

# Introduction

Our paper is "*Unifying Heterogeneous Electronic Health Records Systems via Text-Based Code Embedding*" by [Hur et al. \(2022\)](#). All paper references are provided at the bottom of this notebook.

While EHR (Electronic Health Records) provide an attractive data source to do research on exposures and disease, there are many heterogeneous medical code formats used by different healthcare providers. Thus, clinical studies using EHRs are difficult to scale up due to data incompatibilities. The technical hurdle is that previous systems learned hidden representations (latent embeddings) for each code system which projected the same medical concept with different encodings into different, incompatible semantic spaces. The paper by [Hur et al \(2022\)](#) learns from the unstructured textual descriptions instead of the medcodes. To this end the authors propose a novel method to unify heterogeneous EHR systems via a novel method to learn text embeddings of medical codes from their textual descriptions, which they call description embedding ([DescEmb](#)). themselves, a strategy which the authors termed description embedding (DescEmb). Using DescEmb the authors were able to pool differently structured EHRs from two [datasets](#), namely MIMIC-III and eICU, into one larger pooled dataset and achieved higher accuracy. The authors then tested the performance of the DescEmb on [five tasks](#): mortality prediction, length of stay prediction, readmission prediction and diagnosis. Hur et al. were to our knowledge the first to attempt a unification of different EHR systems via text-based code embeddings. The authors achieved state-of-the-art performance on all five tasks, which is a significant contribution to the field of EHR research.

## Scope of Reproducibility:

We plan to focus on hypotheses 1 - 3 mainly. Hypothesis 4 is addressed by comparing the results of the proposed method with the state-of-the-art methods (see Results section). Hypothesis 5 is our own hypothesis which we will test by using a BERT model pre-trained on a domain-specific dataset. We view this as an extension of the original paper which is optional, thus it will not be included in the draft.

1. Hypothesis 1: *DescEmb* outperforms *CodeEmb*
2. Hypothesis 2: Two EHR datasets with different structure can be used interchangeably with the proposed method.
3. Hypothesis 3: Differently structured EHR datasets can be pooled together to improve the performance of the model.
4. Hypothesis 4: *DescEmb* has a competitive or superior performance compared to the state-of-the-art methods.
5. Hypothesis 5 (ours): Using a BERT model which was pre-trained in a strictly domain-specific manner can improve the performance of the model further (not shown by the authors).

# GitHub repository:

If you entered through a PDF, you can find our GitHub repository [here](#).

## Video

We uploaded our [video walkthrough](#) to YouTube.

## Methodology

## Environment Setup

We provide a [script](#) to create a virtual environment and install necessary packages.

```
./setup.sh
```

In [1]:

```
# Python version
!python --version
```

```
Python 3.12.2
```

## Imports

In [3]:

```
%load_ext autoreload
%autoreload 2
import warnings
warnings.filterwarnings('ignore')
import os
import torch
from pathlib import Path
import numpy as np
import pandas as pd
import umap
import matplotlib.pyplot as plt
from project_code.evaluate.logs import plot_metrics, get_best_epoch, get_metr:
```

```
The autoreload extension is already loaded. To reload it, use:
%reload_ext autoreload
```

## Prediction tasks studied in the paper

The authors assess model performance using five different prediction tasks listed in the following table.

Task Number	Task identifier	Task Description	Classification type
Task 1	Dx	Predicting the diagnosis of a patient given the clinical notes.	Multi-label

Task Number	Task identifier	Task Description	Classification type
Task 2	Mort	Predicting the mortality of a patient given the clinical notes and the treatment codes.	Binary
Task 3	LOS>3	Predicting whether the length of stay of a patient is greater than 3 days given the clinical notes and the treatment codes.	Binary
Task 4	LOS>7	Predicting whether the length of stay of a patient is greater than 7 days given the clinical notes and the treatment codes.	Binary
Task 5	ReAdm	Predicting whether a patient will be readmitted within 30 days given the clinical notes and the treatment codes.	Binary

For task 1 the highest level representation of ICD-9-CM is used, in total 18 representations. MIMIC-III already uses ICD-9, while eICU uses ICD-10. The authors map ICD-10 to ICD-9 using the Clinical Classifications Software (CCS).

## Abbreviations used in the paper

Abbreviation	Description
$p^i$	The $i$ -th patient
$c_i$	The $i$ -th medical event (e.g. diagnoses, prescriptions)
$\mathcal{C}$	The set of all medical events
$t_i$	The $i$ -th time stamp
$w_{i,j}$	The $j$ -th word in the $i$ -th medical event
$\mathcal{W}$	The set of all words, i.e. vocabulary of description
$d^i$	The $i$ -th text description of diagnosis of the $i$ -th patient
$E_\Psi$	The embedding layer of the medical events used in <i>CodeEmb</i>
$\mathbf{c}_i$	Embedded representation of the $i$ -th medical event
$B_\Phi$	The embedding layer of the text descriptions used in <i>DescEmb</i>
$\mathbf{z}_i$	Embedded representation of the $i$ -th text description

## Data

We downloaded two datasets, MIMIC-III v1.4 from [PhysioNet](#) and eICU v2.0 from [MIT Lab](#) and downloaded it via [PhysioNet](#) using `wget`. The datasets are stored in the following paths in our Google Drive:

```
In [4]: storage_dir = Path("/data/DescEmb/output")
mimic_path = storage_dir/"mimic"
eicu_path = storage_dir/"eicu"
pooled_path = storage_dir/"pooled"
```

## Data description

eICU consists of 200,859 patients, 1,139,695 admissions, and 2,839,547 patient unit stays. It is organized in three levels:

A comprehensive description of the dataset can be found in the paper by [Pollard et al.](#)

MIMIC-III consists of 46,520 patients, 58,976 admissions, and 7,875,529 clinical notes. An overview of the dataset:

A comprehensive description of the dataset can be found in the paper by [Johnson et al.](#)

## Preprocessing

We preprocessed the data using the code provided by the authors in their [GitHub repository](#). We ran

```
INPUT_PATH=/home/data
OUTPUT_PATH=/home/data/output
DX_PATH=$INPUT_PATH/ccs_multi_dx_tool_2015.csv

python ../preprocess/preprocess_main.py \
  --src_data mimiciii \
  --dataset_path $INPUT_PATH/mimic \
  --ccs_dx_tool_path $DX_PATH \
  --dest_path $OUTPUT_PATH ;
```

In total this was run six times: For three datasets, i.e. `mimiciii`, `eicu` or `pooled` and `$data_type` was either `predict` or `pretrain` - see our modified `preprocess_run.sh` for all six runs. This script needs to be run from within the author's DescEmb repository, because of all the dependencies which `main.py` imports. Preprocessing was done on a cloud node with 128 GB of RAM, 32 cores, RTX A6000 GPU and 180 GB of storage.

Both, data cleaning and cohort construction are done by running the same script. Below we describe what is happening in those steps.

The datasets are constructed such that three sources of information are available:

- Laboratory results
- Medication
- Infusion

Each of these sources is available in two different datasets: MIMIC-III and eICU. The authors use the following files from the MIMIC-III and eICU datasets to construct the datasets for the tasks:

Item	Source	Filename
------	--------	----------

Item	Source	Filname
Laboratory results	MIMIC-III	labevents.csv
Medication	MIMIC-III	prescriptions.csv
Infusion	MIMIC-III	inpuvents_cv.csv, inpuvents_mv.csv -> Merged together
Laboratory results	eICU	lab.csv
Medication	eICU	medication.csv
Infusion	eICU	infusionDrug.csv

## Data Cleaning

On merging `inpuvents_cv.csv` and `inpuvents_mv.csv`, the authors found that the two datasets have different structures. They used the following steps to clean the data:

- 41 patients were removed which conflicted regarding code systems.
- For patients with multiple ICU stays, the authors used the first ICU stay for the analysis.
- For patients with multiple ICU stays, those with fewer than 5 observed codes were removed.
- Restrict sample to first 150 codes during first 12 hours of ICU stay.

## Cohort Construction

The authors used the following steps to construct the cohorts:

1. Cohort: MICU patients whose first care unit was also the last care unit and of type ICU.
2. Cohort: Patients with multiple ICU stays above age 18
3. Cohort: Patients with multiple ICU stays who remained in ICU for > 12 hours

## Datasets and Dataloaders

In the proposal we intended to leverage `pyhealth` to load the data. However, we found that the authors' code for preprocessing may not be compatible with `pyhealth`. We will use the authors' code to load the data.

Four dataset classes available in `datasets/dataset.py`, all of which inherit from `BaseDataset`, which in turn inherits from `torch.utils.data.Dataset`. The four classes are:

The base class `BaseDataset` provides the following functionality:

- Tokenization of the descriptions with `emilyalsentzer/Bio_ClinicalBERT` via `huggingface transformers`
- Splitting in to folds `train`, `valid` and `test` and return their respective indeces
- A method `mask_tokens` for preparing masked tokens inputs/labels for masked language modeling (MLM). This method samples tokens in each sequence for MLM training with probabilities for masking, replacing with random tokens, or keeping unchanged.

Class	Description	Trainer class	embed_model
CodeDataset	Dataset for the code embeddings	Trainer	codeemb
TokenizedDataset	Dataset for the tokenized descriptions	Trainer	descemb
MLMTokenizedDataset	Dataset for the masked language model tokenized descriptions	Trainer	mlm
Word2VecDataset	Dataset for the word2vec tokenized descriptions	Word2VecTrainer	w2v

`CodeDataset` returns stacked tensors comprising input ID's, sequence length, value and label.

`TokenizedDataset` stacks input IDs, token type IDs, attention masks, sequential lengths, values, and labels into tensors and organizes them into a dictionary, ready for model input.

`MLMTokenizedDataset` returns input IDs, token type IDs, attention masks, MLM labels for masked language model training.

`Word2VecDataset` returns an index dictionary as well as positive and negative word pairs to train the word2vec model using a skip-gram approach.

The datasets are utilized depending on which embedding model is being trained. This decision is made by the user as a command-line argument to `main.py`. There are two trainer classes, one for word2vec and one for all other cases. The table above shows which dataset is used for which embedding model.

## Exploratory Data Analysis

1. Load the data
2. Check the head of the data
3. Check the missing values in the data
4. Check the value counts in the data
5. Check demographics of the patients

Load the data

```
In [5]: mimic_cohort = pd.read_pickle(storage_dir/"mimiciii_cohort.pkl")
        eicu_cohort = pd.read_pickle(storage_dir/"eicu_cohort.pkl")
```

Check the head of the data

```
In [6]: mimic_cohort.head(3)
```

```
Out[6]:
```

	SUBJECT_ID	HADM_ID	ICUSTAY_ID	DBSOURCE	FIRST_CAREUNIT	LAST_CAREUNIT	FIRST_W
0	58526	100001	275225	metavision	MICU	MICU	

SUBJECT_ID	HADM_ID	ICUSTAY_ID	DBSOURCE	FIRST_CAREUNIT	LAST_CAREUNIT	FIRST_W
------------	---------	------------	----------	----------------	---------------	---------

1	54610	100003	209281	metavision	MICU	MICU
---	-------	--------	--------	------------	------	------

2	9895	100006	291788	carevue	MICU	MICU
---	------	--------	--------	---------	------	------

3 rows × 26 columns

In [7]: `eicu_cohort.head(3)`

Out[7]:

	patientunitstayid	patienthealthsystemstayid	gender	age	ethnicity	hospitalid	wardid	apachea
--	-------------------	---------------------------	--------	-----	-----------	------------	--------	---------

0	141392	129109	Female	78	Caucasian	73	97	Seps (includ
---	--------	--------	--------	----	-----------	----	----	--------------

1	141462	129166	Male	80	Caucasian	73	97	Sepsis
---	--------	--------	------	----	-----------	----	----	--------

2	141584	129260	Male	63	Caucasian	73	97	t
---	--------	--------	------	----	-----------	----	----	---

3 rows × 36 columns

Check the missing values in the data

In [8]:

```
# Check the missing values
print(f'MIMIC-III missing values: {mimic_cohort.isnull().sum()}\n')
print(f'eICU missing values: {eicu_cohort.isnull().sum()}\n')
```

```
MIMIC-III missing values: SUBJECT_ID      0
HADM_ID      0
ICUSTAY_ID    0
DBSOURCE      0
FIRST_CAREUNIT  0
LAST_CAREUNIT  0
FIRST_WARDID   0
```

LAST_WARDID	0
INTIME	0
OUTTIME	0
LOS	0
GENDER	0
DOB	0
DOD	8778
DOD_HOSP	11884
DOD_SSN	10321
EXPIRE_FLAG	0
age	0
readmission	0
mortality	0
los_3day	0
los_7day	0
ICD9_CODE	0
12h_obs	0
24h_obs	0
diagnosis	0

dtype: int64)

eICU missing values: patientunitstayid	0
patienthealthsystemstayid	0
gender	0
age	0
ethnicity	24
hospitalid	0
wardid	0
apacheadmissiondx	8
admissionheight	97
hospitaladmittime24	0
hospitaladmitoffset	0
hospitaladmitsource	4083
hospitaldischargeyear	0
hospitaldischargetime24	0
hospitaldischargeoffset	0
hospitaldischargelocation	118
hospitaldischargestatus	104
unittype	0
unitadmittime24	0
unitadmitsource	13
unitvisitnumber	0
unitstaytype	0
admissionweight	240
dischargeweight	4896
unitdischargetime24	0
unitdischargeoffset	0
unitdischargelocation	17
unitdischargestatus	2
uniquepid	0
readmission	0
mortality	0
losday	0
los_3day	0
los_7day	0
diagnosisstring	0
diagnosis	0

dtype: int64)



Check value counts in the data

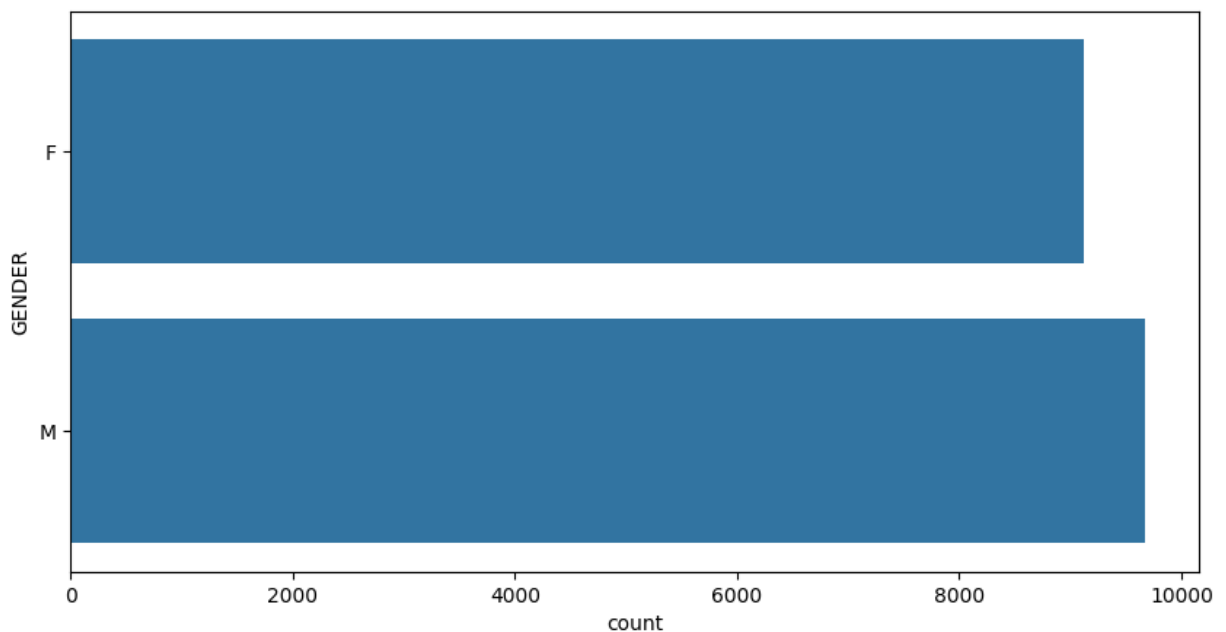
Check demographics of the patients

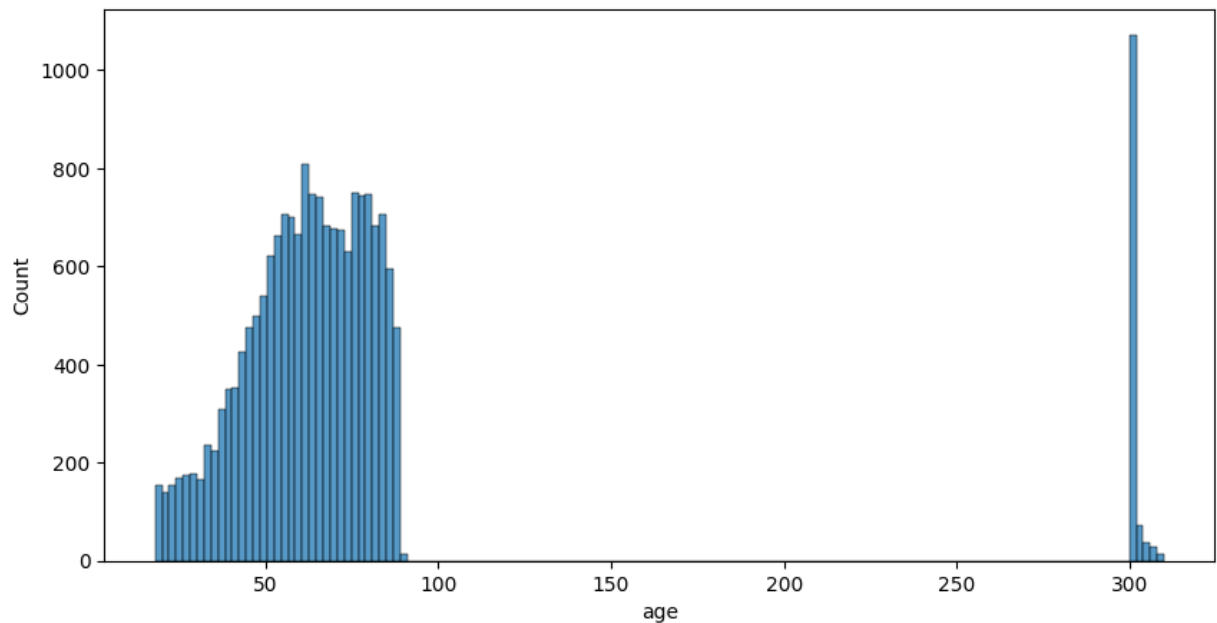
```
In [9]: mimic_cohort.columns
```

```
Out[9]: Index(['SUBJECT_ID', 'HADM_ID', 'ICUSTAY_ID', 'DBSOURCE', 'FIRST_CAREUNIT',  
              'LAST_CAREUNIT', 'FIRST_WARDID', 'LAST_WARDID', 'INTIME', 'OUTTIME',  
              'LOS', 'GENDER', 'DOB', 'DOD', 'DOD_HOSP', 'DOD_SSN', 'EXPIRE_FLAG',  
              'age', 'readmission', 'mortality', 'los_3day', 'los_7day', 'ICD9_CODE',  
              '12h_obs', '24h_obs', 'diagnosis'],  
            dtype='object')
```

```
In [10]: # demographics in MIMIC-III: GENDER, age  
import seaborn as sns  
import matplotlib.pyplot as plt  
  
plt.figure(figsize=(10, 5))  
sns.countplot(mimic_cohort['GENDER'])  
plt.figure(figsize=(10, 5))  
sns.histplot(mimic_cohort['age'])
```

```
Out[10]: <Axes: xlabel='age', ylabel='Count'>
```





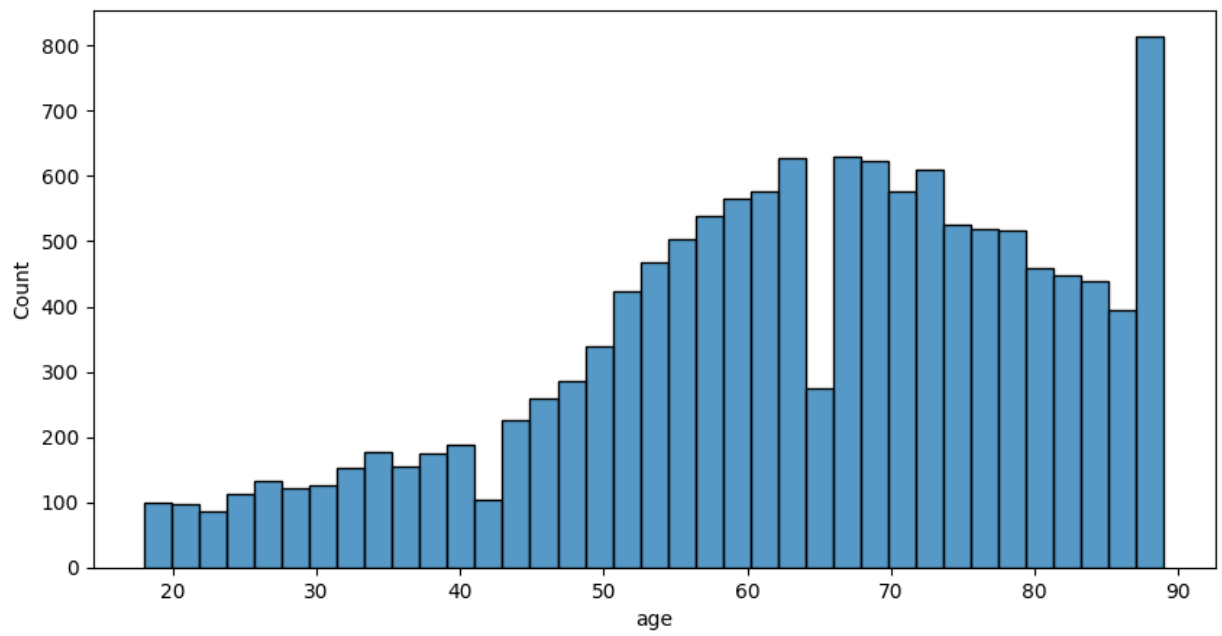
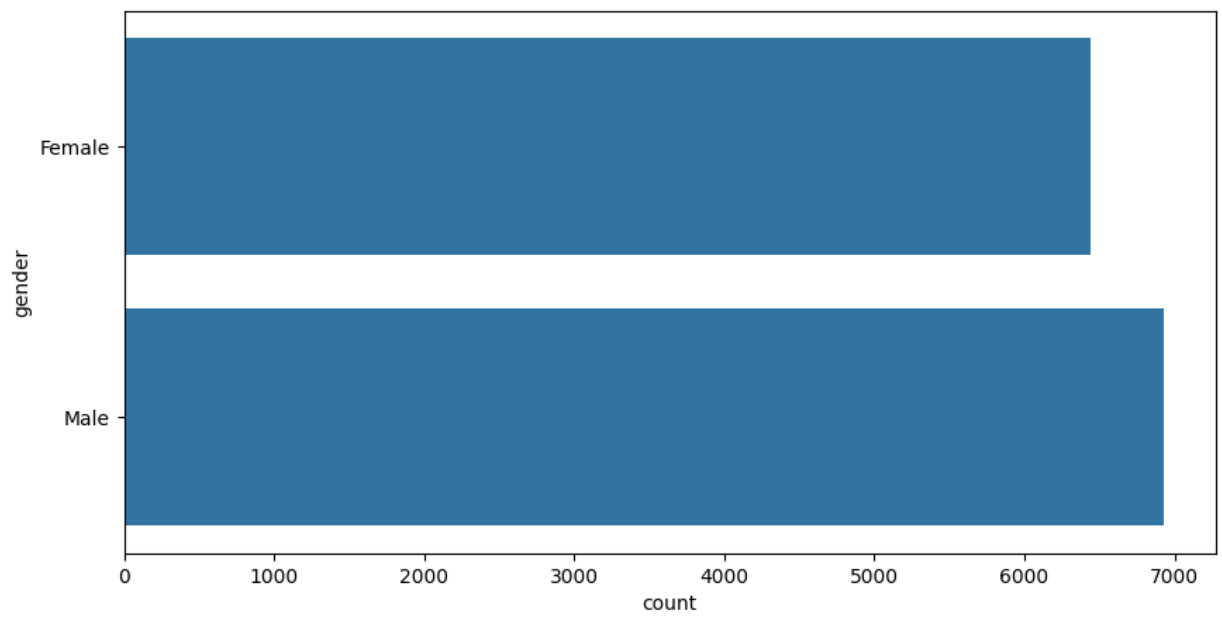
```
In [11]: eicu_cohort.columns
```

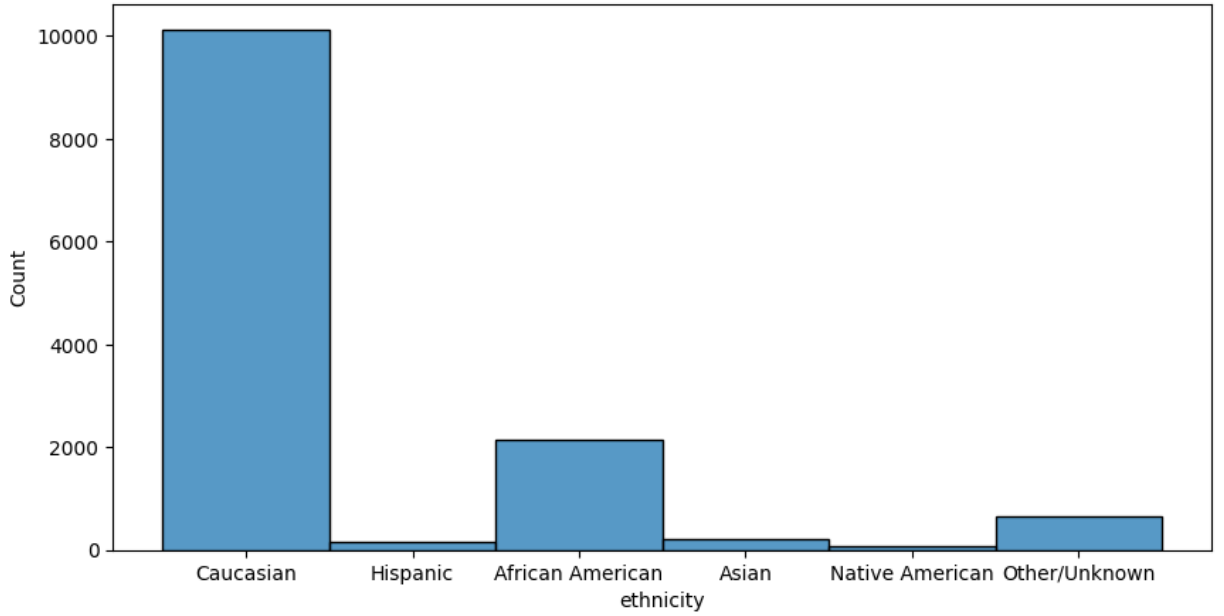
```
Out[11]: Index(['patientunitstayid', 'patienthealthsystemstayid', 'gender', 'age',
               'ethnicity', 'hospitalid', 'wardid', 'apacheadmissiondx',
               'admissionheight', 'hospitaladmittime24', 'hospitaladmitoffset',
               'hospitaladmitsource', 'hospitaldischargeyear',
               'hospitaldischargetime24', 'hospitaldischargeoffset',
               'hospitaldischargelocation', 'hospitaldischargestatus', 'unitttype',
               'unitadmittime24', 'unitadmitsource', 'unitvisitnumber', 'unitstaytyp
               e',
               'admissionweight', 'dischargeweight', 'unitdischargetime24',
               'unitdischargeoffset', 'unitdischargelocation', 'unitdischargestatus',
               'uniquepid', 'readmission', 'mortality', 'losday', 'los_3day',
               'los_7day', 'diagnosisstring', 'diagnosis'],
              dtype='object')
```

```
In [12]: # demographics in eicu: gender, age
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 5))
sns.countplot(eicu_cohort['gender'])
plt.figure(figsize=(10, 5))
sns.histplot(eicu_cohort['age'])
plt.figure(figsize=(10, 5))
sns.histplot(eicu_cohort['ethnicity'])
```

```
Out[12]: <Axes: xlabel='ethnicity', ylabel='Count'>
```





## Model

The model includes the model definition which usually is a class, model training, and other necessary parts.

- Model architecture: layer number/size/type, activation function, etc
- Training objectives: loss function, optimizer, weight of each loss term, etc
- Others: whether the model is pretrained, Monte Carlo simulation for uncertainty analysis, etc
- The code of model should have classes of the model, functions of model training, model validation, etc.
- If your model training is done outside of this notebook, please upload the trained model here and develop a function to load and test it.

## *CodeEmb* vs. *DescEmb* strategies

The authors propose two strategies to embed the data: *CodeEmb* and *DescEmb*. The two strategies are described as follows:

*Code-based Embedding:*

$$\mathbf{c}_i = E_{\Psi}(\mathbf{c}_i)$$

$$\hat{\mathbf{y}} = P_{\Phi}(\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_T)$$

*Description-based Embedding:*

$$\mathbf{d}_i = (w_{i,1}, w_{i,2}, \dots, w_{i,n})$$

$$\mathbf{z}_i = B_{\Psi}(\mathbf{d}_i)$$

$$\hat{\mathbf{y}} = P_{\Phi}(\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_T)$$

Effectively, the *CodeEmb* strategy uses the medical events directly, while the *DescEmb* strategy

uses the text descriptions of the medical events. The authors use a Bi-LSTM for the embedding layer  $E_\Psi$  in *CodeEmb*. For  $B_\Psi$  they ended up using a BERT (Bi-directional Encoder Representations from Transformers). While the authors tested several model depths, we will only use the smallest 2-MSA-layer model **BERT-tiny**. For the most part  $z_i$  corresponds to the BERT output vector for the  $[CLS]$  token. The prediction layer  $P_\Phi$  employs an RNN.

## Value Embedding

The descriptions of medical events often contain numbers (values), which are important for the prediction tasks. The authors propose four different methods to embed these values into the model. The methods is described as follows:

- Value Aggregation (VA): All numbers are used as is and embedded into a single vector. This has two disadvantages
  - Use a lot of rare (in the Zipfian sense) tokens for all the numbers
  - Prone to sub-word tokenization (e.g. 1351 gets split into 13 and 15 )
- Digit-split value aggregation (DSVA): The numbers are split into digits and embedded separately. This has the advantage of not needing rare tokens, but the disadvantage of not capturing the value as a whole.
- DSVA + Digit Place Embedding (DSVA-DPE): The numbers are split into digits and embedded separately. Additionally, the place of the digit is embedded. This has the advantage of capturing the value as a whole, but the disadvantage of needing more tokens.
- Value Concatenated (VC): Numbers and their corresponding physical units are embedded and then concatenated. This has the advantage of capturing the value as a whole and the physical unit, but the disadvantage of needing more tokens. The value embeddings in this case is done using a single layer feed-forward neural network (MLP or `nn.Linear` in PyTorch).

The following figure summarizes the value embedding techniques.

## Model Training

There are three optimization schemes used in the paper:

$$CodeEmb : \underset{\Theta, \Psi}{\operatorname{argmin}} \mathcal{L}(\hat{y}, y) \quad (1)$$

$$DescEmb, \text{ all parameters} : \underset{\Theta, \Phi}{\operatorname{argmin}} \mathcal{L}(\hat{y}, y) \quad (2)$$

$$DescEmb, \text{ class fine-tuning} : \underset{\Psi, z_{CLS}}{\operatorname{argmin}} \mathcal{L}(\hat{y}, y) \quad (3)$$

Eqn. 1 is the optimization scheme for *CodeEmb*, where  $\Theta$  are the parameters of the prediction layer and  $E_\Psi$ , and  $\Psi$  are the parameters of the embedding layer. Eqn. 2 is the optimization scheme for *DescEmb* where all parameters are fine-tuned. Eqn. 3 is the optimization scheme for

*DescEmb* where only the class token  $z_{CLS}$  is fine-tuned, but the parameters of the text embedding layer  $E_{\Psi}$  are fixed.

We used the author's example scripts for running model training and provide our scripts in the `project_code` folder.

Script	Description	Essential Arguments
<a href="#">00_pretrain_codeemb.sh</a>	Pretrain the <i>CodeEmb</i> model (i.e. Embedding layer followed by RNN)	<code>--model ehr_model --embed_model codeemb --value_mode NV --task w2v</code>
<a href="#">01_pretrain_descemb_rnn.sh</a>	Pretrain the <i>DescEmb</i> encoder with RNN architecture and MLM target	<code>--model descemb_rnn --value_mode NV --task mlm</code>
<a href="#">02_pretrain_descemb_bert.sh</a>	Pretrain the <i>DescEmb</i> encoder with BERT architecture and MLM target	<code>--model descemb_bert --value_mode NV --task mlm</code>
<a href="#">00_single_domain_learning_descemb.sh</a>	Train the <i>DescEmb</i> model with fine-tuning on prediction task	Iterate through -- embed_models=('descemb_rnn' 'descemb_bert'), -- tasks=('readmission' 'mortality' 'los_3day' 'los_7day' 'diagnosis') and --value_modes=('NV' 'VA' 'DSVA' 'DSVA_DPE' 'VC')
<a href="#">01_single_domain_learning_codeemb.sh</a>	Train the <i>CodeEmb</i> model with fine-tuning on prediction task	Iterate through -- embed_models=('codeemb'), -- tasks=('readmission' 'mortality' 'los_3day' 'los_7day' 'diagnosis') and --value_modes=('NV' 'VA' 'DSVA' 'VC')

As we progressed with the project the number of scripts we produced grew, but they all followed the same overall logic. You can find a comprehensive list in the [project\\_code](#) folder.

For illustration we reproduce one of them here:

```
#!/bin/bash
# Run training new model using FT with descemb
# run this from project root folder using
# ./project_code/02_train_new_models/03_train_ft_descemb_bert.sh $
(pwd)/project_code

if [ -n "$1" ]; then
    root="${1%}/DescEmb/"
    MODEL_PATH=("${1%}/DescEmb/outputs/2024-04-19/06-18-04/
checkpoints/checkpoint_mimiciii_descemb_bert_mlm_last.pt")
```

```

else
    root='../DescEmb/'
    MODEL_PATH=('../../../../../outputs/2024-04-19/06-18-04/checkpoints/
checkpoint_mimiciii_descemb_bert_mlm_last.pt')
fi

INPUT_PATH=/home/data/output
SRC_DATA=('mimiciii' 'eicu')

embed_models=('descemb_bert')
tasks=('diagnosis' 'los_3day' 'los_7day' 'readmission' 'mortality')
value_modes=('VA' 'DSVA' 'DSVA_DPE' 'VC')

for data in "${SRC_DATA[@]"; do
    for index in "${!embed_models[@]"; do
        emb_model=${embed_models[$index]}
        model_path=${MODEL_PATH[$index]}
        echo "Processing model: $emb_model with data $data"
        for task in "${tasks[@]"; do
            for value_mode in "${value_modes[@]"; do
                echo "Current directory: $(pwd)" && \
                CUDA_VISIBLE_DEVICES=0 python ${root}main.py \
                    --distributed_world_size 1 \
                    --input_path "$INPUT_PATH" \
                    --init_bert_params \
                    --model ehr_model \
                    --embed_model "$emb_model" \
                    --pred_model rnn \
                    --src_data $data \
                    --ratio 100 \
                    --patience 90 \
                    --value_mode "$value_mode" \
                    --save_prefix "checkpoint_${data}_${emb_model}_${
{task}" \
                    --task "$task" ;
            done
        done
    done
done

```

Because we modified the author's code in several places to remove issues we had with preprocessing, GPU utilization and others we kept a copy in a subfolder called [DescEmb](#). The training loop is part of the trainer classes in the `trainer.py` file. The training loop is a standard PyTorch training loop. We did modify these scripts to run on a single GPU, as we only have access to a single GPU. The scripts provided by the authors are for multi-GPU training.

```


## Before the training loop get the available GPU devices (excerpt)
self.n_devices = torch.cuda.device_count() if
torch.cuda.is_available() else 0
self.device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')

```

```

if self.n_devices > 1:
    self.model = nn.DataParallel(model,
device_ids=args.device_ids).to(self.device)
else:
    self.model = model.to(self.device)
## continue trainer init ...
## Inside the training loop (exerpt)
for sample in tqdm.tqdm(self.data_loaders['train']):
    self.optimizer.zero_grad(set_to_none=True)
    net_output = self.model(**sample["net_input"])
    if self.n_devices > 1:
        logits = self.model.module.get_logits(net_output)
        target =
self.model.module.get_targets(sample).to(logits.device)
    else:
        logits = self.model.get_logits(net_output)
        target = self.model.get_targets(sample).to(logits.device)
## continue loop ...

```

The number of hyperparameters needed to uniquely determine each experiment is at least 7, plus some additional parameters regarding data and model paths. For this reason we produced the following table to generate an overview of the hyperparameters used in the paper. We also use this to keep track of which experiments need to be run in order to reproduce the results from the paper. 

The pink cells are the key (combinations) which determine what kind of experiment it is. E.g. for CLS-FT you need to use an `ehr_model` and `init_bert_params_with_freeze`.

## Results and Analysis

### Preprocessing

We had many issues during the preprocessing step, because not all options are documented. So, at first we learned them from the command-line options in the source code. Then the authors released an updated version which fixed some of the issues we had (which we are very appreciative of). There was still a lot of trial and error involved. Still on the the last day before the draft deadline the authors were still making changes.

Some issues were related to the folder structure the scripts are creating to deviate from what some parts of the processing are expecting. We filed an [issue](#) on that. As a workaround we used some manual copying and soft linking of files to the best of our understanding. However, it is likely that we made some mistakes in this process. Still for the draft submission we utilized the preprocessed data thus obtained. After that things had stabilized. Unfortunately, one issue remained invisible to us until very late in the project: pooled training was missing some crucial files with DSVa\_DPE value embedding. More details further below.

### Pretraining *DescEmb* with RNN or BERT encoders



In order to get *DescEmb* to run we need to pretrain the text encoders. We ran the scripts provided by the authors for pretraining the *DescEmb* model with RNN and BERT encoders. We ran the scripts for pretraining the *DescEmb* model with RNN and BERT encoders. The scripts are provided in the `project_code` folder (see table above). A serious issue we had has been that all pretrainings with `--task mlm` got stalled. Debugging showed, that there must be an issue with the `MLMTokenizedDataset` class. Apparently, there is no noticeable progress when loading the first batch of data, even when letting it run for hours (on the same machine we used for pre-processing). At one point we hypothesized that maybe the dataset is too large and created a minimal version MIMIC-III with only 1% of the data randomly sampled. We used [03\\_Creating\\_minimal\\_dataset.ipynb](#) to that end. This did not solve our problem either. Eventually, we found the root cause (see Discussion / Issues summary).

## Experiment Tracking

There is a substantial number of experiments to run if we wish to reproduce the results of the paper. Each of Tables 1 and 2 in the paper reports the AUPRC on five prediction tasks with 3-4 different value embedding methods and different models. This results in 170 experiments to run each. Table 3 reports the AUPRC on single domain learning, transfer learning and pooled learning, which amounts to 120 experiments. We need to keep track not only of the AUPRC metrics, but also the hyperparameters used in each experiment. Likewise, if a run doesn't finish, we need to keep track of the reason why. While popular tools such as [Weights & Biases](#) or [mlflow](#) exist for this purpose, they, too require some overhead to set up. We will use a simple logging system for now: The `Trainer` class in the `trainer.py` file has a `log` method which logs the loss and AUPRC progression, as well as hyperparameters to a file called `train.log`. We will use this to log the hyperparameters and the results of each experiment.

In order to generate an overview as well as results table we produced some helper functions to parse the results from these log files. We provide these functions in the `project_code/evaluate` folder in `logs.py`. Some more helper functions are [in the works](#) to generate training history plots and other visualizations, much like `tensorboard` would do, but simpler and more tailored to our needs.

A global table with all experiments is stored as an Excel file in the `project_code/outputs/experiments.xlsx` file. All training logs and hyperparameters for all runs are documented in `project_code/DescEmb/outputs`. We did not upload all trained models to GitHub, but zipped them and put them on a shared folder in Google Drive.

We create the `experiments.xlsx` and `metrics.xlsx` by running `python parse_experiments.py` in the root folder of this project. The script also applies the overview table of all experiment types shown in the Methods section to introduce a `tag` column. Typical output of the script looks like:

```
Found 549 experiments and 105321 metrics.  
CodeEmb_RD                               94
```

DescEmb-BERT_FT	90
DescEmb-RNN_Scr	78
DescEmb-RNN_Scr-MLM	75
DescEmb-BERT_FT-MLM	69
DescEmb-BERT_CLS-FT	46
DescEmb-BERT_Transfer	32
DescEmb-BERT_Scr	24
DescEmb-RNN_Transfer	24
DescEmb-RNN_Pretrain-MLM	3
DescEmb-BERT_Pooled	3
CodeEmb-Pooled	3
unknown	2
DescEmb-BERT_Pretrain-MLM	2
CodeEmb_Pretrain-W2V	2
CodeEmb_W2V	2

## Model Performance

The authors reported mainly AUPRC, i.e. area under the precision-recall curve, as the evaluation metric. There are three main tables in the paper which we will try to reproduce.

- The first and second table reflect the performance of the *CodeEmb* and *DescEmb* strategies on the five tasks with and without pretraining as well as different value embedding strategies. Table 1 shows the performance on MIMIC-III while Table 2 shows the performance on eICU. This speaks to **Hypothesis 1**.
- The third table provides a comparison of the performance on single domain learning, transfer learning of model trained on MIMIC-III and fine-tuned on eICU (and vice versa) as well as the performance when using pooled data. This table speaks to **Hypothesis 2** and **Hypothesis 3**.

```
In [31]: # Load experiments.xlsx
df_exp = pd.read_excel("outputs/experiments.xlsx")
# Load metrics.xlsx
df_metrics = pd.read_excel("outputs/metrics.xlsx")
```

## Hypothesis 1

We need to reproduce (parts of) Tables 1 and 2 in the paper. We will use the AUPRC metric to evaluate the performance of the models. We will compare the performance of the *CodeEmb* and *DescEmb* strategies on the five tasks with and without pretraining as well as different value embedding strategies.

### Tracking of Experiments

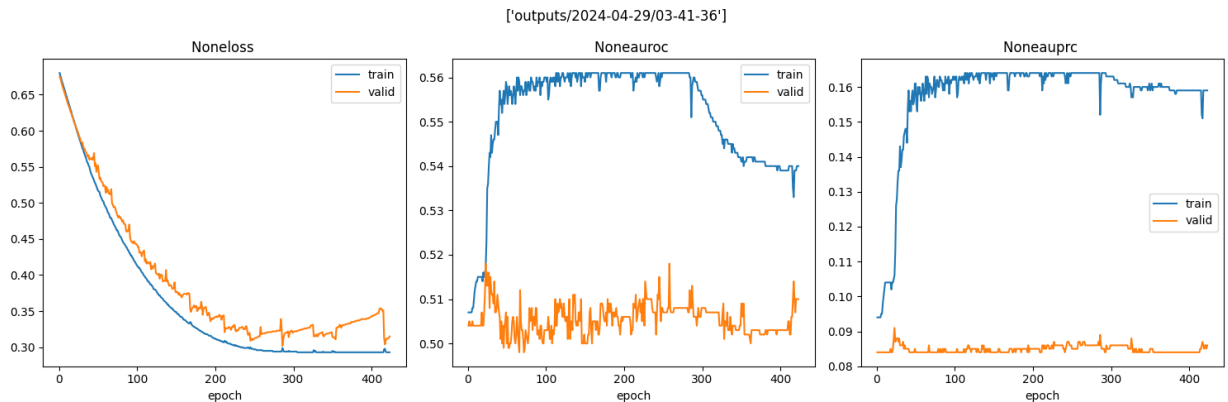
Each run produces a log file which contains the hyperparameters and the metrics (loss, auroc, auprc) of the run. We parse these log files to generate an overview of the experiments and the results. We use this to generate the tables for the paper. Additionally, we use the log files to create diagnostic plots to understand the training process better. In the run below we noticed

overfitting after about 300 epochs. Overfitting shows in two ways: The validation loss starts to increase and the validation AUPRC starts to decrease. During training we used early stopping to prevent overfitting. We kept the model with the best (lowest) validation loss.

```
In [32]: run = 'outputs/2024-04-29/03-41-36'  
df_exp.loc[df_exp['run'] == run, ['task', 'value_mode', 'tag']].values[0]
```

```
Out[32]: array(['mortality', 'DSVA', 'DescEmb-BERT_CLS-FT'], dtype=object)
```

```
In [33]: plot_metrics(df_metrics, run = run, metrics=['loss', 'auroc', 'auprc'], folds=
```



MIMIC-III

In [34]:

[illegible]

```

        'outputs/2024-04-29/10-47-31',
        'outputs/2024-04-29/12-09-48',
        'outputs/2024-04-29/13-15-34',
        'outputs/2024-04-29/14-26-09',
        'outputs/2024-04-29/15-28-54',
        'outputs/2024-04-29/16-34-44',
        'outputs/2024-04-29/19-12-26',
        'outputs/2024-04-29/20-17-52',
        'outputs/2024-04-29/22-56-58',
        'outputs/2024-04-30/00-08-13',
        'outputs/2024-04-30/01-22-03',
        'outputs/2024-04-30/02-27-44',
        'outputs/2024-04-30/03-40-52'],
'DescEmb-RNN_Scr': ['outputs/2024-04-13/18-26-13',
                    'outputs/2024-04-13/19-31-26',
                    'outputs/2024-04-21/12-07-33',
                    'outputs/2024-04-21/23-27-13',
                    'outputs/2024-04-20/15-34-10',
                    'outputs/2024-04-20/17-24-12',
                    'outputs/2024-04-20/18-47-54',
                    'outputs/2024-04-20/21-34-28',
                    'outputs/2024-04-21/04-19-24',
                    'outputs/2024-04-21/06-17-31',
                    'outputs/2024-04-21/00-40-18',
                    'outputs/2024-04-21/08-35-23',
                    'outputs/2024-04-20/08-32-42',
                    'outputs/2024-04-20/05-46-14',
                    'outputs/2024-04-20/14-11-24',
                    'outputs/2024-04-20/11-18-32',
                    'outputs/2024-04-20/04-48-24',
                    'outputs/2024-04-20/02-31-44',
                    'outputs/2024-04-20/01-31-19',
                    'outputs/2024-04-20/03-40-30'],
'DescEmb-RNN_Scr-MLM': ['outputs/2024-04-22/06-02-46',
                        'outputs/2024-04-22/06-25-03',
                        'outputs/2024-04-22/06-35-22',
                        'outputs/2024-04-22/05-04-22',
                        'outputs/2024-04-22/05-11-29',
                        'outputs/2024-04-22/05-18-59',
                        'outputs/2024-04-22/05-30-16',
                        'outputs/2024-04-22/05-40-08',
                        'outputs/2024-04-22/05-48-39',
                        'outputs/2024-04-22/04-42-48',
                        'outputs/2024-04-22/04-50-11',
                        'outputs/2024-04-22/04-57-42',
                        'outputs/2024-04-22/04-27-30',
                        'outputs/2024-04-22/04-30-54',
                        'outputs/2024-04-22/04-34-52']]

```

In [35]:

```

mimic_results_table = create_result_table(tag2run, df_exp, df_metrics)
mimic_analysis_table = create_analysis_table(tag2run, df_exp, df_metrics)

```

```

100%|██████████| 6/6 [00:03<00:00, 1.91it/s]
100%|██████████| 6/6 [00:01<00:00, 3.64it/s]

```

```
In [36]: mimic_results_table
```

Out[36]:

	task	value_mode	CodeEmb_RD	DescEmb-BERT_CLS-FT	DescEmb-BERT_FT	DescEmb-BERT_FT-MLM	DescEmb-RNN_Scr	Dr R
0	diagnosis	VA	0.639	0.645	0.639	0.635	0.647	
1	diagnosis	DSVA	0.633	0.647	0.645	0.638	0.648	
2	diagnosis	DSVA_DPE	NaN	0.641	0.642	NaN	0.645	
3	diagnosis	VC	NaN	0.646	0.641	0.641	0.646	
4	los_3day	VA	0.361	0.352	0.361	0.357	0.366	
5	los_3day	DSVA	0.356	0.357	0.353	0.354	0.346	
6	los_3day	DSVA_DPE	NaN	0.357	0.357	NaN	0.355	
7	los_3day	VC	NaN	0.356	0.357	0.356	0.355	
8	los_7day	VA	0.130	0.130	0.138	0.134	0.136	
9	los_7day	DSVA	0.132	0.129	0.140	0.136	0.137	
10	los_7day	DSVA_DPE	NaN	0.128	0.146	NaN	0.128	
11	los_7day	VC	NaN	0.134	0.134	0.135	0.135	
12	mortality	VA	0.085	0.091	0.085	0.088	0.084	
13	mortality	DSVA	0.084	0.084	0.085	0.086	0.084	
14	mortality	DSVA_DPE	NaN	0.084	0.085	NaN	0.084	
15	mortality	VC	NaN	0.087	0.087	0.087	0.087	
16	readmissionission	VA	NaN	NaN	NaN	NaN	NaN	
17	readmissionission	DSVA	NaN	NaN	NaN	NaN	NaN	
18	readmissionission	DSVA_DPE	NaN	NaN	NaN	NaN	NaN	
19	readmissionission	VC	NaN	NaN	NaN	NaN	NaN	

```
In [37]: mimic_analysis_table
```

Out[37]:

	task	value_mode	run	tag
0	diagnosis	VA	outputs/2024-04-22/06-02-46	DescEmb-RNN_Scr-MLM
1	diagnosis	DSVA	outputs/2024-04-22/06-25-03	DescEmb-RNN_Scr-MLM
2	diagnosis	DSVA_DPE	outputs/2024-04-13/18-26-13	DescEmb-RNN_Scr
3	diagnosis	VC	outputs/2024-04-22/06-35-22	DescEmb-RNN_Scr-MLM
4	diagnosis	VA	outputs/2024-04-22/06-02-46	DescEmb-RNN_Scr-MLM
...	...	...	...	...
115	readmissionission	VC		
116	readmissionission	VA		

	task	value_mode	run	tag
117	readmissionission	DSVA		
118	readmissionission	DSVA_DPE		
119	readmissionission	VC		

120 rows × 4 columns

eICU

In [38]:

```
# Selecting src_data and task by specifying which runs are to be included
tag2run = { 'CodeEmb_RD': ['outputs/2024-04-25/12-05-41',
                           'outputs/2024-04-25/11-56-22',
                           'outputs/2024-04-25/11-38-08',
                           'outputs/2024-04-25/11-35-28',
                           'outputs/2024-04-25/11-45-05',
                           'outputs/2024-04-25/11-43-37',
                           'outputs/2024-04-25/11-28-42',
                           'outputs/2024-04-25/11-26-15',
                           'outputs/2024-04-25/11-17-45',
                           'outputs/2024-04-25/11-20-12',
                           ],
            'DescEmb-BERT_CLS-FT': ['outputs/2024-04-26/11-56-56',
                                    'outputs/2024-04-26/12-44-27',
                                    'outputs/2024-04-26/13-25-15',
                                    'outputs/2024-04-26/14-08-05',
                                    'outputs/2024-04-26/10-33-22',
                                    'outputs/2024-04-26/10-44-20',
                                    'outputs/2024-04-26/10-55-34',
                                    'outputs/2024-04-26/11-09-39',
                                    'outputs/2024-04-26/11-16-30',
                                    'outputs/2024-04-26/11-28-11',
                                    'outputs/2024-04-26/11-38-41',
                                    'outputs/2024-04-26/11-49-07',
                                    'outputs/2024-04-26/10-07-24',
                                    'outputs/2024-04-26/10-16-14',
                                    'outputs/2024-04-26/10-18-43',
                                    'outputs/2024-04-26/10-28-48',
                                    'outputs/2024-04-26/09-49-06',
                                    'outputs/2024-04-26/09-54-41',
                                    'outputs/2024-04-26/09-59-47',
                                    'outputs/2024-04-26/10-03-59',
                                    ],
            'DescEmb-BERT_FT': ['outputs/2024-05-02/06-58-01',
                                'outputs/2024-05-02/08-16-52',
                                'outputs/2024-05-02/09-46-47',
                                'outputs/2024-05-02/11-11-49',
                                'outputs/2024-04-27/12-51-07',
                                'outputs/2024-05-02/12-54-43',
                                'outputs/2024-05-02/13-42-27',
                                'outputs/2024-05-02/14-27-50',
                                'outputs/2024-05-02/15-52-54',
                                'outputs/2024-05-02/16-33-49',
                                'outputs/2024-05-02/18-14-11',
                                'outputs/2024-05-04/11-30-22',
                                'outputs/2024-05-02/23-34-29',
                                'outputs/2024-05-03/00-14-39',
                                'outputs/2024-05-03/01-38-59',
                                'outputs/2024-05-04/12-37-53',
                                'outputs/2024-05-02/19-13-53',
                                'outputs/2024-05-02/20-43-54',
                                'outputs/2024-05-02/22-31-23',
                                'outputs/2024-05-04/12-04-26',
                                ],
            'DescEmb-BERT_FT-MLM': ['outputs/2024-04-30/06-14-20',
                                    'outputs/2024-04-30/07-42-04',
                                    'outputs/2024-04-30/09-06-21',
                                    'outputs/2024-04-30/10-37-13',
                                    ]
}
```



```

        'outputs/2024-04-30/12-03-53',
        'outputs/2024-04-30/12-41-38',
        'outputs/2024-04-22/09-25-28',
        'outputs/2024-04-30/13-23-34',
        'outputs/2024-04-30/14-50-12',
        'outputs/2024-05-01/10-33-13',
        'outputs/2024-05-01/11-12-47',
        'outputs/2024-05-01/11-56-05',
        'outputs/2024-05-01/08-11-56',
        'outputs/2024-05-01/08-59-05',
        'outputs/2024-05-01/09-43-37',
    ],
    'DescEmb-RNN_Scr': [
        'outputs/2024-04-21/13-36-42',
        'outputs/2024-04-22/21-23-49',
        'outputs/2024-04-22/11-11-30',
        'outputs/2024-04-22/00-49-19',
        'outputs/2024-04-20/16-11-15',
        'outputs/2024-04-20/19-45-10',
        'outputs/2024-04-20/17-36-40',
        'outputs/2024-04-20/21-59-10',
        'outputs/2024-04-21/06-34-51',
        'outputs/2024-04-21/05-05-54',
        'outputs/2024-04-21/09-01-57',
        'outputs/2024-04-21/01-11-41',
        'outputs/2024-04-20/14-26-33',
        'outputs/2024-04-20/05-57-07',
        'outputs/2024-04-20/08-51-14',
        'outputs/2024-04-20/11-35-46',
        'outputs/2024-04-20/04-56-00',
        'outputs/2024-04-20/01-51-58',
        'outputs/2024-04-20/03-55-24',
        'outputs/2024-04-20/02-52-27',
    ],
    'DescEmb-RNN_Scr-MLM': [
        'outputs/2024-04-22/13-17-51',
        'outputs/2024-04-22/13-48-48',
        'outputs/2024-04-22/14-25-52',
        'outputs/2024-04-22/12-07-19',
        'outputs/2024-04-22/12-13-50',
        'outputs/2024-04-22/12-22-07',
        'outputs/2024-04-22/12-35-59',
        'outputs/2024-04-22/12-41-30',
        'outputs/2024-04-22/12-48-20',
        'outputs/2024-04-22/11-38-39',
        'outputs/2024-04-22/11-47-27',
        'outputs/2024-04-22/11-56-23',
        'outputs/2024-04-22/11-21-26',
        'outputs/2024-04-22/11-24-19',
        'outputs/2024-04-22/11-27-44',
    ]
}]

```

In [39]:

```

eicu_results_table = create_result_table(tag2run, df_exp, df_metrics)
eicu_analysis_table = create_analysis_table(tag2run, df_exp, df_metrics)

```

```

100%|██████████| 6/6 [00:02<00:00, 2.01it/s]
100%|██████████| 6/6 [00:01<00:00, 3.80it/s]

```

```
In [40]: eicu_results_table
```

Out[40]:

	task	value_mode	CodeEmb_RD	DescEmb-BERT_CLS-FT	DescEmb-BERT_FT	DescEmb-BERT_FT-MLM	DescEmb-RNN_Scr	Dr R
0	diagnosis	VA	0.305	0.441	0.442	0.445	0.444	
1	diagnosis	DSVA	0.315	0.425	0.475	0.479	0.444	
2	diagnosis	DSVA_DPE	NaN	0.449	0.475	NaN	0.381	
3	diagnosis	VC	NaN	0.402	0.464	0.451	0.429	
4	los_3day	VA	0.392	0.483	0.489	0.487	0.492	
5	los_3day	DSVA	0.391	0.522	0.501	0.521	0.508	
6	los_3day	DSVA_DPE	NaN	0.498	0.508	NaN	0.471	
7	los_3day	VC	NaN	0.501	0.514	0.507	0.540	
8	los_7day	VA	0.111	0.118	0.113	0.128	0.124	
9	los_7day	DSVA	0.111	0.140	0.166	0.157	0.123	
10	los_7day	DSVA_DPE	NaN	0.149	0.147	NaN	0.124	
11	los_7day	VC	NaN	0.139	0.160	0.142	0.143	
12	mortality	VA	0.084	0.111	0.110	0.136	0.165	
13	mortality	DSVA	0.073	0.065	0.141	0.138	0.154	
14	mortality	DSVA_DPE	NaN	0.118	0.103	NaN	0.122	
15	mortality	VC	NaN	0.108	0.175	0.138	0.110	
16	readmissionission	VA	NaN	NaN	NaN	NaN	NaN	
17	readmissionission	DSVA	NaN	NaN	NaN	NaN	NaN	
18	readmissionission	DSVA_DPE	NaN	NaN	NaN	NaN	NaN	
19	readmissionission	VC	NaN	NaN	NaN	NaN	NaN	

```
In [41]: eicu_analysis_table
```

Out[41]:

	task	value_mode	run	tag
0	diagnosis	VA	outputs/2024-04-22/13-17-51	DescEmb-RNN_Scr-MLM
1	diagnosis	DSVA	outputs/2024-04-22/13-48-48	DescEmb-RNN_Scr-MLM
2	diagnosis	DSVA_DPE	outputs/2024-04-22/11-11-30	DescEmb-RNN_Scr
3	diagnosis	VC	outputs/2024-04-22/14-25-52	DescEmb-RNN_Scr-MLM
4	diagnosis	VA	outputs/2024-04-22/13-17-51	DescEmb-RNN_Scr-MLM
...	...	...	...	...
115	readmissionission	VC		
116	readmissionission	VA		

	task	value_mode	run	tag
117	readmissionission	DSVA		
118	readmissionission	DSVA_DPE		
119	readmissionission	VC		
120 rows × 4 columns				

Hypothesis 1 asks if *DescEmb* has higher predictive power than direct EHR code embedding using *CodeEmb*. We agree this is largely the case. We take note of the following:

- The AUPRC for MIMIC-III is higher than for eICU. If this is attributable to eICU being a larger dataset could be hypothesized.
- DSVA\_DPE is not available with *CodeEmb*, because it has no way of digit place embedding needed for that.
- DSVA\_DPE did not finish training in experiments involving MLM pretrained parameters. We tried to debug that, but didn't succeed.
- VC value encoding didn't run for CodeEmb-RD. We narrowed it down to a dimension mismatch of  $x$  yielded by the DataLoader and the value encoded version of a digit value (i.e. 151 vs. 150). We couldn't eliminate the root cause, however.
- DSVA and DSVA-DPE performance is very similar.

For the following experiments the authors used CodeEmb-RD + VC and (DescEmb-BERT-FT, DescEmb-RNN-Src, DescEmb-RNN-Src-MLM) + DSVA\_DPE. Since these are mostly not available, but performance of DSVA and DSVA\_DPE were very similar we continued using DSVA value encoding.

## Hypothesis 2 and 3: Single Domain Learning vs. Transfer learning

In [42]:

```
runs4table3 = {
    'Single_Mimic-III': {
        'CodeEmb_RD': ['outputs/2024-05-06/00-52-30',
'outputs/2024-05-06/00-58-20',
'outputs/2024-05-06/01-03-01',
'outputs/2024-05-06/01-07-33',
'outputs/2024-05-06/01-12-31',
],
        'DescEmb-BERT_FT': ['outputs/2024-05-01/13-21-37',
'outputs/2024-05-01/16-57-55',
'outputs/2024-05-01/19-39-34',
'outputs/2024-05-02/00-54-15',
'outputs/2024-05-02/04-59-22',
],
        'DescEmb-RNN_Scr': [
            'outputs/2024-04-20/08-32-42',
'outputs/2024-04-20/02-31-44',
'outputs/2024-04-20/17-24-12',
'outputs/2024-04-21/23-27-13',
'outputs/2024-04-21/04-19-24',
],
        'DescEmb-RNN_Scr-MLM': ['outputs/2024-04-22/04-30-54',
'outputs/2024-04-22/04-50-11',
'outputs/2024-04-22/05-11-29',
'outputs/2024-04-22/05-40-08',
'outputs/2024-04-22/06-25-03',
],
    },
    'Transfer_Mimic-III': {
        'CodeEmb_RD': [],
        'DescEmb-BERT_FT': ['outputs/2024-05-07/15-59-24',
'outputs/2024-05-07/16-31-26',
'outputs/2024-05-07/16-52-37',
'outputs/2024-05-07/17-17-46',
'outputs/2024-05-07/17-38-11',
],
        'DescEmb-RNN_Scr': ['outputs/2024-05-07/06-14-52',
'outputs/2024-05-07/07-01-52',
'outputs/2024-05-07/07-23-01',
'outputs/2024-05-07/07-49-08',
'outputs/2024-05-07/08-12-43',
],
    },
    'Single_eICU': {
        'CodeEmb_RD': ['outputs/2024-05-06/03-52-09',
'outputs/2024-05-06/03-57-53',
'outputs/2024-05-06/04-01-15',
'outputs/2024-05-06/04-06-59',
'outputs/2024-05-06/04-12-51',
],
        'DescEmb-BERT_FT': ['outputs/2024-05-02/08-16-52',
'outputs/2024-05-02/13-42-27',
'outputs/2024-05-02/16-33-49',
'outputs/2024-05-02/20-43-54',
'outputs/2024-05-03/00-14-39',
],
    },
}
```

```

        ],
        'DescEmb-RNN_Scr': ['outputs/2024-04-20/08-51-14',
'outputs/2024-04-20/17-36-40',
'outputs/2024-04-20/02-52-27',
'outputs/2024-04-21/05-05-54',
'outputs/2024-04-22/00-49-19',

        ],
        'DescEmb-RNN_Scr-MLM': ['outputs/2024-04-22/11-24-19',
'outputs/2024-04-22/11-47-27',
'outputs/2024-04-22/12-13-50',
'outputs/2024-04-22/12-41-30',
'outputs/2024-04-22/13-48-48',
]
    },
    'Transfer_eICU': {
        'CodeEmb_RD': [],
        'DescEmb-BERT_FT': [
'outputs/2024-05-07/17-57-16',
'outputs/2024-05-07/19-14-41',
'outputs/2024-05-07/19-43-48',
'outputs/2024-05-07/20-03-01',
'outputs/2024-05-07/20-15-08',
],
        'DescEmb-RNN_Scr': [
'outputs/2024-05-07/08-30-57',
'outputs/2024-05-07/09-33-14',
'outputs/2024-05-07/09-45-20',
'outputs/2024-05-07/10-21-37',
'outputs/2024-05-07/10-32-54',
]
    }
}

```

```

In [43]: table = pd.DataFrame()
table['task'] = sorted(list(['diagnosis', 'mortality', 'los_3day', 'los_7day']
table['tag'] = list(runs4table3['Single_Mimic-III'].keys()*5
fold = 'valid' # 'train'
for col in runs4table3:
    for tag in runs4table3[col]:
        for run in runs4table3[col][tag]:
            task = df_exp.loc[(df_exp['run']==run), 'task'].values[0]
            best = get_best_epoch(df_metrics, run, fold)
            auprc = get_metric_at_epoch(df_metrics, epoch=best, run=run, metr
            table.loc[(table['task']==task) & (table['tag']==tag), col] = auprc

```

In [44]: table

```

Out[44]:

```

	task	tag	Single_Mimic-III	Transfer_Mimic-III	Single_eICU	Transfer_eICU
0	diagnosis	CodeEmb_RD	0.634	NaN	0.324	NaN
1	diagnosis	DescEmb-BERT_FT	0.645	0.647	0.475	0.469

	task	tag	Single_Mimic-III	Transfer_Mimic-III	Single_eICU	Transfer_eICU
2	diagnosis	DescEmb-RNN_Scr	0.648	0.646	0.444	0.452
3	diagnosis	DescEmb-RNN_Scr-MLM	0.598	NaN	0.399	NaN
4	los_3day	CodeEmb_RD	0.356	NaN	0.391	NaN
5	los_3day	DescEmb-BERT_FT	0.353	0.357	0.501	0.522
6	los_3day	DescEmb-RNN_Scr	0.346	0.360	0.508	0.508
7	los_3day	DescEmb-RNN_Scr-MLM	0.356	NaN	0.454	NaN
8	los_7day	CodeEmb_RD	0.132	NaN	0.111	NaN
9	los_7day	DescEmb-BERT_FT	0.140	0.134	0.166	0.140
10	los_7day	DescEmb-RNN_Scr	0.137	0.135	0.123	0.157
11	los_7day	DescEmb-RNN_Scr-MLM	0.137	NaN	0.122	NaN
12	mortality	CodeEmb_RD	0.084	NaN	0.073	NaN
13	mortality	DescEmb-BERT_FT	0.085	0.084	0.141	0.094
14	mortality	DescEmb-RNN_Scr	0.084	0.093	0.154	0.154
15	mortality	DescEmb-RNN_Scr-MLM	0.094	NaN	0.138	NaN
16	readmission	CodeEmb_RD	0.045	NaN	0.113	NaN
17	readmission	DescEmb-BERT_FT	0.046	0.051	0.243	0.109
18	readmission	DescEmb-RNN_Scr	0.045	0.045	0.116	0.112
19	readmission	DescEmb-RNN_Scr-MLM	0.045	NaN	0.115	NaN

## Embedding Visualization

In [45]:

```
eicu_analysis_table
```

Out[45]:

	task	value_mode	run	tag
0	diagnosis	VA	outputs/2024-04-22/13-17-51	DescEmb-RNN_Scr-MLM
1	diagnosis	DSVA	outputs/2024-04-22/13-48-48	DescEmb-RNN_Scr-MLM
2	diagnosis	DSVA_DPE	outputs/2024-04-22/11-11-30	DescEmb-RNN_Scr

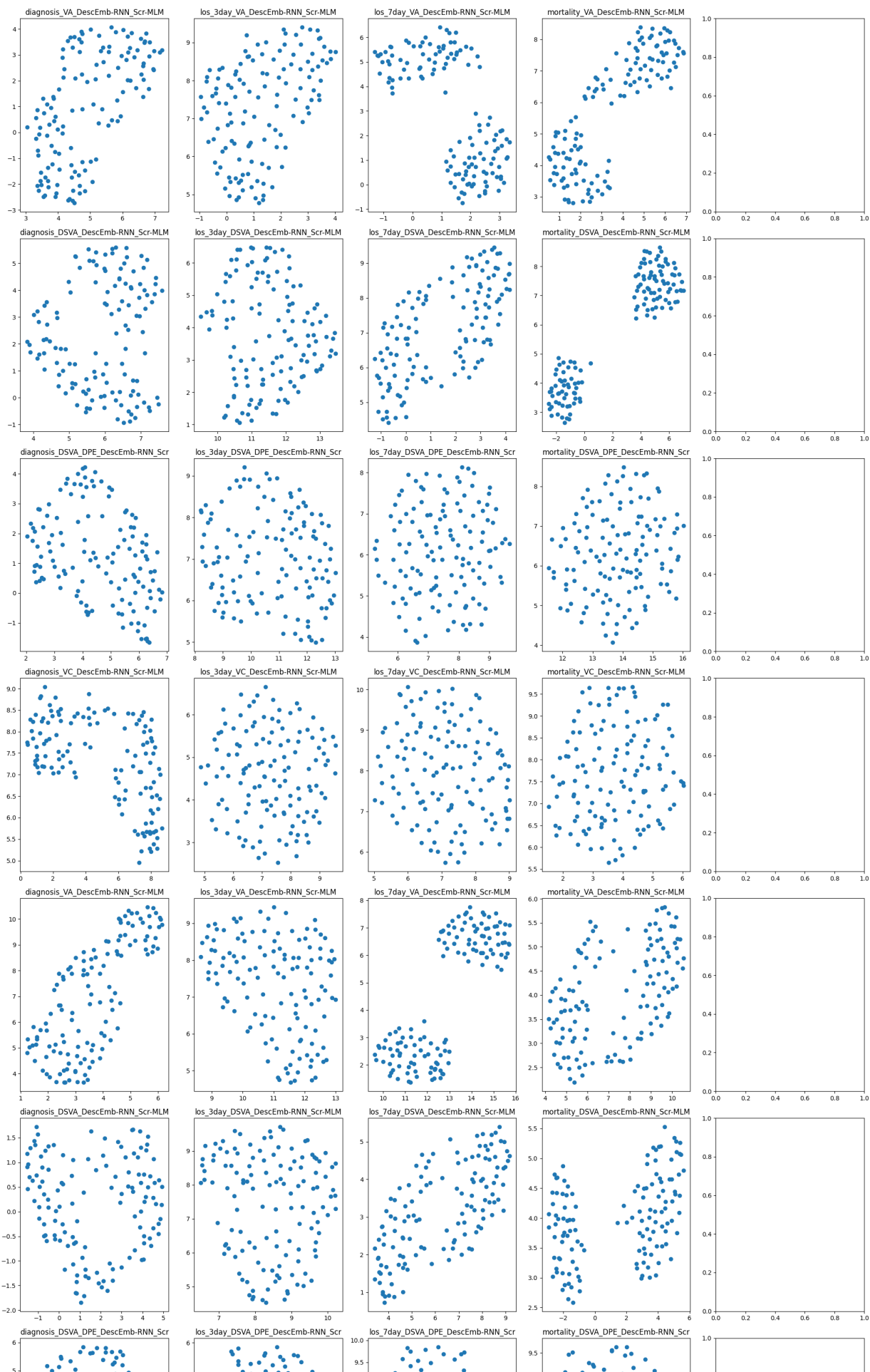
	task	value_mode	run	tag
3	diagnosis	VC	outputs/2024-04-22/14-25-52	DescEmb-RNN_Scr-MLM
4	diagnosis	VA	outputs/2024-04-22/13-17-51	DescEmb-RNN_Scr-MLM
...	...	...	...	...
115	readmissionission	VC		
116	readmissionission	VA		
117	readmissionission	DSVA		
118	readmissionission	DSVA_DPE		
119	readmissionission	VC		

120 rows × 4 columns

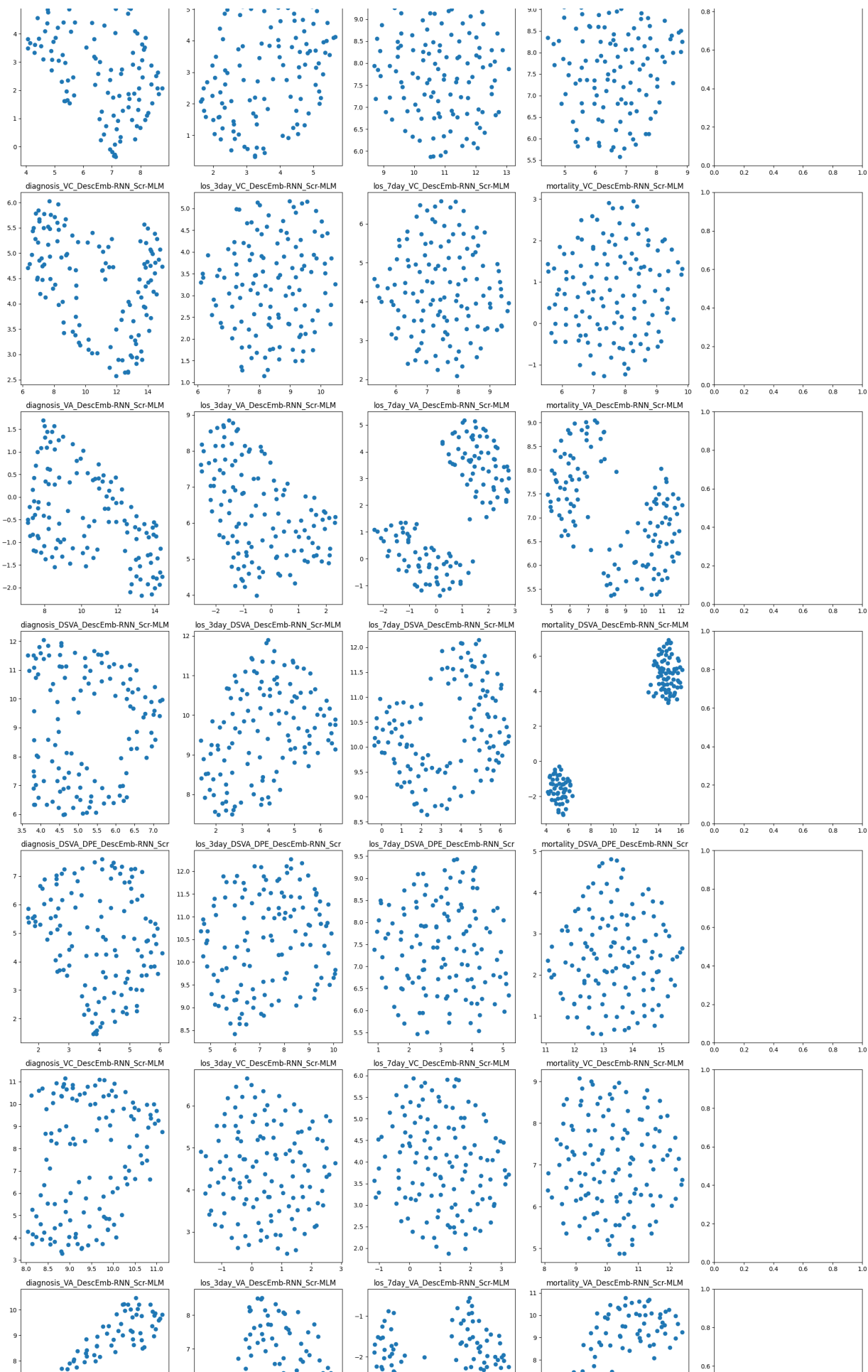
In [48]:

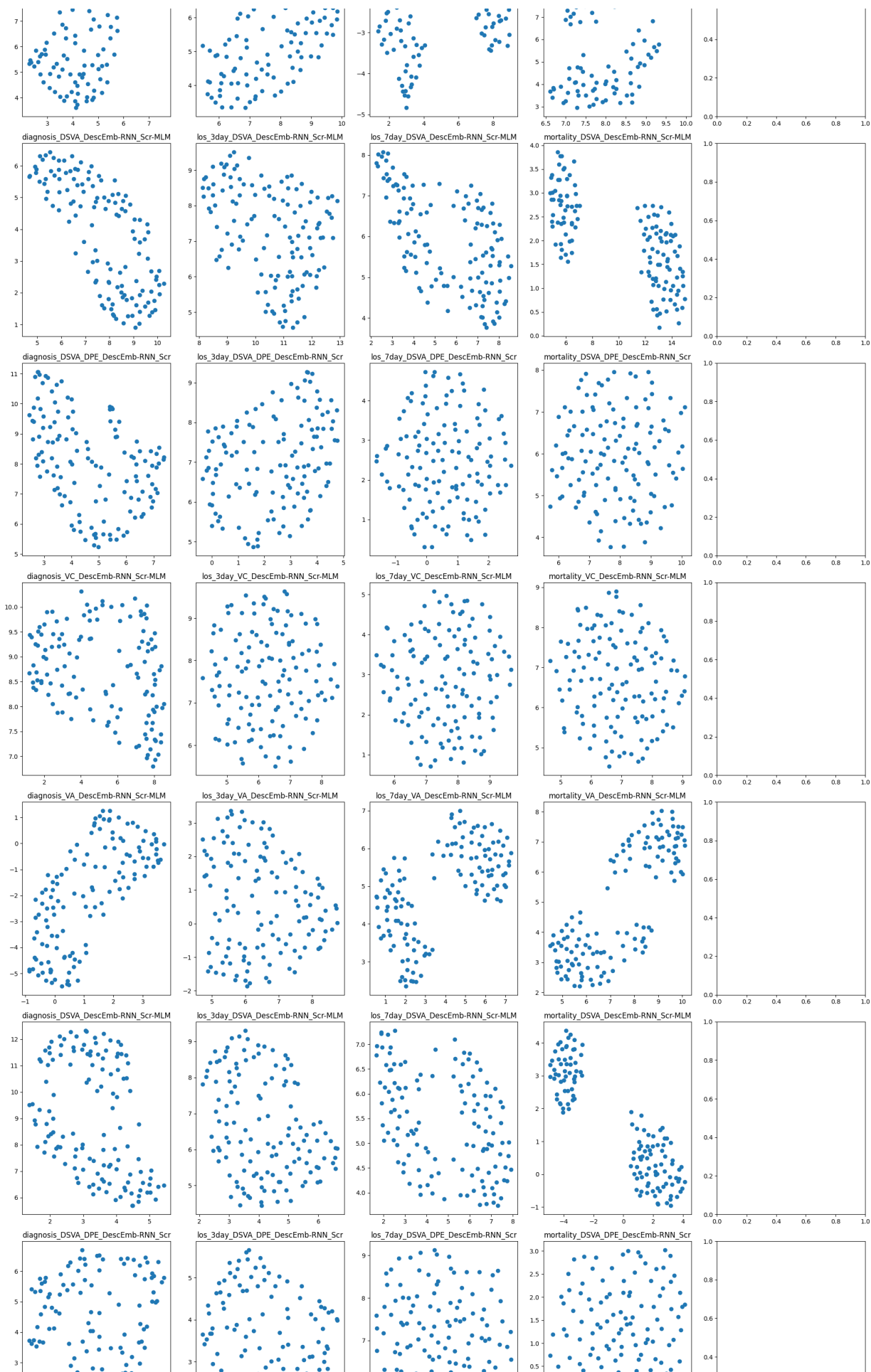
```
plot_all_umaps(mimic_analysis_table)
```

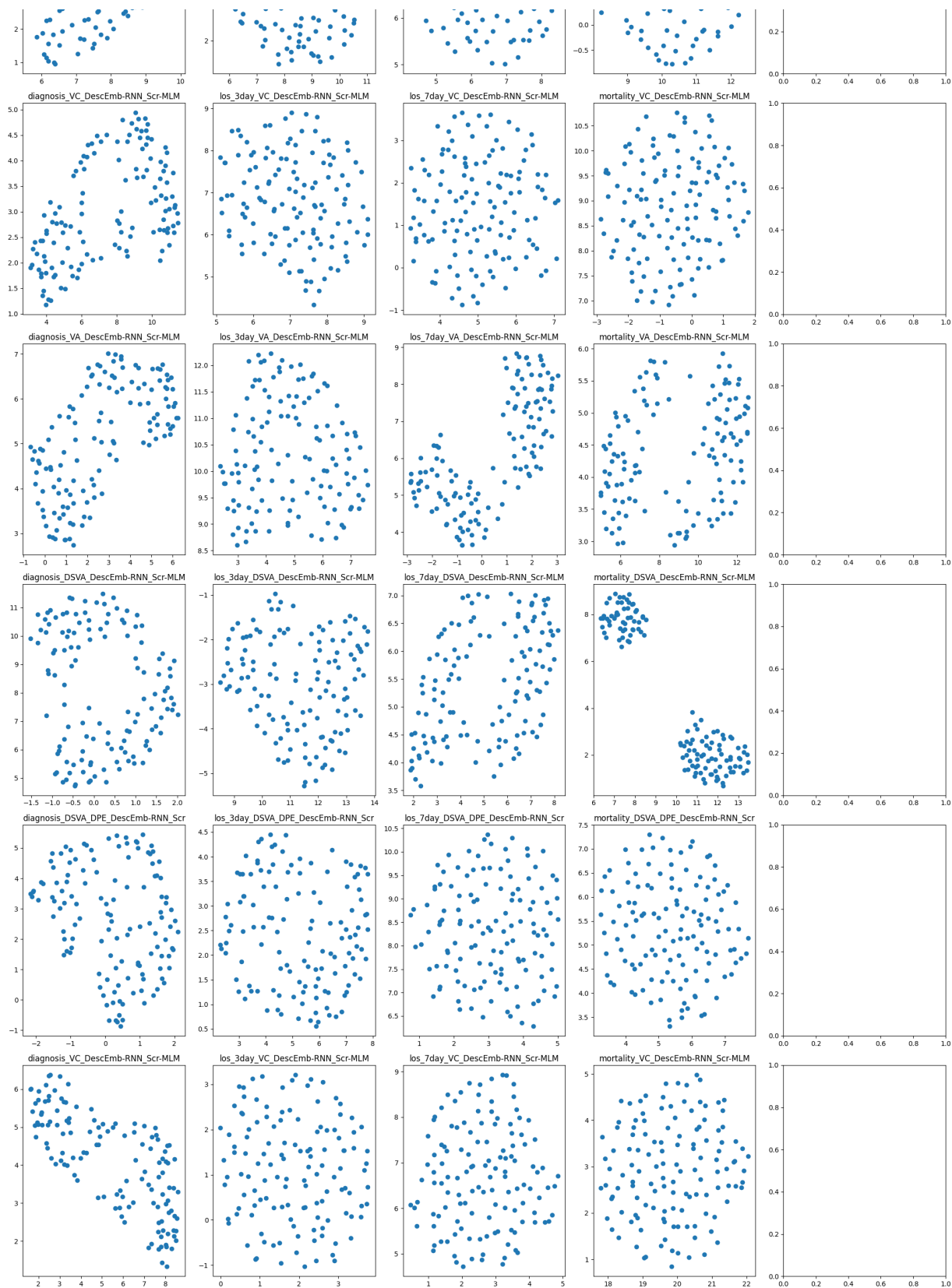
5it [03:57, 47.44s/it]









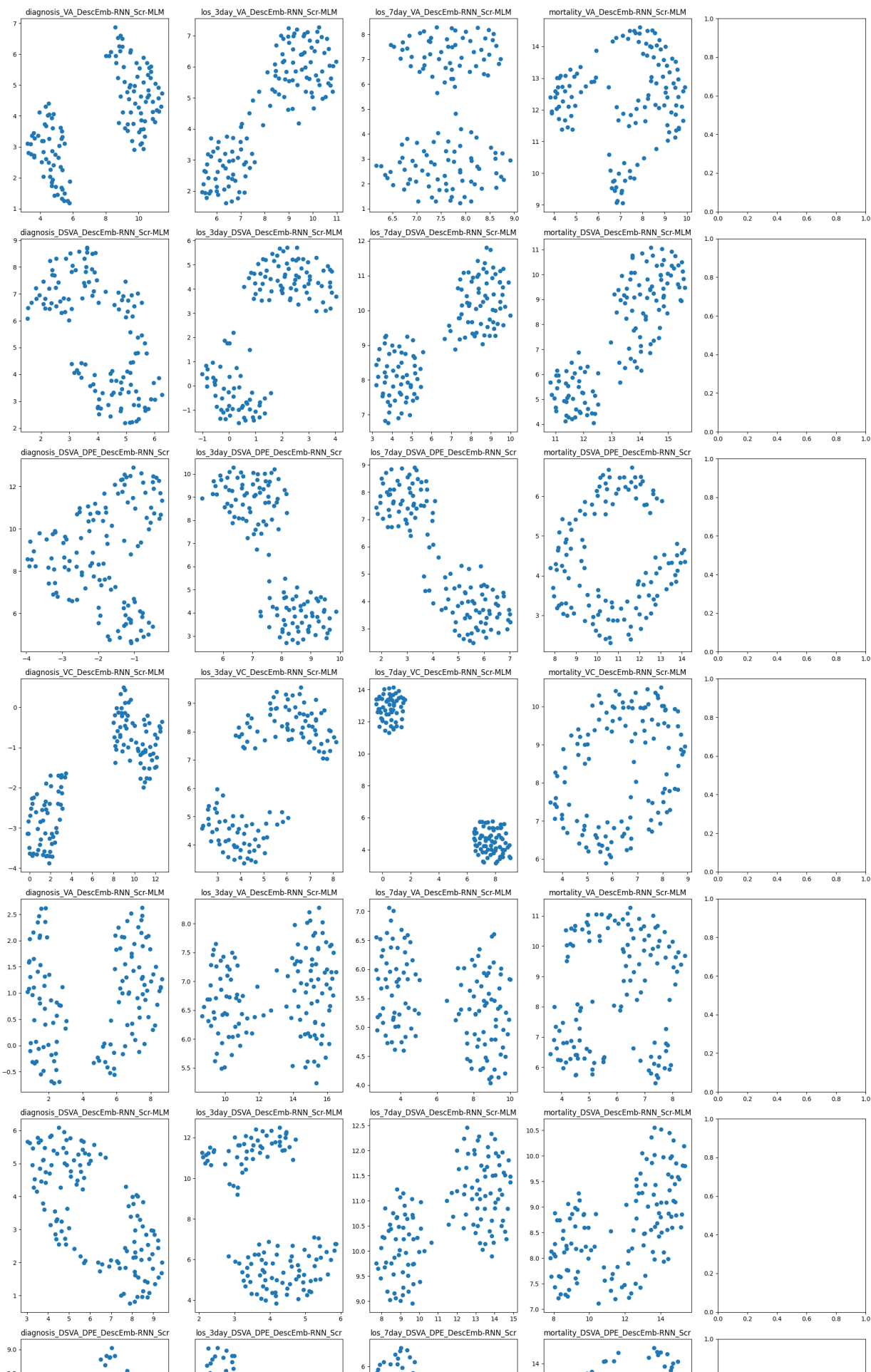


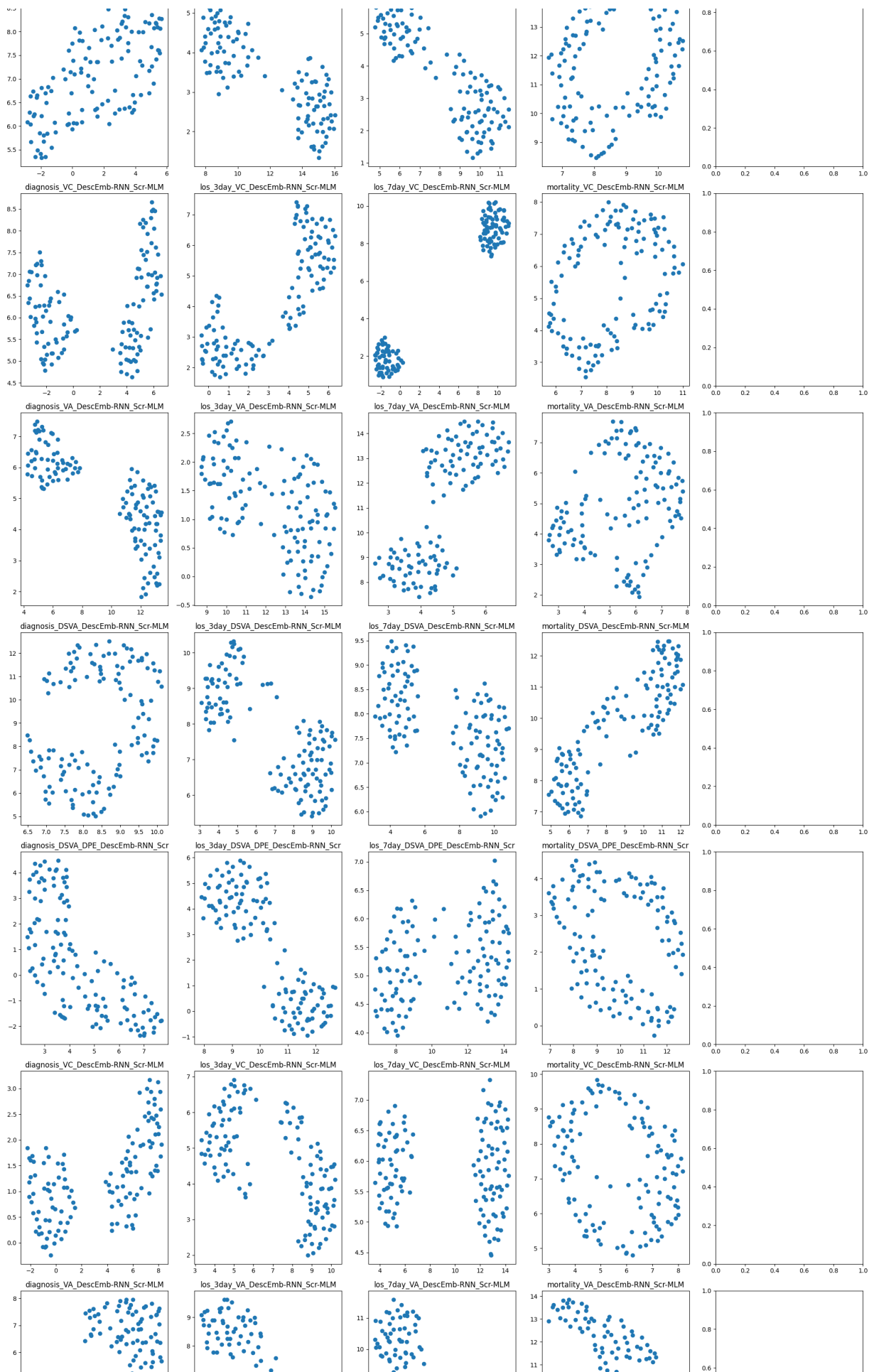




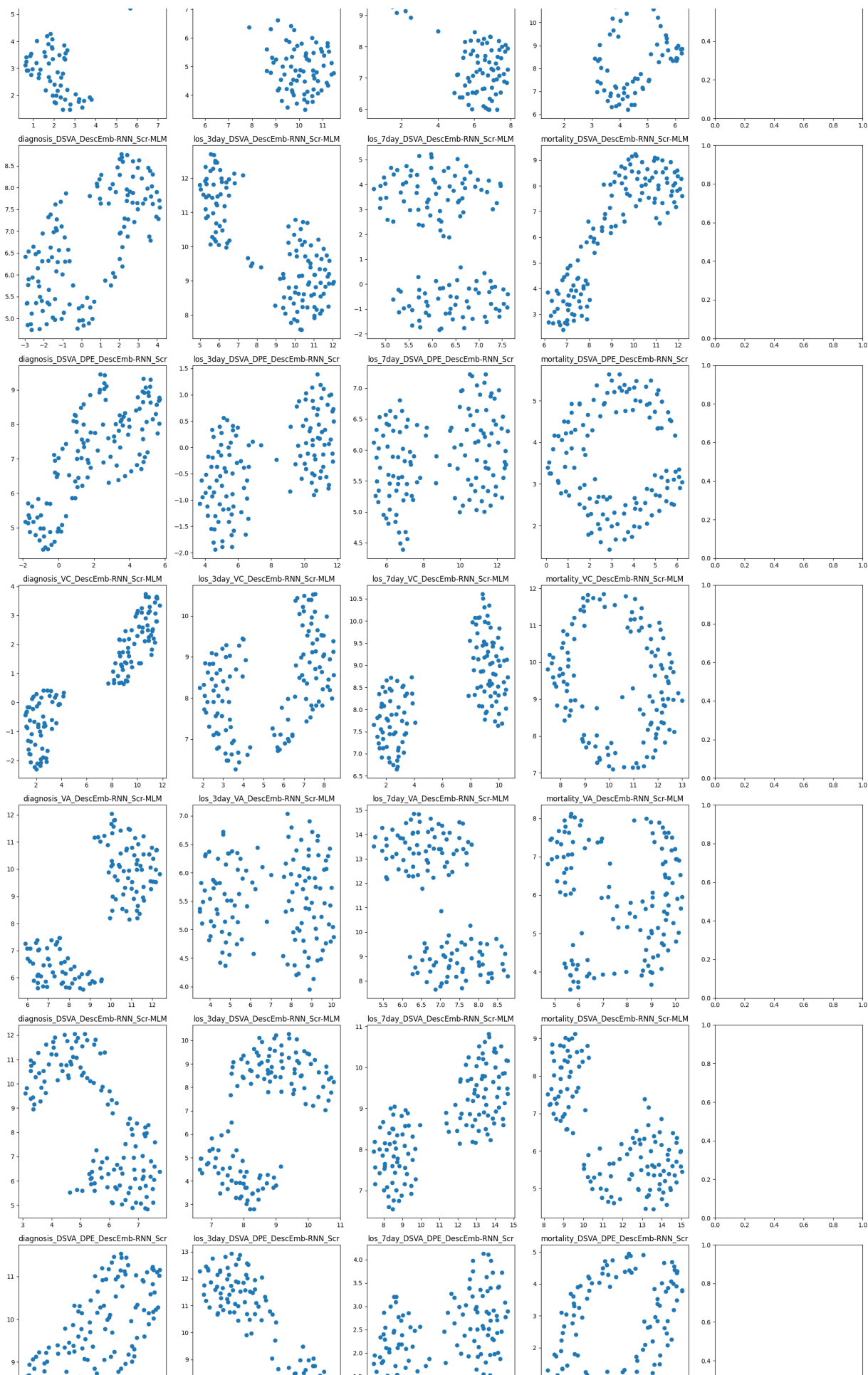
```
In [49]: plot_all_umaps(eicu_analysis_table)
```

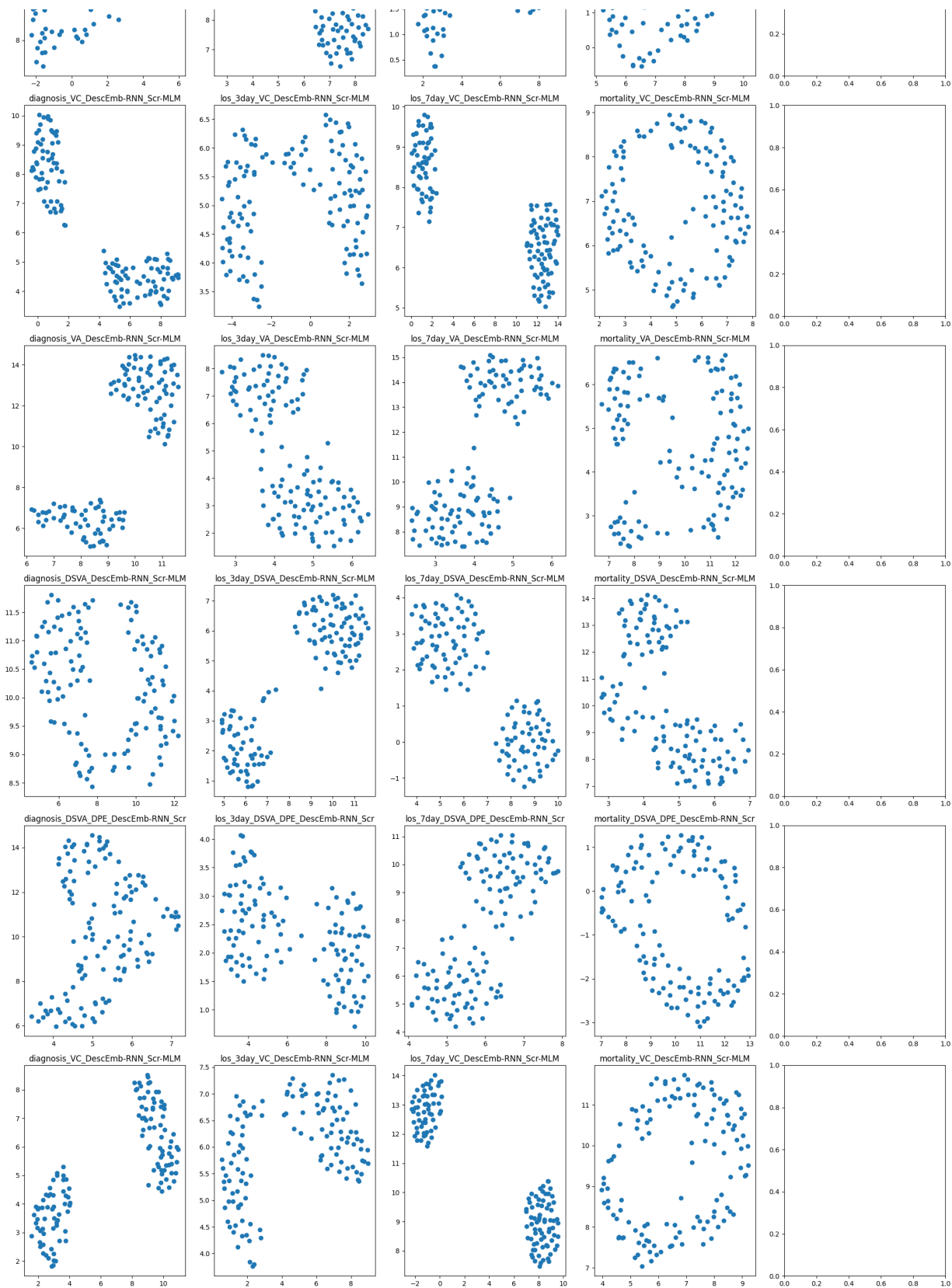
```
5it [03:40, 44.19s/it]
```















## Embedding Takeaways

Above are UMAP plots of the various model embeddings. UMAP is a dimensionality reduction technique that aims to preserve the local and global structure of high-dimensional data in a low-dimensional space. When we reduce the high dimensional embedding space, we are able to see the underlying structure and relationship the data points have with each other. Since most of the tasks are binary classification problems, we should see 2 clusters in many tasks which show that the training model was able to learn embeddings that created a separable space for the two classes. While the labels are not present and are non trivial to gather, we are still able to see the separability in datapoints in the embedding space. We use that to get an idea about the class separation in embedding space. A clear separation is indicative of a good classifier.

The los 7 day and los 3 day tasks consistently separate the best, and readmission task performs the worst with the embedding space looking like a gaussian blob in both the MIMIC and eICU datasets.

It is also interesting to note that the eICU separates better than the MIMIC dataset just from looking at the embeddings, which is consistent with the authors findings and our findings from the AUPRC tables above.

## Discussion

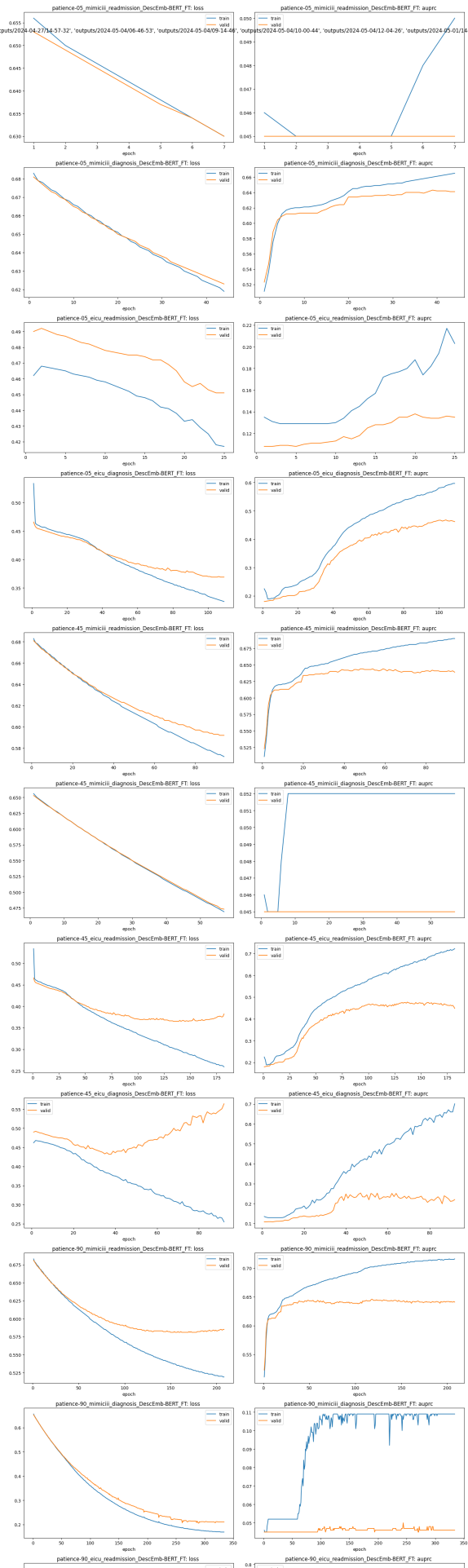
### Hypothesis 1

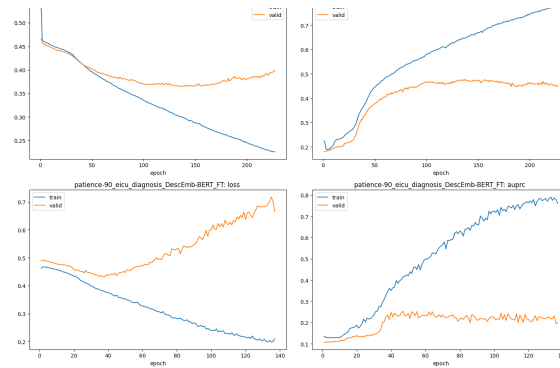
Overall we confirm the main observation of Tables 1 and 2 in the paper in that *DescEmb* achieves higher AUPRC than *CodeEmb*. We did not quite reach the same AUPRC values as reported in the paper. The reason for that is not known to us. For training we used the default parameters suggested by the author's `main.py`. We hypothesized, however, that some training runs might have been run for too few epochs. So, we used different patience settings.

```
In [67]: subtitles = ['patience-05_mimiciii_readmission_DescEmb-BERT_FT',
                    'patience-05_mimiciii_diagnosis_DescEmb-BERT_FT',
                    'patience-05_eicu_readmission_DescEmb-BERT_FT',
                    'patience-05_eicu_diagnosis_DescEmb-BERT_FT',
                    'patience-45_mimiciii_readmission_DescEmb-BERT_FT',
                    'patience-45_mimiciii_diagnosis_DescEmb-BERT_FT',
                    'patience-45_eicu_readmission_DescEmb-BERT_FT',
                    'patience-45_eicu_diagnosis_DescEmb-BERT_FT',
                    'patience-90_mimiciii_readmission_DescEmb-BERT_FT',
                    'patience-90_mimiciii_diagnosis_DescEmb-BERT_FT',
                    'patience-90_eicu_readmission_DescEmb-BERT_FT',
                    'patience-90_eicu_diagnosis_DescEmb-BERT_FT',]

plot_metrics(df_metrics, run=['outputs/2024-04-27/08-38-40',
                             'outputs/2024-04-27/10-29-36',
                             'outputs/2024-04-27/11-28-00',
                             'outputs/2024-04-27/14-57-32',
                             'outputs/2024-05-04/06-46-53',
                             'outputs/2024-05-04/09-14-46',
                             'outputs/2024-05-04/10-00-44',
                             'outputs/2024-05-04/12-04-26',
                             'outputs/2024-05-01/14-15-52',
                             'outputs/2024-05-02/01-48-46',
                             'outputs/2024-05-02/09-46-47',
                             'outputs/2024-05-02/21-41-30'], metrics=['loss', 'auprc'], folds=['train', '\
```

['outputs/2024-04-27/08-38-40', 'outputs/2024-04-27/10-29-36', 'outputs/2024-04-27/11-28-00', 'outputs/2024-05-04/06-46-53', 'outputs/2024-05-04/09-14-14', 'outputs/2024-05-04/10-00-44', 'outputs/2024-05-04/12-04-26', 'outputs/2024-05-01/16-15-52', 'outputs/2024-05-02/01-48-46', 'outputs/2024-05-02/09-46-47', 'outputs/2024-05-02/21-41-30']





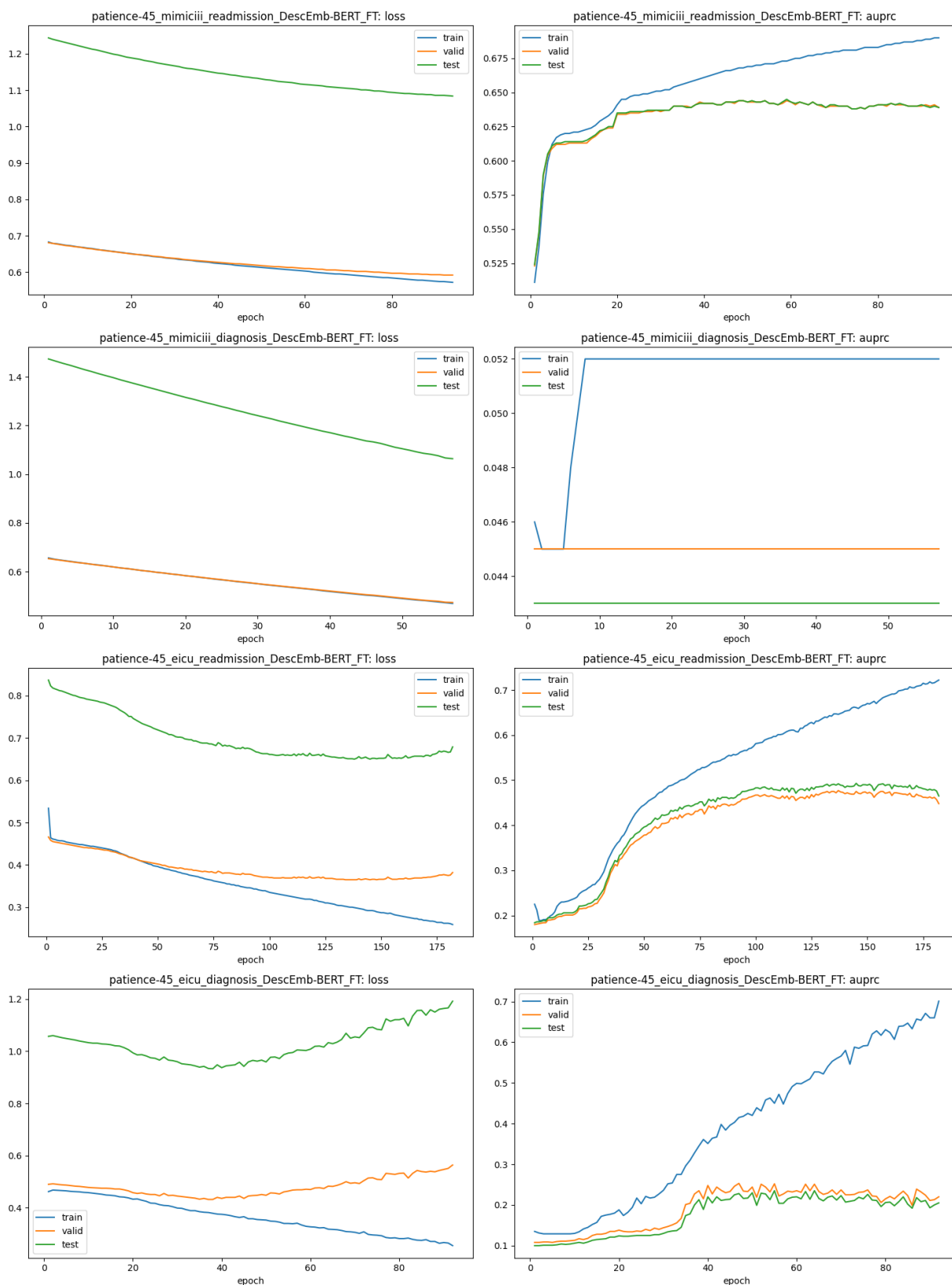
While we can't show the training history for all conditions we picked DescEmb-BERT-FT DSVa\_DPE trained on both datasets with patience settings 5, 45, 90 and for two tasks, namely readmission and diagnosis prediction. We notice:

- The default setting of patience 5 seems to stop too early in all cases.
- The patience setting 45 starts to overfit at different times, depending on task and dataset. While readmission on mimiciii starts overfitting at epoch 40, diagnosis on mimiciii does not. For eicu we get overfitting starting at about 75 and 40 epochs respectively.
- Patience 90 seems too long in all cases. Thus, we found patience 45 to be a reasonable trade-off. Still, we have to rely on the early stopping mechanism to reliably detect the minimum validation loss and stop the experiment. So, we can not exclude that overfitting still happened which might have deteriorated our results.

Another indication of overfitting is that the test results tend to perform worse than validation, indicative of poor generalization of the model. Only our train results come close to what the authors reported.

```
In [69]: plot_metrics(df_metrics, run=['outputs/2024-05-04/06-46-53',
'outputs/2024-05-04/09-14-46',
'outputs/2024-05-04/10-00-44',
'outputs/2024-05-04/12-04-26'], metrics=['loss', 'auprc'], folds=['train', '
'patience-45_mimiciii_diagnosis_DescEmb-BERT_FT',
'patience-45_eicu_readmission_DescEmb-BERT_FT',
'patience-45_eicu_diagnosis_DescEmb-BERT_FT',])
```





## Hypothesis 4 - DescEmb outperforms CodeEmb in predicting patient outcomes

The authors report that *DescEmb* outperforms *CodeEmb* in most tasks. The basis for which they make this claim is the AUPRC metric on five tasks: Dx, Mort, LOS>3, LOS>7, ReAdm. [Wornow et](#)

al. (2023) criticize the use of AUPRC as the sole metric for evaluation in combination with a narrow set of prediction tasks. They argue that the tasks used were not always overlapping with other publications thus making it hard to compare the results. Wornow et al. (2023) provide a taxonomy of medical foundation models. According to this taxonomy the tasks used in this paper would fall under the category of "prediction tasks" by an FEMR.

While in NLP tasks standardized benchmarks exist, such as BLUE and ROUGE the same is not always true in the medical domain. However, there have been some efforts to standardize the evaluation of medical NLP tasks, such as the BLURB benchmark put forward by Gu et al. (2022). To the best of our knowledge the same is unfortunately not true for the tasks used in this paper.

## Hypothesis 5 - Using a BERT model pre-trained on a domain-specific dataset (optional)

To assess if such a domain-specific pre-training improve the can an improve performance we had planned to replace the `AutoTokenizer` in the `BaseDataset` by one derived from `PubMedBERT` as reported by Gu et al (2022). Then we could run preprocessing, pre-training and pooled training with this tokenization. Unfortunately, we did not have enough time to do all these steps, due to other, more basic issues as discussed.

## Issues and Challenges

To start with we really appreciate the authors made comprehensive code available. This is a great help for reproducibility. However, the code has had some issues when we started out. We also had to understand all the available command-line options by reading both documentation and source code as well as a lot of experimentation.

### Summary of issues we experienced

Issue	Impact	Solution
Bugs in preprocessing when we started	Could not train any DescEmb model	Mostly solved by author's fixes made in April 2024
Data with DSVa_DPE value mode not generated in preprocessing	Could not do pooled training	Modification to <code>preprocess_main.py</code> (too late in the project)
Pretraining with MLM target took excessively long	MLM pretraining not possible	We narrowed it down to <code>MLMTokenizedDataset</code> and made modifications to it (see below)
Training CodeEmb with VC value mode did not run	We could not include CodeEmb in Table 3 which used VC	None so far, we used DSVa instead
Training CodeEmb-W2V did not run	Less fair comparison between CodeEmb and DescEmb	None so far

Issue	Impact	Solution
Transfer learning DescEmb-RNN with DSVa_DPE did not run	Can not include in Table 3	None so far, we used DSVa instead
Pooled training didn't run	Can't speak to how pooling data impacts model performance	None so far. We solved part the issues with preprocessing, but ultimately, training still failed

## Changes made to MLMTokenizedDataset

The challenges we had were such that cuda was not properly initialized. This may have been due to the fact that the authors trained on a multi-GPU machine and used `nn.DataParallel`s which handles GPU initialization differently. So, we had to assign tensors to GPUs in several places within the dataset class, for example:

```
input = {"input_ids": torch.stack([s["input_ids"] for s in
samples]).to(self.device)}
        input["seq_len"] = torch.stack([s["seq_len"] for s in
samples]).to(self.device)
        input["value"] = torch.stack([s["value"] for s in
samples]).to(self.device)
        out = {"label": torch.stack([s["label"] for s in
samples]).to(self.device)}
```

Here `self.device` was

```
self.device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
```

Given the age of the publication the authors used `nn.DataParallel`s for multi-GPU training, which is now deprecated in PyTorch. We did NOT try to update the code to use

`torch.nn.parallel.DistributedDataParallel` instead, because we wished to avoid multi-GPU training due to cost considerations.

## Challenges

Generally, the sheer number of experiments is very large (175 per dataset for tables 1 and 2, 80 additional ones for table 3) and the amount of RAM and VRAM necessary partially went beyond of what we could run on our local machines (mostly for preprocessing, pretraining and some variants of DescEmb-BERT with FT and FT-MLM). The memory limitation was overcome by renting a cloud node with 1 RTX6000 Ada GPU that had 48 GB of VRAM, an AMD EPYC 7713 64-Core Processor and 1 TB of RAM (the latter was not fully used). Over the course of the experiments we used up about 180 GB of storage, which was just about enough (when deleting some experiments we had downloaded from the server).

As for the number of experiments we resorted to bash scripting of the author's `main.py`. We used nested loops wherever possible to run a whole batch of experiments, e.g. DescEmb-BERT-FT with both datasets, all five tasks and all four value modes. The scripts are part of our

repository. For the mentioned example see [here](#). To keep track of the experiments we wrote a `parse_experiments.py` which recursively parsed all runs in output folders and extracted the most important hyperparameters. It also added a `tag` column which helped us to quickly filter what kind of experiment each row corresponds to. This was particularly useful to manage contributions from a distributed team working asynchronously.

We did not train BERT-tiny from scratch, because it is very computationally expensive and the authors found that it did not help very much over fine-tuning.

## References

1. Kyunghoon Hur, Jiyoung Lee, Jungwoo Oh, Wesley Price, Young-Hak Kim, and Edward Choi. Uni- fying Heterogeneous Electronic Health Records Systems via Text-Based Code Embedding, March 2022, arXiv:2108.03625, doi: 10.48550/arXiv.2108.03625.
2. Pollard, Tom, et al. "eICU Collaborative Research Database" (version 2.0). PhysioNet (2019), <https://doi.org/10.13026/C2WM1R>.
3. Johnson, Alistair, et al. "MIMIC-III Clinical Database" (version 1.4). PhysioNet (2016), <https://doi.org/10.13026/C2XW26>.
4. Wornow, Michael, Yizhe Xu, Rahul Thapa, Birju Patel, Ethan Steinberg, Scott Fleming, Michael A. Pfeffer, Jason Fries, and Nigam H. Shah. "The Shaky Foundations of Clinical Foundation Models: A Survey of Large Language Models and Foundation Models for EMRs." March 2023, arXiv:2303.12961, doi: 10.48550/arXiv.2303.12961.
5. Gu, Yu, Robert Tinn, Hao Cheng, Michael Lucas, Naoto Usuyama, Xiaodong Liu, Tristan Naumann, Jianfeng Gao, and Hoifung Poon. "Domain-Specific Language Model Pretraining for Biomedical Natural Language Processing." January 2022, arXiv:2007.15779, doi: 10.1145/3458754.