Important Links:

Final Project Google Colab Notebook Link (.ipynb)

Github Link

Video Link

Introduction

Longitudinal Electronic Health Records (EHRs) successfully used for clinical disease and outcome prediction using Deep Learning models. State-of-the-art (SOTA) models outperform traditional ML models by using pretrain-finetune methods in EHR-based predictive modeling. However, their pre-training objectives are limited in predicting fraction of ICD codes within each visit. In real life scenarios, patients have multiple diseases which can be correlated and can contribute to disease progression and change in outcome. Additionally, generalizing the same model on out-of-domain data in different medical settings with limited computing resources is a major challenge today.

Paper explanation

The paper proposes TransformEHR, which is a generative encoder-decoder model with a transformer that is pre-trained using a new strategy, to predict the complete set of diseases and outcomes of patients at a future visit from previous visits. The model is generalizable and can be finetuned for various clinical prediction tasks with limited data.

TransformEHR uses encoder-decoder transformer architecture. The encoder processes the input embeddings and generates a set of hidden representations. The model performs cross-attention over hidden representations from the encoder and assigns an attention weight for each representation. The decoder generates ICD codes following the sequential order of code priority within a visit. It includes the date of each visit as a feature to integrate temporal information. The model uses 3 unique components compared to state-of-the-art models (BERT - Bidirectional Encoder Representations from Transformers) i.e. Visit masking, Encoder-decoder architecture and time embedding.

As per author, during the pretraining with a larger set of longitudinal EHR data, TransformEHR model learned the probability distribution of ICD codes through correlation of cross attention. Later It was fine-tuned to the predictions of a single disease or outcome.

Scope of Reproducibility:

Hypothesis1:

Whether using the TransformEHR model can effectively predict the complete set of diseases and outcomes of a patient from past visits (i.e. Disease or outcome agnostic prediction (DOAP) task). The result will be compared to the state-of-the-art model (BERT - Bidirectional Encoder Representations from Transformers) that is usually trained to predict a fraction of ICD codes within each visit. The model will use a transformer architecture and seek to outperform bidirectional encode-only models.

Hypothesis2:

The other hypothesis could be that the TransformEHR model can perform better for single disease outcomes with pre-training than without pre-training. After carefully reviewing the large data requirement of the VHA dataset for pretraining, we realized that testing this hypothesis may not be feasible with limited computational resources.

Abalations planned:

1. To evaluate the effectiveness of 3 of the unique components of the TransformEHR model, which are visit date masking, encoder-decoder architecture, and time embedding.
2. To assess the performance of an Encoder-only architecture model (BERT) compared to an encoder-decoder (BART) architecture.

Important Instructions Before Running the Code

To get the Dataset mounted on Google Drive:

- Go to MIMIC IV Website
- Download the dataset to your drive
- Validate the dataset is present under -> MyDrive/mimiciv/2.2/hosp. Example:
    - /content/drive/MyDrive/mimiciv/2.2/hosp/admissions.csv.gz
    - /content/drive/MyDrive/mimiciv/2.2/hosp/diagnoses_icd.csv.gz

Imports Modules

```
#Required installations
!pip install --upgrade accelerate
!pip install --upgrade transformers
!pip install --upgrade tqdm
!pip install --upgrade scikit-learn
!pip install --upgrade datasets
```

```
Collecting accelerate
  Downloading accelerate-0.30.0-py3-none-any.whl (302 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 302.4/302.4 kB 3.9 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from accelerate) (1.25.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from accelerate) (24.0)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from accelerate) (5.9.5)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (from accelerate) (6.0.1)
Requirement already satisfied: torch>=1.10.0 in /usr/local/lib/python3.10/dist-packages (from accelerate) (2.2.1+cu1
Requirement already satisfied: huggingface-hub in /usr/local/lib/python3.10/dist-packages (from accelerate) (0.20.3
Requirement already satisfied: safetensors>=0.3.1 in /usr/local/lib/python3.10/dist-packages (from accelerate) (0.4.
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->accelerate)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.1
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->accelerate) (1.
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->accelerate)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->accelerate) (
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->accelerate) (
Collecting nvidia-cuda-nvrtc-cu12==12.1.105 (from torch>=1.10.0->accelerate)
  Using cached nvidia_cuda_nvrtc_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (23.7 MB)
Collecting nvidia-cuda-runtime-cu12==12.1.105 (from torch>=1.10.0->accelerate)
  Using cached nvidia_cuda_runtime_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (823 kB)
```

```
Collecting nvidia-cuda-cupti-cu12==12.1.105 (from torch>=1.10.0->accelerate)
  Using cached nvidia_cuda_cupti_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (14.1 MB)
Collecting nvidia-cudnn-cu12==8.9.2.26 (from torch>=1.10.0->accelerate)
  Using cached nvidia_cudnn_cu12-8.9.2.26-py3-none-manylinux1_x86_64.whl (731.7 MB)
Collecting nvidia-cublas-cu12==12.1.3.1 (from torch>=1.10.0->accelerate)
  Using cached nvidia_cublas_cu12-12.1.3.1-py3-none-manylinux1_x86_64.whl (410.6 MB)
Collecting nvidia-cufft-cu12==11.0.2.54 (from torch>=1.10.0->accelerate)
  Using cached nvidia_cufft_cu12-11.0.2.54-py3-none-manylinux1_x86_64.whl (121.6 MB)
Collecting nvidia-curand-cu12==10.3.2.106 (from torch>=1.10.0->accelerate)
  Using cached nvidia_curand_cu12-10.3.2.106-py3-none-manylinux1_x86_64.whl (56.5 MB)
Collecting nvidia-cusolver-cu12==11.4.5.107 (from torch>=1.10.0->accelerate)
  Using cached nvidia_cusolver_cu12-11.4.5.107-py3-none-manylinux1_x86_64.whl (124.2 MB)
Collecting nvidia-cusparse-cu12==12.1.0.106 (from torch>=1.10.0->accelerate)
  Using cached nvidia_cusparse_cu12-12.1.0.106-py3-none-manylinux1_x86_64.whl (196.0 MB)
Collecting nvidia-nccl-cu12==2.19.3 (from torch>=1.10.0->accelerate)
  Using cached nvidia_nccl_cu12-2.19.3-py3-none-manylinux1_x86_64.whl (166.0 MB)
Collecting nvidia-nvtx-cu12==12.1.105 (from torch>=1.10.0->accelerate)
  Using cached nvidia_nvtx_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (99 kB)
Requirement already satisfied: triton==2.2.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->acceler
Collecting nvidia-nvjitlink-cu12 (from nvidia-cusolver-cu12==11.4.5.107->torch>=1.10.0->accelerate)
  Using cached nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (21.1 MB)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from huggingface-hub->accelerate
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub->accele
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.10.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->h
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->huggingface-h
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->hugging
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->hugging
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch>=1.10.0->a
Installing collected packages: nvidia-nvtx-cu12, nvidia-nvjitlink-cu12, nvidia-nccl-cu12, nvidia-curand-cu12, nvidia
Successfully installed accelerate-0.30.0 nvidia-cublas-cu12-12.1.3.1 nvidia-cuda-cupti-cu12-12.1.105 nvidia-cuda-nvr
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.40.1)
Collecting transformers
  Downloading transformers-4.40.2-py3-none-any.whl (9.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.0/9.0 MB 31.4 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.14.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.19.3 in /usr/local/lib/python3.10/dist-packages (from transfo
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.25.2)
```

```
#Adding all modules to import for the project.
from google.colab import drive
```

```
import pandas as pd
from datetime import datetime
import random
import math
import numpy as np
import pandas as pd
from google.colab import drive
import logging
import os
from typing import Callable, Dict, List, Optional, Tuple
import csv
import json, time
from collections import defaultdict
from itertools import combinations, islice
import pickle

from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import train_test_split

import torch
torch.__version__
import torch.nn.functional as F
from torch import Tensor, nn

from torch.utils.data.dataloader import DataLoader
from torch.utils.data.dataset import Dataset
from torch.utils.data.distributed import DistributedSampler
from torch.utils.data.sampler import RandomSampler, Sampler, SequentialSampler

from transformers.data.data_collator import DataCollator
from transformers.modeling_utils import PreTrainedModel
from transformers.optimization import AdamW, get_linear_schedule_with_warmup
from transformers.trainer_utils import PREFIX_CHECKPOINT_DIR, EvalPrediction, PredictionOutput, TrainOutput
from transformers.training_args import TrainingArguments

from transformers.activations import ACT2FN
from transformers.models.bart.configuration_bart import BartConfig
from transformers import BertTokenizer, BartTokenizer, BartForConditionalGeneration, Trainer, TFTrainingArguments
from transformers import DefaultDataCollator

from transformers import (
    CONFIG_MAPPING,
    MODEL_WITH_LM_HEAD_MAPPING,
    AutoConfig,
    AutoModelWithLMHead,
    AutoTokenizer,
    BertTokenizer,
    DataCollatorForLanguageModeling,
    HfArgumentParser,
    LineByLineTextDataset,
    PreTrainedTokenizer,
    TextDataset,
    TrainingArguments,
    set_seed,
)

from dataclasses import dataclass, field

from datasets import Dataset, DatasetDict, load_dataset
from tqdm import tqdm
```

Methodology

Environment

Operating systems:

- Ubuntu 20.04.5 LTS
- GPU T4
- Google colab environment
- Python 3.8.11 with libraries:

- NumPy (currently tested on version 1.20.3)
- PyTorch (currently tested on version 1.9.0+cu111)
- Transformers (currently tested on version 4.16.2)
- tqdm==4.62.2
- scikit-learn==0.24.2

Mount Notebook to Google Drive

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

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

```
#Check the File listing

!ls -lr /content/drive/MyDrive/mimiciv/2.2/*
```

```
     -rw------- 1 root root  2884 Jan  6  2023 /content/drive/MyDrive/mimiciv/2.2/SHA256SUMS.txt
     -rw------- 1 root root  2518 Jan  6  2023 /content/drive/MyDrive/mimiciv/2.2/LICENSE.txt
     -rw------- 1 root root   789 Mar 29 00:31 /content/drive/MyDrive/mimiciv/2.2/index.html
     -rw------- 1 root root 13332 Jan  5  2023 /content/drive/MyDrive/mimiciv/2.2/CHANGELOG.txt

     /content/drive/MyDrive/mimiciv/2.2/icu:
     total 3077969
     -rw------- 1 root root   20717852 Jan  5  2023 procedureevents.csv.gz
     -rw------- 1 root root   38747895 Jan  5  2023 outputevents.csv.gz
     -rw------- 1 root root  324218488 Jan  5  2023 inputevents.csv.gz
     -rw------- 1 root root  251962313 Jan  5  2023 ingredientevents.csv.gz
     -rw------- 1 root root       1336 Mar 29 00:31 index.html
     -rw------- 1 root root    2614571 Jan  5  2023 icustays.csv.gz
     -rw------- 1 root root      57476 Jan  5  2023 d_items.csv.gz
     -rw------- 1 root root   45721062 Jan  5  2023 datetimeevents.csv.gz
     -rw------- 1 root root 2467761053 Jan  5  2023 chartevents.csv.gz
     -rw------- 1 root root      35893 Jan  5  2023 caregiver.csv.gz

     /content/drive/MyDrive/mimiciv/2.2/hosp:
```

```
total 4429839
-rw------- 1 root root   36158338 Jan  5  2023 transfers.csv.gz
-rw------- 1 root root    6781247 Jan  5  2023 services.csv.gz
-rw------- 1 root root     122507 Jan  5  2023 provider.csv.gz
-rw------- 1 root root    6027067 Jan  5  2023 procedures_icd.csv.gz
-rw------- 1 root root  458817415 Jan  5  2023 prescriptions.csv.gz
-rw------- 1 root root   25477219 Jan  5  2023 poe_detail.csv.gz
-rw------- 1 root root  498505135 Jan  5  2023 poe.csv.gz
-rw------- 1 root root  398753125 Jan  5  2023 pharmacy.csv.gz
-rw------- 1 root root    2312631 Jan  5  2023 patients.csv.gz
-rw------- 1 root root   36124944 Jan  5  2023 omr.csv.gz
-rw------- 1 root root   96698496 Jan  5  2023 microbiologyevents.csv.gz
-rw------- 1 root root 1939088924 Jan  5  2023 labevents.csv.gz
-rw------- 1 root root       2907 Mar 29 00:31 index.html
-rw------- 1 root root    1767138 Jan  5  2023 hcpcsevents.csv.gz
-rw------- 1 root root  471096030 Jan  5  2023 emar_detail.csv.gz
-rw------- 1 root root  508524623 Jan  5  2023 emar.csv.gz
-rw------- 1 root root    7426955 Jan  5  2023 drgcodes.csv.gz
-rw------- 1 root root      12900 Jan  5  2023 d_labitems.csv.gz
-rw------- 1 root root     578517 Jan  5  2023 d_icd_procedures.csv.gz
-rw------- 1 root root     859438 Jan  5  2023 d_icd_diagnoses.csv.gz
-rw------- 1 root root   25070720 Jan  5  2023 diagnoses_icd.csv.gz
-rw------- 1 root root     427468 Jan  5  2023 d_hcpcs.csv.gz
-rw------- 1 root root   15516088 Jan  5  2023 admissions.csv.gz
```

Data

MIMIC-IV dataset comprises data from intensive care unit patients admitted to the Beth Israel Deaconess Medical Center in Boston, Massachusetts. Although the dataset spans from 2008 to 2019, the implementation of ICD-10CM began in October 2015. As per authors implementation plan, to align with the cohorts from the Veterans Health Administration (VHA) dataset pretrained model, only patients with ICD-10CM records will be selected.

Data Load - Extract below data files from MIMIC-IV dataset https://physionet.org/content/mimiciv/2.2 :

1. admissions.csv.gz
2. diagnoses_icd.csv.gz

```
# Read diagnoses_icd.csv.gz
diagnoses_df = pd.read_csv('/content/drive/MyDrive/mimiciv/2.2/hosp/diagnoses_icd.csv.gz',
                           nrows=None,
                           compression='gzip',
                           dtype={'subject_id': str, 'hadm_id': str, 'icd_code': str, 'icd_version': str},
#                           error_bad_lines=False)
                           on_bad_lines = 'skip')
print(f'Number of Rows and Columns in Diagnoses_icd file: {diagnoses_df.shape}')
#print(diagnoses_df.head(5))


# print(diagnoses_df)
```

```
     Number of Rows and Columns in Diagnoses_icd file: (4756326, 5)
```

```
# Read admissions.csv.gz file
admissions_df = pd.read_csv('/content/drive/MyDrive/mimiciv/2.2/hosp/admissions.csv.gz',
                            nrows=None,
                            compression='gzip',
                            dtype={'subject_id': str, 'hadm_id': str},
#                            error_bad_lines=False)
                           on_bad_lines = 'skip')
print(f'Number of Rows and Columns in Admissions file: {admissions_df.shape}')
#print(admissions_df.head(5))
```

```
     Number of Rows and Columns in Admissions file: (431231, 16)
```

Preprocess Data

- Create subset dataset by extracting ICD10 version records to align with the cohorts from Pretrained model.
- Create subset of data for Project purpose due to Project run time and computational resources limitations. Subset of data consists of 1500 Patients and related records from admissions and diagnosis_icd files.
- Map the ICD codes to CUI codes [Concepts to concept Unique Identifiers (CUIs) from the United Medical Language System (UMLS)]

```
# Number of subject
num_of_subjects = 1500

# read_diagnoses_and_admissions_data()

# Filter icd_version=10
diagnoses_df = diagnoses_df[diagnoses_df['icd_version'] == '10']
print(f'Number of Rows and Columns in Diagnoses_icd_10 file: {diagnoses_df.shape}')

# Select number of unique subject_id from the filtered diagnoses file
selected_subject_ids = diagnoses_df['subject_id'].unique()[:num_of_subjects]
#selected_subject_ids = diagnoses_df['subject_id'].unique()
# print(f'Selected {num_of_subjects} Patients from Diagnoses_icd_10 file: {selected_subject_ids}')


# Filter both the Diagnoses and admissions to include only the selected subject_id
diagnoses_df = diagnoses_df[diagnoses_df['subject_id'].isin(selected_subject_ids)]
print(f'Number of Rows and Columns in Subset dataset Diagnoses_icd_10 file: {diagnoses_df.shape}')
admissions_df = admissions_df[admissions_df['subject_id'].isin(selected_subject_ids)]
print(f'Number of Rows and Columns in Subset dataset Admissions file: {admissions_df.shape}')
#print(len(data_df))
#print(len(admissions_data_df))

#print(diagnoses_df)
```

```
     Number of Rows and Columns in Diagnoses_icd_10 file: (1989449, 5)
     Number of Rows and Columns in Subset dataset Diagnoses_icd_10 file: (37613, 5)
     Number of Rows and Columns in Subset dataset Admissions file: (4103, 16)
```

```
#Print the Max and Min of discharge date in admissions subset
visitid2dischargedate = {}
for ind, row in admissions_df.iterrows():
    visitid2dischargedate[row['hadm_id']] = row['dischtime'][0:10]

print(min(visitid2dischargedate.values()))
print(max(visitid2dischargedate.values()))
```

```
#print(visitid2dischargedate)
```

```
     2110-01-18
     2210-06-22
```

```
#Create Patient dictionary
#Code reference: https://github.com/whaleloops/TransformEHR/blob/main/preprocess.py

patients = defaultdict(lambda: defaultdict(list))
print("Number of rows:" , diagnoses_df.shape[0])
for ind, row in diagnoses_df.iterrows():
    hadm_id = row['hadm_id']
    scrssn = row['subject_id']
    visit_date = visitid2dischargedate[hadm_id]
    patients[scrssn][visit_date].append(row['icd_version'] +'-'+ row['icd_code'])

num_icd_pat = defaultdict(int)
for k,v in patients.items():
    for kv, vv in v.items():
        for icdcode in vv:
            if icdcode.startswith("10-"):
                num_icd_pat[k] += 1
                break

#print(len(patients))
#print(len(num_icd_pat))
num_pos = 0
for k,v in num_icd_pat.items():
    if v > 1:
        num_pos += 1
#print(num_pos)
#print(f'num_icd_pat dictionary: {num_icd_pat}')
#print("Done")
```

```
     Number of rows: 37613
```

```
#Map the ICD codes to CUI codes [Concepts to concept Unique Identifiers (CUIs) from the United Medical Language System (
#Code reference: https://github.com/whaleloops/TransformEHR/blob/main/preprocess.py
```

```
def icd2cui(patients, logging_step=50000):
    dictionary = defaultdict(int)
    # cuis_li = []
    cuis_di = {}
    date_di = {}
    num_idx = 0
    for pssn,v in patients.items():
        num_idx += 1
        if num_idx%logging_step == 0:
            print("|{} - Processed {}".format(time.asctime(time.localtime(time.time())), num_idx), flush=True)
        cuis_di[pssn] = []
        cuis_li_tmp = []
        date_li_tmp = []
        for datetime_str in sorted(v.keys()): # sort by time
            datetime_object = datetime.strptime(datetime_str, '%Y-%m-%d') # make sure time str is correct
            infos = v[datetime_str]
            if len(infos) > 0:
                # cuis_di[pssn].append((cuis, ext_cuis, strs))
                cuis_li_tmp.append((infos, [], []))
                date_li_tmp.append(datetime_str)
            for cui_id in infos:
                dictionary[cui_id] += 1
        if len(cuis_li_tmp) > 0:
            cuis_di[pssn] = cuis_li_tmp
            date_di[pssn] = date_li_tmp
    return cuis_di, date_di, dictionary

patients_few = dict(islice(patients.items(), 0, 200))
# cuis, date, dictionary = icd2cui(patients_few, logging_step=50000)
cuis, date, dictionary = icd2cui(patients, logging_step=50000)

dir_apth = '/content/drive/My Drive/'
print("Number of cui in dictionary: {}".format(len(dictionary)), flush=True)
with open(dir_apth + '/dict.txt', 'w') as handle: #TODO
    handle.write("[PAD]"+"\n")
    for i in range(99):
        handle.write("[unused{}]".format(i)+"\n")
    handle.write("[UNK]"+"\n")
    handle.write("[CLS]"+"\n")
    handle.write("[SEP]"+"\n")
    handle.write("[MASK]"+"\n")
    for i in range(99,194):
        handle.write("[unused{}]".format(i)+"\n")
    for k,v in dictionary.items():
        handle.write("{}\n".format(k))
# save data
print("Saving patient data...", flush=True)
f1 = open(dir_apth + '/value.pickle', 'wb')
f3 = open(dir_apth + '/dates.pickle', 'wb')
f2 = open(dir_apth + '/key.txt', 'w')
for k,v in cuis.items():
    pickle.dump(v, f1, protocol=pickle.HIGHEST_PROTOCOL)
    pickle.dump(date[k], f3, protocol=pickle.HIGHEST_PROTOCOL)
    f2.write("{}\n".format(k))
f1.close()
f3.close()
f2.close()

print("Done")
```

```
Number of cui in dictionary: 4084
Saving patient data...
Done
```

Load Data

```
# Load key.txt, value.pickle and dates.pickle files
# Code reference: https://github.com/whaleloops/TransformEHR/blob/main/sample_load.py

import json

# Minimum number of visit
min_num_of_visits = 3
```

```
include_patient_id = True
include_icd_codes = True
include_visit_dates = True

# Load data from files
dir_path = "/content/drive/My Drive/"
f1 = open(dir_path + "/value.pickle", "rb")
f3 = open(dir_path + "/dates.pickle", "rb")
f2 = open(dir_path + "/key.txt", "r")
keys = f2.readlines()

patients = []

# Iterate over patient data
for key in keys:
    patient_idd = key.strip()
    each_visit = pickle.load(f1)
    f1obj = []
    visit_dates = []
    icd_codes = []

    for (cuis, ext_cuis, strs), date in zip(each_visit, pickle.load(f3)):
        icd_code = [icd.split("-")[1] for icd in cuis]
        icd_codes.append(icd_code)
        visit_dates.append(date)

    if len(icd_codes) >= min_num_of_visits:
        patient_data = {}

        if include_patient_id:
            patient_data["patient_id"] = patient_idd

        if include_icd_codes:
            patient_data["icd_codes"] = str(icd_codes)

        if include_visit_dates:
            patient_data["visit_dates"] = visit_dates
        patients.append(patient_data)


f1.close()
f3.close()
f2.close()

print(f'Number of patients in the sample dataset: {len(patients)}')
print(f'Sample dataset: {patients}')
```

```
Number of patients in the sample dataset: 258
Sample dataset: [{'patient_id': '10000980', 'icd_codes': "[['D500', 'I5023', 'N184', 'E118', 'K2970', 'Z23', 'K259',
```

Train Test Data Preparation

```
#Split the Subset dataset into Train and Test (80/20 split)
s = pd.Series(patients)
train_dataset , test_dataset  = [i.to_dict() for i in train_test_split(s, train_size=0.8)]
print("length of train dataset:", len(train_dataset))
# print(f'Train dataset: {json.dumps(train_dataset, indent=4)}')
print("length of test dataset:", len(test_dataset))
# print(f'Test dataset: {json.dumps(test_dataset, indent=4)}')

with open('/content/drive/MyDrive/patients.json', 'w') as json_file:
  json.dump(patients, json_file, indent=4)
```

```
length of train dataset: 206
length of test dataset: 52
```

```
# train_dataset = Dataset.from_dict(train_dataset)

huggingface_dataset = load_dataset("json", data_files="/content/drive/MyDrive/patients.json")
huggingface_dataset = huggingface_dataset['train'].train_test_split(test_size=0.1)
# huggingface_dataset_test_valid = huggingface_dataset['test'].train_test_split(test_size=0.5)
# huggingface_dataset_test_valid['validation'] = huggingface_dataset_test_valid['train']

# huggingface_final_dataset = DatasetDict({
#                                            "train": huggingface_dataset['train'],
#                                            "validation": huggingface_dataset_test_valid['validation'],
#                                            "test": huggingface_dataset_test_valid['test']
#                                          })

print("Print Huggingface dataset:",huggingface_dataset)
print("Print 5 rows of Huggingface dataset in JSON format",json.dumps(huggingface_dataset['train'][:5], indent=4))
```

```
Generating train split:    258/0 [00:00<00:00, 4151.72 examples/s]
```

```
Print Huggingface dataset: DatasetDict({
    train: Dataset({
        features: ['icd_codes', 'visit_dates', 'patient_id'],
        num_rows: 232
    })
    test: Dataset({
        features: ['icd_codes', 'visit_dates', 'patient_id'],
        num_rows: 26
    })
})
Print 5 rows of Huggingface dataset in JSON format {
    "icd_codes": [
        "[['T465X2A', 'T434X2A', 'Y929', 'B356', 'F17210'], ['T434X2A', 'Y929', 'F259', 'F4310', 'F909', 'Z915', 'Z
        "[['A4152', 'K830', 'D696', 'E860', 'A408', 'K828', 'G309', 'F0280', 'M25511', 'W19XXXA', 'F329', 'K580', 'K
        "[['Z5111', 'C9000', 'R740', 'E039', 'D6851', 'M3500'], ['C9000', 'D6851', 'D701', 'M3500', 'R5081', 'R112',
        "[['K5732', 'R079', 'K219', 'E1122', 'I129', 'N182', 'E785', 'E860', 'G44209', 'E8881', 'R933', 'E669', 'Z6
        "[['J95851', 'I214', 'G9340', 'G1221', 'N179', 'N10', 'E873', 'D721', 'J9610', 'Z9911', 'N390', 'B965', 'Z4
    ],
    "visit_dates": [
        [
            "2128-05-02",
            "2129-02-25",
            "2129-02-27"
        ],
        [
            "2198-04-11",
            "2198-12-26",
            "2199-02-21"
            ],
            [
                "2129-05-12",
                "2129-07-06",
                "2131-02-08"
            ],
            [
                "2135-12-21",
                "2136-03-03",
                "2136-11-30"
            ],
            [
                "2178-07-08",
                "2178-07-16",
                "2178-07-25"
            ]
        ],
        "patient_id": [
            "10093625",
            "10167779",
            "10083754",
            "10033552",
            "10016742"
        ]
    }
```

Model

The Paper user the TransformEHR which has an encoder–decoder transformer architecture. The encoder processes the input embeddings and generates a set of hidden representations for each predictor. TransformEHR performs cross-attention over the hidden representations from the encoder and assigns an attention weight for each representation. These weighted representations are then fed to the decoder, which generates ICD codes of the future visit. The decoder generates ICDcodes following the sequential order of code priority within a visit.

Here is the Github link for the Paper's Model code: https://github.com/whaleloops/TransformEHR/blob/main/icdmodelbart.py The code was not reproducible due to missing Datacollator. We tried to contact the Author of the paper but could not receive the missing code. There were multiple attempts made to make Author's model work with standard Datacollator but unable to successfully train the model.

Here is the link for the Project Draft with which tries to replicate the Author's code: https://colab.research.google.com/drive/1RMl4T3FsAQaJDColj0xPma8t3Xwk0c3z

During review with TA, considering the code limitations, we decided to change the plan and implement below models from Hugging face https://huggingface.co/docs/hub/en/transformers to train MIMICIV subset data with ICD codes and Visit date tokenization:

1. BERT (Bidirectional Encoder Representations from Transformers)
2. BART (Bidirectional Encoder and left-to-right Decoder from Transformers)

We still made sure we run the models to perform below Ablations:

1. To assess the performance of an Encoder-only architecture (BERT) model compared to an encoder-decoder (BART) architecture.
2. To assess the performance of models using only ICD code tokenization and using both ICD code and Visit date tokenization.

Model 1: Bert model - bert-base-cased - With ICD code and Visit date tokenization

```
#Model1: Bert model - bert-base-cased - With ICD code and Visit date tokenization
from transformers import AutoTokenizer, AutoModelForMaskedLM, DataCollatorWithPadding
tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")

def tokenize_function(examples):
    #print(len(examples["icd_codes"]))
    #print(len(examples["visit_dates"]))
    assert len(examples["icd_codes"]) == len(examples["visit_dates"])
    icd_token = tokenizer(examples["icd_codes"], return_tensors="pt", padding="max_length", truncation=True, max_length=
    #print(icd_token["input_ids"].dtype)
    #print(icd_token["input_ids"].shape)
    #print(icd_token)
```

```
        visit_dates = examples["visit_dates"]
        visit_dates = [str(date) for date in visit_dates]
        visit_token = tokenizer(visit_dates, return_tensors="pt", padding="max_length", truncation=True, max_length=256)
        #print(visit_token["input_ids"].dtype)
        #print(visit_token["input_ids"].shape)
        #print(visit_token)
        assert icd_token["input_ids"].shape == visit_token["input_ids"].shape
        #combined_token = {key: torch.cat([icd_token[key], visit_token[key]], dim=-1) for key in icd_token.keys()}
        comb_input_ids = torch.cat([icd_token["input_ids"], visit_token["input_ids"]], dim=1)
        #print(comb_input_ids.shape)
        comb_att_mask = torch.cat([icd_token["attention_mask"], visit_token["attention_mask"]], dim=1)
        #print(comb_att_mask.shape)
        combined_token = {"input_ids": comb_input_ids, "attention_mask": comb_att_mask}
        print("Combined token: ", combined_token)
        return combined_token

    tokenized_data = huggingface_dataset.map(tokenize_function, batched=True, batch_size=None)
    #print(len(tokenized_data))
    #print("Tokenized Training data sample ['train'][0] ", tokenized_data['train'][0])
    #print("Tokenized Test data sample ['test'][0] ", tokenized_data['test'][0])

    config = AutoConfig.from_pretrained("bert-base-cased")
    print("Bert Configuration: ", config)
    data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm_probability=0.15)
    bert_model_v = AutoModelForMaskedLM.from_pretrained("bert-base-cased", config=config)
    print("Bert (bert-base-cased) Model: ",bert_model_v)
    # data_collator = DataCollatorWithPadding(tokenizer=tokenizer, padding= "max_length", max_length=512)
    print("Data Collator For LanguageModeling: ", data_collator)
```

Model1 Training - Bert model - bert-base-cased With ICDcodes and Visit date tokenization

```
#Training Model1 - Bert model - bert-base-cased With ICDcodes and Visit date tokenization
from transformers import Trainer, TrainingArguments

training_args = TrainingArguments(
    output_dir="output-mlm",
    evaluation_strategy = "no",
    learning_rate=1e-5,
    weight_decay=0.01,
    logging_steps=10,
    num_train_epochs=3,
    report_to="none",
    disable_tqdm = False,
    load_best_model_at_end=True,
    save_strategy = 'no',
    metric_for_best_model="accuracy",
)
#print(tokenized_data["train"].shape)
#print(tokenized_data["test"].shape)
bert_trainer_v = Trainer(
    model=bert_model_v,
    args=training_args,
    train_dataset=tokenized_data["train"],
    eval_dataset=tokenized_data["test"],
    data_collator=data_collator
)
bert_train_output_v = bert_trainer_v.train()
bert_train_output_v_global_step, bert_train_output_v_training_loss, bert_train_output_v_metrics = bert_train_output_v
bert_train_output_v
```

Model1 Evaluation - Bert model - bert-base-cased With ICDcodes and Visit date tokenization

```
#Evaluating Model1 - Bert model - bert-base-cased With ICDcodes and Visit date tokenization
bert_eval_output_v = bert_trainer_v.evaluate()
bert_eval_output_v
```

Model 2 Bert model - bert-base-cased with ICD codes tokenization

```
#Model1: Bert model - bert-base-cased with ICD Codes tokenization
```

```
from transformers import AutoTokenizer, AutoModelForMaskedLM
tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")

def tokenize_function(examples):
    token = tokenizer(examples["icd_codes"], return_tensors="pt", padding="max_length", truncation=True, max_length=256
    print("token: ", token)
    return token

tokenized_data = huggingface_dataset.map(tokenize_function, batched=True, batch_size=None)
#print("Tokenized Training data sample ['train'][0] ", tokenized_data['train'][0])
#print("Tokenized Test data sample ['test'][0] ", tokenized_data['test'][0])

config = AutoConfig.from_pretrained("bert-base-cased")
print("Bert Configuration: ", config)
data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm_probability=0.15)
bert_model = AutoModelForMaskedLM.from_pretrained("bert-base-cased", config=config)
print("Bert (bert-base-cased) Model: ",bert_model)
```

Model 2 Training - Bert model - bert-base-cased with ICD Codes tokenization

```
#Training Model 2 - Bert model - bert-base-cased with ICD Codes tokenization
```

```
from transformers import Trainer, TrainingArguments

training_args = TrainingArguments(
    output_dir="output-mlm",
    evaluation_strategy = "no",
    learning_rate=1e-5,
    weight_decay=0.01,
    logging_steps=10,
    num_train_epochs=3,
    report_to="none",
    disable_tqdm = False,
    load_best_model_at_end=True,
    save_strategy = 'no',
    metric_for_best_model="accuracy",
)

bert_trainer = Trainer(
    model=bert_model,
    args=training_args,
    train_dataset=tokenized_data["train"],
    eval_dataset=tokenized_data["test"],
    data_collator=data_collator
)
bert_train_output = bert_trainer.train()
bert_train_output_global_step, bert_train_output_training_loss, bert_train_output_metrics = bert_train_output
bert_train_output
```

Model 2 Evaluation: Bert model - bert-base-cased with ICD Codes tokenization

```
#Evaluating Model1 - Bert model - bert-base-cased with ICD Codes tokenization
bert_eval_output = bert_trainer.evaluate()
bert_eval_output
```

Model 3 - Bart model - facebook/bart-base With ICDcodes and Visit date tokenization

```
#Model3: Bart model - facebook/bart-base With ICDcodes and Visit date tokenization
from transformers import BartTokenizer, BartForConditionalGeneration, DataCollatorWithPadding
tokenizer = BartTokenizer.from_pretrained("facebook/bart-base")

def tokenize_function(examples):
    #print(len(examples["icd_codes"]))
    #print(len(examples["visit_dates"]))
    assert len(examples["icd_codes"]) == len(examples["visit_dates"])
    icd_token = tokenizer(examples["icd_codes"], return_tensors="pt", padding="max_length", truncation=True, max_length=
    #print(icd_token["input_ids"].dtype)
    #print(icd_token["input_ids"].shape)
    #print(icd_token)
    visit_dates = examples["visit_dates"]
```

```
        visit_dates = examples["visit_dates"]
        visit_dates = [str(date) for date in visit_dates]
        visit_token = tokenizer(visit_dates, return_tensors="pt", padding="max_length", truncation=True, max_length=256)
        #print(visit_token["input_ids"].dtype)
        #print(visit_token["input_ids"].shape)
        #print(visit_token)
        assert icd_token["input_ids"].shape == visit_token["input_ids"].shape
        #combined_token = {key: torch.cat([icd_token[key], visit_token[key]], dim=-1) for key in icd_token.keys()}
        comb_input_ids = torch.cat([icd_token["input_ids"], visit_token["input_ids"]], dim=1)
        #print(comb_input_ids.shape)
        comb_att_mask = torch.cat([icd_token["attention_mask"], visit_token["attention_mask"]], dim=1)
        #print(comb_att_mask.shape)
        combined_token = {"input_ids": comb_input_ids, "attention_mask": comb_att_mask}
        print("combined_token: ", combined_token)
        return combined_token

    tokenized_data = huggingface_dataset.map(tokenize_function, batched=True, batch_size=None)
    #print(tokenized_data)
    #print("Tokenized Training data sample ['train'][0] ", tokenized_data['train'][0])
    #print("Tokenized Test data sample ['test'][0] ", tokenized_data['test'][0])

    config = BartConfig.from_pretrained("facebook/bart-base")
    print("Bart Configuration: ", config)
    bart_model_v = BartForConditionalGeneration.from_pretrained("facebook/bart-base", config=config)

    #os.environ["CUDA_VISIBLE_DEVICES"] = "0"
    #bart_model_v
    print("Bart facebook/bart-base Model: ",bart_model_v)
    data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm_probability=0.15)
    # data_collator = DataCollatorWithPadding(tokenizer=tokenizer) #padding= "max_length", max_length=512)
    print("Data Collator For LanguageModeling: ", data_collator)
```

Model 3 Training Bart Model - facebook/bart-base With ICDcodes and Visit date tokenization

```
#Training Model3 - Bart Model - facebook/bart-base With ICDcodes and Visit date tokenization
from transformers import Trainer, TrainingArguments

training_args = TrainingArguments(
    output_dir="output-mlm",
    evaluation_strategy = "no",
    learning_rate=1e-5,
    weight_decay=0.01,
    logging_steps=10,
    num_train_epochs=3,
    report_to="none",
    disable_tqdm = False,
    load_best_model_at_end=True,
    save_strategy = 'no',
    metric_for_best_model="accuracy",
)

bart_trainer_v = Trainer(
    model=bart_model_v,
    args=training_args,
    train_dataset=tokenized_data["train"],
    eval_dataset=tokenized_data["test"],
    data_collator=data_collator
)
bart_train_output_v = bart_trainer_v.train()
bart_train_output_v_global_step, bart_train_output_v_training_loss, bart_train_output_v_metrics = bart_train_output_v
```

```
bart_train_output_v
```

Model 3 Evaluation Bart Model - facebook/bart-base With ICDcodes and Visit date tokenization

```
#Evaluating Model3 - Bart Model - facebook/bart-base With ICDcodes and Visit date tokenization
bart_eval_output_v = bart_trainer_v.evaluate()
bart_eval_output_v
```

Model 4 Bart model - facebook/bart-base with ICD code tokenization

```
#Model4: Bart model - facebook/bart-base with ICD code tokenization
from transformers import BartTokenizer, BartForConditionalGeneration
tokenizer = BartTokenizer.from_pretrained("facebook/bart-base")

def tokenize_function(examples):
    token = tokenizer(examples["icd_codes"], return_tensors="pt", padding="max_length", truncation=True, max_length=256
    print("token: ", token)
    return token

tokenized_data = huggingface_dataset.map(tokenize_function, batched=True, batch_size=None)
#print("Tokenized Training data sample ['train'][0] ", tokenized_data['train'][0])
#print("Tokenized Test data sample ['test'][0] ", tokenized_data['test'][0])

config = BartConfig.from_pretrained("facebook/bart-base")
print("Bart Configuration: ", config)
data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm_probability=0.15)
bart_model = BartForConditionalGeneration.from_pretrained("facebook/bart-base", config=config)
print("Bart facebook/bart-base Model: ",bart_model)
#os.environ["CUDA_VISIBLE_DEVICES"] = "0"
#bart_model
```

Model 4 Training Bart Model - facebook/bart-base with ICD code tokenization

```
#Training Model4 - Bart Model - facebook/bart-base with ICD code tokenization
from transformers import Trainer, TrainingArguments

training_args = TrainingArguments(
    output_dir="output-mlm",
    evaluation_strategy = "no",
    learning_rate=1e-5,
    weight_decay=0.01,
    logging_steps=10,
    num_train_epochs=3,
    report_to="none",
    disable_tqdm = False,
    load_best_model_at_end=True,
    save_strategy = 'no',
    metric_for_best_model="accuracy",
)

bart_trainer = Trainer(
    model=bart_model,
    args=training_args,
    train_dataset=tokenized_data["train"],
    eval_dataset=tokenized_data["test"],
    data_collator=data_collator
)
bart_train_output = bart_trainer.train()
bart_train_output_global_step, bart_train_output_training_loss, bart_train_output_metrics = bart_train_output
bart_train_output
```

Model 4 Evaluation Bart Model - facebook/bart-base with ICD code tokenization

```
#Evaluating Model2 - Bart Model - facebook/bart-base with ICD code tokenization
bart_eval_output = bart_trainer.evaluate()
bart_eval_output
```

Results

As per the Training and Evaluation results present in the below Model Comparison section BERT model performs better than the BART model. Visit tokenization addition does not make much difference in the results. Refer the Model comparison table below.

Per Author's paper, the TransformEHR model (ICDBART) outperforms bidirectional encode-only (BERT) models. Please note that Author uses sequence to sequence transformer model that was pretrained on huge dataset. It uses cross-attention by identifying relevant ICD codes from previous visits to predict future ICD codes.

Model comparison

```
### Print out the Model comparison from Model training and Evaluation in Tabular format
result_data = {
    'Model Name': ['Bert Model with ICD and Visit Date tokenization', 'Bert Model with ICD tokenization', 'Bart Model wi
    'Train Loss': [bert_train_output_v_metrics['train_loss'], bert_train_output_metrics['train_loss'], bart_train_output
    'Train RunTime (Sec)': [bert_train_output_v_metrics['train_runtime'], bert_train_output_metrics['train_runtime'], ba
    'Train Epochs': [bert_train_output_v_metrics['epoch'], bert_train_output_metrics['epoch'], bart_train_output_v_metri
    'Eval Loss': [bert_eval_output_v['eval_loss'], bert_eval_output['eval_loss'], bart_eval_output_v['eval_loss'], bart_
    'Eval RunTime (Sec)': [bert_eval_output_v['eval_runtime'], bert_eval_output['eval_runtime'], bart_eval_output_v['eva
    'Eval Epochs': [bert_eval_output_v['epoch'], bert_eval_output['epoch'], bart_eval_output_v['epoch'], bart_eval_outpu
}

result_df = pd.DataFrame(data=result_data)
result_df.set_index('Model Name', inplace=True)
result_df
```

Discussion

The code was not reproducible due to missing Datacollator. We tried to contact the Author of the paper but could not receive the missing code. There were multiple attempts made to make Author's model with available Datacollator but unable to successfully train the model. During review with TA, considering the code limitations, we decided to change the plan and implement below models from Hugging face. https://huggingface.co/docs/hub/en/transformers :

1. BERT (Bidirectional Encoder Representations from Transformers)
2. BART (Bidirectional Encoder and left-to-right Decoder from Transformers)

The paper uses Transformers library and customized few classes for the implementation. During the DLH course assignments, we have not used this library therefore understanding the usage of transformers library and implementing the code parallelly was the biggest challenge.

We traverse through a major learning curve by understanding Transformers Library and learning BERT and BART model implementations from Hugging face.

Getting the MIMIC IV data preprocess and ready for these implementation was another challenging task. Most of the time spent on getting this huge dataset extracted and creating the required subset to run with existing computing resources.

We still made sure we run the models to perform below Ablations:

1. To assess the performance of an Encoder-only architecture model compared to an encoder-decoder architecture.
2. To assess the performance of models using only ICD code tokenization and using ICD code and Visit date tokenization.

Below are our recommendations for the author:

1. Custom Data Collator code availability. This will help streamline the data processing process and ensure that the input data is properly formatted and orgainzed for the training model.
2. Improve the code readability by Modularizing code pertaining to the different experiments performed in the paper.
3. Prepare a comprehensive README.md file describing the steps to the code corresponding to each experiment.

References

1. Citation to Original Paper: Yang, Z., Mitra, A., Liu, W. et al. TransformEHR: transformer-based encoder-decoder generative model to enhance prediction of disease outcomes using electronic health records. Nat Commun 14, 7857 (2023). https://doi.org/10.1038/s41467-023-43715-z. 2023 Nov 29;14(1):7857. doi: 10.1038/s41467-023-43715-z. PMID: 38030638;

PMCID: PMC10687211.

2. https://www.nature.com/articles/s41467-023-43715-z
3. https://physionet.org/content/mimiciv/2.2/icu/#files-panel
4. https://github.com/whaleloops/TransformEHR
5. https://huggingface.co/docs/hub/en/transformers