A Detailed Review and Reproducibility Analysis on "Time-Aware Transformer-based Network for Clinical Notes Series Prediction" Zhang et al., 2020

by

- Ethan Shen ethans11@illinois.edu
- Ajay Ramsunder ajayr4@illinois.edu
- Huyen Lai hlai9@illinois.edu

for CS598 Deep Learning for Healthcare - UIUC

Please refer to https://drive.google.com/drive/folders/1hy5GfU9eopCpi9ZTXbhhEbDCRvVKwZ06 for the data files, model training code, and model objects. Our video presentation is also located in the drive.

Notebook Setup

```
!pip install torch
!pip install pytorch-pretrained-bert==0.6.2
!pip install pytorch-transformers==1.2.0
!pip install dotmap
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.2.1+cu121)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.14.0)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch) (4.11.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.3)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.3)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch) (2023.6.0)
Collecting nvidia-cuda-nvrtc-cu12==12.1.105 (from torch)
  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)
  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)
  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)
  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)
  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)
  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)
  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)
  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)
  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)
  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)
  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) (2.2.0)
Collecting nvidia-nvjitlink-cu12 (from nvidia-cusolver-cu12==11.4.5.107->torch)
  Using cached nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (21.1 MB)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch) (2.1.5)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch) (1.3.0)
Installing collected packages: nvidia-nvtx-cu12, nvidia-nvjitlink-cu12, nvidia-nccl-cu12, nvidia-curand-cu12, nvidia-cufft-c
Successfully installed nvidia-cublas-cu12-12.1.3.1 nvidia-cuda-cupti-cu12-12.1.105 nvidia-cuda-nvrtc-cu12-12.1.105 nvidia-cu
Collecting pytorch-pretrained-bert==0.6.2
  Downloading pytorch_pretrained_bert-0.6.2-py3-none-any.whl (123 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 123.8/123.8 kB 3.2 MB/s eta 0:00:00
Requirement already satisfied: torch>=0.4.1 in /usr/local/lib/python3.10/dist-packages (from pytorch-pretrained-bert==0.6.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from pytorch-pretrained-bert==0.6.2) (1.25.
Collecting boto3 (from pytorch-pretrained-bert==0.6.2)
  Downloading boto3-1.34.99-py3-none-any.whl (139 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 139.3/139.3 kB 8.9 MB/s eta 0:00:00
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from pytorch-pretrained-bert==0.6.2) (2.
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from pytorch-pretrained-bert==0.6.2) (4.66.2
Requirement already satisfied: regex in /usr/local/lib/python3.10/dist-packages (from pytorch-pretrained-bert==0.6.2) (2023.
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch>=0.4.1->pytorch-pretrained-be
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch>=0.4.1->pytor
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch>=0.4.1->pytorch-pretrained-bert=
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=0.4.1->pytorch-pretrained-be
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=0.4.1->pytorch-pretrained-bert
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch>=0.4.1->pytorch-pretrained-bert
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch>=0.4.
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch>=0.
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch>=0.4.
Requirement already satisfied: nvidia-cudnn-cu12==8.9.2.26 in /usr/local/lib/python3.10/dist-packages (from torch>=0.4.1->py
Requirement already satisfied: nvidia-cublas-cu12==12.1.3.1 in /usr/local/lib/python3.10/dist-packages (from torch>=0.4.1->p
```

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

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

```
# mount notebook to google drive
import sys
sys.path.append('/content/drive/MyDrive/')
sys.path.append('/content/drive/MyDrive/DL4H_Team_67_Final_Project/common_functions')
sys.path.append('/content/drive/MyDrive/DL4H_Team_67_Final_Project/common_functions/file_utils.py')
sys.path.append('/content/drive/MyDrive/DL4H_Team_67_Final_Project/common_functions/modeling_patient.py')
sys.path.append('/content/drive/MyDrive/DL4H_Team_67_Final_Project/common_functions/modeling_readmission.py')
sys.path.append('/content/drive/MyDrive/DL4H_Team_67_Final_Project/common_functions/other_func.py')
sys.path.append('/content/drive/MyDrive/DL4H_Team_67_Final_Project/common_functions/utils_ftltrans.py')
proj_dir = ('/content/drive/MyDrive/DL4H_Team_67_Final_Project/')
```

```
# import  packages you need
import numpy as np
from google.colab import drive
from tqdm import tqdm, trange
import pandas as pd
import io
import os
import time
import numpy as np
import matplotlib.pyplot as plt
import re
import argparse
import torch
from pytorch_transformers import BertTokenizer

import time
import os
import torch
import random
from pytorch_transformers import BertTokenizer, BertConfig

import pandas as pd
import io
import numpy as np
import matplotlib.pyplot as plt
from dotmap import DotMap
from torch import nn
import logging

#custom functions
from other_func import Tokenize_with_note_id_hour, concat_by_id_list_with_note_chunk_id_time
from other_func import convert_note_ids, flat_accuracy, write_performance, reorder_by_time
import modeling_readmission

from sklearn.metrics import roc_curve, precision_recall_curve, \
    auc, matthews_corrcoef, accuracy_score, precision_score, recall_score, f1_score
```

```
in the modeling class
```

Please refer to this link for our shared Google drive: https://drive.google.com/drive/u/1/folders/1hy5GfU9eopCpi9ZTXbhhEbDCRvVKwZ06

It includes all of the data, training scripts, and model objects for this project draft.

Introduction

The downstream task of clinical prediction (of in-hospital mortality, readmission, and certain infection types) is one of the most important tasks in deep learning in healthcare because of its apparent preventative benefits. If we know the specific risks of in-hospitcal mortality or readmission or certain infections, we could save patients' life or patients' time, and reduce heathcare system's waste in finance and fundamental resources.

This task is a big challenge because unlike traditional data input normally coming in a single structure or strings, clinical data, specifically clinical notes, are structured on multiple levels, for example, each note consists of sequences of words, and several notes belong to different visits for each patient. Additionally, although it might seem that we can just feed the chunks of data to the models, there is no consideration of discrepancy in time when inputting these chunks. That means no attention has been given to different length of time in between a patients' visits, while obviously in healthcare this is an essential factor.

There have been several State Of The Art models including pretrained BERT and ClinicalBERT that can efficiently perform the downstream task of clinical prediction. BERT, with its transformer encoder architecture and bi-directional self-attetion, has made a breakthrough for neural network in general. BERT's pretraining methods consist of two unsupervised tasks: masked language modeling and next sentence prediction. ClinicalBERT adopt this model with these same unsupervised tasks, but on clinical corpus from the MIMICIII dataset. Both of these preprocess text by splitting notes into equal-length sequences, i.e. chunks. However, they still lack the consideration for position within the multi-level structure of clinical data, and time awareness factor for data representation.

Paper explanation

The Flexible Time-aware LSTM Transformer (FTL-Trans) is trying to solve this problem with methods that better account for the irregularity in the spacing of clinical notes. While it uses content embeddings from a pre-trained ClinicalBERT model, the FTL-Trans accounts for the irregularity in the spacing of notes and incorporates both the position-enhanced chunk representation generated from content embeddings and sequential information as well as the time information into a patient representation. This patient representation is then fed into the classification layer.

This problem is challenging to tackle because it requires sophisticated integration of typical multi-level structure of clinical notes on top of splitting sequences previously done by SOTAs. At the same time, the model need to have a time-aware design so that there's attention learnt in each piece of data chunk.

The FTL-Trans model is innovative because it does not introduce completely new methods or techniques but rather make use of existing effective ones in deep learning and improve them on top of that. For dealing specifically with the three above-mentioned limitations in SOTAs, the authors use three different layers (out of 4 total layers): a transformer-encoder layer called Chunk Content Embedding Layer, a a merging layer called Position-Enhanced Chunk Embedding Layer that includes Global Position embedding for note position, and Local Position embedding for chunk position, and lastly, a Time-Aware Layer that employs a Flexible Time-aware LSTM architecture. How is this enhanced LSTM layer different from traditional LSTM is that it does not just account for the order of the input data, but also the fact that clinical notes are not equally-spaced in time. It allows temporal information, both long-term and short-term, to have trainable parameters.

FLT-Trans was tested in comparison with BERT and ClinicalBERT on 5 different downstream tasks:

- In-hospital mortality prediction
- 30-day readmission prediction
- Escherichia Coli Infection prediction
- Enterococcus Sp. infection prediction
- Klebsiella pneumoniae infection prediction

On these tasks, the paper claims its metrics show an overall increase of 5% in AUROC and 6% in accuray compared to the SOTAs. Specifically, for mortality cohort, FTL-Trans supposedly scored a 0.95 AUC against those of 0.88-0.94 range for the other baseline models. The paper also claims that downstream task training FTL-Trans with different types of embedding shows that multi-level position embedding performed the best, esp. in mortality cohort (biggest cohort - doubling the size of the second largest dataset), drawing a conclusion that a shortage of data or limited size of data would deter the training success.

In this analysis, we will compare the paper's metrics obtained from reproducing (or an attempt to) FTL-Trans and running two other baseline models, on two main tasks of readmission and mortality predictions, in our setup following the paper's guide.

- The reason why we chose bert-sm and tlsthm models as our two baselines is because we would like to experiment with the simplest option as well as almost the most complex one. Bert-sm is the core model of all the models, without any time attention and with only single-level data structure, also pre-trained on non-clinical dataset. ClinicalBert-tlstm gives us the clinical data pre-training, the multi-level data structure but not the same time attention as FTL, but with some fixed time attention. Compared against these two baselines, the effectiveness of the flexible time attention in FTL, we expect, can be easily detected.
- The reason why we chose these two cohort is because they have highest number of notes and patients of all the cohorts by significant amount. The general usability of these two cohorts, in our opinion, is also of higher degree than the other specific-type-infection-based three.

This paper, if proved effective and correct, will be a significant contribution to the research regime because it accounts for more meaningful and possibly conclusive information obtained from data. For example, because BERT and ClinicalBERT do not have the information of which chunk belonging to which note, they do not have attention necessary for a comprehensive understanding of a patient's condition at a certain moment (visit), which can be crucial in assessing that patient's overall health progress or deterioration. Same observation applies to time-aware data representation - the order of the notes and how they are spread out especially can help depict the patient's overall health status over time.

Scope of Reproducibility

Clinical Notes are really good indicators of the health of patients if analysed correctly. The consideration of the more complex structure of clinical notes under a visit under a certain patient, and of time difference, not just of the order of notes in time, but also the uneven distribution of these notes along a certain timeline, are said to perform better than the models that don't incorporate attention to these details. Therefore, the hypotheses that we intend to explore are:

1. Hypothesis 1: If the FTL-Transformer is used to predict patients over time as the temporal learning of health status per person is a crucial part of this model, it will perform better than other transformers that are being used to make predictions on the simple chunks of sequences.
2. Hypothesis 2: The types of embeddings that will be used in the FTL-Transformer and see which one will provide better accuracy. From the paper, it can be seen that if multi-level position embeddings are used, then the general metrics will improve significantly.

We plan to run our FTL-Trans model and two baseline models: (1) a flat model such as BERT simple mean (BERT-sm) (Devlin et al., 2018) without multi-level tokenising and without time awareness, and (2) a model with time awareness based on a bidirectional T-LSTM (Baytas et al., 2017), called TL-Trans, which assumes that "temporal influence will always decay in a fixed mode" and therefore do not allow short-term attention to have trainable parameters.

Based on the above-mentioned experiment, we hope to come to conclusion regarding the paper's two main hypotheses.

Methodology

Data

The data is collected from the MIMIC III dataset, this dataset has many different tables and so a subset was taken. The data was grabbed from the NOTEEVENTS table and the ADMISSIONS table and then preprocessed so only the data that fit the criteria that was provided in the paper was present.

Mortality

For the mortality data set, we use the hospital expire flag from the Admissions table. If this flag is set to 1, this indicates that the patient has passed away in the critical care unit. We then use the clinical notes of a patient from their admission until one day before the patient's death. This is to make sure no direct mention of a patient's outcome is included in the data. Therefore, the patients who have only stayed one day are filtered out, because all of their notes are from the date of death or discharge. This then makes up the positive cohort and the negative cohort samples the same amount of patients and shows the noteevents for those patients

Readmission

In the Admissions table, re-admitted patients without scheduled appointments within 30 days of a prior discharge date are marked with a readmission flag. All other admissions are considered negative. We then filter out the in-hospital death and newborn admissions. Then we sample the same amount of negative patients.

Below is the code that was used to process the raw MIMIC data.

```
# def mortality_place(admissions_df, note_events_df):
#     admissions_df['ADMITTIME'] = pd.to_datetime(admissions_df['ADMITTIME'])
#     admissions_df['DISCHTIME'] = pd.to_datetime(admissions_df['DISCHTIME'])
#     positive_admissions_df = admissions_df[(admissions_df['HOSPITAL_EXPIRE_FLAG'] == 1) &
#                                            (admissions_df['ADMITTIME'].dt.date != admissions_df['DISCHTIME'].dt.date)]
#     print(len(positive_admissions_df['HADM_ID'].unique()))
#     negative_admissions = admissions_df[admissions_df['HOSPITAL_EXPIRE_FLAG'] == 0]
#     negative_admissions_df = negative_admissions.sample(n=len(positive_admissions_df), replace=False)
#     filtered_admissions_df = pd.concat([positive_admissions_df, negative_admissions_df], ignore_index=True)
#     note_events_df['CHARTDATE'] = pd.to_datetime(note_events_df['CHARTDATE'])
#     merged_df = note_events_df.merge(filtered_admissions_df[['HADM_ID', 'HOSPITAL_EXPIRE_FLAG', 'DISCHTIME']], on='HADM_ID', h
#     mortality_positive_notes = merged_df[(merged_df['HOSPITAL_EXPIRE_FLAG'] == 1) &
#                                          (merged_df['CHARTDATE'] < merged_df['DISCHTIME'].dt.date)]
#     negative_notes = merged_df[(merged_df['HOSPITAL_EXPIRE_FLAG'] == 0) &
#                                (merged_df['CATEGORY'] != 'Discharge summary') &
#                                (merged_df['CHARTDATE'] < merged_df['DISCHTIME'].dt.date - pd.Timedelta(days=1))]
#     mortality_positive_notes = mortality_positive_notes.rename(columns={'HADM_ID': 'Adm_ID',
#                                                                         'ROW_ID': 'Note_ID',
#                                                                         'CHARTDATE': 'chartdate',
#                                                                         'CHARTTIME': 'charttime',
#                                                                         'TEXT': 'TEXT'})
#     mortality_positive_notes['Label'] = 1
#     negative_notes = negative_notes.rename(columns={'HADM_ID': 'Adm_ID',
#                                                     'ROW_ID': 'Note_ID',
#                                                     'CHARTDATE': 'chartdate',
#                                                     'CHARTTIME': 'charttime',
#                                                     'TEXT': 'TEXT'})
#     negative_notes['Label'] = 0
#     final_dataset = pd.concat([mortality_positive_notes, negative_notes], ignore_index=True)
#     final_dataset.drop(columns=['SUBJECT_ID', 'STORETIME', 'CATEGORY', 'DESCRIPTION', 'CGID', 'ISERROR', 'HOSPITAL_EXPIRE_FLAG
#     final_dataset_df=final_dataset.sample(frac=1).reset_index(drop=True)
#     final_dataset_df.to_csv('final_dataset_mortality.csv', index=False)


# def readmit_place(admissions_df, note_events_df):
#     admissions_df['ADMITTIME'] = pd.to_datetime(admissions_df['ADMITTIME'])
#     admissions_df['DISCHTIME'] = pd.to_datetime(admissions_df['DISCHTIME'])
#     admissions_df['READMIT'] = 0
#     admissions_df['NEXT_ADMITTIME'] = admissions_df.groupby('SUBJECT_ID')['ADMITTIME'].shift(-1)  # Get next admission time fo
#     admissions_df['DAYS_BETWEEN'] = (admissions_df['NEXT_ADMITTIME'] - admissions_df['DISCHTIME']).dt.days  # Compute days bet
#     admissions_df.loc[(admissions_df['DAYS_BETWEEN'] >= 0) & (admissions_df['DAYS_BETWEEN'] <= 30), 'READMIT'] = 1
#     admissions_df.loc[(admissions_df['DAYS_BETWEEN'] >= 0) & (admissions_df['DAYS_BETWEEN'] <= 30) &
#                       (admissions_df['ADMISSION_TYPE'] == 'ELECTIVE'), 'READMIT'] = 0
#     admissions_df = admissions_df[(admissions_df['HOSPITAL_EXPIRE_FLAG'] == 0) & (admissions_df['ADMISSION_TYPE'] != 'NEWBORN'
#     positive_admissions_df = admissions_df[admissions_df['READMIT'] == 1]
#     print(len(positive_admissions_df['HADM_ID'].unique()))
#     negative_admissions = admissions_df[admissions_df['READMIT'] == 0]
#     negative_admissions_df = negative_admissions.sample(n=len(positive_admissions_df), replace=False)
#     filtered_admissions_df = pd.concat([positive_admissions_df, negative_admissions_df], ignore_index=True)
#     merged_df = note_events_df.merge(filtered_admissions_df[['HADM_ID', 'READMIT']], on='HADM_ID', how='left')
#     readmit_positive_notes = merged_df[(merged_df['READMIT'] == 1)].copy()
#     readmit_negative_notes = merged_df[(merged_df['READMIT'] == 0)].copy()
#     readmit_positive_notes['Label'] = 1
#     readmit_negative_notes['Label'] = 0
#     final_dataset = pd.concat([readmit_positive_notes, readmit_negative_notes], ignore_index=True)
#     final_dataset = final_dataset.rename(columns={'HADM_ID': 'Adm_ID',
#                                                     'ROW_ID': 'Note_ID',
#                                                     'CHARTDATE': 'chartdate',
#                                                     'CHARTTIME': 'charttime',
#                                                     'TEXT': 'TEXT'})
#     final_dataset.drop(columns=['SUBJECT_ID', 'STORETIME', 'CATEGORY', 'DESCRIPTION', 'CGID', 'ISERROR', 'READMIT'], inplace=T
#     final_dataset_df=final_dataset.sample(frac=1).reset_index(drop=True)
#     final_dataset_df.to_csv('final_dataset_readmission.csv', index=False)
```

Data Loading

For this notebook, the compute required to run this code is very high so we will not be placing the model training within the notebook. However, in order to load the data that we are placing in this notebook, here are the instructions.

1. Place the folder this notebook is in into your drive
2. Connect to a runtime
3. Load the code below in sequential order

If these steps are followed the statistics of the data and the model training will be placed.

Data Statistics

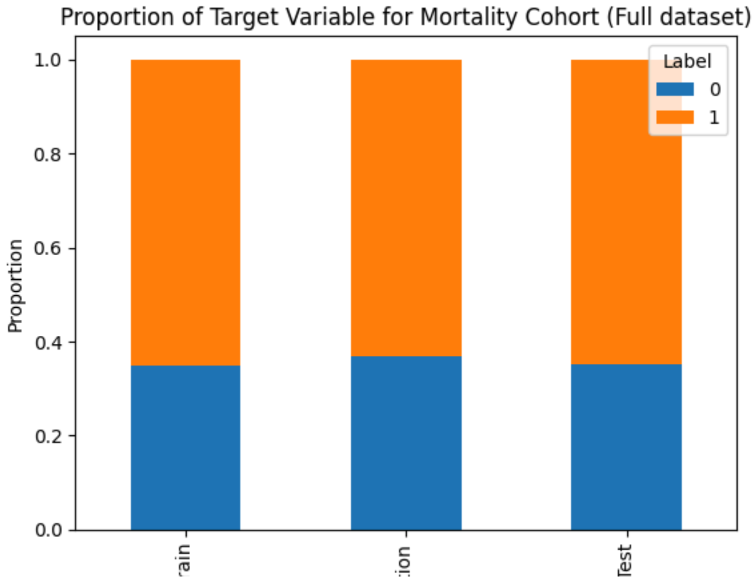
We first display summary statistics of our processed training, validation and test datasets. In addition, we also display summary statistics of the subset of the data that we used to train, validate and test our models on.

Here is the code we used to generate the random 1/500 subset of the processed data. We show statistics on this subset because this the data we used to train, validate and test our models on. We discuss this more in future sections, but the main reason for using a small subset of the data is due to a lack of compute resources to train the models on the full datasets.

```
# pd.read_csv(r'drive/MyDrive/Final_Project_DLH/data/full/train_readmission.csv').sample(frac=1/500, random_state = 42).to_csv(r
# pd.read_csv(r'drive/MyDrive/Final_Project_DLH/data/full/val_readmission.csv').sample(frac=1/500, random_state = 42).to_csv(r'd
# pd.read_csv(r'drive/MyDrive/Final_Project_DLH/data/full/test_readmission.csv').sample(frac=1/500, random_state = 42).to_csv(r'

# pd.read_csv(r'drive/MyDrive/Final_Project_DLH/data/full/train_mortality.csv').sample(frac=1/500, random_state = 42).to_csv(r'd
# pd.read_csv(r'drive/MyDrive/Final_Project_DLH/data/full/val_mortality.csv').sample(frac=1/500, random_state = 42).to_csv(r'dri
# pd.read_csv(r'drive/MyDrive/Final_Project_DLH/data/full/test_mortality.csv').sample(frac=1/500, random_state = 42).to_csv(r'dr
```

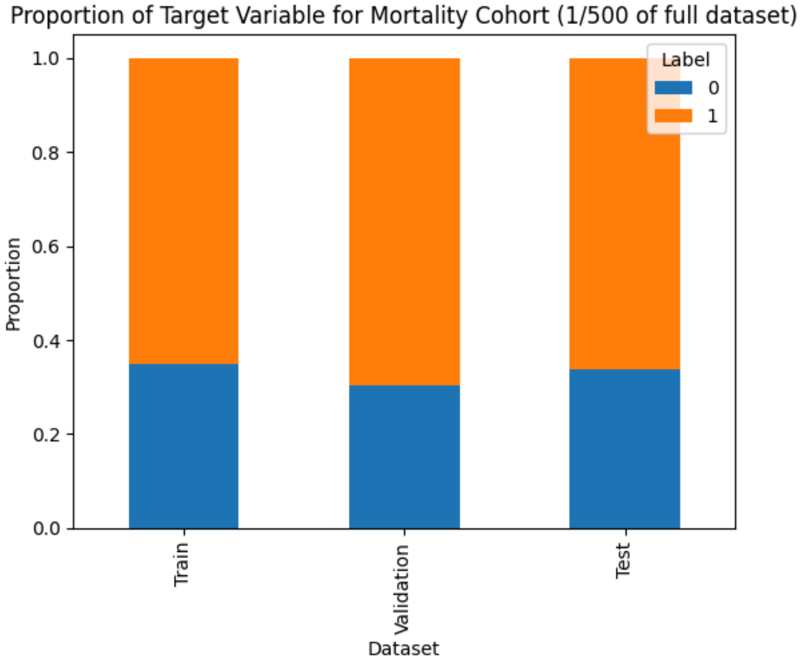
```
cohort_list = ["mortality", "readmission"]


def get_label_props(cohort, data_dir = proj_dir, subset=True):
  if subset:
    csv_label = ""
    title_label = "1/500 of full dataset"
    data_dir = os.path.join(data_dir, "data/subset_1_500")
  else:
    csv_label = ""
    title_label = "Full dataset"
    data_dir = os.path.join(data_dir, "data/full")


  cohort_df = pd.DataFrame()
  data_type_dict = {
      'train': 'training',
      'val': 'validation',
      'test': 'test'
  }
  print("COHORT: " + cohort.capitalize())
  for data_type in ['train', 'val', 'test']:
    df = pd.read_csv(os.path.join(data_dir, data_type + "_" + cohort + csv_label + ".csv"))

    print("Size of the " + data_type_dict[data_type] + " dataset:", df.shape[0])

    print("\nLabel distribution for " + data_type_dict[data_type] + " dataset:")
    print(df['Label'].value_counts())
    print("\n")

    df = df.groupby("Label")[['Label']].count().rename(columns={'Label': 'Count'})
    df['prop'] = df['Count'] / df['Count'].sum()
    df['data_type'] = data_type
    df = df.groupby(['data_type', 'Label'])['prop'].sum().unstack().fillna(0)
    df.reset_index(inplace=True)

    cohort_df = pd.concat([cohort_df, df])
  cohort_df['cohort'] = cohort
  cohort_df['data_type'] = np.where(
      cohort_df['data_type'] == 'val', 'validation', cohort_df['data_type']
  )

  cohort_df['data_type'] = cohort_df['data_type'].str.capitalize()
  print(cohort_df.to_string(index=False))


  cohort_df.plot(kind='bar', stacked=True, x = 'data_type')
  plt.title('Proportion of Target Variable for ' + cohort.capitalize() + ' Cohort (' + title_label + ')')
  plt.xlabel('Dataset')
  plt.ylabel('Proportion')
  plt.show()

  print("\n")


get_label_props(cohort = 'mortality', subset=False)
```

```
COHORT: Mortality
Size of the training dataset: 1028619

Label distribution for training dataset:
Label
1    669965
0    358654
Name: count, dtype: int64


Size of the validation dataset: 111416

Label distribution for validation dataset:
Label
1    70237
0    41179
Name: count, dtype: int64


Size of the test dataset: 146820

Label distribution for test dataset:
Label
1    95362
0    51458
Name: count, dtype: int64


 data_type        0        1    cohort
     Train 0.348675 0.651325 mortality
Validation 0.369597 0.630403 mortality
      Test 0.350484 0.649516 mortality
```



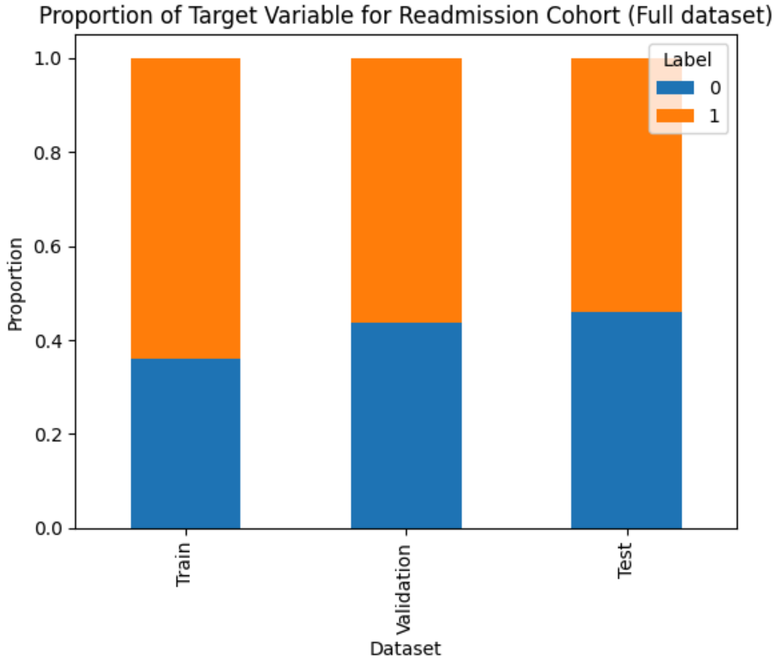
```
get_label_props(cohort = 'mortality', subset=True)
```

```
0     719
Name: count, dtype: int64


 Size of the validation dataset: 223

 Label distribution for validation dataset:
Label
1    155
0     68
Name: count, dtype: int64


 Size of the test dataset: 294

 Label distribution for test dataset:
Label
1    195
0     99
Name: count, dtype: int64


  data_type        0        1    cohort
      Train 0.349538 0.650462 mortality
 Validation 0.304933 0.695067 mortality
       Test 0.336735 0.663265 mortality
```



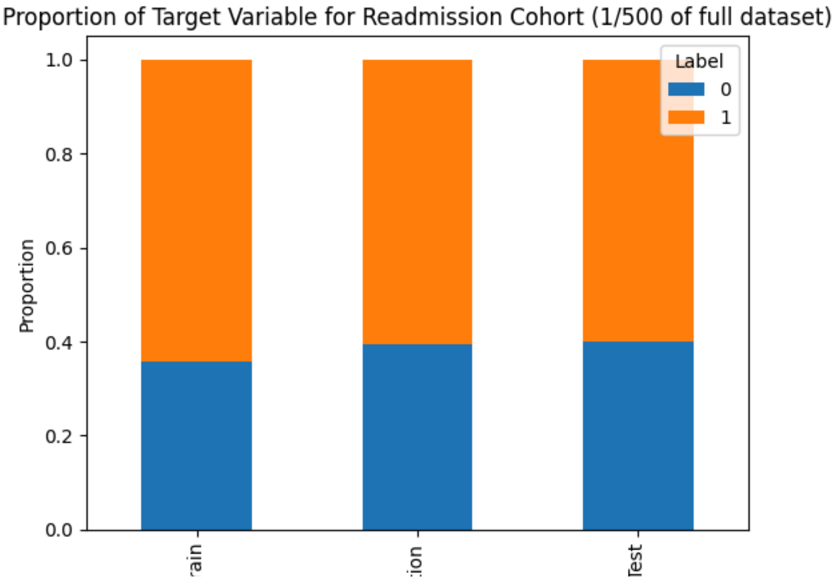
```
get_label_props(cohort = 'readmission', subset=False)
```

```
0    233058
Name: count, dtype: int64


Size of the validation dataset: 73260

Label distribution for validation dataset:
Label
1    41188
0    32072
Name: count, dtype: int64


Size of the test dataset: 78463

Label distribution for test dataset:
Label
1    42480
0    35983
Name: count, dtype: int64


 data_type        0        1      cohort
     Train 0.361303 0.638697 readmission
Validation 0.437783 0.562217 readmission
      Test 0.458598 0.541402 readmission
```



```
get_label_props(cohort = 'readmission', subset=True)
```

```
COHORT: Readmission
Size of the training dataset: 1290

Label distribution for training dataset:
Label
1    829
0    461
Name: count, dtype: int64


Size of the validation dataset: 147

Label distribution for validation dataset:
Label
1    89
0    58
Name: count, dtype: int64


Size of the test dataset: 157

Label distribution for test dataset:
Label
1    94
0    63
Name: count, dtype: int64


 data_type        0        1      cohort
     Train 0.357364 0.642636 readmission
Validation 0.394558 0.605442 readmission
      Test 0.401274 0.598726 readmission
```



Data Preprocessing

The data preprocessing code was taken from the Github that was given for the paper. In this, the data is loaded and then tokenized using a BERT tokenizer. Then, that data is then broken into chunks and then saved in temporary files. These temporary files are then read and placed and broken into train, validation and test sets. Finally, this data is then split into chunks so that way the data can be passed through the BERT model. This code is commented out because it is incredibly computationally intensive and actually took the group almost 2 days to run. We also used a random sample that was 1/500 of the original datasets because of how large the data is even after preprocessing.

```
# raw_data_dir = '/content/drive/MyDrive/Final_Project_DLH/'
# RANDOM_SEED = 42
# TEMP_DIR = raw_data_dir + 'temp_data'
# LOG_PATH = 'log.txt'
# MAX_LEN = 128


# # dir and function to load raw data
# def load_raw_data(raw_data_dir, filename):
#   original_df = pd.read_csv(raw_data_dir + filename, header=0)
#   return original_df
# raw_data_mortality = load_raw_data(raw_data_dir, 'final_dataset_mortality.csv')
# raw_data_readmission = load_raw_data(raw_data_dir, 'final_dataset_readmission.csv')


# # calculate statistics
# def calculate_stats(raw_data):
#   dataset_size = raw_data.shape
#   print("Size of the dataset (rows, columns):", dataset_size)
# calculate_stats(raw_data_ecoli)


# def write_log(content, log_path, print_content=True):
#     if os.path.exists(log_path):
#         with open(log_path, 'a') as f:
#             f.write("Time: " + time.ctime() + "\n")
#             f.write(content + "\n")
#             f.write("=====================\n")
#     else:
#         with open(log_path, 'w') as f:
#             f.write("Time: " + time.ctime() + "\n")
#             f.write(content + "\n")
#             f.write("=====================\n")
#     if print_content:
#         print(content)

# def preprocess1(x):
#     y = re.sub('\\[(.*?)\\]', '', x)  # remove de-identified brackets
#     y = re.sub('[0-9]+\.', '', y)  # remove 1.2. since the segmenter segments based on this
#     y = re.sub('dr\.', 'doctor', y)
#     y = re.sub('m\.d\.', 'md', y)
#     y = re.sub('admission date:', '', y)
#     y = re.sub('discharge date:', '', y)
#     y = re.sub('--|__|==', '', y)
#     return y


# def preprocessing(df_less_n, tokenizer):
#     df_less_n['TEXT'] = df_less_n['TEXT'].fillna(' ')
#     df_less_n['TEXT'] = df_less_n['TEXT'].str.replace('\n', ' ')
#     df_less_n['TEXT'] = df_less_n['TEXT'].str.replace('\r', ' ')
#     df_less_n['TEXT'] = df_less_n['TEXT'].apply(str.strip)
#     df_less_n['TEXT'] = df_less_n['TEXT'].str.lower()

#     df_less_n['TEXT'] = df_less_n['TEXT'].apply(lambda x: preprocess1(x))

#     sen = df_less_n['TEXT'].values
#     tokenized_texts = [tokenizer.tokenize(x) for x in sen]
#     print("First sentence tokenized")
#     print(tokenized_texts[0])
#     input_ids = [tokenizer.convert_tokens_to_ids(x) for x in tokenized_texts]
#     df_less_n['Input_ID'] = input_ids
#     return df_less_n[['Adm_ID', 'Note_ID', 'TEXT', 'Input_ID', 'Label', 'chartdate', 'charttime']]
# # process raw data
# def process_data(raw_data, filename, output_dir):
#   if os.path.exists(TEMP_DIR) and os.listdir(TEMP_DIR):
#       raise ValueError("Temp Output directory ({}) already exists and is not empty.".format(TEMP_DIR))
#   os.makedirs(TEMP_DIR, exist_ok=True)
#   tokenizer = BertTokenizer.from_pretrained("bert-base-uncased", do_lower_case=True)
#   for i in range(int(np.ceil(len(raw_data) / 10000))):
#         write_log("chunk {} tokenize start!".format(i), LOG_PATH)
#         df_chunk = raw_data.iloc[i * 10000:(i + 1) * 10000].copy()
#         df_processed_chunk = preprocessing(df_chunk, tokenizer)
#         df_processed_chunk = df_processed_chunk.astype({'Adm_ID': 'int64', 'Note_ID': 'int64', 'Label': 'int64'})
#         temp_file_dir = os.path.join(TEMP_DIR, 'Processed_{}.csv'.format(i))
#         df_processed_chunk.to_csv(temp_file_dir, index=False)
#   dfs = []
#   for i in range(int(np.ceil(len(raw_data) / 10000))):
#       temp_file_dir = os.path.join(TEMP_DIR, 'Processed_{}.csv'.format(i))
#       df_chunk = pd.read_csv(temp_file_dir, header=0)
#       write_log("chunk {} has {} notes".format(i, len(df_chunk)), LOG_PATH)
#       dfs.append(df_chunk)
#   df = pd.concat(dfs, ignore_index=True)
#   result = df.Label.value_counts()
#   write_log(
#         "In the full dataset Positive Patients' Notes: {}, Negative Patients' Notes: {}".format(result[1],
#                                                                                            result[0]),
#         LOG_PATH)

#   dead_ID = pd.Series(df[df.Label == 1].Adm_ID.unique())
#   not_dead_ID = pd.Series(df[df.Label == 0].Adm_ID.unique())
#   write_log("Total Positive Patients' ids: {}, Total Negative Patients' ids: {}".format(len(dead_ID), len(not_dead_ID)), LOG_F
#   id_num_pos = len(df[df.Label == 1].Adm_ID.unique())
#   id_num_neg = len(df[df.Label == 0].Adm_ID.unique())
#   id_num_pos = id_num_neg if id_num_neg < id_num_pos else id_num_pos
#   not_dead_ID_use = not_dead_ID.sample(n=id_num_pos, random_state=RANDOM_SEED)
#   dead_ID_use = dead_ID.sample(n=id_num_pos, random_state=RANDOM_SEED)

#   id_val_test_t = dead_ID_use.sample(frac=0.2, random_state=RANDOM_SEED)
#   id_val_test_f = not_dead_ID_use.sample(frac=0.2, random_state=RANDOM_SEED)

#   id_train_t = dead_ID_use.drop(id_val_test_t.index)
#   id_train_f = not_dead_ID_use.drop(id_val_test_f.index)

#   id_val_t = id_val_test_t.sample(frac=0.5, random_state=RANDOM_SEED)
#   id_test_t = id_val_test_t.drop(id_val_t.index)
#   id_val_f = id_val_test_f.sample(frac=0.5, random_state=RANDOM_SEED)
#   id_test_f = id_val_test_f.drop(id_val_f.index)

#   id_test = pd.concat([id_test_t, id_test_f])
#   test_id_label = pd.DataFrame(data=list(zip(id_test, [1] * len(id_test_t) + [0] * len(id_test_f))),
#                                 columns=['id', 'label'])

#   id_val = pd.concat([id_val_t, id_val_f])
#   val_id_label = pd.DataFrame(data=list(zip(id_val, [1] * len(id_val_t) + [0] * len(id_val_f))),
#                               columns=['id', 'label'])

#   id_train = pd.concat([id_train_t, id_train_f])
#   train_id_label = pd.DataFrame(data=list(zip(id_train, [1] * len(id_train_t) + [0] * len(id_train_f))),
#                                 columns=['id', 'label'])

#   mortality_train = df[df.Adm_ID.isin(train_id_label.id)]
#   mortality_val = df[df.Adm_ID.isin(val_id_label.id)]
#   mortality_test = df[df.Adm_ID.isin(test_id_label.id)]
#   mortality_not_use = df[
#       (~df.Adm_ID.isin(train_id_label.id)) & (~df.Adm_ID.isin(val_id_label.id) & (~df.Adm_ID.isin(test_id_label.id)))]

#   train_result = mortality_train.Label.value_counts()

#   val_result = mortality_val.Label.value_counts()

#   test_result = mortality_test.Label.value_counts()

#   no_result = mortality_not_use.Label.value_counts()

#   mortality_train.to_csv(os.path.join(output_dir, 'train_' + filename +'.csv'), index=False)
#   mortality_val.to_csv(os.path.join(output_dir, 'val_' + filename + '.csv'), index=False)
#   mortality_test.to_csv(os.path.join(output_dir, 'test_' + filename + '.csv'), index=False)
#   mortality_not_use.to_csv(os.path.join(output_dir, 'not_use_' + filename + '.csv'), index=False)
#   df.to_csv(os.path.join(output_dir, 'full_' + filename + '.csv'), index=False)
#   if len(no_result) == 2:
#       write_log(("In the train dataset Positive Patients' Notes: {}, Negative Patients' Notes: {}\n"
#                   "In the validation dataset Positive Patients' Notes: {}, Negative Patients' Notes: {}\n"
#                   "In the test dataset Positive Patients' Notes: {}, Negative Patients' Notes: {}\n"
#                   "In the not use dataset Positive Patients' Notes: {}, Negative Patients' Notes: {}").format(
#           train_result[1],
#           train_result[0],
#           val_result[1],
#           val_result[0],
#           test_result[1],
#           test_result[0],
#           no_result[1],
#           no_result[0]),
#           LOG_PATH)
#   else:
#       try:
#           write_log(("In the train dataset Positive Patients' Notes: {}, Negative  Patients' Notes: {}\n"
#                       "In the validation dataset Positive Patients' Notes: {}, Negative Patients' Notes: {}\n"
#                       "In the test dataset Positive Patients' Notes: {}, Negative Patients' Notes: {}\n"
#                       "In the not use dataset Negative Patients' Notes: {}").format(train_result[1],
#                                                                                     train_result[0],
#                                                                                     val_result[1],
#                                                                                     val_result[0],
#                                                                                     test_result[1],
#                                                                                     test_result[0],
#                                                                                     no_result[0]),
#                     LOG_PATH)
#       except KeyError:
#           write_log(("In the train dataset Positive Patients' Notes: {}, Negative  Patients' Notes: {}\n"
#                       "In the validation dataset Positive Patients' Notes: {}, Negative Patients' Notes: {}\n"
#                       "In the test dataset Positive Patients' Notes: {}, Negative Patients' Notes: {}\n"
#                       "In the not use dataset Positive Patients' Notes: {}").format(train_result[1],
#                                                                                     train_result[0],
#                                                                                     val_result[1],
#                                                                                     val_result[0],
#                                                                                     test_result[1],
#                                                                                     test_result[0],
#                                                                                     no_result[1]),
#                     LOG_PATH)

#   write_log("Data saved in the {}".format(output_dir), LOG_PATH)
# def split_into_chunks(df, max_len):
#     input_ids = df.Input_ID.apply(lambda x: x[1:-1].replace(' ', '').split(','))
#     df_len = len(df)
#     Adm_ID, Note_ID, Input_ID, Label, chartdate, charttime = [], [], [], [], [], []
#     for i in tqdm(range(df_len)):
#         x = input_ids[i]
#         n = int(len(x) / (max_len - 2))
#         for j in range(n):
#             Adm_ID.append(df.Adm_ID[i])
#             Note_ID.append(df.Note_ID[i])
#             sub_ids = x[j * (max_len - 2): (j + 1) * (max_len - 2)]
#             sub_ids.insert(0, '101')
#             sub_ids.append('102')
#             Input_ID.append(' '.join(sub_ids))
#             Label.append(df.Label[i])
#             chartdate.append(df.chartdate[i])
#             charttime.append(df.charttime[i])
#         if len(x) % (max_len - 2) > 10:
#             Adm_ID.append(df.Adm_ID[i])
#             Note_ID.append(df.Note_ID[i])
#             sub_ids = x[-((len(x)) % (max_len - 2)):]
#             sub_ids.insert(0, '101')
#             sub_ids.append('102')
#             Input_ID.append(' '.join(sub_ids))
#             Label.append(df.Label[i])
#             chartdate.append(df.chartdate[i])
#             charttime.append(df.charttime[i])
#     new_df = pd.DataFrame({'Adm_ID': Adm_ID,
#                             'Note_ID': Note_ID,
#                             'Input_ID': Input_ID,
#                             'Label': Label,
#                             'chartdate': chartdate,
#                             'charttime': charttime})
#     new_df = new_df.astype({'Adm_ID': 'int64', 'Note_ID': 'int64', 'Label': 'int64'})
#     return new_df


# def split_chunks(output_dir, filename, raw_data_dir):
#   train_file_path = os.path.join(raw_data_dir, 'train_' + filename +'.csv')
#   val_file_path = os.path.join(raw_data_dir, 'val_' + filename + '.csv')
#   test_file_path = os.path.join(raw_data_dir, 'test_' + filename + '.csv')
#   train_df = pd.read_csv(train_file_path)
#   val_df = pd.read_csv(val_file_path)
#   test_df = pd.read_csv(test_file_path)

#   new_train_df = split_into_chunks(train_df, MAX_LEN)
#   new_val_df = split_into_chunks(val_df, MAX_LEN)
#   new_test_df = split_into_chunks(test_df, MAX_LEN)

#   train_result = new_train_df.Label.value_counts()
#   val_result = new_val_df.Label.value_counts()
#   test_result = new_test_df.Label.value_counts()

#   write_log(("In the train dataset Positive Patients' Chunks: {}, Negative Patients' Chunks: {}\n"
#               "In the validation dataset Positive Patients' Chunks: {}, Negative Patients' Chunks: {}\n"
#               "In the test dataset Positive Patients' Chunks: {}, Negative Patients' Chunks: {}").format(train_result[1],
#                                                                                                 train_result[0],
#                                                                                                 val_result[1],
#                                                                                                 val_result[0],
#                                                                                                 test_result[1],
#                                                                                                 test_result[0]),
#             LOG_PATH)

#   new_train_df.to_csv(os.path.join(output_dir, 'train_' + filename +'.csv'), index=False)
#   new_val_df.to_csv(os.path.join(output_dir, 'val_' + filename + '.csv'), index=False)
#   new_test_df.to_csv(os.path.join(output_dir, 'test_' + filename + '.csv'), index=False)

#   write_log("Split finished", LOG_PATH)


# output_process_dir = raw_data_dir + 'output_preprocess_data'
# processed_data = process_data(raw_data_ecoli, 'ecoli', output_process_dir)
# for file in os.listdir(TEMP_DIR)[:53334]:
#     os.remove(TEMP_DIR+'/' + file)
# output_split_dir = raw_data_dir + 'output_split_data'
# raw_data_dir_split = raw_data_dir + 'output_preprocess_data'
# split_chunks(output_split_dir, 'ecoli', raw_data_dir_split)
```

Models

BERT-SM

The model architecture of the BERT-sm includes a multi-layer bidirectional Transformer encoder. We're running BERT base models across all experiments, this comes with 12 layers (Transformer blocks), 12 attention heads and total parameters of 110M. BERT's creators use GeLU (rather than ReLU) activation function (Devlin et al., 2018). The training objectives are cross entropy loss with Adams optimiser at a learning rate of 2e-5 (Devlin et al., 2018). BERT based was pretrained on Book-Corpus (ref) and English Wikipedia (Devlin et al., 2018). The sequence of notes is split into chunks before being fed into the model and afterwards given a probability against a threshold number to predict which label it should be.

The majority of the BERT-sm code we use has been adapted from the paper's code. The complexity and compute limitation forced us to train this model on the two downstream tasks in an environment outside of this notebook. We have saved the trained model to import in the following code.

```
bert_sm_model_readmission = torch.load(os.path.join(proj_dir, 'bert/bert_sm_readmission_1_500.pt'))
bert_sm_model_mortality = torch.load(os.path.join(proj_dir, 'bert/bert_sm_mortality_1_500.pt'), map_location=torch.device('cpu')
```

TL-Trans

The model architecture of the TL-Trans is very similar to that of the FTL-Trans, but employing bidirectional T-LSTM encoder architecture. Our experiment uses 1 T-LSTM layer. The layer uses Tanh as activation function (Baytas et al., 2017). THe T-LSTM uses a non-increasing function of the elapsed time which transforms the time lapse into an appropriate weight. Mini-batch stochastic Adam optimizer of learning rate of 2×10−5 was employed and all the weights were learned simultaneously and in a data-driven manner (Baytas et al., 2017). This model has ClinicalBERT core, which is a pre-trained BERT using a medical corpus (Zhang et al., 2020).

Similarly as BERT-sm, the code we use in our experiment has been largely adapted from the paper's code. The complexity and compute limitation have us finish training this model on the two downstream tasks in an environment outside of this notebook. We have saved the trained model to import in the following code.

```
# Loading readmission TLSTM model
tlstm_model_readmission = torch.load(os.path.join(proj_dir, 'tlstm/tlstm_readmission_1_500.pt'), map_location=torch.device('cpu'
tlstm_layer_readmission = torch.load(os.path.join(proj_dir, 'tlstm/tlstm_readmission_lstmlayer_1_500.pt'), map_location=torch.de

# Loading mortality TLSTM model
tlstm_model_mortality = torch.load(os.path.join(proj_dir, 'tlstm/tlstm_mortality_1_500.pt'), map_location=torch.device('cpu')) #
tlstm_layer_mortality = torch.load(os.path.join(proj_dir, 'tlstm/tlstm_mortality_lstmlayer_1_500.pt'), map_location=torch.device
```

FTL-Trans

This is the paper's proposed model.

The model contains four layers: Chunk Content Embedding Layer is a ClinicalBERT layer, the chunk embeddings output of this layer (instead of being directly used for prediction as in ClinicalBERT), are then fed into Position-Enhanced Chunk Embedding Layer, which merges chunk and note information (local and global positions) into a single representation. These embeddings then go to a Flexible T-LSTM which does not employ a non-increasing function, but rather a flexible and universal decay function where all parameters involved are train-able. The last layer is Classification Layer consists of a dropout layer, a single perceptron and lastly a sigmoid function (Zhang et al., 2020). The training objectives include a flexible and universal decay function where all parameters involved are train-able, BertAdam optimizer with an initial learning rate of 2×10−5 and a warm-up proportion of 0.1 (Zhang et al., 2020). As mentioned above, the core of the first layer is ClinicalBERT which was pre-trained BERT using a medical corpus.

Similarly as other models, the code we use in our experiment has been largely adopted from the paper's code. The complexity and compute limitation have us finish training this model on the two downstream tasks in an environment outside of this notebook. We have saved the trained model to import in the following code.

```
# Loading readmission ftlstm model
ftlstm_model_readmission = torch.load(os.path.join(proj_dir, 'ftl_trans/ftlstm_readmission_1_500.pt'), map_location=torch.device
ftlstm_layer_readmission = torch.load(os.path.join(proj_dir, 'ftl_trans/ftlstm_lstmlayer_readmission_1_500.pt'), map_location=tc

# Loading mortality ftlstm model
ftlstm_model_mortality = torch.load(os.path.join(proj_dir, 'ftl_trans/ftlstm_1_500_mortality.pt'), map_location=torch.device('cp
ftlstm_layer_mortality = torch.load(os.path.join(proj_dir, 'ftl_trans/ftlstm_lstmlayer_1_500_mortality.pt'), map_location=torch.
```

Training

We do not provide any of the training code in this document. The training scripts for all three models are extremely long and are provided as separate files in the Google drive. Please refer to `run_[model].ipynb` for the training code for any specific model - the three model files are located in the Google drive.

For the results below, we trained, validated and tested all three models on 1/500 of their respective original datasets for 5 epochs on a GPU. On average, training the readmission models took roughly 12 minutes per epoch and the mortality models took roughly 20 minutes per epoch. All three models were trained with a learning rate of 2e-5, a batch size of 1, a hidden size of 768, and a dropout probability of 0.1.

Results

We evaluate the models on the test set with the following metrics: Area Under the Receiver Operating Characteristic curve (AUROC), Accuracy, Area Under Precision-Recall curve (AUPR) and F1 score.

Unlike the paper, the FTL-Trans actually does not significantly outperform the alternative methods across the readmission and mortality cohorts. For both tasks, all three models report the same accuracy and F1 scores. The FTL-Trans has the highest AUPR for both cohorts. It also reports the highest AUC on the readmission cohort, while the BERT-sm reports the highest AUC on the mortality cohort. The reason why all three models have the same accuracy is because the models only predict a target label of 1. If we're able to use more data to train the models, this would definitely improve the models' predictive power.

The biggest limitation is using only 1/500 of the original training data, so it's hard for the more complex models TL-Trans and FTL-Trans to properly learn signals from the data.

Please see the table below for a comparison of these metrics across the three models and two cohorts.

With respect to our hypotheses, we expected the FTL-Trans model to perform the best due to its use of multi-level position embeddings. However, it is clear that none of these models perform that well. Compared to the paper, the results are much worse - this is because we were only able to train our models on 1/500 of the original training data.

```
def model_auc(y_true, y_pred):
    fpr, tpr, thresholds = roc_curve(y_true, y_pred)
    auc_score = auc(fpr, tpr)
    return auc_score, fpr, tpr, thresholds


def model_aupr(y_true, y_pred):
    precision, recall, thresholds = precision_recall_curve(y_true, y_pred)
    aupr_score = auc(recall, precision)
    return aupr_score, precision, recall, thresholds

def write_performance(flat_true_labels, flat_predictions, flat_logits):
    test_accuracy = accuracy_score(flat_true_labels, flat_predictions)

    test_f1 = f1_score(flat_true_labels, flat_predictions, average='binary')

    test_auc, _, _, _ = model_auc(flat_true_labels, flat_logits)

    test_aupr, _, _, _ = model_aupr(flat_true_labels, flat_logits)


    return pd.DataFrame({'test_Accuracy': test_accuracy,
            'test_F1': test_f1,
            'test_AUC': test_auc,
            'test_AUPR': test_aupr}, index=[0])
```

```
def process_test_data(data_path):
  df = pd.read_csv(data_path)
  logging.info("Data has been loaded")
  random.seed(42)
  np.random.seed(42)
  torch.manual_seed(42)

  tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)

  test_df = reorder_by_time(df)

  test_labels, test_inputs, test_masks, test_note_ids, test_times = Tokenize_with_note_id_hour(df, 128,
                                                                                                tokenizer)
  test_inputs = torch.tensor(test_inputs)
  test_labels = torch.tensor(test_labels)
  test_masks = torch.tensor(test_masks)
  test_times = torch.tensor(test_times)


  (test_labels, test_inputs,
  test_masks, test_ids,
  test_note_ids, test_chunk_ids, test_times) = concat_by_id_list_with_note_chunk_id_time(test_df, test_labels,
                                                                                        test_inputs, test_masks,
                                                                                        test_note_ids, test_times,
                                                                                        128)

  return (test_labels, test_inputs,
    test_masks, test_ids,
    test_note_ids, test_chunk_ids, test_times)

def print_results_bert(data_path, model):

  (test_labels, test_inputs,
    test_masks, test_ids,
    test_note_ids, test_chunk_ids, test_times) = process_test_data(data_path)

  m = torch.nn.Softmax(dim=1)


  if model is not None:
    model.eval()
    logging.info("Model is in eval mode")

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

  # Tracking variables
  predictions, true_labels = [], []

  # Predict
  te_ids_num = len(test_ids)
  for step in range(te_ids_num):
      b_input_ids = test_inputs[step][-64:, :].to(device)
      b_input_mask = test_masks[step][-64:, :].to(device)
      b_labels = test_labels[step].repeat(b_input_ids.shape[0])
      # Telling the model not to compute or store gradients, saving memory and speeding up prediction
      with torch.no_grad():
          # Forward pass, calculate logit predictions
          outputs = model(b_input_ids, token_type_ids=None, attention_mask=b_input_mask)

      # Move logits and labels to CPU
      logits = outputs[-1]
      logits = m(logits).detach().cpu().numpy()[:, 1].mean()
      label_ids = b_labels.numpy().max()

      # Store predictions and true labels
      predictions.append(logits)
      true_labels.append(label_ids)

  flat_logits = predictions
  flat_predictions = (np.array(flat_logits) >= 0.5).astype(int)
  flat_true_labels = true_labels

  output_df = pd.DataFrame({'logits': flat_logits,
                            'pred_label': flat_predictions,
                            'label': flat_true_labels,
                            'Adm_ID': test_ids})
  res = write_performance(flat_true_labels, flat_predictions, flat_logits)
  return output_df, res


def print_results_lstm(data_path, model, layer):

  if model is not None:
    model.eval()
    logging.info("Model is in eval mode")
  if layer is not None:
    layer.eval()
    logging.info("LSTM layer is in eval mode")

  (test_labels, test_inputs,
    test_masks, test_ids,
    test_note_ids, test_chunk_ids, test_times) = process_test_data(data_path)
  logging.info("Data has been loaded")

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

  # Tracking variables
  predictions, true_labels = [], []

  # Predict
  te_ids_num = len(test_ids)
  for step in range(te_ids_num):
      b_input_ids = test_inputs[step][-64:, :].to(device)
      b_input_mask = test_masks[step][-64:, :].to(device)
      b_note_ids = test_note_ids[step][-64:]
      b_new_note_ids = convert_note_ids(b_note_ids).to(device)
      b_chunk_ids = test_chunk_ids[step][-64:].unsqueeze(0).to(device)
      b_labels = test_labels[step]
      b_labels.resize_((1))
      with torch.no_grad():
          _, whole_output = model(b_input_ids, token_type_ids=None, attention_mask=b_input_mask)
          whole_input = whole_output.unsqueeze(0)
          b_new_note_ids = b_new_note_ids.unsqueeze(0)
          b_times = test_times[step][-64:].unsqueeze(0).to(device)
          pred = layer(whole_input, b_times, b_new_note_ids, b_chunk_ids).detach().cpu().numpy()
      label_ids = b_labels.numpy()[0]
      predictions.append(pred)
      true_labels.append(label_ids)

  # Flatten the predictions and true values for aggregate Matthew's evaluation on the whole dataset
  flat_logits = [item for sublist in predictions for item in sublist]
  flat_predictions = np.asarray([1 if i else 0 for i in (np.array(flat_logits) >= 0.5)])
  flat_true_labels = np.asarray(true_labels)

  output_df = pd.DataFrame({'pred_prob': flat_logits,
                            'pred_label': flat_predictions,
                            'label': flat_true_labels,
                            'Adm_ID': test_ids})

  res = write_performance(flat_true_labels, flat_predictions, flat_logits)

  return output_df, res
```

```
output_df_bert_readmission, res_bert_readmission = print_results_bert(
    data_path = os.path.join(proj_dir, 'data/subset_1_500/test_readmission.csv'),
    model = bert_sm_model_readmission
)
```

```
    100%|██████████| 231508/231508 [00:00<00:00, 908583.44B/s]
```

```
output_df_bert_mortality, res_bert_mortality = print_results_bert(
    data_path = os.path.join(proj_dir, 'data/subset_1_500/test_mortality.csv'),
    model = bert_sm_model_mortality
)
```

```
output_df_tlstm_readmission, res_tlstm_readmission = print_results_lstm(
    data_path = os.path.join(proj_dir, 'data/subset_1_500/test_readmission.csv'),
    model = tlstm_model_readmission,
    layer = tlstm_layer_readmission
)


output_df_tlstm_mortality, res_tlstm_mortality = print_results_lstm(
    data_path = os.path.join(proj_dir, 'data/subset_1_500/test_mortality.csv'),
    model = tlstm_model_mortality,
    layer = tlstm_layer_mortality
)


output_df_ftl_trans_readmission, res_ftl_trans_readmission = print_results_lstm(
    data_path = os.path.join(proj_dir, 'data/subset_1_500/test_readmission.csv'),
    model = ftlstm_model_readmission,
    layer = ftlstm_layer_readmission
)


output_df_ftl_trans_mortality, res_ftl_trans_mortality  = print_results_lstm(
    data_path = os.path.join(proj_dir, 'data/subset_1_500/test_mortality.csv'),
    model = ftlstm_model_mortality,
    layer = ftlstm_layer_mortality
)


# print(output_df_bert_readmission['pred_label'].value_counts())
# print(output_df_tlstm_readmission['pred_label'].value_counts())
# print(output_df_ftl_trans_readmission['pred_label'].value_counts())
# print(output_df_bert_mortality['pred_label'].value_counts())
# print(output_df_tlstm_mortality['pred_label'].value_counts())
# print(output_df_ftl_trans_mortality['pred_label'].value_counts())


res_bert_readmission['Model'] = 'BERT-sm'
res_tlstm_readmission['Model'] = 'TL-Trans'
res_ftl_trans_readmission['Model'] = 'FTL-Trans'
readmission_results = pd.concat([res_bert_readmission, res_tlstm_readmission, res_ftl_trans_readmission])

readmission_results.columns = readmission_results.columns.str.replace('test_', '')
print("Readmission")
print(readmission_results[['Model', 'AUC', 'Accuracy', 'AUPR', 'F1']])
```

```
Readmission
       Model       AUC  Accuracy      AUPR        F1
0    BERT-sm  0.489013  0.650485  0.599368  0.788235
0   TL-Trans  0.385365  0.650485  0.652907  0.788235
0  FTL-Trans  0.500000  0.650485  0.825243  0.788235
```

```
res_bert_mortality['Model'] = 'BERT-sm'
res_tlstm_mortality['Model'] = 'TL-Trans'
res_ftl_trans_mortality['Model'] = 'FTL-Trans'
mortality_results = pd.concat([res_bert_mortality, res_tlstm_mortality, res_ftl_trans_mortality])

mortality_results.columns = mortality_results.columns.str.replace('test_', '')
print("Mortality")
print(mortality_results[['Model', 'AUC', 'Accuracy', 'AUPR', 'F1']])
```

```
Mortality
       Model       AUC  Accuracy      AUPR        F1
0    BERT-sm  0.521684  0.723164  0.752738  0.839344
0   TL-Trans  0.380580  0.723164  0.679433  0.839344
0  FTL-Trans  0.500000  0.723164  0.861582  0.839344
```

Ablations

1. Optimization Change

All the models we were testing use Adaptive Moment Estimation (Adam) algorithm for their optimization. While we understand why the authors chose to use this, thanks to the optimization method's attention as well as bias correction to both first and second moments of the gradient, we would like to experiment with a little simpler optimization algorithm such as SGD (Stochastic Gradient Descent). We think that maybe a simpler and more random-but-of-general-approach algorithm can expose what might be the problem why FTL-Trans did not work or give us some interesting observations. However, after trying with SGD the performances are somewhat similar and no difference was found with the case of FTL-Trans. As also mentioned above, this SGD experiment did not help to fix the FTL-Trans model.

```
bert_sm_model_readmission_sgd = torch.load(os.path.join(proj_dir, 'bert/bert_sm_readmission_sgd.pt'), map_location=torch.device(
bert_sm_model_mortality_sgd = torch.load(os.path.join(proj_dir, 'bert/bert_sm_mortality_1_500_sgd.pt'), map_location=torch.devic

# Loading readmission TLSTM model
tlstm_model_readmission_sgd = torch.load(os.path.join(proj_dir, 'tlstm/tlstm_readmission_sgd.pt'), map_location=torch.device('cp
tlstm_layer_readmission_sgd = torch.load(os.path.join(proj_dir, 'tlstm/tlstm_readmission_lstmlayer_sgd.pt'), map_location=torch.

# Loading mortality TLSTM model
tlstm_model_mortality_sgd = torch.load(os.path.join(proj_dir, 'tlstm/tlstm_mortality_sgd.pt'), map_location=torch.device('cpu'))
tlstm_layer_mortality_sgd = torch.load(os.path.join(proj_dir, 'tlstm/tlstm_mortality_lstmlayer_sgd.pt'), map_location=torch.devi

# Loading readmission ftlstm model
ftlstm_model_readmission_sgd = torch.load(os.path.join(proj_dir, 'ftl_trans/ftlstm_1_500_SGD_readmission.pt'), map_location=torc
ftlstm_layer_readmission_sgd = torch.load(os.path.join(proj_dir, 'ftl_trans/ftlstm_lstmlayer_1_500_SGD_readmission.pt'), map_loc

# Loading mortality ftlstm model
ftlstm_model_mortality_sgd = torch.load(os.path.join(proj_dir, 'ftl_trans/ftlstm_1_500_SGD_mortality.pt'), map_location=torch.de
ftlstm_layer_mortality_sgd = torch.load(os.path.join(proj_dir, 'ftl_trans/ftlstm_lstmlayer_1_500_SGD_mortality.pt'), map_locatic
```

```
output_df_bert_readmission_sgd, res_bert_readmission_sgd = print_results_bert(
    data_path = os.path.join(proj_dir, 'data/subset_1_500/test_readmission.csv'),
    model = bert_sm_model_readmission_sgd
)

output_df_bert_mortality_sgd, res_bert_mortality_sgd = print_results_bert(
    data_path = os.path.join(proj_dir, 'data/subset_1_500/test_mortality.csv'),
    model = bert_sm_model_mortality_sgd
)

output_df_tlstm_readmission_sgd, res_tlstm_readmission_sgd = print_results_lstm(
    data_path = os.path.join(proj_dir, 'data/subset_1_500/test_readmission.csv'),
    model = tlstm_model_readmission_sgd,
    layer = tlstm_layer_readmission_sgd
)

output_df_tlstm_mortality_sgd, res_tlstm_mortality_sgd = print_results_lstm(
    data_path = os.path.join(proj_dir, 'data/subset_1_500/test_mortality.csv'),
    model = tlstm_model_mortality_sgd,
    layer = tlstm_layer_mortality_sgd
)

output_df_ftl_trans_readmission_sgd, res_ftl_trans_readmission_sgd = print_results_lstm(
    data_path = os.path.join(proj_dir, 'data/subset_1_500/test_readmission.csv'),
    model = ftlstm_model_readmission_sgd,
    layer = ftlstm_layer_readmission_sgd
)

output_df_ftl_trans_mortality_sgd, res_ftl_trans_mortality_sgd  = print_results_lstm(
    data_path = os.path.join(proj_dir, 'data/subset_1_500/test_mortality.csv'),
    model = ftlstm_model_mortality_sgd,
    layer = ftlstm_layer_mortality_sgd
)
```

```
res_bert_readmission_sgd['Model'] = 'BERT-sm'
res_tlstm_readmission_sgd['Model'] = 'TL-Trans'
res_ftl_trans_readmission_sgd['Model'] = 'FTL-Trans'
readmission_results = pd.concat([res_bert_readmission_sgd, res_tlstm_readmission_sgd, res_ftl_trans_readmission_sgd])

readmission_results.columns = readmission_results.columns.str.replace('test_', '')
print("Readmission with SGD")
print(readmission_results[['Model', 'AUC', 'Accuracy', 'AUPR', 'F1']])
```

```
Readmission with SGD
       Model       AUC  Accuracy      AUPR        F1
0    BERT-sm  0.469735  0.660194  0.658327  0.792899
0   TL-Trans  0.387231  0.650485  0.626296  0.788235
0  FTL-Trans  0.500000  0.650485  0.825243  0.788235
```

```
res_bert_mortality_sgd['Model'] = 'BERT-sm'
res_tlstm_mortality_sgd['Model'] = 'TL-Trans'
res_ftl_trans_mortality_sgd['Model'] = 'FTL-Trans'
mortality_results = pd.concat([res_bert_mortality_sgd, res_tlstm_mortality_sgd, res_ftl_trans_mortality_sgd])

mortality_results.columns = mortality_results.columns.str.replace('test_', '')
print("Mortality with SGD")
print(mortality_results[['Model', 'AUC', 'Accuracy', 'AUPR', 'F1']])
```

```
Mortality with SGD
       Model       AUC  Accuracy      AUPR        F1
0    BERT-sm  0.470823  0.723164  0.722666  0.839344
0   TL-Trans  0.444675  0.723164  0.721900  0.839344
0  FTL-Trans  0.500000  0.723164  0.861582  0.839344
```

2. Learning rate tuning

While Adam optimization has built-in adaptation for learning rates based on its attention to first and second moments of the gradients, because the way the model converges fast and the training losses over the 5 epochs behave (very high value in the .9x range for bert-sm model, and increasing along the training process), we suspect the authors' learning rate is too high. So we experimented with different lower rates to observe the performance. After initially experimenting with the simplest model (BERT-sm) on cohort `readmission` we did noticed a significant decrease in training losses, but the performance metrics were not different, so we did not proceed with other models.

```
bert_sm_model_readmission_lr_2eminus6 = torch.load(os.path.join(proj_dir, 'bert/bert_sm_readmission_1_500_lr_2e-6.pt'), map_loca
bert_sm_model_readmission_lr_2eminus7 = torch.load(os.path.join(proj_dir, 'bert/bert_sm_readmission_1_500_lr_2e-7.pt'), map_loca
```

```
output_df_bert_readmission_lr_26, res_bert_readmission_lr_26 = print_results_bert(
    data_path = os.path.join(proj_dir, 'data/subset_1_500/test_readmission.csv'),
    model = bert_sm_model_readmission_lr_2eminus6
)
output_df_bert_readmission_lr_27, res_bert_readmission_lr_27 = print_results_bert(
    data_path = os.path.join(proj_dir, 'data/subset_1_500/test_readmission.csv'),
    model = bert_sm_model_readmission_lr_2eminus7
)
```

```
res_bert_readmission_lr_26['Model'] = 'BERT-sm'
res_bert_readmission_lr_27['Model'] = 'BERT-sm'
res_bert_readmission_lr_26['Learning Rate'] = '2e-6'
res_bert_readmission_lr_27['Learning Rate'] = '2e-7'
readmission_results = pd.concat([res_bert_readmission_lr_26, res_bert_readmission_lr_27])

readmission_results.columns = readmission_results.columns.str.replace('test_', '')
print("Readmission with Different Learning Rates")
print(readmission_results[['Model', 'Learning Rate', 'AUC', 'Accuracy', 'AUPR', 'F1']])
```

```
Readmission with Different Learning Rates
     Model Learning Rate       AUC  Accuracy      AUPR        F1
0  BERT-sm          2e-6  0.423715  0.650485  0.607721  0.788235
0  BERT-sm          2e-7  0.391791  0.650485  0.599030  0.788235
```

Discussion

Hypothetically, the paper should be fully reproducible - the main constraint that we ran into was the lack of compute resources. With our initial set up (using 1/1000 of the training data and 5 epochs), it took approximately 3 hours to train the models on the readmission cohort and 6 hours for the mortality cohort. Even though the models take a while to train, the main issue was running out of the GPU or TPU compute resources on Colab, so some of the models were trained on our local machines.

We spent a lot of time figuring out what percentage of the training data we could use and how many epochs to train the models on before we ran out of compute resources on Colab. This turned out to be one of the more challenging parts of the reproduction, as we had multiple model runs crash due to a lack of compute resources.

However, once we finalized the subset of training data to use, the code the authors provided was very straightforward and easy to model. They provided end-to-end scripts for each model introduced in the paper. There were lots of comments and descriptions of all the arguments used in the training scripts. Based on this, we do not have any suggestions for the authors on how to improve the paper's reproducibility.

For this final submission, we planned to use a larger subset of the training data, such as 1/10, 1/4 or 1/2 in the hopes of producing results more similar to those reported in the paper. Unfortunately, while we were able to increase number of training epochs to 5, we were only able to increase the training dataset from 1/1000th of the dataset to 1/500th of that. This was reflected in a little wider range of variations in our metrics but overall performance remains relatively the same.

We did reach out to the paper's main author, hoping for some clarification and/or guidance, but unfortunately did not hear from him. In an attempt to make this work, we also contacted other teams that were also working on this paper and the one who did responded confirmed they also ran into the same issue.

Revisiting our two initial Hypotheses, we have the following conclusions:

1. Hypothesis 1: If the FTL-Transformer is used to predict patients over time as the temporal learning of health status per person is a crucial part of this model, it will perform better than other transformers that are being used to make predictions on the simple chunks of sequences. ==> Refutation/Non conclusive. As our reproduced FTL-Trans did not produce meaningful metrics, and our baseline models are downstream task trained on small datasets, the scores are more random than reflective of the performance of flexible (or even non-flexible) time layer added.
2. Hypothesis 2: The types of embeddings that will be used in the FTL-Transformer and see which one will provide better accuracy. From the paper, it can be seen that if multi-level position embeddings are used, then the general metrics will improve significantly. ==> Refutation/Non conclusive. Because the reproduced FTL-Trans model did not work we were not able to compare against different types of data to draw a concusion.

Even with many attempts and tweaks in multiple places, the results were not as good as our expectations. We hope this paper will get more attention by professionals and key players in the industry because its potential is significant, and it would be a shame if nobody could adapt this model in a practical product to help with important life-saving tasks in the context of healthcare.

FTL-Trans Model Problems

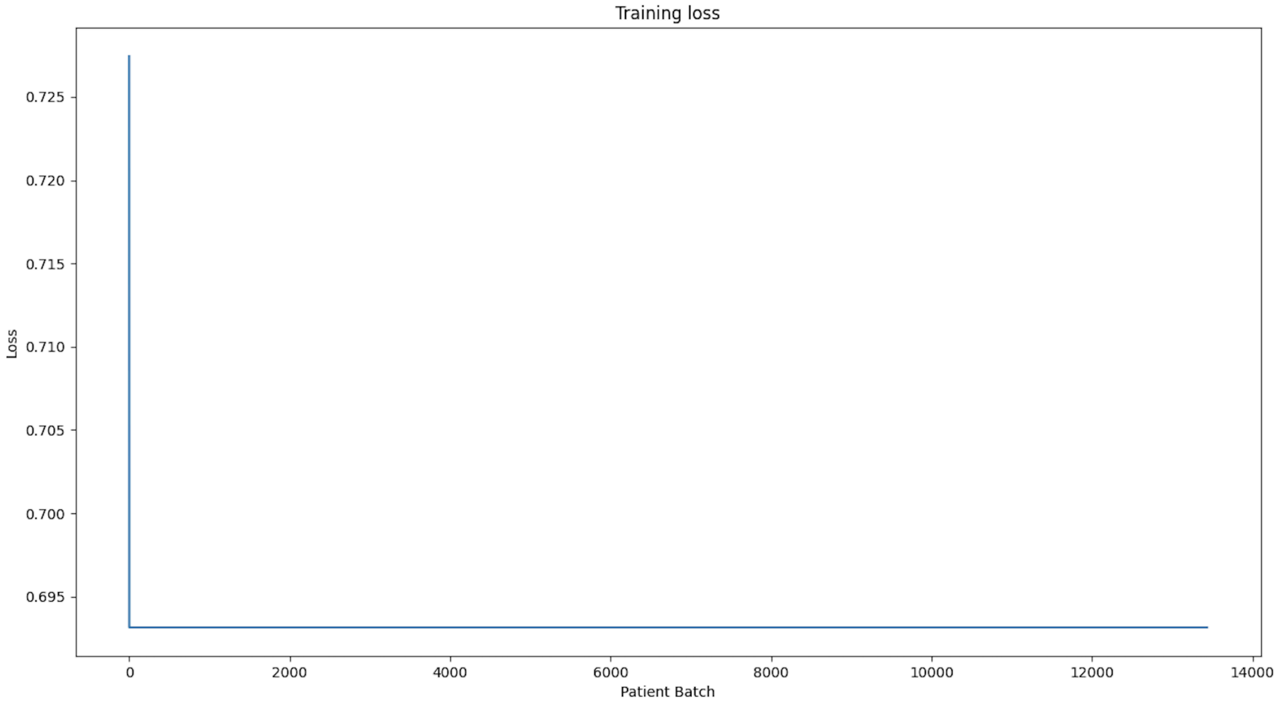
The FTL-Trans model while the model that was presented by the paper's authors and whose code was placed in a Github that actually had many different issues that resulted in the model not changing the prediction probablities and predict anything other than a 1 for the data. This was noticed when training on different versions of the datasets and realizing that the train loss was not changing at all. Once this was noticed, we attempted to diagnose the problem. When we diagnosed the problem, we were training on a dataset that was 1/4 of the original set so we thought that training on the full dataset would allow for the training to work better as the FTL-Trans is a time based model so maybe having more data would allow the model to train better and maybe allow the model to work correctly. So, that prompted us to train the model on a full cohort of the readmission data for a three epochs. After training on the full readmission data, the same thing that happened for the daller dataset happened with this dataset. Below is the value counts from the full readmission csv.

```
show_df = pd.read_csv(proj_dir + "images/test_predictions_full_readmission_ftl.csv", header=0)
print(show_df["pred_prob"].value_counts())
```

```
pred_prob
0.5    560
Name: count, dtype: int64
```

Below is the graph that was created for the train loss from the full readmission data

```
# def
img_dir = os.path.join(proj_dir, 'images/fullReadmissionGraph.png')

import cv2
from google.colab.patches import cv2_imshow
img = cv2.imread(img_dir)
cv2_imshow(img)
```



So it can be seen from the graph that the model did not change the train loss at all and from the CSV value counts it can be seen that the prediction probablities are not changing at all even on the full dataset as all the values are 0.5. Since our first hypothesis was incorrect, we then moved onto maybe seeing if we could email the paper's authors and see if they had any insight. We sent them an email, however, we received no response and so then we attempted to debug the code with the information provided in the paper. Below is the commented out version of the code that was used to train and test the model.

```
# class FTLSTM(nn.Module):
#     def __init__(self, input_size, hidden_size, config, batch_first=True, bidirectional=True):
#         super(FTLSTM, self).__init__()
#         self.input_size = input_size
#         self.hidden_size = hidden_size
#         self.batch_first = batch_first
#         self.bidirectional = bidirectional
#         self.c1 = torch.Tensor([1]).float()
#         self.c2 = torch.Tensor([np.e]).float()
#         self.c3 = torch.Tensor([0.]).float()
#         self.ones = torch.ones([1, self.hidden_size]).float()
#         self.register_buffer('c1_const', self.c1)
#         self.register_buffer('c2_const', self.c2)
#         self.register_buffer('c3_const', self.c3)
#         self.register_buffer("ones_const", self.ones)
#         # Input Gate Parameter
#         self.Wi = Parameter(torch.normal(0.0, config.initializer_range, size=(self.input_size, self.hidden_size)))
#         self.Ui = Parameter(torch.normal(0.0, config.initializer_range, size=(self.hidden_size, self.hidden_size)))
#         self.bi = Parameter(torch.zeros(self.hidden_size))
#         # Forget Gate Parameter
#         self.Wf = Parameter(torch.normal(0.0, config.initializer_range, size=(self.input_size, self.hidden_size)))
#         self.Uf = Parameter(torch.normal(0.0, config.initializer_range, size=(self.hidden_size, self.hidden_size)))
#         self.bf = Parameter(torch.zeros(self.hidden_size))
#         # Output Gate Parameter
#         self.Wog = Parameter(torch.normal(0.0, config.initializer_range, size=(self.input_size, self.hidden_size)))
#         self.Uog = Parameter(torch.normal(0.0, config.initializer_range, size=(self.hidden_size, self.hidden_size)))
#         self.bog = Parameter(torch.zeros(self.hidden_size))
#         # Cell Layer Parameter
#         self.Wc = Parameter(torch.normal(0.0, config.initializer_range, size=(self.input_size, self.hidden_size)))
#         self.Uc = Parameter(torch.normal(0.0, config.initializer_range, size=(self.hidden_size, self.hidden_size)))
#         self.bc = Parameter(torch.zeros(self.hidden_size))
#         # Decomposition Layer Parameter
#         self.W_decomp = Parameter(
#             torch.normal(0.0, config.initializer_range, size=(self.hidden_size, self.hidden_size)))
#         self.b_decomp = Parameter(torch.zeros(self.hidden_size))
#         # Decay Parameter
#         self.W_decay_1 = Parameter(torch.tensor([[0.33]]))
#         self.W_decay_2 = Parameter(torch.tensor([[0.33]]))
#         self.W_decay_3 = Parameter(torch.tensor([[0.33]]))
#         self.a = Parameter(torch.tensor([1.0]))
#         self.b = Parameter(torch.tensor([1.0]))
#         self.m = Parameter(torch.tensor([0.02]))
#         self.k = Parameter(torch.tensor([2.9]))
#         self.d = Parameter(torch.tensor([4.5]))
#         self.n = Parameter(torch.tensor([2.5]))

#     def FTLSTM_unit(self, prev_hidden_memory, inputs, times):
#         prev_hidden_state, prev_cell = prev_hidden_memory
#         x = inputs
#         t = times
#         T = self.map_elapse_time(t)
#         C_ST = torch.tanh(torch.matmul(prev_cell, self.W_decomp) + self.b_decomp)
#         C_ST_dis = torch.mul(T, C_ST)
#         prev_cell = prev_cell - C_ST + C_ST_dis

#         # Input Gate
#         i = torch.sigmoid(torch.matmul(x, self.Wi) +
#                           torch.matmul(prev_hidden_state, self.Ui) + self.bi)
#         # Forget Gate
#         f = torch.sigmoid(torch.matmul(x, self.Wf) +
#                           torch.matmul(prev_hidden_state, self.Uf) + self.bf)
#         # Output Gate
#         o = torch.sigmoid(torch.matmul(x, self.Wog) +
#                           torch.matmul(prev_hidden_state, self.Uog) + self.bog)
#         # Candidate Memory Cell
#         C = torch.sigmoid(torch.matmul(x, self.Wc) +
#                           torch.matmul(prev_hidden_state, self.Uc) + self.bc)
#         # Current Memory Cell
#         Ct = f * prev_cell + i * C

#         # Current Hidden State
#         current_hidden_state = o * torch.tanh(Ct)

#         return current_hidden_state, Ct

      #  def map_elapse_time(self, t):
      #           T_1 = torch.div(self.c1_const, torch.mul(self.a, torch.pow(t, self.b)))
      #           T_2 = self.k - torch.mul(self.m, t)
      #           T_3 = torch.div(self.c1_const, (self.c1_const + torch.pow(torch.div(t, self.d), self.n)))
      #           T = torch.mul(self.W_decay_1, T_1) + torch.mul(self.W_decay_2, T_2) + torch.mul(self.W_decay_3, T_3)
      #           T = torch.max(T, self.c3_const)
      #           T = to rch.min(T, self.c1_const)
      #           T = torch.matmul(T, self.ones_const)
      #           return T


#     def forward(self, inputs, times):
#         device = inputs.device
#         write_log("Training logits1", "drive/MyDrive/log.txt")
#         if self.batch_first:
#             batch_size = inputs.size()[0]
#             inputs = inputs.permute(1, 0, 2)
#             times = times.transpose(0, 1)
#         else:
#             batch_size = inputs.size()[1]
#         prev_hidden = torch.zeros((batch_size, self.hidden_size), device=device)
#         prev_cell = torch.zeros((batch_size, self.hidden_size), device=device)
#         seq_len = inputs.size()[0]
#         hidden_his = []
#         for i in range(seq_len):
#             prev_hidden, prev_cell = self.FTLSTM_unit((prev_hidden, prev_cell), inputs[i], times[i])
#             hidden_his.append(prev_hidden)
#         hidden_his = torch.stack(hidden_his)
#         if self.bidirectional:
#             second_hidden = torch.zeros((batch_size, self.hidden_size), device=device)
#             second_cell = torch.zeros((batch_size, self.hidden_size), device=device)
#             second_inputs = torch.flip(inputs, [0])
#             second_times = torch.flip(times, [0])
#             second_hidden_his = []
#             for i in range(seq_len):
#                 if i == 0:
#                     time = times[i]
#                 else:
#                     time = second_times[i-1]
#                 second_hidden, second_cell = self.FTLSTM_unit((second_hidden, second_cell), second_inputs[i], time)
#                 second_hidden_his.append(second_hidden)
#             second_hidden_his = torch.stack(second_hidden_his)
#             hidden_his = torch.cat((hidden_his, second_hidden_his), dim=2)
#             prev_hidden = torch.cat((prev_hidden, second_hidden), dim=1)
#             prev_cell = torch.cat((prev_cell, second_cell), dim=1)
#         if self.batch_first:
#             hidden_his = hidden_his.permute(1, 0, 2)
#         return hidden_his, (prev_hidden, prev_cell)
# class FTLSTMLayer(SelfDefineBert):

#     def __init__(self, config, num_labels):
#         super(FTLSTMLayer, self).__init__()
#         self.config = config
#         self.ftlstm = FTLSTM(self.config.hidden_size,
#                            self.config.hidden_size // 2,
#                            self.config,
#                            batch_first=True,
#                            bidirectional=True)
#         self.dropout = nn.Dropout(self.config.hidden_dropout_prob)
#         self.embeddings = PatientLevelEmbedding(config)
#         self.classifier = nn.Linear(self.config.hidden_size, num_labels)

#         self.apply(self.init_weights)

#     def forward(self, inputs, times, new_note_ids=None, new_chunk_ids=None, labels=None):
#         new_input = self.embeddings(inputs, new_note_ids, new_chunk_ids)
#         lstm_output, hidden = self.ftlstm(new_input, times.float())
#         loss_fct = BCEWithLogitsLoss()
#         drop_input = lstm_output[0, -1, :]
#         class_input = self.dropout(drop_input)
#         logits = self.classifier(class_input)
#         logits = torch.where(torch.isnan(logits), torch.zeros_like(logits), logits)
#         logits = torch.where(torch.isinf(logits), torch.zeros_like(logits), logits)
#         pred = torch.sigmoid(logits)
#         pred = torch.where(torch.isnan(pred), torch.zeros_like(pred), pred)
#         pred = torch.where(torch.isinf(pred), torch.zeros_like(pred), pred)
#         if labels is not None:
#             loss = loss_fct(logits, labels.float().view(1))
#             write_log("Training labels %d\n" % (labels), "drive/MyDrive/log.txt")
#             return loss, pred
#         else:
#             return pred
# #!/usr/bin/env python3
# # -*- coding: utf-8 -*-
# """
# @author: Dongyu Zhang
# """

# def main():
#     parser = argparse.ArgumentParser(description='Example Argument Parser')
#     ## Required parameters
#     parser.add_argument("--data_dir",
#                         default='./output_split_data',
#                         type=str,
#                         required=True,
#                         help="The input data dir. Should contain the .tsv files (or other data files) for the task.")

#     parser.add_argument("--train_data",
#                         default='train_readmission.csv',
#                         type=str,
#                         required=True,
#                         help="The input training data file name."
#                              " Should be the .tsv file (or other data file) for the task.")

#     parser.add_argument("--val_data",
#                         default='val_readmission.csv',
#                         type=str,
#                         required=True,
#                         help="The input validation data file name."
#                              " Should be the .tsv file (or other data file) for the task.")

#     parser.add_argument("--test_data",
#                         default='test_readmission.csv',
#                         type=str,
#                         required=True,
#                         help="The input test data file name."
#                              " Should be the .tsv file (or other data file) for the task.")

#     parser.add_argument("--log_path",
#                         default='./log.txt',
#                         type=str,
#                         required=True,
#                         help="The log file path.")

#     parser.add_argument("--output_dir",
#                         default='./exp_FTL-Trans',
#                         type=str,
#                         required=True,
#                         help="The output directory where the model checkpoints will be written.")

#     parser.add_argument("--save_model",
#                         default=True,
#                         action='store_true',
#                         help="Whether to save the model.")

#     parser.add_argument("--bert_model",
#                         default="bert-base-uncased",
#                         type=str,
#                         required=True,
#                         help="Bert pre-trained model selected in the list: bert-base-uncased, "
#                              "bert-large-uncased, bert-base-cased, bert-base-multilingual, bert-base-chinese.")

#     parser.add_argument("--embed_mode",
#                         default='all',
#                         type=str,
#                         required=True,
#                         help="The embedding type selected in the list: all, note, chunk, no.")

#     parser.add_argument("--task_name",
#                         default="FTLSTM_with_ClBERT_mortality",
#                         type=str,
#                         required=True,
#                         help="The name of the task.")

#     ## Other parameters
#     parser.add_argument("--max_seq_length",
#                         default=128,
#                         type=int,
#                         help="The maximum total input sequence length after WordPiece tokenization. \n"
```

```
#                              "Sequences longer than this will be truncated, and sequences shorter \n"
#                              "than this will be padded.")
#     parser.add_argument("--max_chunk_num",
#                         default=64,
#                         type=int,
#                         help="The maximum total input chunk numbers after WordPiece tokenization.")
#     parser.add_argument("--train_batch_size",
#                         default=1,
#                         type=int,
#                         help="Total batch size for training.")
#     parser.add_argument("--eval_batch_size",
#                         default=1,
#                         type=int,
#                         help="Total batch size for eval.")
#     parser.add_argument("--learning_rate",
#                         default=2e-5,
#                         type=float,
#                         help="The initial learning rate for Adam.")
#     parser.add_argument("--warmup_proportion",
#                         default=0.0,
#                         type=float,
#                         help="Proportion of training to perform linear learning rate warmup for. "
#                              "E.g., 0.1 = 10%% of training.")
#     parser.add_argument("--num_train_epochs",
#                         default=5,
#                         type=int,
#                         help="Total number of training epochs to perform.")
#     parser.add_argument('--seed',
#                         type=int,
#                         default=42,
#                         help="random seed for initialization")
#     parser.add_argument('--gradient_accumulation_steps',
#                         type=int,
#                         default=1,
#                         help="Number of updates steps to accumualte before performing a backward/update pass.")

#     args_values = ['--data_dir', 'drive/MyDrive/Final_Project_DLH/output_split_data/subset_1_500', '--train_data', 'train_mor
#                    '--log_path', 'drive/MyDrive/log.txt',  '--output_dir',  'drive/MyDrive/exp_FTL-Trans-1-500-mortality', '-

#     args = parser.parse_args(args_values)
#     print("Arguments:")
#     print(args)
#     if os.path.exists(args.output_dir) and os.listdir(args.output_dir) and args.save_model:
#         raise ValueError("Output directory ({}) already exists and is not empty.".format(args.output_dir))
#     os.makedirs(args.output_dir, exist_ok=True)

#     LOG_PATH = args.log_path
#     MAX_LEN = args.max_seq_length

#     config = DotMap()
#     config.hidden_dropout_prob = 0.1
#     config.layer_norm_eps = 1e-12
#     config.initializer_range = 0.02
#     config.max_note_position_embedding = 1000
#     config.max_chunk_position_embedding = 1000
#     config.embed_mode = args.embed_mode
#     config.layer_norm_eps = 1e-12
#     config.hidden_size = 768
#     config.lstm_layers = 1

#     config.task_name = args.task_name

#     write_log(("New Job Start! \n"
#                "Data directory: {}, Directory Code: {}, Save Model: {}\n"
#                "Output_dir: {}, Task Name: {}, embed_mode: {}\n"
#                "max_seq_length: {},  max_chunk_num: {}\n"
#                "train_batch_size: {}, eval_batch_size: {}\n"
#                "learning_rate: {}, warmup_proportion: {}\n"
#                "num_train_epochs: {}, seed: {}, gradient_accumulation_steps: {}\n"
#                "FTLSTM Model's lstm_layers: {}").format(args.data_dir,
#                                                         args.data_dir.split('_')[-1],
#                                                         args.save_model,
#                                                         args.output_dir,
#                                                         config.task_name,
#                                                         config.embed_mode,
#                                                         args.max_seq_length,
#                                                         args.max_chunk_num,
#                                                         args.train_batch_size,
#                                                         args.eval_batch_size,
#                                                         args.learning_rate,
#                                                         args.warmup_proportion,
#                                                         args.num_train_epochs,
#                                                         args.seed,
#                                                         args.gradient_accumulation_steps,
#                                                         config.lstm_layers),
#               LOG_PATH)

#     content = "config setting: \n"
#     for k, v in config.items():
#         content += "{}: {} \n".format(k, v)
#     write_log(content, LOG_PATH)

#     device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
#     n_gpu = torch.cuda.device_count()
#     write_log("Number of GPU is {}".format(n_gpu), LOG_PATH)
#     for i in range(n_gpu):
#         write_log(("Device Name: {},"
#                    "Device Capability: {}").format(torch.cuda.get_device_name(i),
#                                                    torch.cuda.get_device_capability(i)), LOG_PATH)

#     train_file_path = os.path.join(args.data_dir, args.train_data)
#     val_file_path = os.path.join(args.data_dir, args.val_data)
#     test_file_path = os.path.join(args.data_dir, args.test_data)
#     train_df = pd.read_csv(train_file_path)
#     val_df = pd.read_csv(val_file_path)
#     test_df = pd.read_csv(test_file_path)

#     random.seed(args.seed)
#     np.random.seed(args.seed)
#     torch.manual_seed(args.seed)
#     if n_gpu > 0:
#         torch.cuda.manual_seed_all(args.seed)

#     tokenizer = BertTokenizer.from_pretrained(args.bert_model, do_lower_case=True)

#     write_log("Tokenize Start!", LOG_PATH)
#     train_df = reorder_by_time(train_df)
#     val_df = reorder_by_time(val_df)
#     test_df = reorder_by_time(test_df)
#     train_labels, train_inputs, train_masks, train_note_ids, train_times = Tokenize_with_note_id_hour(train_df, MAX_LEN,
#                                                                                                       tokenizer)
#     validation_labels, validation_inputs, validation_masks, validation_note_ids, validation_times = Tokenize_with_note_id_hou
#         val_df, MAX_LEN, tokenizer)
#     test_labels, test_inputs, test_masks, test_note_ids, test_times = Tokenize_with_note_id_hour(test_df, MAX_LEN,
#                                                                                                  tokenizer)
#     write_log("Tokenize Finished!", LOG_PATH)
#     train_inputs = torch.tensor(train_inputs)
#     validation_inputs = torch.tensor(validation_inputs)
#     test_inputs = torch.tensor(test_inputs)
#     train_labels = torch.tensor(train_labels)
#     validation_labels = torch.tensor(validation_labels)
#     test_labels = torch.tensor(test_labels)
#     train_masks = torch.tensor(train_masks)
#     validation_masks = torch.tensor(validation_masks)
#     test_masks = torch.tensor(test_masks)
```