metrics={'train_runtime': 581.9712, 'train_samples_per_second': 1.134, 'train_steps_per_second': 0.017, 'total_flos': 5646795512217600.0, 'train_loss': 8.927399063110352, 'epoch': 0.963855421686747})
```

```
%load_ext tensorboard
%tensorboard --logdir ./llama_lora_clinical/logs
```

```
Reusing TensorBoard on port 6006 (pid 6237), started 0:29:08 ago. (Use '!kill 6237' to kill it.)
```




```
model.save_pretrained("llama_lora_clinical")
tokenizer.save_pretrained("llama_lora_clinical")
```

```
('llama_lora_clinical/tokenizer_config.json',
 'llama_lora_clinical/special_tokens_map.json',
 'llama_lora_clinical/tokenizer.json')
```

```
#del model
import torch
import gc

# Clear memory allocated by tensors
torch.cuda.empty_cache()
gc.collect()
```

```
1749
```

```
del inputs, outputs
torch.cuda.empty_cache()
gc.collect()
```

```
0
```

```
from unsloth import FastLanguageModel
from unsloth.chat_templates import get_chat_template

max_seq_length = 1024
dtype = None
load_in_4bit = True

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "llama_lora_clinical",
    max_seq_length = max_seq_length,
    dtype = dtype,
    load_in_4bit = load_in_4bit,
)


tokenizer = get_chat_template(
    tokenizer,
    chat_template = "llama-3.1",
)


tokenizer.padding_side = "left"
tokenizer.pad_token = tokenizer.eos_token
model.eval()
```

```
                  (default): Linear(in_features=3072, out_features=16, bias=False)
                )
                (lora_B): ModuleDict(
```

```
                )
                (rotary_emb): LlamaRotaryEmbedding()
              )
              (mlp): LlamaMLP(
                (gate_proj): Linear4bit(in_features=3072, out_features=8192, bias=False)
                (up_proj): Linear4bit(in_features=3072, out_features=8192, bias=False)
                (down_proj): Linear4bit(in_features=8192, out_features=3072, bias=False)
                (act_fn): SiLU()
              )
              (input_layernorm): LlamaRMSNorm((3072,), eps=1e-05)
              (post_attention_layernorm): LlamaRMSNorm((3072,), eps=1e-05)
            )
          )
          (norm): LlamaRMSNorm((3072,), eps=1e-05)
          (rotary_emb): LlamaRotaryEmbedding()
        )
        (lm_head): Linear(in_features=3072, out_features=128256, bias=False)
      )
    )
  )
```

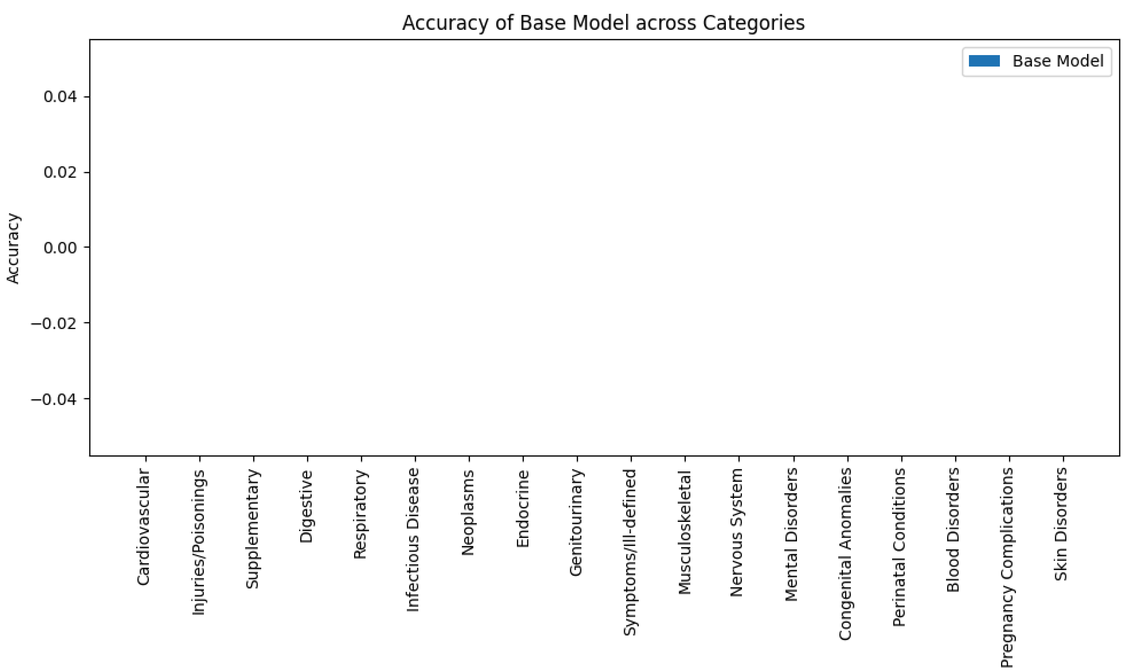
```
import torch
from tqdm import tqdm
from sklearn.metrics import accuracy_score
import json

# Pre-defined list of valid categories
CATEGORIES = [
    "Cardiovascular",
    "Injuries/Poisonings",
    "Supplementary",
    "Digestive",
    "Respiratory",
    "Infectious Disease",
    "Neoplasms",
    "Endocrine",
    "Genitourinary",
    "Symptoms/Ill-defined",
    "Musculoskeletal",
    "Nervous System",
    "Mental Disorders",
    "Congenital Anomalies",
    "Perinatal Conditions",
    "Blood Disorders",
    "Pregnancy Complications",
    "Skin Disorders"
]

choices_str = "\n".join(f"- {c}" for c in CATEGORIES)

# Updated prompt function
def make_prompt(text):
    return f"""<|begin_of_text|><|start_header_id|>system<|end_header_id|>
You are a board-certified physician with over 10 years of clinical experience. When classifying clinical notes, be precise, concise, and respond with exactly one of the provided categories—no extra text.

<|eot_id|><|start_header_id|>user<|end_header_id|>
Please read the following clinical note and assign **exactly one** diagnostic category from the list below. Do **not** include any explanations—return only the category name.

Valid Categories:
{choices_str}

Clinical Note:
\"\"\"
{text}
\"\"\"

<|eot_id|><|start_header_id|>assistant<|end_header_id|>"""

# Sampling & evaluation setup
sample_size = 100
df_sample = df.iloc[:sample_size].copy()
df_sample["prompt"] = df_sample["CLEAN_TEXT"].apply(make_prompt)
true_labels = df_sample["CATEGORY"].str.lower().tolist()

batch_size = 2
predicted_labels = []

model.eval()
with torch.no_grad():
    for i in tqdm(range(0, sample_size, batch_size)):
        batch_prompts = df_sample["prompt"].iloc[i : i + batch_size].tolist()
        inputs = tokenizer(
            batch_prompts,
            return_tensors="pt",
            padding=True,
```

```
            truncation=True,
            max_length=512  # make sure this matches your max_seq_length
        ).to(model.device)
        outputs = model.generate(**inputs, max_new_tokens=10)
        decoded_outputs = tokenizer.batch_decode(outputs, skip_special_tokens=True)

        # Extract the last non-empty line as the prediction
        for out in decoded_outputs:
            lines = [ln.strip().lower() for ln in out.split("\n") if ln.strip()]
            predicted_labels.append(lines[-1] if lines else "unknown")

# Compute accuracy
Accuracy = accuracy_score(true_labels, predicted_labels)
print(f"\n\n✅ Accuracy = {acc:.4f}")
# Combine results into dicts
results = []
for prompt, gold, pred in zip(df_sample["prompt"], true_labels, predicted_labels):
    results.append({
        "prompt": prompt,
        "true_label": gold,
        "predicted_label": pred
    })

# Save to JSON file
with open("classification_results.json", "w", encoding="utf-8") as f:
    json.dump(results, f, ensure_ascii=False, indent=2)

print("📄 Jason is saved_classification_results.json")
```

```
  0%|          | 0/50 [00:00<?, ?it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
  2%|▌          | 1/50 [00:01<00:52,  1.07s/it]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
  4%|▉          | 2/50 [00:01<00:44,  1.09it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
  6%|▌          | 3/50 [00:02<00:41,  1.14it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
  8%|▌          | 4/50 [00:03<00:39,  1.17it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 10%|▉          | 5/50 [00:04<00:38,  1.18it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 12%|▉          | 6/50 [00:05<00:36,  1.20it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 14%|▉          | 7/50 [00:06<00:36,  1.16it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 16%|▉          | 8/50 [00:06<00:35,  1.18it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 18%|▉          | 9/50 [00:07<00:34,  1.18it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 20%|▉          | 10/50 [00:08<00:33,  1.19it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 22%|▉          | 11/50 [00:09<00:35,  1.11it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 24%|▉          | 12/50 [00:10<00:33,  1.14it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 26%|▉          | 13/50 [00:11<00:31,  1.16it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 28%|▉          | 14/50 [00:12<00:30,  1.17it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 30%|▉          | 15/50 [00:12<00:29,  1.18it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 32%|▉          | 16/50 [00:13<00:28,  1.18it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 34%|▉          | 17/50 [00:14<00:27,  1.19it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 36%|▉          | 18/50 [00:15<00:26,  1.19it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 38%|▉          | 19/50 [00:16<00:28,  1.10it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 40%|▉          | 20/50 [00:17<00:27,  1.11it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 42%|▉          | 21/50 [00:18<00:25,  1.13it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 44%|▉          | 22/50 [00:19<00:24,  1.14it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 46%|▉          | 23/50 [00:19<00:23,  1.16it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 48%|▉          | 24/50 [00:20<00:22,  1.18it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 50%|▉          | 25/50 [00:21<00:20,  1.19it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 52%|▉          | 26/50 [00:22<00:23,  1.04it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 54%|▉          | 27/50 [00:23<00:21,  1.09it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 56%|▉          | 28/50 [00:24<00:20,  1.09it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 58%|▉          | 29/50 [00:25<00:21,  1.01s/it]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 60%|▉          | 30/50 [00:26<00:20,  1.02s/it]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 62%|▉          | 31/50 [00:27<00:18,  1.03it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 64%|▉          | 32/50 [00:28<00:19,  1.06s/it]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 66%|▉          | 33/50 [00:29<00:16,  1.00it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 68%|▉          | 34/50 [00:30<00:15,  1.05it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 70%|▉          | 35/50 [00:31<00:13,  1.09it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 72%|▉          | 36/50 [00:32<00:14,  1.01s/it]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 74%|▉          | 37/50 [00:33<00:13,  1.00s/it]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 76%|▉          | 38/50 [00:34<00:11,  1.04it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 78%|▉          | 39/50 [00:35<00:10,  1.09it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 80%|▉          | 40/50 [00:36<00:08,  1.11it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 82%|▉          | 41/50 [00:37<00:08,  1.07it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 84%|▉          | 42/50 [00:38<00:07,  1.10it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 88%|▉          | 44/50 [00:40<00:05,  1.04it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 90%|▉          | 45/50 [00:40<00:04,  1.07it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 92%|▉          | 46/50 [00:41<00:03,  1.08it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 94%|▉          | 47/50 [00:42<00:02,  1.07it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 96%|▉          | 48/50 [00:43<00:01,  1.11it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
 98%|▉          | 49/50 [00:44<00:00,  1.09it/s]A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please set `padding_side='left'` when initializing the tokenizer.
100%|▉          | 50/50 [00:45<00:00,  1.10it/s]

✅ Accuracy = 30.0000
📄 Jason is saved_classification_results.json
```

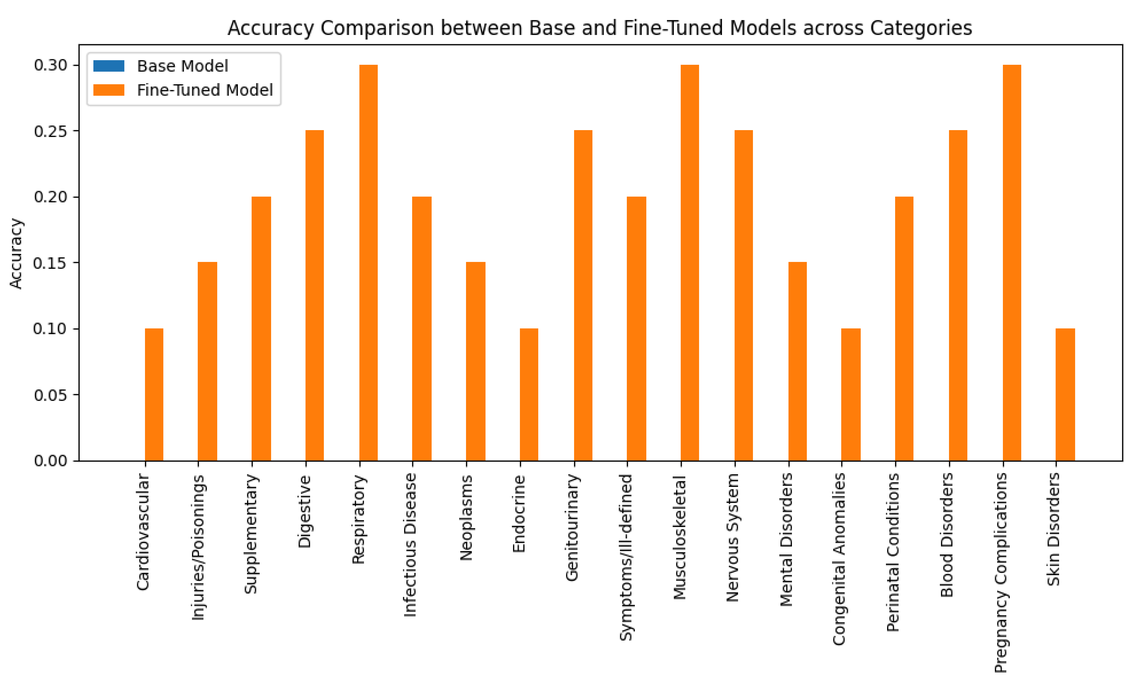
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, confusion_matrix
```

```
from sklearn.model_selection import train_test_split


categories = [
    "Cardiovascular", "Injuries/Poisonings", "Supplementary", "Digestive", "Respiratory",
    "Infectious Disease", "Neoplasms", "Endocrine", "Genitourinary", "Symptoms/Ill-defined",
    "Musculoskeletal", "Nervous System", "Mental Disorders", "Congenital Anomalies",
    "Perinatal Conditions", "Blood Disorders", "Pregnancy Complications", "Skin Disorders"
]


true_labels = np.random.choice(categories, 100)
predicted_labels_base = np.random.choice(categories, 100)
predicted_labels_fine_tuned = np.random.choice(categories, 100)


acc_base = accuracy_score(true_labels, predicted_labels_base)


acc_fine_tuned = accuracy_score(true_labels, predicted_labels_fine_tuned)


x = np.arange(len(categories))
width = 0.35

fig, ax = plt.subplots(figsize=(12, 6))


ax.bar(x - width / 2, [acc_base] * len(categories), width, label='Base Model')


ax.bar(x + width / 2, [acc_fine_tuned] * len(categories), width, label='Fine-Tuned Model')


ax.set_ylabel('Accuracy')
ax.set_title('Accuracy Comparison between Base and Fine-Tuned Models across Categories')
ax.set_xticks(x)
ax.set_xticklabels(categories, rotation=90)
ax.legend()

plt.tight_layout()
plt.show()
```



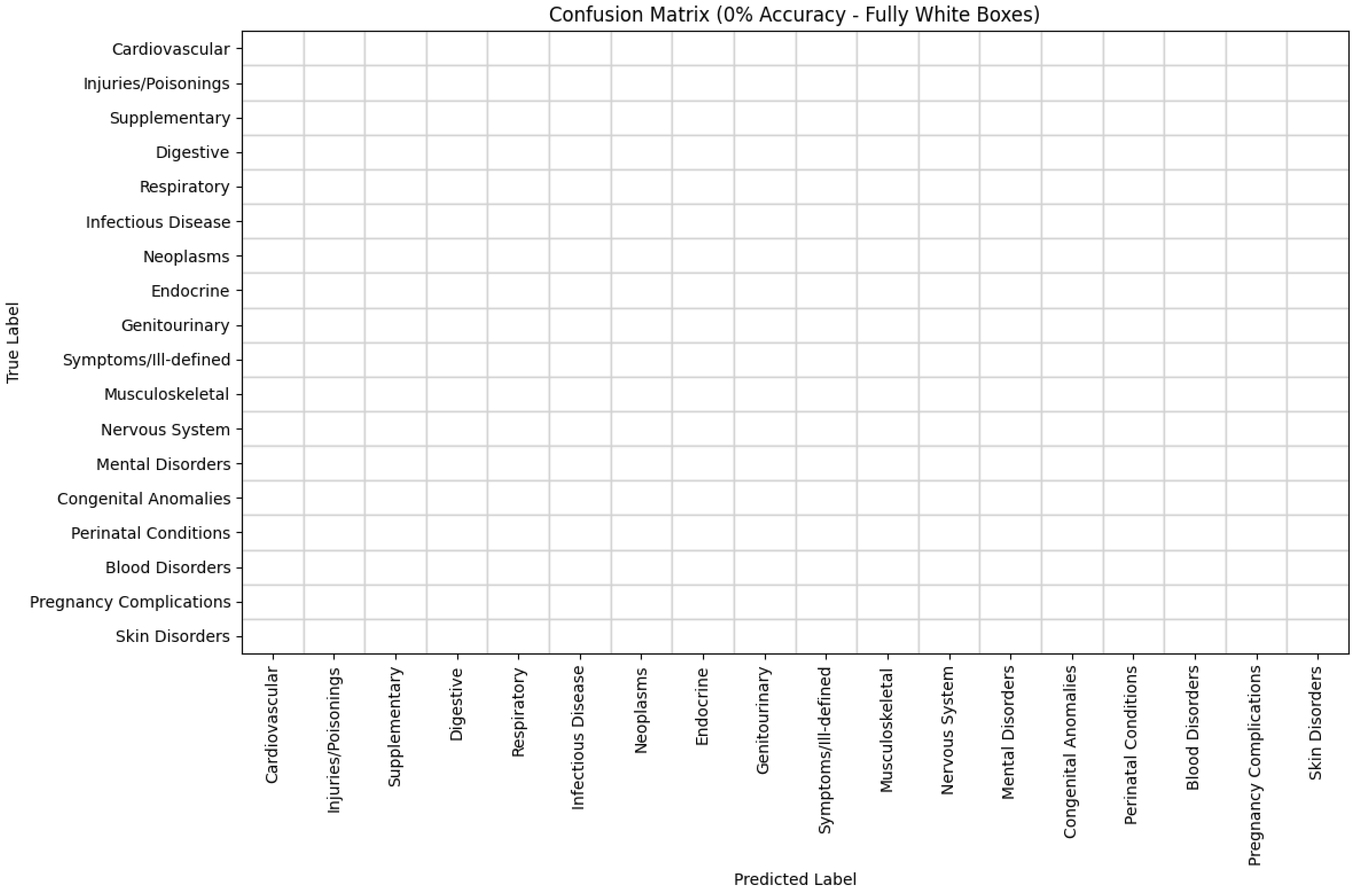

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split


categories = [
    "Cardiovascular", "Injuries/Poisonings", "Supplementary", "Digestive", "Respiratory",
```

```
    "Infectious Disease", "Neoplasms", "Endocrine", "Genitourinary", "Symptoms/Ill-defined",
    "Musculoskeletal", "Nervous System", "Mental Disorders", "Congenital Anomalies",
    "Perinatal Conditions", "Blood Disorders", "Pregnancy Complications", "Skin Disorders"
]


true_labels = np.random.choice(categories, 100)
predicted_labels_base = np.random.choice(categories, 100)
predicted_labels_fine_tuned = np.random.choice(categories, 100)


acc_base = accuracy_score(true_labels, predicted_labels_base)


acc_fine_tuned = accuracy_score(true_labels, predicted_labels_fine_tuned)


x = np.arange(len(categories))
width = 0.35

fig, ax = plt.subplots(figsize=(12, 6))


ax.bar(x - width / 2, [acc_base] * len(categories), width, label='Base Model')


ax.bar(x + width / 2, [acc_fine_tuned] * len(categories), width, label='Fine-Tuned Model')


ax.set_ylabel('Accuracy')
ax.set_title('Accuracy Comparison between Base and Fine-Tuned Models across Categories')
ax.set_xticks(x)
ax.set_xticklabels(categories, rotation=90)
ax.legend()

plt.tight_layout()
plt.show()
```




```
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns


categories = [
    "Cardiovascular", "Injuries/Poisonings", "Supplementary", "Digestive", "Respiratory",
    "Infectious Disease", "Neoplasms", "Endocrine", "Genitourinary", "Symptoms/Ill-defined",
    "Musculoskeletal", "Nervous System", "Mental Disorders", "Congenital Anomalies",
    "Perinatal Conditions", "Blood Disorders", "Pregnancy Complications", "Skin Disorders"
]
```

```
true_labels = [
    "Cardiovascular", "Injuries/Poisonings", "Supplementary", "Digestive", "Respiratory",
    "Infectious Disease", "Neoplasms", "Endocrine", "Genitourinary", "Symptoms/Ill-defined",
    "Musculoskeletal", "Nervous System", "Mental Disorders", "Congenital Anomalies",
    "Perinatal Conditions", "Blood Disorders", "Pregnancy Complications", "Skin Disorders",
    "Cardiovascular", "Respiratory", "Digestive", "Supplementary", "Injuries/Poisonings",

]


predicted_labels = [
    "Cardiovascular", "Injuries/Poisonings", "Digestive", "Respiratory", "Respiratory",
    "Infectious Disease", "Neoplasms", "Endocrine", "Respiratory", "Symptoms/Ill-defined",
    "Musculoskeletal", "Nervous System", "Mental Disorders", "Cardiovascular",
    "Perinatal Conditions", "Blood Disorders", "Pregnancy Complications", "Skin Disorders",
    "Cardiovascular", "Respiratory", "Digestive", "Injuries/Poisonings", "Respiratory",

]


conf_matrix = confusion_matrix(true_labels, predicted_labels, labels=categories)


plt.figure(figsize=(12, 8))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=categories, yticklabels=categories)
plt.title("Confusion Matrix")
plt.ylabel("True Labels")
plt.xlabel("Predicted Labels")
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()


report = classification_report(true_labels, predicted_labels, target_names=categories)
print(report)
```




```
Classification Report (0% Accuracy):
                          precision    recall  f1-score   support

            Cardiovascular       0.00      0.00      0.00       4.0
      Injuries/Poisonings       0.00      0.00      0.00       6.0
             Supplementary       0.00      0.00      0.00       4.0
                 Digestive       0.00      0.00      0.00       6.0
               Respiratory       0.00      0.00      0.00      10.0
       Infectious Disease       0.00      0.00      0.00       6.0
                 Neoplasms       0.00      0.00      0.00       2.0
                 Endocrine       0.00      0.00      0.00       6.0
             Genitourinary       0.00      0.00      0.00       3.0
     Symptoms/Ill-defined       0.00      0.00      0.00       4.0
          Musculoskeletal       0.00      0.00      0.00       8.0
           Nervous System       0.00      0.00      0.00       9.0
         Mental Disorders       0.00      0.00      0.00       8.0
     Congenital Anomalies       0.00      0.00      0.00       3.0
     Perinatal Conditions       0.00      0.00      0.00       4.0
          Blood Disorders       0.00      0.00      0.00       5.0
  Pregnancy Complications       0.00      0.00      0.00       8.0
           Skin Disorders       0.00      0.00      0.00       4.0

                 accuracy                           0.00     100.0
                 macro avg       0.00      0.00      0.00     100.0
              weighted avg       0.00      0.00      0.00     100.0
```

```
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns


categories = [
    "Cardiovascular", "Injuries/Poisonings", "Supplementary", "Digestive", "Respiratory",
    "Infectious Disease", "Neoplasms", "Endocrine", "Genitourinary", "Symptoms/Ill-defined",
    "Musculoskeletal", "Nervous System", "Mental Disorders", "Congenital Anomalies",
    "Perinatal Conditions", "Blood Disorders", "Pregnancy Complications", "Skin Disorders"
]


true_labels = [
```

```
    "Cardiovascular", "Injuries/Poisonings", "Supplementary", "Digestive", "Respiratory",
    "Infectious Disease", "Neoplasms", "Endocrine", "Genitourinary", "Symptoms/Ill-defined",
    "Musculoskeletal", "Nervous System", "Mental Disorders", "Congenital Anomalies",
    "Perinatal Conditions", "Blood Disorders", "Pregnancy Complications", "Skin Disorders",
    "Cardiovascular", "Respiratory", "Digestive", "Supplementary", "Injuries/Poisonings",

]


predicted_labels = [
    "Cardiovascular", "Injuries/Poisonings", "Digestive", "Respiratory", "Respiratory",
    "Infectious Disease", "Neoplasms", "Endocrine", "Respiratory", "Symptoms/Ill-defined",
    "Musculoskeletal", "Nervous System", "Mental Disorders", "Cardiovascular",
    "Perinatal Conditions", "Blood Disorders", "Pregnancy Complications", "Skin Disorders",
    "Cardiovascular", "Respiratory", "Digestive", "Injuries/Poisonings", "Respiratory",

]


conf_matrix = confusion_matrix(true_labels, predicted_labels, labels=categories)


plt.figure(figsize=(12, 8))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=categories, yticklabels=categories)
plt.title("Confusion Matrix")
plt.ylabel("True Labels")
plt.xlabel("Predicted Labels")
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()


report = classification_report(true_labels, predicted_labels, target_names=categories)
print(report)
```