

Master's Programme in Computer, Communication and Information Sciences

Contrastive Learning with Time-aware Transformer on EHR data

Linh Tran

© 2025

This work is licensed under a [Creative Commons](https://creativecommons.org/licenses/by-nc-sa/4.0/) “Attribution-NonCommercial-ShareAlike 4.0 International” license.



Author Linh Tran

Title Contrastive Learning with Time-aware Transformer on EHR data

Degree programme Computer, Communication and Information Sciences

Major Machine Learning, Data Science and Artificial Intelligence

Supervisor Prof. Pekka Marttinen

Advisors Prof. Risto Renkonen, Dr. Miika Koskinen

Collaborative partner Helsinki University in collaboration with University
Helsinki Hospital

Date 31 October 2025

Number of pages 71

Language English

Abstract

Patient representation learning has been a hot topic in the medical fields in recent years due to the fact that personal indexes of each person can be different from the population levels. Therefore, researchers have been trying contrastive learning using EHR data to achieve a personalized representation of each patient, resulting in more individual and specific recommendations.

Earlier contrastive learning work has made use of scan images or clinical notes during hospitalization to analyze the survival status or predict the upcoming events of patients. However, the impact of both categorical and numerical data in a multimodal setting has not yet been focused on, even though hospitals and clinical sites have a considerable amount of such information collected over years. Therefore, in this paper, we concentrate on contrastive learning using multimodal data, also including special types with hierarchical structure such as ICD-10 and medication codes.

We conduct exhaustive experiment with different types of data to see how each of them can be augmented for a contrastive learning framework. We also come up with a novel pretrained model schema for understanding the hierarchy of ICD-10 and medication codes, which is integrated into a unified multimodal model. For learning the temporal and semantic information of patients event sequences in a unified manner, we follow SimCLR framework with TAAT being the main encoder module and InfoNCE loss acting as the objective function.

The obtained results show that our pretrained models can perfectly understand the inherent hierarchy within ICD-10 and medication codes, and our augmentation methods work very well for both normal data types and special data types with hierarchical structure. The preliminary findings from the multimodal model suggest that it has learned meaningful latent representations for patients and can form clusters of the ones with similar features, and these representations can be utilized for downstream tasks such as prediction and classification. We believe this would contribute a further step toward the goal of personalized healthcare with artificial intelligence.

Keywords contrastive learning, BERT, time-aware transformer, EHR data, data augmentation, hierarchical structure

Preface

I want to thank my advisors Prof. Risto Renkonen and Dr. Miika Koskinen for their dedication, guidance, and advice throughout the whole time. I am grateful for having met such wonderful advisors, as well as for all the patience and help I have received.

I would not make it to this day without my research group, special thanks to all the members for the ideas, experiments, talks and, of course, hangouts.

I also want to thank my family and friends for all the emotional support and for being by my side when I struggle.

Last but not least, I would like to include myself here, thank you for not giving up.

Otaniemi, 31 October 2025

Linh Tran

Contents

Abstract	3
Preface	4
Contents	5
Symbols and abbreviations	7
1 Introduction	9
2 Background	11
2.1 Medical and EHR data	11
2.1.1 Laboratory tests	11
2.1.2 ICD-10 Codes	11
2.1.3 Procedure Codes	12
2.1.4 ATC Codes	12
2.2 Contrastive Learning	13
2.2.1 Data Augmentation	13
2.2.2 Contrastive Loss	14
2.3 Transformer	16
3 Literature Review	18
3.1 Related work on Time-series Transformer with EHR	18
3.2 Related work on Contrastive Learning with EHR data	19
4 Methodology	21
4.1 SimCLR	21
4.2 Time-aware Attention-based Transformer	23
4.3 ModernTAAT	25
4.4 Validation Metrics	28
5 Results	30
5.1 Data Preprocessing	30
5.2 Augmented Views	32
5.2.1 Event time	32
5.2.2 Event type	33
5.2.3 Event values	36
5.2.4 Event name	36
5.2.5 Encoded Augmentation	39
5.3 ICD-10 Codes Experiments	42
5.3.1 First Experiment of Hierarchical Model for ICD-10 Codes	42
5.3.2 Second Experiment of Hierarchical Model for ICD-10 Codes	43
5.3.3 Third Experiment of Hierarchical Model for ICD-10 Codes	44
5.4 Final Multimodal Model	44

5.4.1	Pretrain on ICD-10 Codes	47
5.4.2	Pretrain on ATC Codes	48
5.4.3	TAAT-based Encoder Module	49
5.5	Evaluation	53
5.5.1	ICD-10 Codes Embedding Vectors	53
5.5.2	Representation Clustering	59
6	Conclusion	66
6.1	Discussion	66
6.2	Limitations and Future Directions	66
7	References	68

Symbols and abbreviations

Symbols

$\{x_k\}_{k=1}^n$	a sequence of x_1, x_2, \dots, x_n
Δt	change in time
$[a, \dots, b]$	range from a to b (inclusive)
\in	belong to
x^T	transpose of vector/matrix x
$\ x\ $	norm of x
\mathcal{L}	loss function
n -D or n D	n -dimension
$\mathbb{R}^{(n \times d)}$	$(n \times d)$ -dimensional real vector space
W_x	projection matrix

Operators

\sqrt{x}	square root of x
\sum	sum over an index
$a \odot b$	element-wise product of a and b
$d(\cdot, \cdot)$	distance metrics
\exp	exponential function
$f(\cdot)$	encoding function
\log	logarithmic function
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution with mean μ and standard deviation σ
$\text{sim}(\cdot, \cdot)$	similarity score

Abbreviations

ATC	Anatomical Therapeutic Chemical (ATC) Classification
BERT	Bidirectional Encoder Representations from Transformers
BMI	Body Mass Index
CNN	Convolutional Neural Network
CVE	Continuous Value Embedding
EHR	Electrical Health Records
ICD-10	International Classification of Diseases, 10th Revision
InfoNCE	Information Noise Contrastive Estimation
GeGLU	Gated Linear Unit (GLU) with Generalized Linear Unit (GeLU) activation
GPU	Graphics Processing Unit
LSTM	Long Short-Term Memory
MHA	Multi Head Attention
MTAA	Multi-head Time-Aware Attention
(N)MI	(Normalized) Mutual Information
ReLU	Rectified Linear Unit
ROC-AUC	Receiver Operating Characteristic Area Under the Curve
RNN	Recurrent Neural Network
RQ	Research Question
SimCLR	Simple Framework for Contrastive Learning
STRaTS	Self-Supervised Transformer for Sparse and Irregularly Sampled Time-Series
TAA	Time-Aware Attention
TAAT	Time-Aware Attention-based Transformer
TRE	Time Relation Estimation
UMAP	Uniform Manifold Approximation and Projection (UMAP) for Dimension Reduction

1 Introduction

The human body is a complicated entity made up of organs, cells, and systems, which seamlessly cooperate with different components, to ensure that life is provided to us human and our body functions normally [1]. Considering how each human body is different from the other, there have been some concerns about this in the medical field related to the generalization problem. Human diseases are inherently as complex as our body [2], so interpolating between individuals is not reasonable. Each patient has a distinct disease trajectory or a pattern of how a disease develops within their body, and while some indices might seem normal to one, they could be fatal to others. Traditionally, researchers work on the population-level averages and make diagnosis or recommendations based on these numbers, resulting in overly wide ranges of ‘normal’ indices. However, a person whose measure indices lie totally within these ranges might still catch a disease or get infected simply because their personal ranges are much narrower compared to the population. And this is not a rare case, but a rather common situation due to the fact that our bodies are composed differently. Therefore, it is urgent to move the direction towards personalized studies for each individual [2]. Patients’ data have been collected for years and years with a detailed portrait of each profile, but few have been implemented to make use of that tremendous information. In recent years, with the development of artificial intelligence and machine learning, researchers have started to apply them to medical usage. Several attempts, including DoctorAI [3], Med2Vec [4], BEHRT [5], have been made to achieve personalized health analysis using machine learning. They take advantage of scan images, laboratory tests, along with notes or information during hospitalization to obtain a distinct profile of a patient. To further analyze the longitudinal relationship between medical events over a long time period, we introduce a method based on a time-aware transformer combined with contrastive learning. Transformer, introduced in 2017 by [6], is popular for its ability to learn the positional correlation between inputs, and instead of position, we use timestamps in this model. The data input going through this turns into a unique representation of that patient. And by utilizing contrastive learning, we achieve a state where similar patient trajectories would result in similar representation and stay close in the latent space.

In this paper, our aim is to show how contrastive learning using longitudinal patient profiles can be conducted. This can further be elaborated to achieve a personalized representation, which can be adapted to more specific downstream tasks such as survival analysis and time-to-event prediction. The current contrastive learning methods available mostly focus on clinical notes or scan images [7] [8], ignoring the great potential of numerical data such as laboratory tests and categorical data such as ICD-10 codes. There are also some work using imputation and interpolation for missing data [9] [10], and this can become critical depending on how personalized the data are. Moreover, timestamps are usually simply embedded so that each stamp corresponds to a separate embedding value, leading to the lack of correlation between the distance in timeline. Therefore, in this report, we introduce a novel way of integrating multimodal data (including numbers, categories, and even hierarchical structure into one single model) as well as making use of transformer to learn the

longitudinal effects of timestamps instead of treating them as simple indexes without the need of imputation. This includes how to obtain high-quality embedding vectors between hierarchical medical indicators such as ICD-10 codes and ATC (medication) codes in patients' event sequences, which have yet to be explored in earlier works. Furthermore, we explain how this unified model can work perfectly with a simple yet popular contrastive learning framework SimCLR [11] by showing the extensive data augmentation of each modality in the experiment.

With the combination of these methods, the ultimate goal is to explain how contrastive learning can be applied to Electrical Health Records (EHR) data, especially the longitudinal records of patients. This can be broken down into multiple research questions (RQs) that we aim to answer as follows:

- **RQ1:** How to perform augmentation with regard to multimodal data including different types such as numbers and categories?
- **RQ2:** How to train the model to understand the inherent hierarchical structure of ICD-10 codes and medication codes, as well as how to conduct augmentation on these hierarchical data?
- **RQ3:** Which model should be used as the encoder for sparse time series without imputation and interpolation to best capture the semantic and temporal information?
- **RQ4:** How are all these components (type-specific data augmentation, encoder, contrastive loss) integrated to produce a unified system for contrastive learning with EHR data?

Continuing from this introduction chapter, Chapter 2 goes through the background information about medical data and contrastive learning, along with transformer models and their improved versions. Chapter 3 explores existing work on time-aware transformer models and current contrastive learning models using EHR data. Following is Chapter 4, where we discuss the main methodologies and detailed structure of our model, as well as its development history. Then, Chapter 5 explains how the data is processed to pass through the models and evaluates the obtained results of our models. Finally, Chapter 6 concludes the findings of this paper and suggests possible future directions for further implementation.

2 Background

This chapter discusses the general level of information that the report revolves around, including the basics of EHR data, the contrastive learning concept together with its components, and a brief on the transformer model that has been increasingly applied in medical field during the last few years.

2.1 Medical and EHR data

Electrical Health Records [12], or EHR, is the systematic collection of patients information over time stored digitally. This information may be shared across clinical sites for a comprehensive and well-documented view of the profiles. EHR contains a wide variety of data, including demographics information, patients' sickness records, and different types of tests. In this paper, we focus on the elaboration of laboratory tests, ICD-10 codes, procedure codes, and ATC codes (or medication codes).

2.1.1 Laboratory tests

Laboratory tests are one of the methods to assess the current situation of one's body, as well as to monitor the effects of any treatment used on the patients. Some of the most common tests are blood tests and urine tests. The results obtained from these might indicate whether the patient's index is falling within a normal range (based on the population), or even within their own normal range (even when this might be different from the population level). There are two types of lab results: qualitative and quantitative. Quantitative results give the exact numerical value of a test, indicating the corresponding index of the patient and whether that number is normal or abnormal. Qualitative results, on the other hand, tell us about the presence of a substance in the test samples; therefore, the output will mostly be either positive or negative. For example, blood test is the most common quantitative test, which is conducted to check the number of blood cells and measure different types of blood chemistries. These indexes can detect a variety of issues, including infections, cancer, and heart disease. For qualitative tests, a common one is the pregnancy test, whose result is either pregnant or not. Given how much information they offer, laboratory tests are an essential process that allows both doctors and patients to prevent possible diseases, diagnose the current condition, and monitor other related treatments.

2.1.2 ICD-10 Codes

International Classification of Diseases, Tenth Revision [13], also referred to as ICD-10, is the internationally standardized classification of medical conditions published by the World Health Organization. These conditions are expressed as codes, and there are codes for symptoms, diseases, and causes of abnormal situations. Within the base classification only, there are more than 14000 codes to specify for the details of the diagnosis. ICD-10 codes have a structured format for neatly displaying information with 3-7 characters. They have layers of information, and the longer the code is,

the more information it contains. This serves different purposes whether we want to have a rough estimate or a detailed report of the severity, place, and manifestation of the diseases. The codes always start with an alphabetical character followed by a numerical one. The third character can be either a number or an alpha; however, within the scope of this report, all the third characters are numerical. These three first characters determine the category of the diagnosis. After that, there must be a decimal before we get to the next part of the code. The fourth to sixth characters tell us about the etiology, including the causes of the diseases and other vital clinical notes. The last character is called the extension, which provides details about the encounter of the disease. One code needs not to have all these characters, but rather we read the information based on how many characters there are within the code, from general to specific. Due to the nature of ICD-10 codes, they possess the inherent hierarchy within the code. That is, if the two codes share some first characters, the longer one belongs to the shorter one, similar to a tree structure. This can be further exploited in machine learning to study the correlation between diagnoses and patients' profiles.

2.1.3 Procedure Codes

Procedure codes, as the name suggests, include the information of the medical treatments performed to a patient profile. These codes are in use universally and serve as a means of communication between different healthcare and medical parties. Each code describes the medical and surgical services that the patient has undergone. Similarly to ICD-10 codes, procedure codes also determine which events occurred in the code itself. However, compared to the systematic structure and levels of details in ICD-10 codes, procedure codes are expressive as themselves. That means they do not have the same hierarchical structure as the way ICD-10 codes are designed. While ICD-10 codes tell us the causes of a disease, procedure codes tell us the corresponding actions taken to deal with the disease.

2.1.4 ATC Codes

Anatomical Therapeutic Chemical (ATC) Classification System [14] is the system that organizes the drugs' active ingredients according to their corresponding organs in the body as well as the chemical and pharmaceutical features. This organization is expressed as a code that we have been referring to as medication codes. ATC codes resemble ICD-10 codes in the sense that they are both strictly hierarchical. Each character in the code expresses a level of information, and the next character elaborates on top of that. ATC codes are divided into 5 levels with a further description as in Table 1.

By the structure of the codes, we can already tell the first three levels construct a hierarchical graph with the more general being the parent and the more specific being the child. These groups explain the details of the organ on which the drug is supposed to be used. The last two levels, on the other hand, are more of a self-expressive code themselves, which tells us about the chemical substances included in the drug. By making use of this systematic drug classification, we can conduct analysis of drugs

Level	Characters	Content
First	1 alphabetical character	Anatomical main group, indicating which organ or system of the body the drug is targeted to
Second	2 numerical characters	Therapeutic subgroup with hierarchy (codes starting with 0_ should all belong to group 0)
Third	1 alphabetical character	Detailed therapeutic and pharmacological subgroup
Fourth	1 alphabetical character	Chemical subgroup
Fifth	2 numerical characters	Active ingredients

Table 1: ATC classification system levels

patterns and, even further, the interaction between drugs and other medical events within one’s EHR profile.

2.2 Contrastive Learning

Contrastive learning, whose original idea about grouping similar inputs and magnifying the difference dates back to 2006 in [15], is an emerging machine learning method that has been utilized in multiple fields, especially medical one, within the last 5 years. The core idea of this technique is to treat each input in the dataset as an instance, so if there are multiple inputs, they should be correlated as being in the same set but at the same time different from each other because they are distinct subjects. Contrastive learning focuses on this goal and tries to maximize the distinction between each sample, or contrastivity between them. Based on the contrastivity, there exist positive and negative counterparts to an input, with the positive ones being more similar than the negative ones. To achieve this, there needs to be a medium to infer the similarity between different data samples. Therefore, the two most crucial elements contributing to this goal are data augmentation and contrastive loss - the former injects noise to the sample and the latter learns to recognize that noisy version is similar to the original sample.

2.2.1 Data Augmentation

Data augmentation [16] is the grounding factor in contrastive learning, featuring the noise injection to data inputs for more enriched versions. The most straightforward way to perform data augmentation is to create a slightly different version of the input. The reason why this works is that it produces an almost the same version as the

original one, but with small room for noise. This space allows the contrastive model to learn the similarity between different samples because technically these augmented versions are counted to be the same as the original. So, if there exists any sample that resembles the augmented version, the model knows that this sample and the original version of the augmented one are a positive pair. Based on the types of data, there are several ways to augment it [17], the most basic of which are described shortly as follows. Pixel erasing is an image augmentation where we randomly mask out some pixels in the original image. Another image method is photometric transformation, in which we change the total color and style of the image. And this could even lead to a completely different input in machine language, but similar objects to our eyes. Moving forward to text data type, there are several methods to perform related to tokens (either words or sentences in simple meaning). Tokens replacement is the random substitution of words with alternatives in a sentence, leading to generally similar and slightly distinct versions. Tokens adding is simple as the name suggests, where we add synonyms or punctuations to introduce a bit more complex sentences while maintaining the original meaning. Token deletion is to delete some random words within the sentence. Similarly, numerical values have their own augmentation strategies. The most common is to inject random noise, such as Gaussian, into the data, transforming the original values according to the distribution of the added noise. Scaling is almost the same as adding, but we perform multiplication instead of addition in order to scale up or down the numbers. In addition, we can also use the shifting method, for example, by adding a static number to all the data samples. This is more related to sequences where the order of events matters. Besides these basic data types and augmentation techniques, there are other complicated and customized methods which might be more effective for some specific cases.

2.2.2 Contrastive Loss

For the model to learn the similarity and differences between samples, a suitable loss is inevitable. Depending on the problem that is being solved, there are multiple contrastive losses that have been the primary foundation for further explored and tailored loss functions. To start with, a pair loss [15] is the most simple and straightforward loss function.

$$\mathcal{L}_{pair} = y_{ij}d(f(x_i), f(x_j))^2 + (\mathbb{1} - y_{ij}) \max(0, m - d(f(x_i), f(x_j)))^2$$

where (x_i, x_j) is a pair of samples input, $f(\cdot)$ is an encoder function, $d(\cdot, \cdot)$ is a distance metric to measure how far the two inputs are, $y_{ij} = \{0, 1\}$ indicating whether they are a positive or negative pair, and m is the margin, or the minimum acceptable distance to be consider as negative. In this setting, the processed input (rather than the ‘raw’ input that will be augmented) includes a pair of data sample: one is the original data, and the other is either the positive counterpart (one similar sample or the augmented version) or the negative counterpart (one different sample). The loss will set the positive to be close and the negative to be far from the original one. Each pair of inputs will be handled one at a time, so it is useful when we need to compare just two samples together like FaceID verification. However, this feature also limits

the capability of this loss function. We need to carefully handle the data in pair and design what is the optimal choices for a point being positive/negative, let alone that it is very inefficient if we have to compare just a single pair at a time when it comes to large datasets.

The next version of the loss is the triplet loss [18], where the data input is a tuple of three elements: the original data, the positive counterpart, and the negative counterpart.

$$\mathcal{L}_{triplet} = \max(0, d(f(x_i), f(x_i^+)) - d(f(x_i), f(x_i^-)) + m)$$

where x is the anchor input that is to be compared to, x_i^+ is the positive counterpart and x_i^- is the negative counterpart, $f(\cdot)$, $d(\cdot, \cdot)$, and m are similar to \mathcal{L}_{pair} . In this version, there is more information introduced to the model, letting it know the relative distance between a similar element and a different element. The similar ones will be closer to the original than the negative ones. And there is a different magnitude of distance: the hard negative (very different) should be further than the negative, and this applies to similar elements as well. This is better as the model is now able to grasp the understanding of a positive element and negative element at the same time. However, for this loss to work, there must be a difference between levels of negative elements, otherwise it would be the same as the pair loss. And this triplet loss also requires careful data annotation and selection as it processes a single triplet at a time. A similar loss function to this is N-pair loss [19], which is basically an upgraded version of triplet loss.

$$\mathcal{L}_{N-pair} = \frac{1}{N} \sum_{i=1}^N \log \left(1 + \sum_{j \neq i} \exp(\text{sim}(f(x_i), f(x_j)) - \text{sim}(f(x_i), f(x_i))) \right)$$

where $\text{sim}(\cdot, \cdot)$ is the similarity function to help define the loss. This allows the model to learn one positive pair against multiple negative pairs simultaneously, speeding up the time needed to process the data and signifying the slight distinction between the original input and those negative counterparts. However, the problem still remains as this requires thoughtful and informative design of pairs. All of these losses have some extent of demands for the data to include at least one positive counterpart of the input to work.

Then, InfoNCE loss [20] was introduced and quickly became popular among contrastive learning models due to its flexibility and effectiveness. For InfoNCE loss, in a batch (subset of all data) of inputs, only the input and its augmented version are considered to be a positive pair, and the remaining datapoints in the batch will be the negative counterparts.

$$\mathcal{L}_{InfoNCE} = -\frac{1}{N} \sum_{i=1}^N \log \frac{\exp(\text{sim}(f(x_i), f(x_i^+))/\tau)}{\sum_{j=1}^N \exp(\text{sim}(f(x_i), f(x_j^{(+)})/\tau)}$$

The idea is simple but useful in a way that the model learns to focus on the positive pair; but at the same time, due to the noise introduced by augmentation, it learns more about the relative correlation between the (augmented) input and the other ones in

the batch. Since it is most likely that there will not exist two completely matched data points when it comes to a real world dataset, comparing the original data and the remaining will not give us the best performance. However, if there exists one sample that is similar to the augmented version, it will automatically be pulled closer to the original point than the one that is distinct from the augmented version. Furthermore, one considerable improvement of this loss is that we do not need to manually assign a positive/negative tag to every point according to each input, and that will significantly enhance the effectiveness as well as decrease the laborious work needed when using the previous loss functions.

2.3 Transformer

Transformer [6] is one of the groundbreaking machine learning methods that was introduced in 2017, especially when working with sequential data such as text and time series. Before that, the most common way is to use recurrent [21] and convolutional neural network [22] to exhaustively study the temporal relationship between each element. Transformer presents a new mechanism called self-attention for sequence modeling. The idea is that each element can look directly at other elements to study their relationship, regardless of their distance. This cannot be achieved by recurrent network due to its diminishing dependencies over a long time. Transformer can also be conducted in parallel and perform multiple attention mechanism at the same time, allowing the model to discover different patterns among the elements. This novel invention has laid the ground for further experiments and research, resulting in a variety of transformer-based models.

Among the models that base their modules on transformer, the most popular ones are Bidirectional Encoder Representations from Transformers (BERT) [23], Generative Pretrained Transformer (GPT) [24], and Visual Transformer (ViT) [25]. BERT was originally made for predicting if a sentence comes next given a context in advance by randomly masking out some words during the training process, so that the model learns to predict them. GPT is still in continuous development up until now and is the first transformer to come into mainstream use (OpenAI nowadays). This model family is famous for the ability to complete sentences, write a story, and even reason with users. ViT is applied mostly in computer vision to leverage the traditional models by integrating transformer. It is still in use for common tasks such as image classification, detection, and segmentation.

To further improve the basic BERT model in order to achieve faster and better performance, HuggingFace introduced ModernBERT [26] with some fundamental changes related to the architecture and efficiency design. Regarding the architecture, the authors did not use bias terms to cut down on the resources spent on unnecessary calculations, and they also used GeGLU instead of ReLU activation function (default option) as this has been showing consistently better performance in [27]. Considering the efficiency, the paper introduced the usage of FlashAttention [28] and unpadding sequences [29]. The former is a clever way to perform calculations on hardware. Instead of the normal way to set up full-size matrices where untouched values might take up hardware spaces and memories, FlashAttention only initiates the spaces for

those that are being calculated, ignoring the ones to come. This efficiently reduces the waste of resources on inactive calculations. The latter technique also contributes to reducing the memory taken up by redundant padded tokens. In the setting of transformer, it is essential to have equal-size inputs (in a batch), and to do so, we need to pad all the inputs with dummy tokens to match with the longest sequences. The wasted resources on calculating those tokens can be massive when the length of inputs varies substantially. This technique will cut down all those padded tokens and concatenate the remaining elements from all the inputs into one big component. Now, a batch of multiple padded inputs has become a batch of one unpadded input. Of course, some modifications to the structure of the model are required to accommodate this change.

3 Literature Review

This chapter highlights several recent papers on the main topics of this report including time-series transformer and contrastive learning on EHR data. Each of these topics is discussed in a smaller section to learn about the state-of-the-art methods.

3.1 Related work on Time-series Transformer with EHR

STRaTS [30] introduced in 2022 is a transformer-based model aiming at learning the temporal information of patients' series with the novelty of treating time series as a set of triplets (time, event, value) instead of a conventional dense matrix. With regular time series where the data are sampled with a fixed step size, this matrix is the observations formatted into a 2D matrix with rows being the timestamps and columns being the features. This is a neat way to present the features along with time and is compatible with many machine learning architectures. However, the approach is only ideal when the data is regularly sampled and does not contain missing values; otherwise, those missing ones will need to be imputed or interpolated from neighboring observations. Imputation/Interpolation is usually artificial, especially in the field of medical data. This means that the data generated does not belong to any actual observation, and by introducing them, we are including more false information to our model, making it even harder to learn the individual representation of each patient profile. Real-world data cannot be easily implied due to the variety of features between individual. As said, a human body is constructed by innumerable number of components, and there is no one that is exactly the same as another. Using one's data to imply others' is not the good choice in learning the patient profile. Therefore, in the STRaTS implementation, the time series is represented as a set of triplets

$$(t, f, v)$$

where t is the recorded timestamp, f is the feature and v is the value of that feature. By presenting time series this way, there is no need for neatly formatted tables, equal to no imputation or interpolation. Another novelty introduced in this paper is the embedding of continuous values. Normally, when we want to learn something from a corpus of text using embedding layers, we need to create a lookup table for each word present. However, that method is feasible because words are discrete and countable. For continuous values and numbers, discretization does not work. That is why the authors create and use Continuous Value Embedding (CVE) to embed time (how much time has passed since the first timestamp) and variable numerical values (corresponding to the features). Each component inside the triplet has its own embedding layer to be transformed into a higher dimension for rich and informative representation that will be passed through a transformer. Overall, this model has achieved visibly better results on both MIMIC-III [31] and PhysioNet-2012 [32] datasets than other models at that time, with its ROC-AUC score of 0.891 for the former and 0.839 for the latter when training for the downstream task of mortality prediction.

[10] is another transformer-based model compatible with time-series EHR data. The

goal of this model is to predict early sepsis based on the records of time-series data. Instead of learning the simple lookup table embeddings of time and features like STRaTS, this model implements a neural network such as convolutional neural network (CNN) [22] and long short-term memory (LSTM) [33] to learn the rich and diverse latent representation of input features and passes the output from those layers to a transformer. The authors claim that this allows the model to capture the complex patterns within the data. The reason behind this architecture is that LSTM is a form of recurrent neural network (RNN) [21], allowing it to recognize the hierarchical structure and cyclic dependencies within temporal features. Hence, instead of embedding time and other data separately, this LSTM method can integrate time information within its learning process. LSTM excels at studying sequential data due to its nature of receiving information from the previous steps, and the more steps there are, the better LSTM is. In the results section, the authors show that by joining LSTM and transformer, the achieved performance is better than other single models such as RNN and LSTM only. Additionally, when the window time span is increased, LSTM-Transformer performs noticeably better when the predictions are closer to the true labels of patients. One thing that could be an improvement for the next iteration of the model is that now the data are preprocessed to be regularly sampled. This is because the inputs for LSTM would be best if regularly spaced so that the model can learn the hidden patterns behind all the numbers. Hence, for those missing observations, the authors implemented data imputation by filling the missing ones from the previous values. This would naturally go against our goal of learning a distinct representation for each patient trajectory. Nevertheless, in the scope of early sepsis prediction, the model has performed very well with the imputed and regularly sampled data, more than other methods at the time.

3.2 Related work on Contrastive Learning with EHR data

In previous years, contrastive learning had received attention from medical fields due to its ability to distinguish different profiles given the trajectories. An important feature of this contrastive learning framework is the contrastive loss, which must be carefully chosen to discover the contrastivity between each input. This loss function is designed to create positive and negative pairs of inputs for the model to learn the difference. However, this loss usually does not go further than helping the model to separate the inputs. Therefore, in order to extend the original contrastive learning to clinical prediction problems, in 2021, SCEHR [34] introduced a new loss called general supervised contrastive loss, which combines two losses: contrastive cross-entropy loss and supervised contrastive regularizer loss. The former serves the same purpose as the contrastive loss for binary classification, where the model tries to detect which samples are positive or negative compared to the inputs. The latter focuses on the multi-label classification based on the labels of each input. Then within one loss function, the model is able to learn two objectives: to contrast against positive/negative pairs and to contrast against the labels. By integrating the two objectives, the model can perform the clinical prediction in a unified manner without having to pretrain a contrastive model then fine-tune on the classification task. The authors have stated that by

using their loss function in combination with an encoder for input data representation generation, the model can outperform the baselines and even state-of-the-art models on clinical risk prediction tasks. Furthermore, this loss works extremely well with imbalanced datasets, whereas real-world datasets are not always perfectly balanced. Hence, it can even replace the cross-entropy loss for binary or multi-label classification when training a model according to the authors.

Multimodal clinical data are valuable due to its rich and informative statistics of patients; but at the same time, it is also difficult to be effectively integrated into model training because of the high dimensionality, sparsity, and multiple types of data present in the same profile. [35] leverages the contrastive learning to multimodal data including electrical health records as well as clinical notes during the stay of patients. The novelty of the model is how it handles temporal information across multimodal data using cross-attention transformer and dynamic embedding, so that the sparsity and irregularities of timestamps can be handled and do not require imputation or interpolation. To further study the relationship between medical time series and clinical notes during the stay, a global contrastive learning loss is introduced to guide the model toward the positive pairs and away from negative pairs based on the discharge summaries. This integration of the loss can lead the model to learn from inputs such as laboratory tests and vital signs; at the same time, it also takes into account the similarities between patients using clinical notes and discharge summaries. The text input (including notes and summaries) is handled by a pretrained large language model to extract useful information in a machine-compatible format. This whole framework has proved to exceed other state-of-the-art models in the task of occurrence prediction using real-world datasets collected from the UF health system [36].

A recent published work in the medical field that makes use of contrastive learning is [37]. In recent years, the problem of automatic medical code assignment has been a hot topic to work on due to the benefits it brings back. This automatic assignment can reduce the laborious work of professionals and speed up the general process of diagnosis and corresponding medical responses. Although this paper does not work with time-series data, it shows a new aspect of implementing contrastive learning with large language models. The proposed framework revolves around the use of language models. GPT-4 [38] is used to generate patients' description based on the ICD codes. The use of contrastive learning shows an advantage here when it injects a variety of synonyms into the original code description to create a positive augmentation for the model to learn the similarities between lines of text. Furthermore, the authors also use a pretrained language model to process the text (clinical notes) as input and then extract its latent representations. Then the label learned by contrastive learning contributes to the label attention stage, where the model studies the relationship between the notes and the codes, so that it can identify which code should be assigned to an input clinical note. This framework has successfully handled the long wall of text when it comes to clinical notes. It has also mitigated the problem of label scarcity within ICD codes, since the majority of diagnosis only focus on some specific codes, while there are thousands of other codes as well.

4 Methodology

This chapter explains the ground for our model in more detail, including SimCLR framework [11] for contrastive learning and Time-aware Attention-based Transformer (TAAT) [39] as the foundation for the encoder within SimCLR. In addition, we elaborate on the metrics that can be used to validate the results to make sure that our research goals are achieved.

4.1 SimCLR

We follow SimCLR [11] for contrastive learning framework where the model is trained to distinguish between similar patient profiles and different ones. Contrastive learning

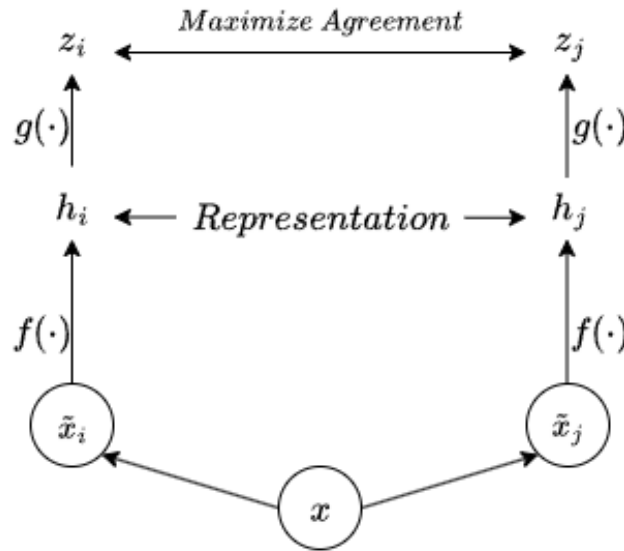


Figure 1: SimCLR Framework adapted from [11]

following the SimCLR framework means that we first need to create a view of our original input, also known as an augmented or noisy version of it. This augmented view will become our pseudo-label, where we want the model to recognize that the original data and this noisy version are from the same patient. By adding noise to the data, we expect that the model is able to recognize and group more similar patients who do not have exact trajectory but same patterns. The goal of this contrastive learning is to pull similar patients close together and push those who are not alike far away. According to SimCLR, the complete model includes an encoder where the data is represented in latent space and a projection layer that is later passed through the loss function for optimization. Given a batch sampled from the whole dataset $\{x_k\}_{k=1}^N$, two augmentation methods t and t' , a constant for temperature τ (smaller temperature means more separation but at the risk of unstable training, higher temperature means more clustering but at the risk of weak distinction), an encoder function $f(\cdot)$, and a

projection layer $g(\cdot)$, the general process for SimCLR framework is as follows:

$$\begin{aligned}\tilde{\mathbf{x}}_{2k-1} &= t(\mathbf{x}_k) \text{ and } \tilde{\mathbf{x}}_{2k} = t'(\mathbf{x}_k) \\ \mathbf{h}_{2k-1} &= f(\tilde{\mathbf{x}}_{2k-1}) \text{ and } \mathbf{h}_{2k} = f(\tilde{\mathbf{x}}_{2k}) \\ \mathbf{z}_{2k-1} &= g(\mathbf{h}_{2k-1}) \text{ and } \mathbf{z}_{2k} = g(\mathbf{h}_{2k})\end{aligned}$$

we obtain the pairwise similarity and the corresponding loss between these two views given $i, j \in \{1, \dots, 2N\}$:

$$\begin{aligned}s_{i,j} &= \mathbf{z}_i^T \mathbf{z}_j / (\|\mathbf{z}_i\| \|\mathbf{z}_j\|) \\ \ell(i, j) &= -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{k \neq i} \exp(s_{i,k}/\tau)}\end{aligned}$$

Our objective is to maximize the mutual information between these two augmented views, or minimize the InfoNCE loss [20] over the batch:

$$\mathcal{L} = \frac{1}{2N} \sum_{k=1}^N [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$$

The essence of this InfoNCE loss is that it will treat the two augmentations as their own positive counterparts, and all the other inputs in the same batch will be the negative ones. This loss function helps the model to pull these augmented views closer, while pushing the other inputs further away. In other words, this can be inferred as a multiclass classification loss, where one of the augmentations should be classified the same as the other, and the model will do its best not to categorize that augmented view to be the remaining inputs in the batch. For example, given an augmentation x'_n of x_n , the preferred output from InfoNCE loss would be similar to a probability distribution of all other x_i in the batch and their corresponding augmented views, with the highest chances being x_n , as visualized in Figure 2. To work along with

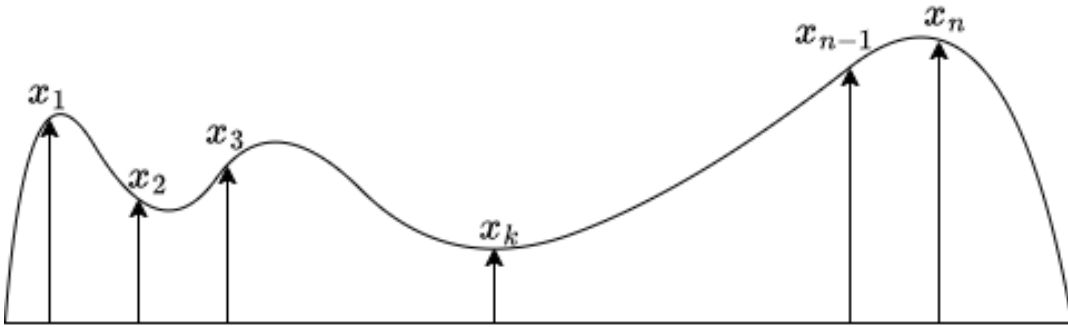


Figure 2: Probability of x'_n being similar to other inputs

this SimCLR framework, we build our model encoder based on TAAT, a time-aware transformer making use of multiple layers between time to learn the relationship between time and events, at the same time, it also considers the time sparsity and does

not perform interpolation. We do not want to create artificial data inferred from the population because it is one of our goal to obtain an individual-level representation and interpolation goes against the personalization.

4.2 Time-aware Attention-based Transformer

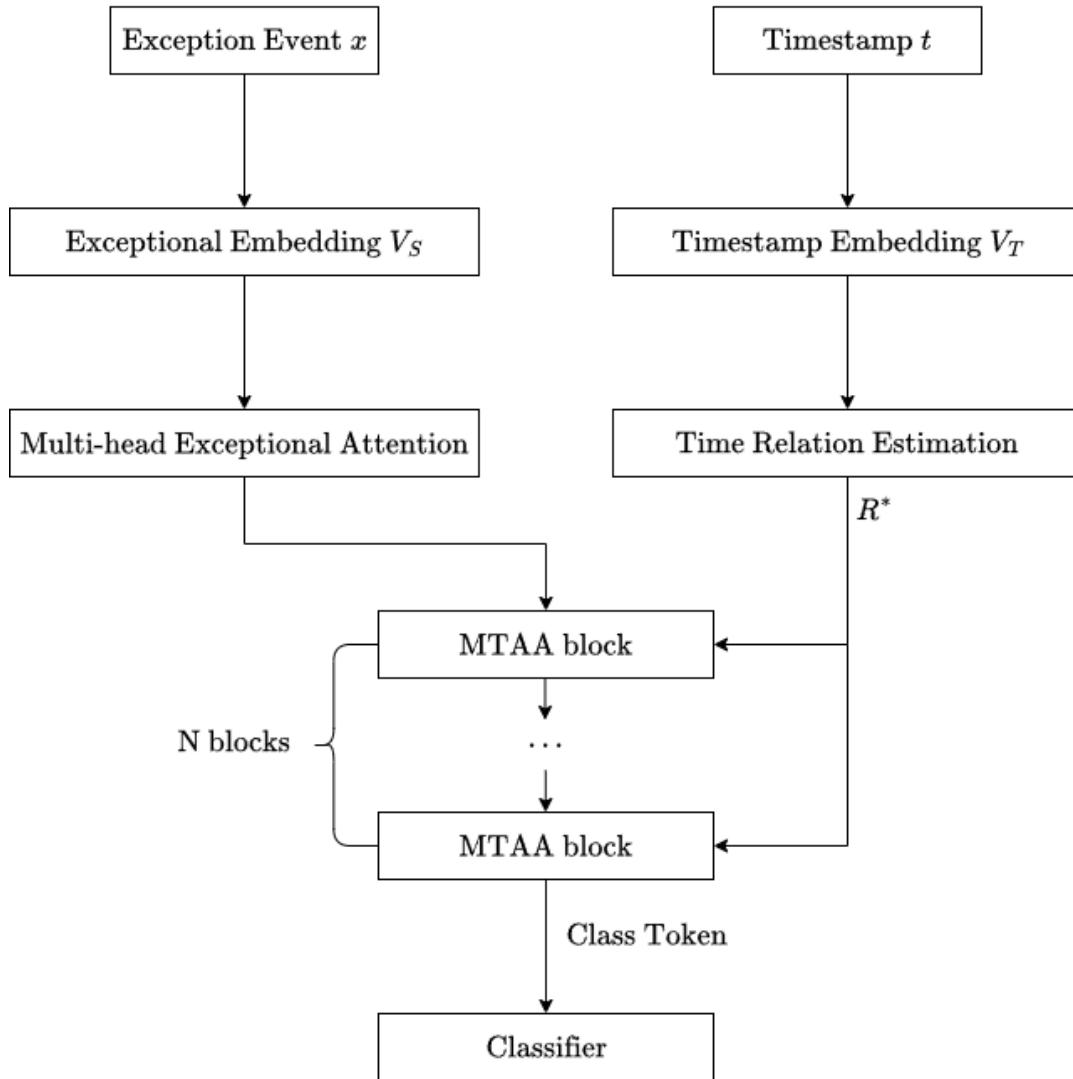


Figure 3: TAAT model adapted from [39]

Time-aware Attention-based Transformer [39], or TAAT, is not a direct EHR transformer model, but it shares many similar features. This model was created by the Alibaba group in the context where they usually have minor and major shutdowns of their hosted servers like websites or apps. The goal of TAAT is to predict the next shutdown of the servers based on collective error logs with time information and to indicate whether this shutdown is fatal or not. Therefore, instead of time and clinical features, they have time and error logs, which are also grouped together and converted

to be machine readable. The novelty of TAAT compared to other previous papers is how it handles timestamps in the time series. If we have the time to be embedded as continuous values in STRaTS [30] and added to other features embeddings, then in TAAT, the time is divided into pyramid levels of time - Day, Hour, Minute and Second. Each of these layers has its own embedding that can be learned throughout the process as presented in Figure 4 with a slight modification to start from Year-level. The parameter ϕ controls how many levels the time is decomposed into. For example, in this case it is 4, so there are four levels: Year - Month - Day - Hour.

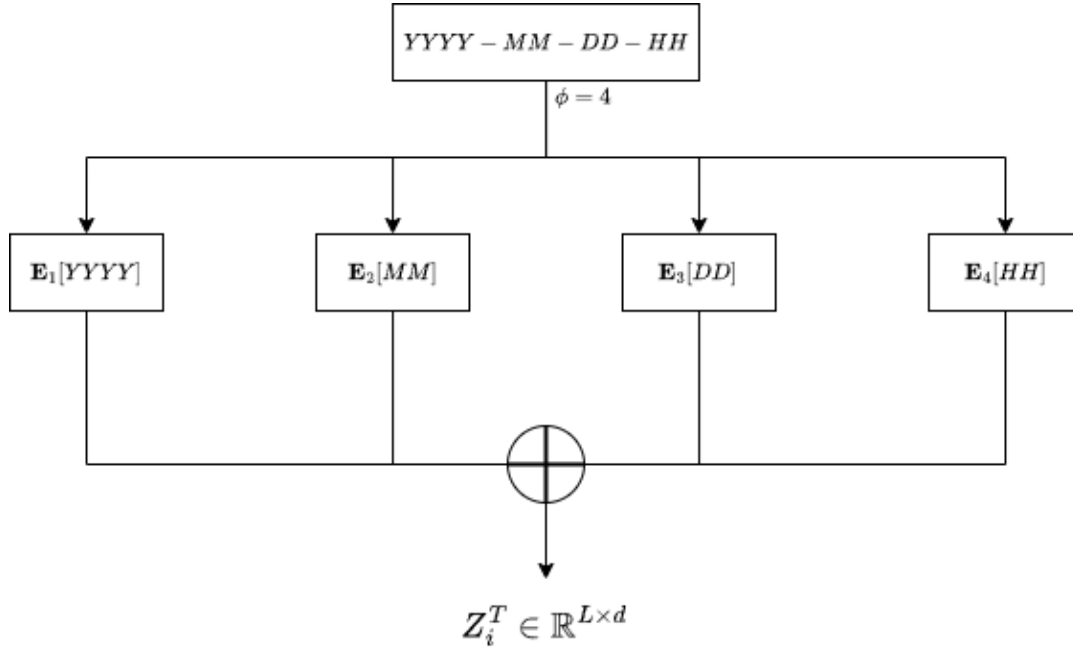


Figure 4: Example of Timestamp Embedding with $\phi = 4$

The multiple layers of time are then used to calculate the bias score called Time Relation Estimation (TRE) with regard to the temporal information. This bias score is later added to the attention score in the transformer module, together with all other input features embedding vectors, to achieve the semantic and temporal relationship between events. The decomposition of time allows the model to look at the whole sequence without too much sparsity, which normally appears in an EHR time series. This temporal score, along with attention score in transformer module, has enabled the model to look at both the relation between error logs considering the error type and time. This TAAT model outperforms other baseline models such as traditional supervised methods, deep learning models (CNN, RNN) and early transformer-based models (Transformer, BERT) when trained for the failure prediction task. With some modifications, this can be applied to EHR data where, instead of error logs, we have clinical events together with a separate embedding such as Continuous Value Embedding (CVE) from [30] for event values. In our work, the TAAT-based model works as an encoder in the SimCLR framework for the exploration of information from patients data. The adaptation of this model also answers to our **RQ3** about which

model to use for the EHR time-series data so that we can capture both semantic and temporal information without imputation or interpolation.

4.3 ModernTAAT

Beside the modifications of TAAT to match with medical setting, we also implement our model in the same way as ModernBERT [26]. To be more precise, we make the following modifications as briefly discussed in the introduction of ModernBERT. Instead of using ReLU as the activation function like the majority of transformers, we use GeGLU as it has been increasingly utilized more due to the better performance. The formula for ReLU given input x is:

$$\text{ReLU}(x) = \max(0, x)$$

This gives us a very simple and effective activation function, which is suitable for positive inputs so that they can be computed without vanishing gradient (the problem where the gradient fed back to neural networks becomes so small that the model cannot learn anything). However, if there are many negative inputs, this would result in a lot of ‘dead neurons’, which means we only have 0 in our inputs. On the other hand, GeGLU is a more complicated activation function but it handles the inputs smoothly to avoid both the vanishing gradient and the ‘dead neurons’. It is a combination of a linear transformation and a gate unit (which produces a gating signal whose value is between 0 and 1 without the need for the inputs to be positive). Given an input x , we have:

$$\begin{aligned} u &= xW_u \\ v &= xW_v \end{aligned}$$

where W_u and W_v are learnable projection matrices. Together, we obtain:

$$\text{GeGLU}(x) = u \odot g = (xW_u) \odot \text{GELU}(xW_v)$$

With this GeGLU activation function, we have a smoother gradient flow without nullifying any negative inputs, and due to the learnable matrices, the model can better select features information.

FlashAttention is implemented to boost the speed and reduce the burden for the computing machine. In standard attention calculation:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

the QK^T matrix is fully initialized as $n \times n$, and this would take up a lot of memory during calculation. Especially when dealing with long sequences, this quadratic multiplication can take up a considerable amount of spaces, let alone the memory needed for the reading and writing of this matrix. Instead of doing the whole matrix at once, FlashAttention [28] proceeds with a smaller block at a time that fits perfectly in fast GPU memory. The big computation is now divided into smaller parts, and

each of these parts is processed and stored individually, so that we do not calculate or store the whole $n \times n$ matrix at any time. This idea speeds up the computing process significantly, especially when it comes to longer inputs.

Unpadding [29] allows us to include more patients into a batch and decrease the computation waste on padding tokens in short trajectories. For contrastive learning, the more inputs there are in a batch (a batch refers to a subset of the whole dataset which the model is currently trained on), the better the model learns. The reason behind this is that there is no label in contrastive learning and the model has to base solely on one input against the remaining in the batch. If we have enriched data with a variety of profiles, the model is exposed to more trajectories and then is able to tell why each of them is not like the others. Therefore, it is crucial to have a large batch size so that we can pass as many patients to the model at the same time as possible. Initially, we follow the unpadding logic of concatenating all patients (in a batch) into one, meaning that we only have one record for that whole batch. The goal is to reduce the waste of padding tokens, as there might exist some patients with extremely short or long length of events. However, this did not work out as well as we expected. The general process speeded up, but at the cost of a very small batch size that makes it impossible to learn from a diversity of patients. The reason is that due to the quadratic computation of TRE, if given an input which is too long, the square matrix will be massive and too big for just one computation. To break this down into numbers, let us assume that we have a batch size of 64 patients and each patient has 100 events. For the normal padding method, we expect a matrix of 100x100 to pass through TRE, which is easily handled by most GPUs. However, if all events are concatenated into one, we will have a pseudo-patient whose event length is $64 \times 100 = 6400$. And this will result in a square matrix of 6400x6400 in the computation step. How big this seems depends on the GPU, but even if we are using a GPU with an enormous computing power, this unpadding method will not surpass the padding one with regard to how many patients it can handle at a time. A normal GPU can handle 1024 patients per batch if padded, but only 64 for unpadding. Therefore, we came to another solution where we do not concatenate every patient into one, but rather group small ones while not exceeding the maximum sequence length of the original batch. This can be further visualized as the graph below.

As can be seen from the graph, we greedily group those small ones whose summed length is still within the maximum length together. While we still use padding tokens here, the number of them has been drastically decreased. For example, with a batch of 128 patients, the post-unpadded batch contains 66 records. Similarly, for a 256-patient batch, it returns 121 patients. This means that we are most likely able to fit twice the number of patients compared to the normal padding version. This is highly preferable as we would like to have as many patients as possible in one batch. Another note is that we only conduct unpadding during the main computation step, meaning that at the beginning and at the end, the data will still be padded and batched normally to accommodate the process. After going through the model and continuing to the pooling layer, the data is extracted using masked average pooling. This is almost the same as just taking the average of all events in a single trajectory, but now considering as well which one is actual and which one is padded. Then, we only take into

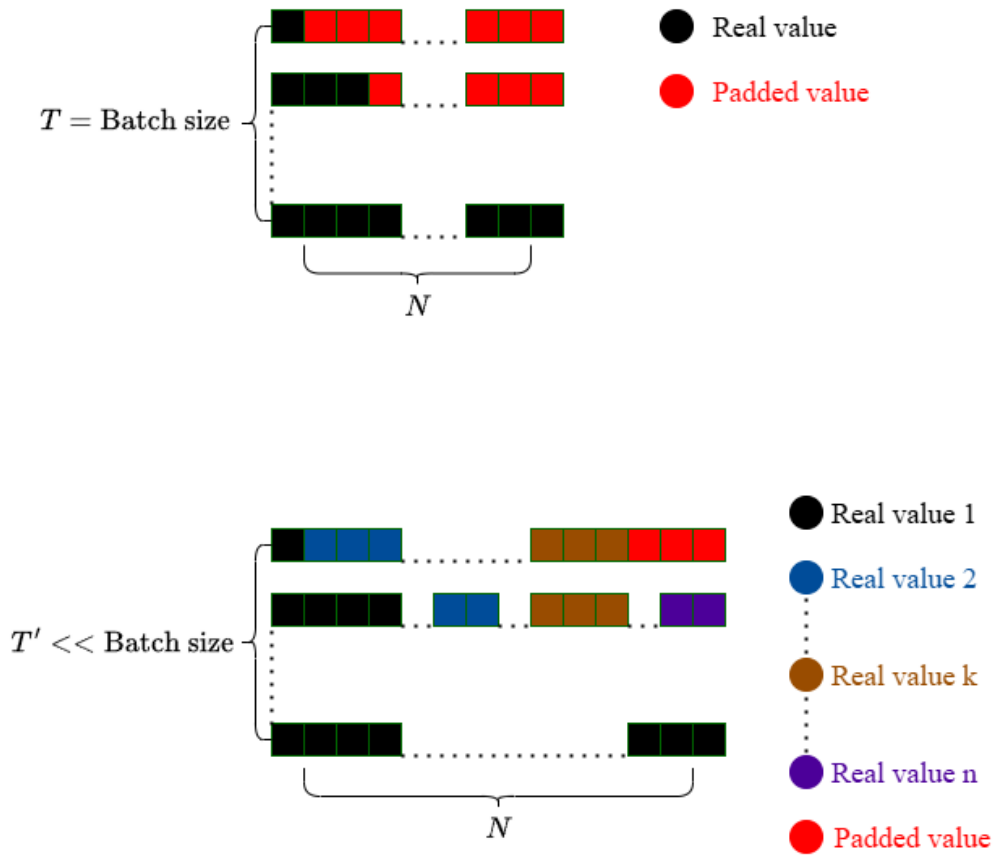


Figure 5: Group Unpadding

account those actual events and calculate the average of them to achieve a 1D vector representing the patient which captures the semantics of all events in that patient's trajectory. However, in the end, we have to resort to the normal padded version due to the limitation in computing and processing resources. We found out a way to make unpadding work, but it is now at the cost of long preprocessing time, especially given the length of input data. Hence, within the scope of this project, we only use the padded version to prove the validity of the model and its results, since this version can handle a large amount of data.

In addition, we also implement autocast [40] for 32-bit and 16-bit variables during the training process. This will also reduce the memory burden and increase the capacity of the machine. The model will decide itself where the variables can be cast into a smaller bit version, allowing more space for computation. This technique has shown that it can increase the capacity up to 2 times more. All of these modifications are to help the model have more data to analyze and learn the contrastivity between different profiles of patients.

A reasonable objective or loss function also plays an important role in guiding the model in which direction it should go. For this implementation, we make use of InfoNCE loss [20] - a simple but effective loss that matches perfectly with our goals. InfoNCE loss considers the original input sequence and its augmented view to be

similar (positive), while all the other sequences (in the same batch) are marked as different (negative). The core idea is that each patient is unique, and by using its own augmentation as the only positive view, we achieve the goal of learning a meaningful representation of the patient. The positive view, though resembles most of the details of the original, includes some noise in itself, allowing other patients whose trajectory is very close to stay near the original patient. Repeating the process will eventually lead to clusters of patients representation where those with comparable behavior are placed closely.

4.4 Validation Metrics

To be able to assess the correctness of some tasks such as ICD-10 codes hierarchy and similarity/distance between objects, we introduce the metrics that will be used in this report and explain the choices. The most commonly used is cosine similarity [41], with the formula

$$s_{a,b} = \frac{a^T b}{\|a\| \|b\|}$$

where a and b are vectors. This is a simple yet effective metrics for vectors comparison, as it cares more about the direction than the magnitude, so as long as the vectors are expressing the same type of objects (pointing towards the same direction), it will not matter if a is many times larger or smaller than b . This is very useful when it comes to checking the distance between the embedding vectors of ICD-10 codes, as it can easily tell us whether the learned embeddings correctly express the codes in the same group. Furthermore, this can be applied as clustering metrics, where we use this to measure the distance between the centroids of clusters to learn how far apart they are. We can use it to calculate the intra-cluster similarity, where we take the average of all the pairwise similarities within a cluster to see how well the clustering is. The same metric can be used for measuring the effect of data augmentation to see how similar the original and augmented views are. Another validation metrics used for hierarchical structure assessment is the Normalized Mutual Information (NMI) [42], which is used to measure the similarity between clusters. In our case, it would be the ground truth groups of codes with the learned embeddings of them. NMI is a variant of Mutual Information (MI), whose formula when given two variables X and Y is:

$$MI(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

where $H(\cdot)$ is the entropy and $H(\cdot|\cdot)$ is the conditional entropy. The concept of entropy in brief summary is the uncertainty of a random variable, for example, X ; so high entropy means unpredictable and low entropy means predictable. The conditional entropy measures the remaining uncertainty of a random variable X when we already know about another variable Y . Therefore, in the case where X and Y are independent, $H(X|Y) = H(X)$. Also, when X depends on Y completely, $H(X|Y) = 0$, which means there is no uncertainty about X when the information of Y is known. Figure 6 visualizes how these components of MI relate to each other. The normalized variant

of MI is NMI, whose formula is

$$NMI(X; Y) = \frac{2MI(X; Y)}{H(X) + H(Y)}$$

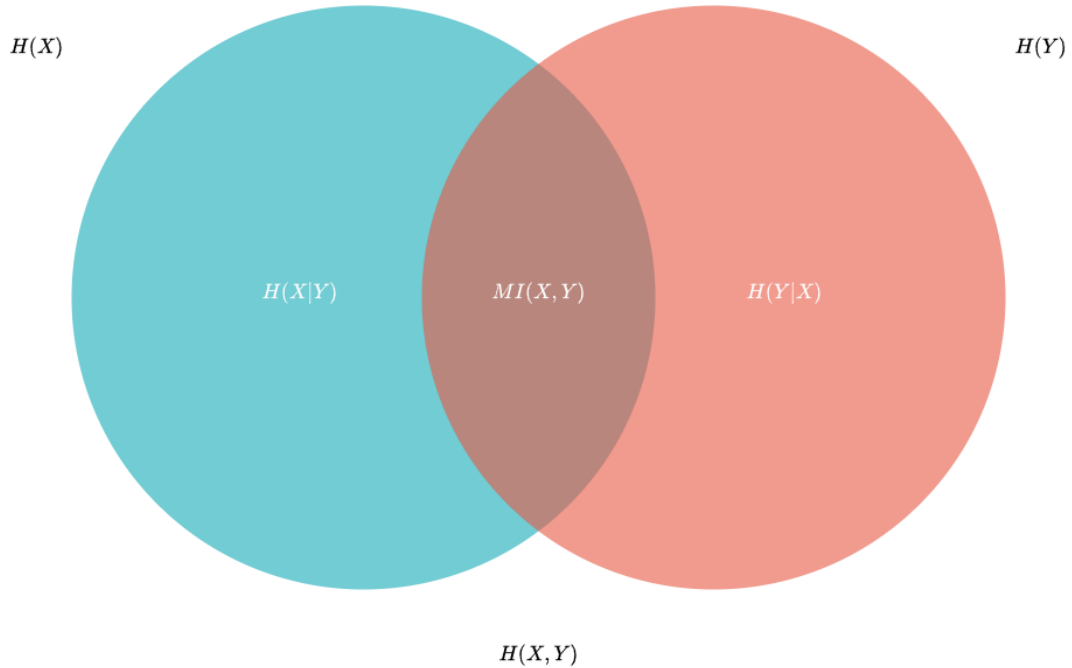


Figure 6: Mutual Information Venn diagram

In this version, the score is normalized to be within the range $[0, 1]$, making it easier to read the number and interpret the results. 0 means that there is no mutual information between the clusters, or that they are completely independent without any shared components. On the contrary, 1 means perfect agreement between them, which means they share the exact same components within clusters. Regarding this, our preferable results would be 1 so that the learned embeddings of ICD-10 codes are identical to the true hierarchy structure as explained previously. In addition to these two main validation methods, there will be some other metrics to evaluate the results obtained from our model.

5 Results

This chapter describes the implemented model for learning patient representation in details. It explicitly explains how the development process went and why the final version is chosen. Sub-modules or components of the overall model are also discussed. It describes the details of data, what it includes and how it is preprocessed for passing through the model. The two main results of the report, which are the ICD-10 codes embedding vector and patient latent representation, are evaluated and analyzed with more information.

5.1 Data Preprocessing

The statistics is provided by Findata [43], the Finnish Social and Health Data Permit Authority, with data request HUS/355/2025, and Findata was responsible for data integration and producing the anonymized statistics. The data are collected as trajectories of cancer patients between 1.1.2010 and 31.5.2019. For this project solely, we focus on the time span starting at the diagnosis point as starting time until 5 years later. The dataset includes a total of 100,000 records, 70% of which is for training, 15% for validation and the remaining 15% as a test set. Due to the privacy features of the data, it cannot be displayed at patient-level details, but rather more general.

Table 2 presents the description of each feature present in the dataset, with Dynamic being values subject to time and Static being values independent of time presented in. As introduced, the data contain different categories with multiple ranges; therefore, it is not possible to use it directly. Even within those numerical event values, there exists already a wide gap between events, let alone the existence of binary and trinary values. In order to solve this problem, the data has been scaled using Min-Max Scaling [44] to put everything into the same range of [0, 1]. To be more specific, each numerical event value is grouped by its event name, and then the whole event is scaled accordingly. For the category event value, it is straightforward after being grouped by event name: strings 'negative' and '-1' become 0, while strings 'positive' and '1' become 1 (with some special cases where there are more than just binary values). At this point, the data has been transformed into the same value scale and ready to be prepared as inputs for the model.

The computer itself cannot understand any meaning of these event features, so we have to change it into a machine readable format, numbers. The timestamps are transformed similarly to what was described previously, into four pyramid levels of time delta counting from the starting time. For numerical values such as age and event values, they are kept the same as this is already compatible with the machine. For simple categorical values, including lab tests, BMI measurements, procedure codes, and remaining demographics information, each of their unique categories is assigned an index until all categories have their own number. Using this way, we make sure that each category still remains as a separate factor while being comprehensible to the model. For special inputs with hierarchical structure such as ICD-10 codes and medication codes, in addition to indexing the code into numbers, we pretrain separate models that are specifically designed to learn the hierarchy within these codes, and

Feature	Type	Description
Timestamps	N/A	The main indication of time, with each timestamp being associated to an event of the patient.
Age at diagnosis	Static	Age of the patient at the first cancer diagnosis time. Though age can change over time, we only care about the demographics information in this number, and aging is natural, not a symptom or special event in cancer trajectory.
Gender	Static	Gender of the patient, being either Female or Male.
Cancer diagnosis	Static	The first cancer that the patient is diagnosed with.
Subgroup	Static	The group of body which the cancer is connected to.
Event type	Dynamic	All the types of events in the patient record, including 'Labs' (laboratory tests), 'Diagnosis' (ICD-10 codes diagnosis), 'BMI', 'Medication', and 'Procedure'.
Event name	Dynamic	The specific name of each event to provide further details and clarity.
Event value	Dynamic	The value associated with each event. It can have multiple ranges of values, depending on which event it is related to. For a test or dosage, this will be numerical value; for a diagnosis or use of medication, this will be mostly categorical - ('negative', 'positive'), ('-1', '1'), ('-1', '0', '1'). or some other special cases.

Table 2: Description of Patient Record's Features

extract the embedding vectors from them to process the codes in the integrated model. Another preparation step before working with the model is the augmented views of our data, and we cover also the augmentation methods for each type of inputs.

5.2 Augmented Views

The main framework of this project is about contrastive learning, specifically SimCLR [11], meaning that we need to prepare an augmentation for each input. As we know that the data has multimodal inputs, we cannot apply the same augmentation technique to all of them. Furthermore, since the static information does not change over time, we decide to keep it the same as this describes the demographics of the patient; hence, only the dynamic information is augmented to create a new view of the patient trajectory. There is no strict rule on how we should perform augmentation, nor is there any recommendation of the hierarchy augmentation. Therefore, we experiment with different methods of data augmentation on multiple features to explore the difference between the original and augmented views of the data. The metrics score used to evaluate this difference is cosine similarity, which allows us to see the difference directly. This evaluation is performed on the test dataset to evaluate without bias since the data augmentation is based on the training dataset. All of the augmentation is conducted at the same time with the data preprocessing, and afterwards both original and augmented versions are converted into machine-readable format.

5.2.1 Event time

To create some small noise in the timeline, we add a small number of hours to each timestamp. At the same time, we need to make sure that these added hours will not mess up with the order of the original timeline. Based on the distribution of time deltas between events in Figure 7, we can see that most of the events time difference is within a day. Therefore, we limit the possibility of added hours to be random from the

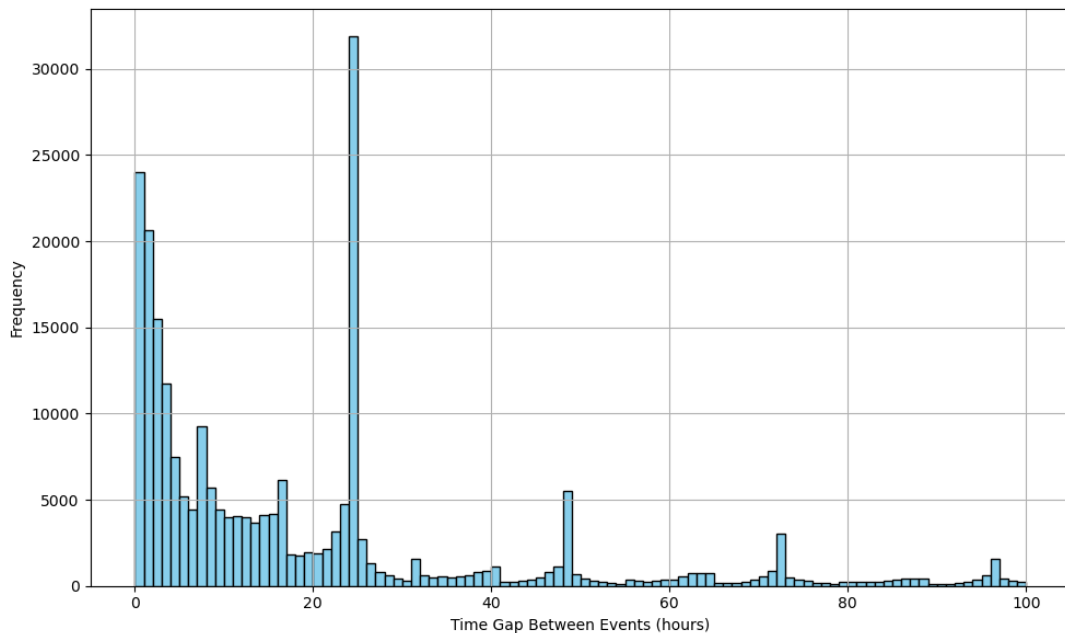


Figure 7: Event Time Delta

range $[0, 24]$. This is viable as can be seen in Figure 8, where the augmented views

clearly get introduced a lot more variation. Despite the more varied data, the main information seems to remain as most of the similarity scores are above 0.8, meaning the two views are quite close to each other. Another way to perform augmentation is

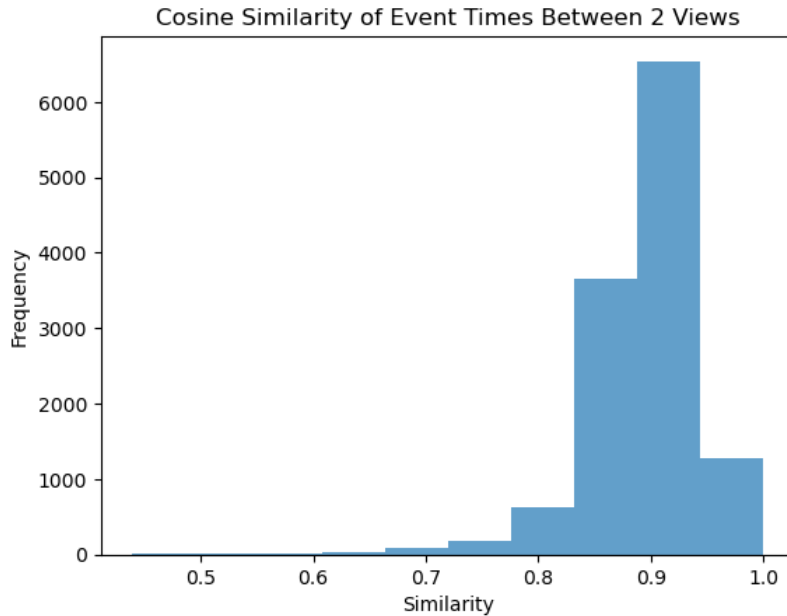


Figure 8: Augmented Time with Noise

to randomly drop out a window of the sequence. We experiment with two options to drop out: 20% and 5% of the whole sequence. The results are plotted in Figures 9 and 10, which shows that this method is also feasible as it creates a customizable amount of variation to the event time. For 20% window, it clearly introduces much more noise and modifies the structure of time series where the majority similarities fluctuate around 0.7. Regarding this, the 5% window performs a little more desirably, as the general similarity is the same as the case where we add noise to the time directly, with most of it above 0.8. In addition, we experiment how randomly permutation within a window frame can affect the similarity between original and augmented views. For this one, we also try with a 20% and 5% window frame, but instead of permuting the time directly, we fix the time and permuting the corresponding event features, which also leads to the same results. Then, for these permutation cases, the results are discussed separately for each event feature, including event type, event name, and event value further below.

5.2.2 Event type

Event type contains all category data, with the full instances being 'BMI', 'Diagnosis', 'Laboratory', 'Medication', and 'Procedure'. Since these are all meaningful words, it would not make sense to randomize any part of it. That would not create noise, but rather transform the sequences into a whole new one. Replacing an instance with another one is not practical either due to their unrelatedness. Therefore, we make

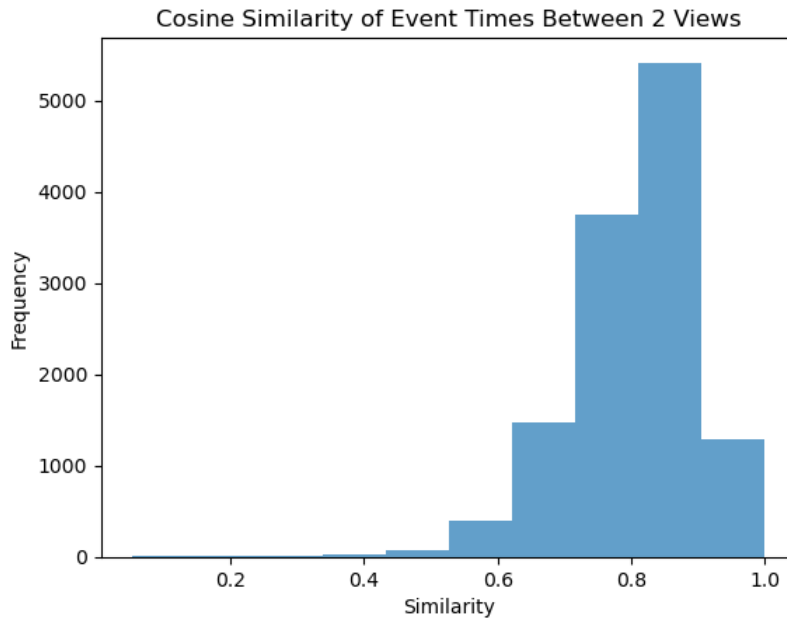


Figure 9: Augmented Time with 20% Dropout

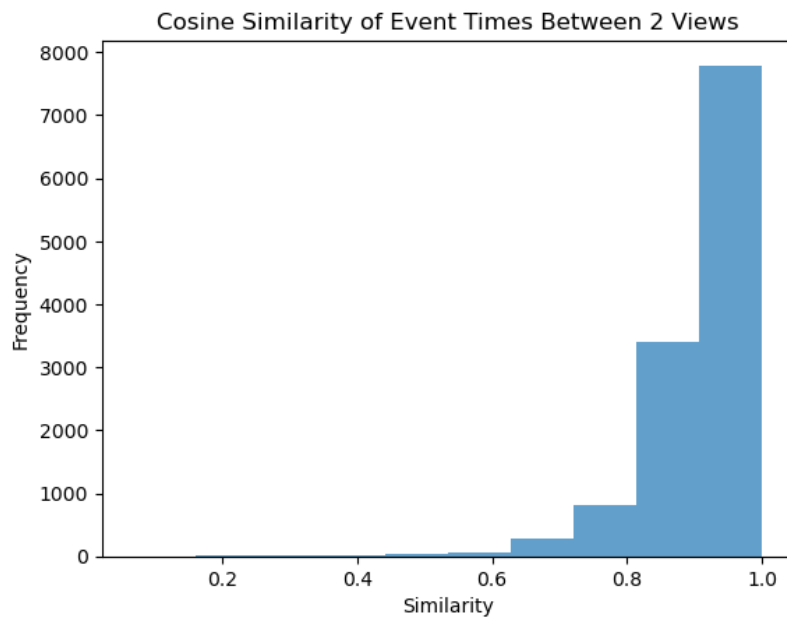


Figure 10: Augmented Time with 5% Dropout

use of a mask token to randomly replace the event type for a small percentage of the sequence. This both creates noise and pushes the model to learn the true value behind the masked token. Using this approach, the model will not mess up the types, and the augmented view will have enough variation from the original. This method creates enough amount of variation in the data as presented in Figure 11, where most of the original data are kept the same with some mask tokens for the model to learn. Additionally, in the permutation cases mentioned above, we permute the order of these

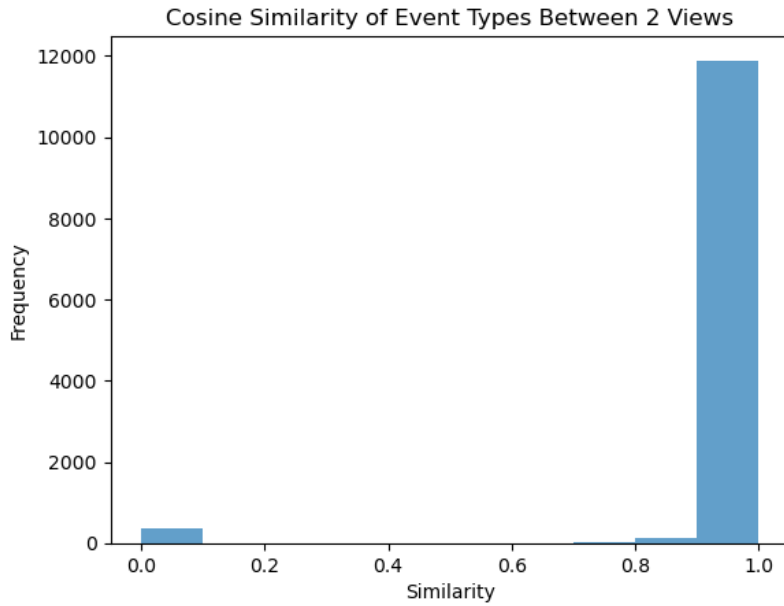


Figure 11: Augmented Type with Random Masking

event types as well as perform the augmentation, leading to the results as in following Figures 13 and 12. As expected, while both of these configurations add some noise to the data, permuting only 5% of the sequence creates noticeably less variation, which might not be a good fit if we want to explore event types in details. The amount of variation introduced by permuting a 20% window frame seems more reasonable and not as strict as the previous version.

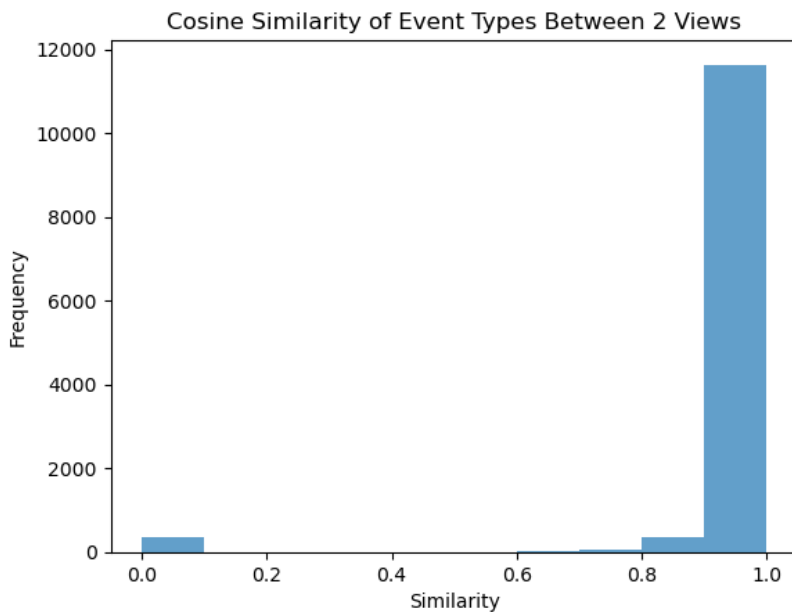


Figure 12: Augmented Type with 20% Time Permutation

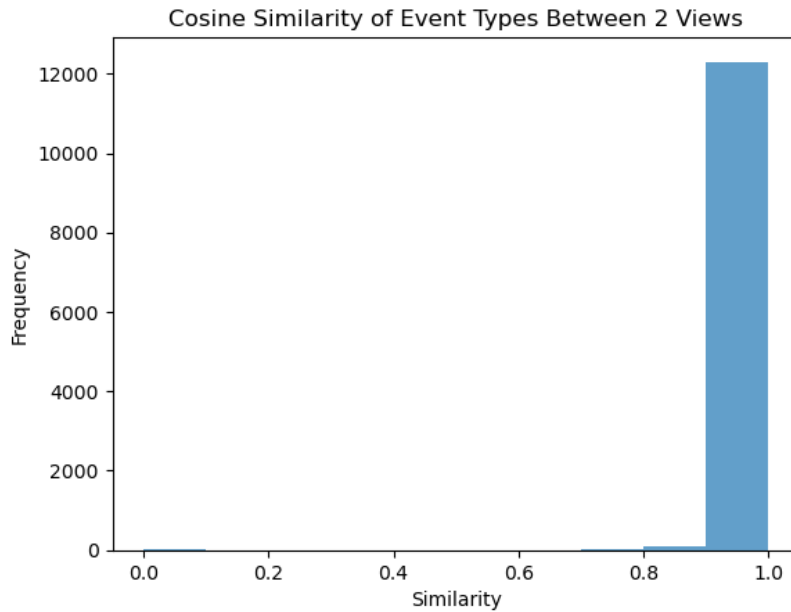


Figure 13: Augmented Type with 5% Time Permutation

5.2.3 Event values

As these are already scaled to be within $[0, 1]$, it is straightforward when we add a random noise from $\mathcal{N}(0, 1)$ to the values. By utilizing the Gaussian noise, we make sure that the augmented values will not exceed the expected range while creating some variation in the sequences. The before and after version difference can be seen clearly from Figure 14, and we also witness a bit of the Normal Distribution pattern from the noise in the graph. This variation added to the input leaves a lot of room for the model to recognize the similarity between different patients, while still maintaining the core information of each individual trajectory. Similarly to event types with regard to time permutation methods, the resulting similarity between before and after augmentation also shows the difference in the effects of the window ratio, with 20% clearly introducing more variation than 5% as shown in Figures 15 and 16. We can clearly see that using this method together with event values' own augmentation, the variation introduced to the data is substantial, making the similarity much lower than before. This might not be ideal where we want to keep the core semantics information of the patients with some little space for noise only.

5.2.4 Event name

This is by far the most complicated input, since it includes different types of data in itself. For laboratory tests, it is hard to tell if they are related by the name and changing some characters within the name does not make any sense. Therefore, the laboratory tests remain the same. Similarly, the procedure codes are also kept unchanged, as this one has no structure in the code itself. On the contrary, the ICD-10 codes and medication codes are hierarchical, and this can be seen right in the code. For ICD-10

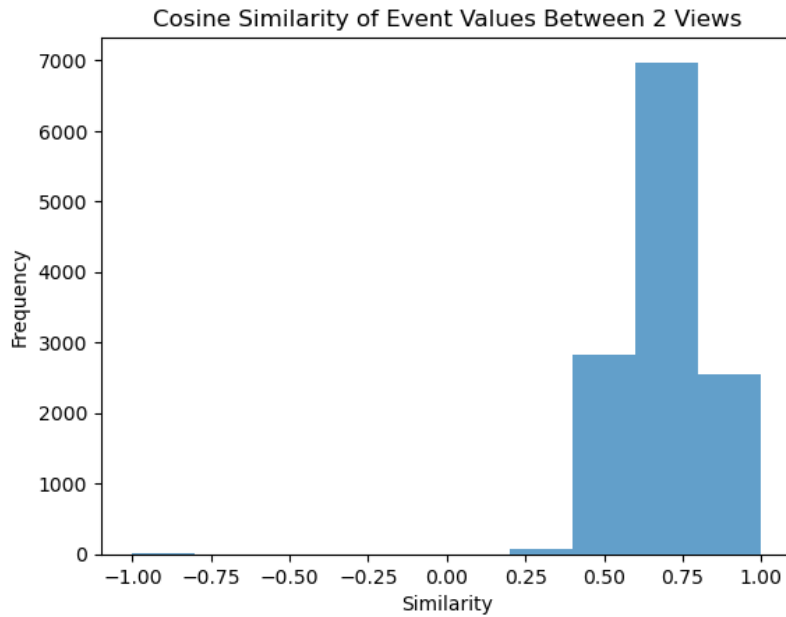


Figure 14: Augmented Value with Random Noise

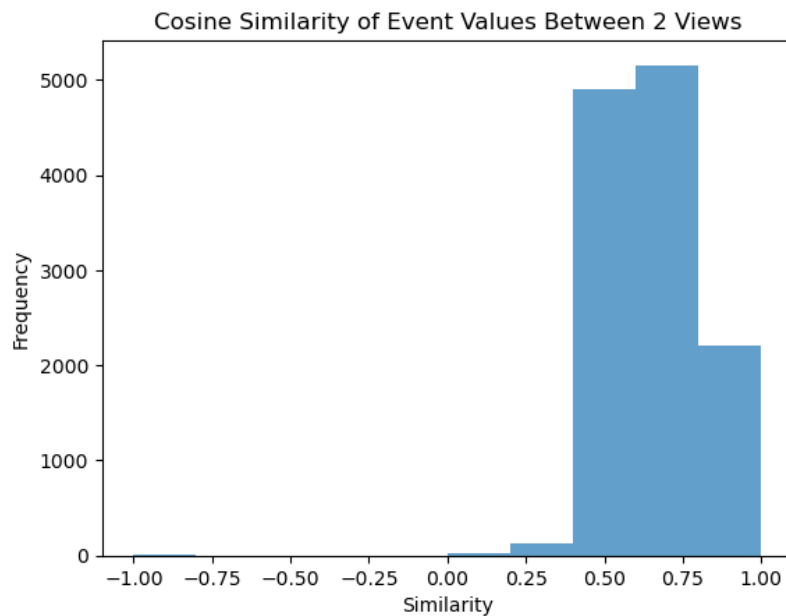


Figure 15: Augmented Value with 20% Time Permutation

codes in the current implementation, which have three characters, we randomly change the last character with a probability of 75% and change the last two 25% of the times (all must be switched to existing options in the training dataset, otherwise they would be masked out). This creates a noisy view of the codes, but will not go too far away from the group of the original code to make sure they will still be related. Likewise, we randomly swapped the last two parts of the medication codes before the ingredients

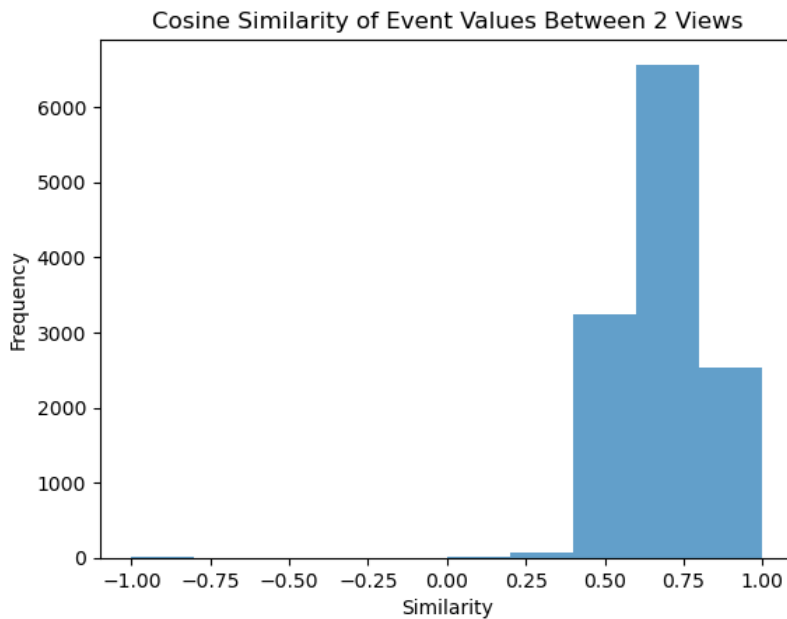


Figure 16: Augmented Value with 5% Time Permutation

to create the augmented view with the same probability. Given the current setting of changing ratio being 25% (for both two digits), we obtain the graph as in Figure 17. There is not as much variation here as for event values because we only augment

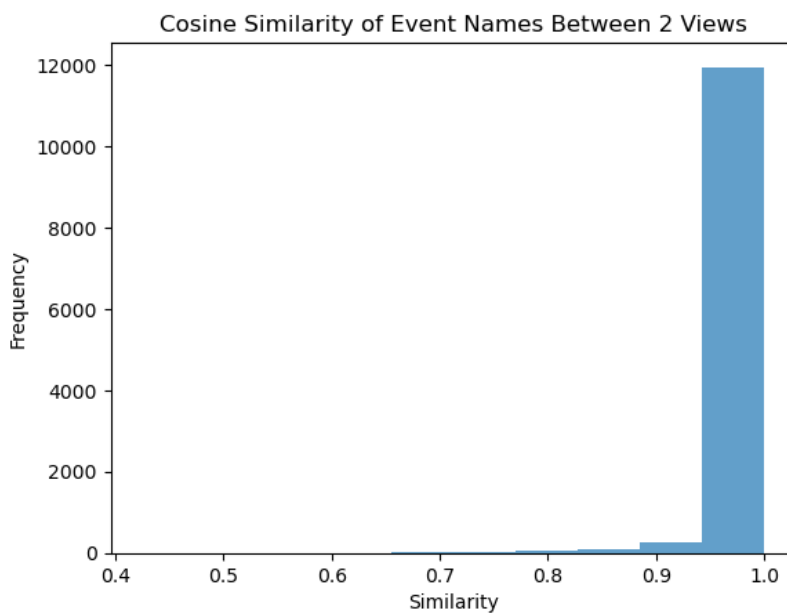


Figure 17: Augmented Name with 25% Ratio

ICD-10 codes and medication codes due to the rigidity of other inputs. Another reason is due to the smaller existence of these codes compared to the remaining data, especially the laboratory tests which can easily take up most of the sequence. Still,

this already spares the model some space for noise, allowing it to explore between patients to know which ones are more similar. To see how the changing ratio affects the data, we set it to be 0 in the next experiment and get the following Figure 18. When

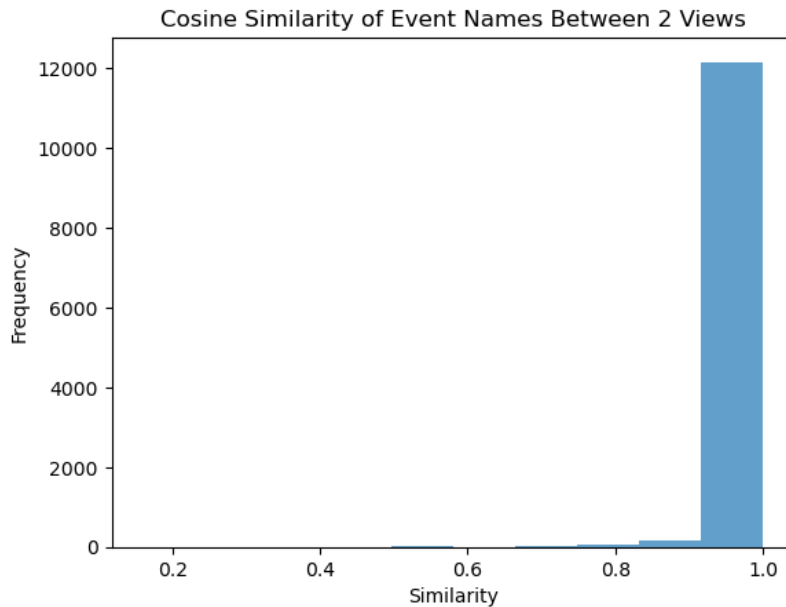


Figure 18: Augmented Name with 0% Ratio

changing only the last parts of ICD-10 codes and medication codes, the variation seems to shrink a bit compared to the previous version; however, the general difference stays the same and this ratio does not affect the structure of the augmented views. The event names are also affected by the time permutation in the cases mentioned above, where Figures 19 and 20 present how 20% and 5% window permutation can have an impact on the similarity between two views of event names. When it is only 5%, the graph looks similar to the first augmentation method we have for event names. However, as the size increases to 20% of the whole sequence, there is a noticeable decrease in the similarity, as this method now affects not just ICD-10 codes and medication codes, but also for the remaining as well.

5.2.5 Encoded Augmentation

To better see how each method performs within the context of a Transformer encoder, we have divided into 6 cases as follows and each of these will be passed through an untrained model to assess the similarity between the encoded original view and the encoded augmented view:

- Case 1: event time with noise, event type masked randomly, event name with codes last parts changed using ratio 0.25, and event value with Gaussian noise
- Case 2: event time with noise, event type masked randomly, event name with codes last parts changed using ratio 0, and event value with Gaussian noise

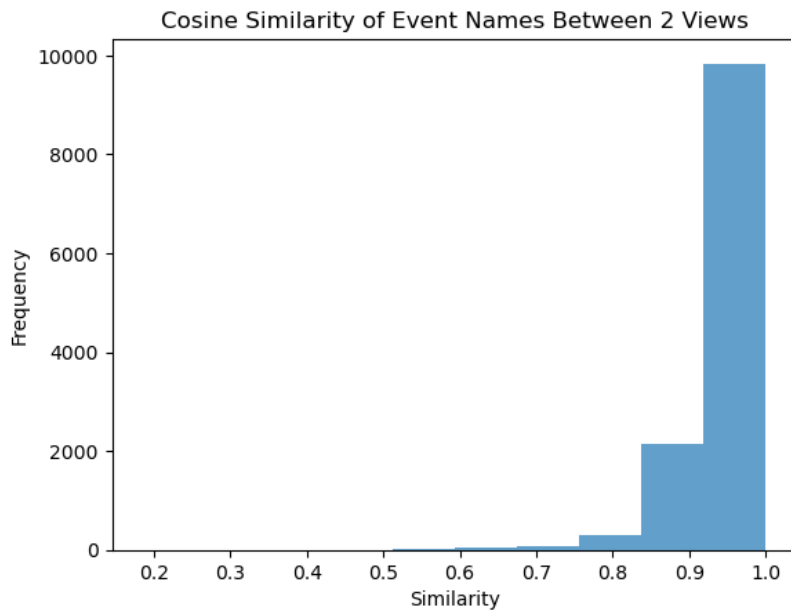


Figure 19: Augmented Name with 20% Time Permutation

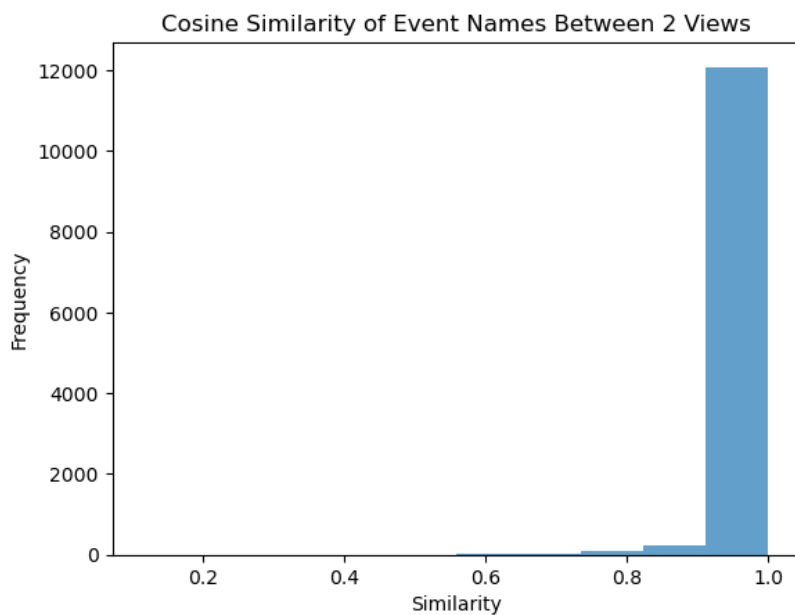


Figure 20: Augmented Name with 20% Time Permutation

- Case 3: event time with 20% dropout, event type masked randomly, event name with codes last parts changed using ratio 0.25, and event value with Gaussian noise
- Case 4: event time with 5% dropout, event type masked randomly, event name with codes last parts changed using ratio 0.25, and event value with Gaussian noise

- Case 5: event time with 20% permutation, event type masked randomly, event name with codes last parts changed using ratio 0.25, and event value with Gaussian noise
- Case 6: event time with 5% dropout, event type masked randomly, event name with codes last parts changed using ratio 0.25, and event values with Gaussian noise

Passing each of these scenarios to the encoding module, we are able to extract the embedded vector representation of the patient for both the perfect version and the noisy version. We again plot the cosine similarity between the two views to see how each augmentation technique affects the results of our model. In the end, we observe that the general shape and trend share between different scenarios, with before and after similarity fluctuating around 0.8 like Figure 21. This is quite a decent number as it means the latent representations are able to maintain the core of the trajectories while leaving out some noise to match with other similar patients. One interesting note that can be spotted easily is that no matter what technique we are using, the embedded vectors in the latent space do not vary much between cases. Therefore, as long as the augmentation method is valid, we are good to experiment with them. In our model, we decide to stick with Case 1 for its simplicity and straightforwardness. This exhaustive investigation into different methods and combinations of augmentation with regard to data types gives us an answer to the **RQ1** about how each data can be augmented and what approaches can be used.

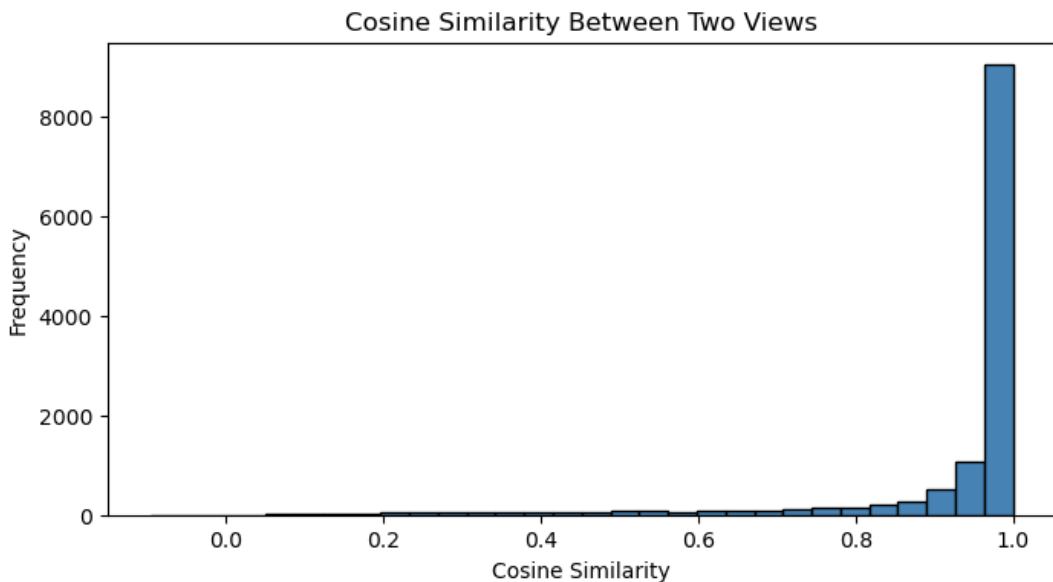


Figure 21: General Histogram of All Cases

5.3 ICD-10 Codes Experiments

To handle the hierarchy of ICD-10 codes and ATC codes, we need to look at how they can be broken down into multiple parts and find a way to embed this correctly before integrating everything into one unified model. There have been multiple attempts at integrating this ICD-10 codes training in the main encoder. We document different strategies that have been implemented to show the development process and why each method works or does not. The following part describes previous versions of the ICD-10 codes model as well as the final one.

5.3.1 First Experiment of Hierarchical Model for ICD-10 Codes

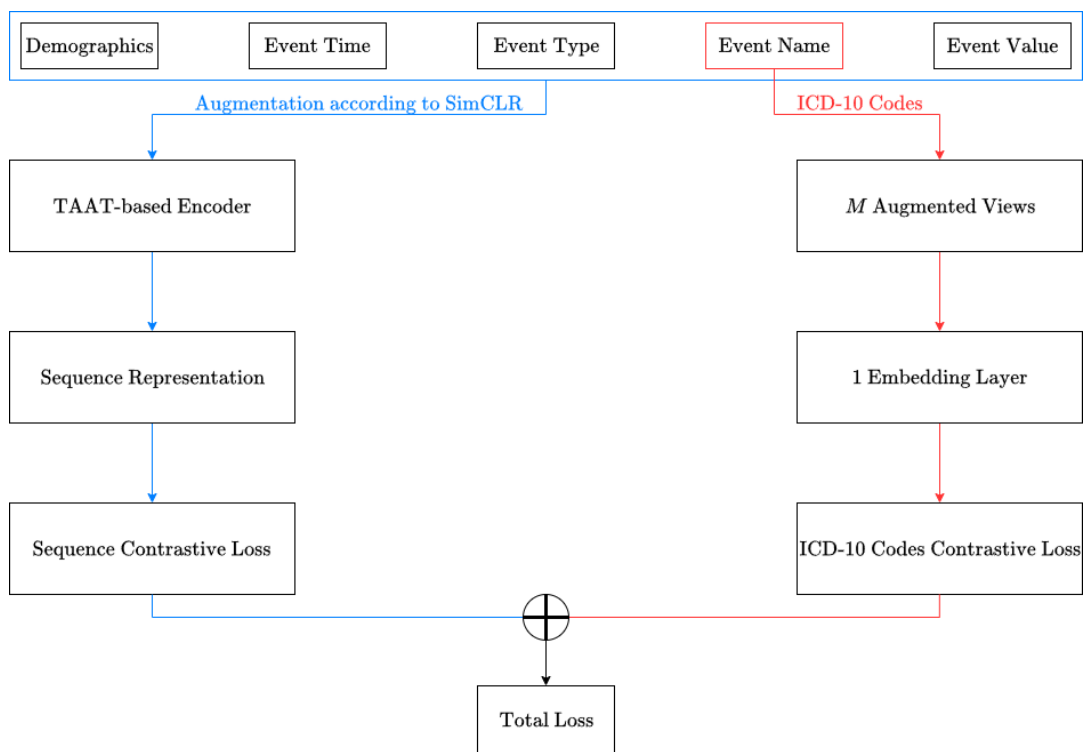


Figure 22: First attempt overview

For the first attempt, besides creating an augmented view of the event features (including timestamp, type, name, and value), we also have a separate part for contrastive learning of the ICD-10 codes. There are M augmented views of an ICD-10 codes, and each of them (including the original) will be passed through an embedding layer (simple lookup table) together with a projection layer, then the ICD-10 codes loss would be the sum of losses between every view and the original ICD-10 codes. For the whole sequence, the event features are passed through the TAAT-based encoder and projection layer to extract the projected representation for loss computation. These two losses of ICD-10 codes and whole sequence will be optimized during the training process. However, the final results of learning the relationship between ICD-10 codes are not promising, partially due to the fact that the loss is dominated by the event

features more. We use average cosine similarity between chapter-level groups and category-level groups to assess the model performance. As can be seen from Figure 24, the model barely learns any relationship between codes of the same groups. Also, the time it takes for training is extremely long (2 minutes per epoch) since this is basically training on the whole dataset. Therefore, we proceed to only training on ICD-10 codes.

5.3.2 Second Experiment of Hierarchical Model for ICD-10 Codes

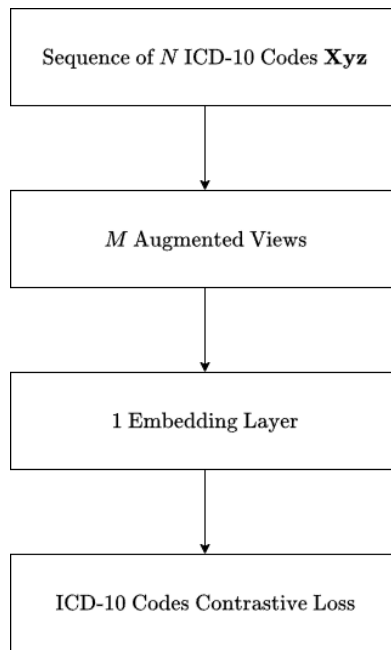


Figure 23: Second attempt overview

In the next trial, we put the encoder part aside and focus only on the ICD-10 codes embedding training so that the loss will not be affected by event features. We use a similar architecture for learning ICD-10 codes as the first version, with an embedding layer and a projection head to learn the similarity between an original code and its M augmented views. This time, when the model is very light and small, the training process goes really fast, allowing us to have more modifications to the hyperparameters. The model can run 1000 epochs in just a few minutes. Hence, we experiment with different numbers of M , from small to large, to see if that creates any difference on the final outcomes. Unfortunately, the results obtained do not differ much from the first version, even though there is a very slight increase in performance such that the average cosine similarity between codes is a small bit higher, but no more. We realize that the inherent hierarchy structure of the ICD-10 codes cannot be learned using just a single embedding head; hence, we divide the codes into three layers of embedding in our next attempt.

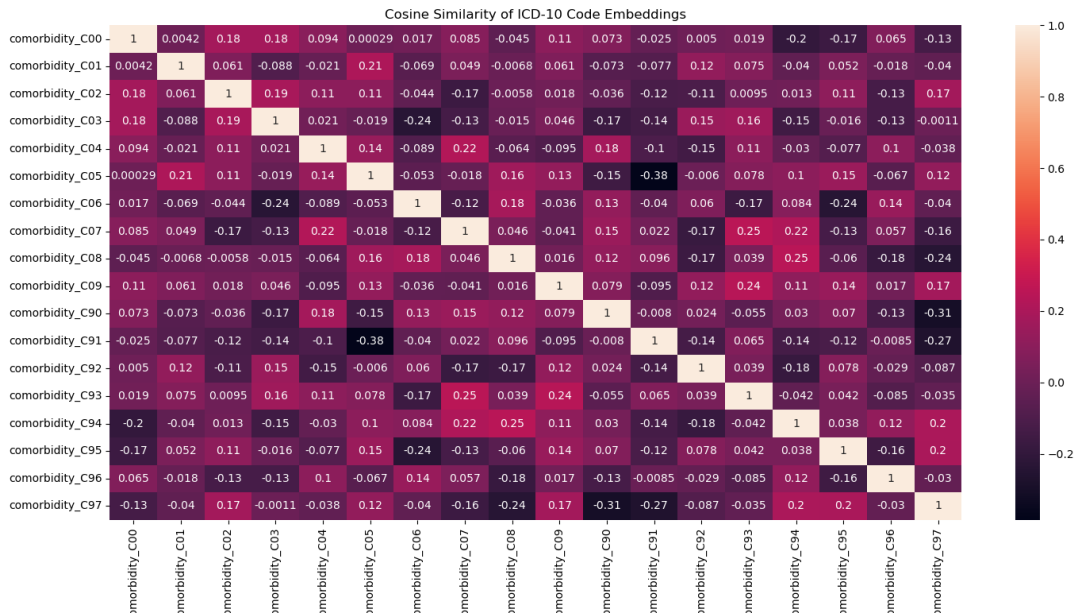


Figure 24: ICD-10 Codes in training with other Event Names

5.3.3 Third Experiment of Hierarchical Model for ICD-10 Codes

In this attempt, we divide the codes into three parts: the first character, the second character, and the last character. Each of these will be learned by a separate embedding layer, then summed together to go through the loss function. This schema sounds more reasonable as it is able to maintain the hierarchical structure of the codes, while still learning the similarity between codes that are of the same chapter or category. This can be validated by the results presented in Figure 26, which shows exactly what we want for the ICD-10 codes with a network graph: codes in one category should be more similar compared to those that are outside of the category. In this graph, each code is colored by the first character, and we notice that those with the same colors form their own clusters. Therefore, we choose to develop the model in a manner that each of these hierarchical medical indicators will be pretrained using layered embeddings, and integrated later into the main encoder part. Further evaluation of this architecture is explained in Subsection 5.5.1, with Figure 30 exhibiting an ideal representation of the hierarchy between codes within and without the same group. A similar strategy is also applied for learning the hierarchy of medication codes, and the results also turned out to be great, indicating the success of our pretrained models. They have helped with answering the **RQ2** about the methods to learn the hierarchical structure of ICD-10 and medication codes.

5.4 Final Multimodal Model

This is where we put everything together to solve the question of **RQ4** about the unified framework with regard to our cancer patients EHR data. After several experiments on how to improve the quality of ICD-10 codes embedding, we have come to a

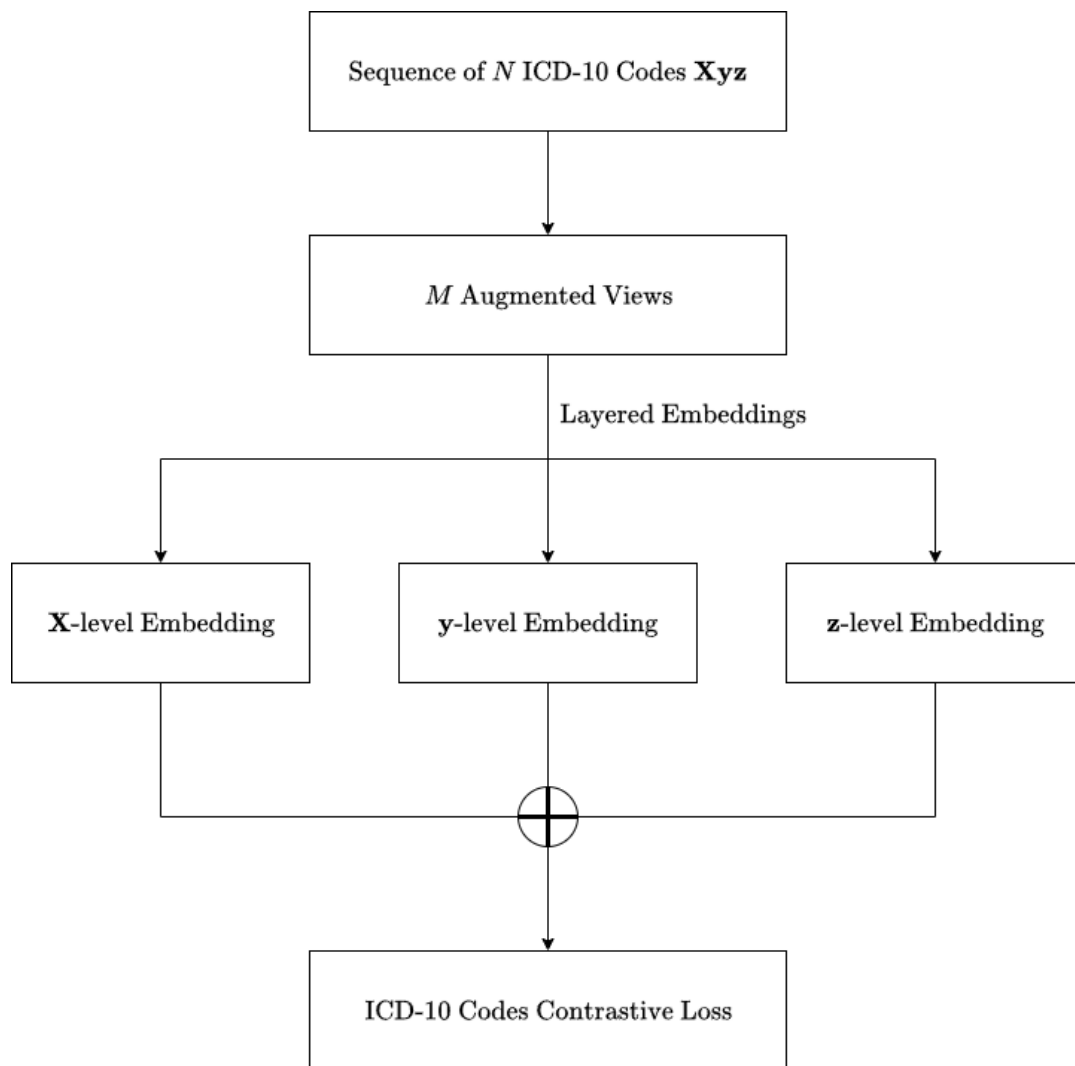


Figure 25: Third attempt overview

final version where all the hierarchical medical indicators are pretrained separately before integrated into the big picture. This final model includes a TAAT-based encoder and a projection head for latent embeddings. Before passed to this encoder, the data is handled accordingly. Event timestamp is divided into multiple layers including years, months, days, and hours. Each is embedded and summed together afterwards. Event type is embedded normally using a lookup table. Event value is embedded using continuous value embedding CVE which is designated for continuous numbers rather than discrete ones. Event name is the most complicated as it contains different types of data: laboratory tests, ICD-10 codes, ATC codes, procedure codes, and 'measurement'. Laboratory tests, procedure codes and 'measurement' can be embedded using straightforward lookup tables, but the remaining ICD-10 and ATC codes are self-related in the sense that the codes have hierarchical structure and codes within the same group are related to an extent. Therefore, each of them are passed through a separate pretrained model that returns the same output shape, allowing

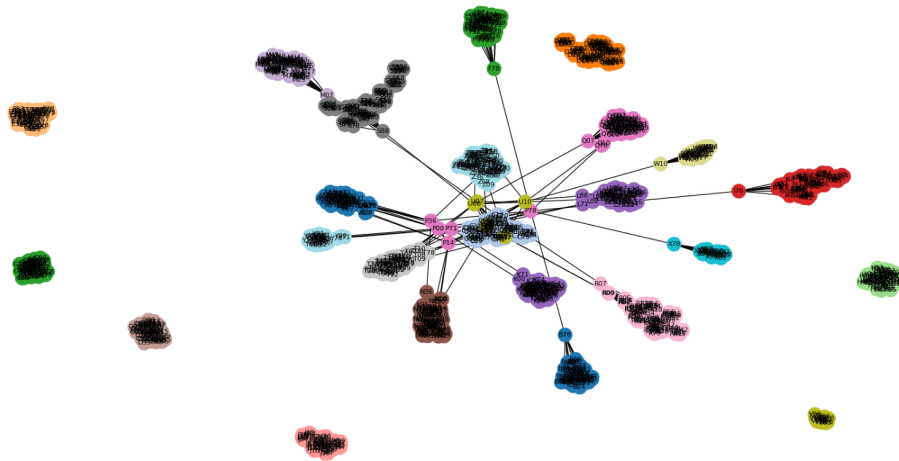


Figure 26: ICD-10 Codes Network

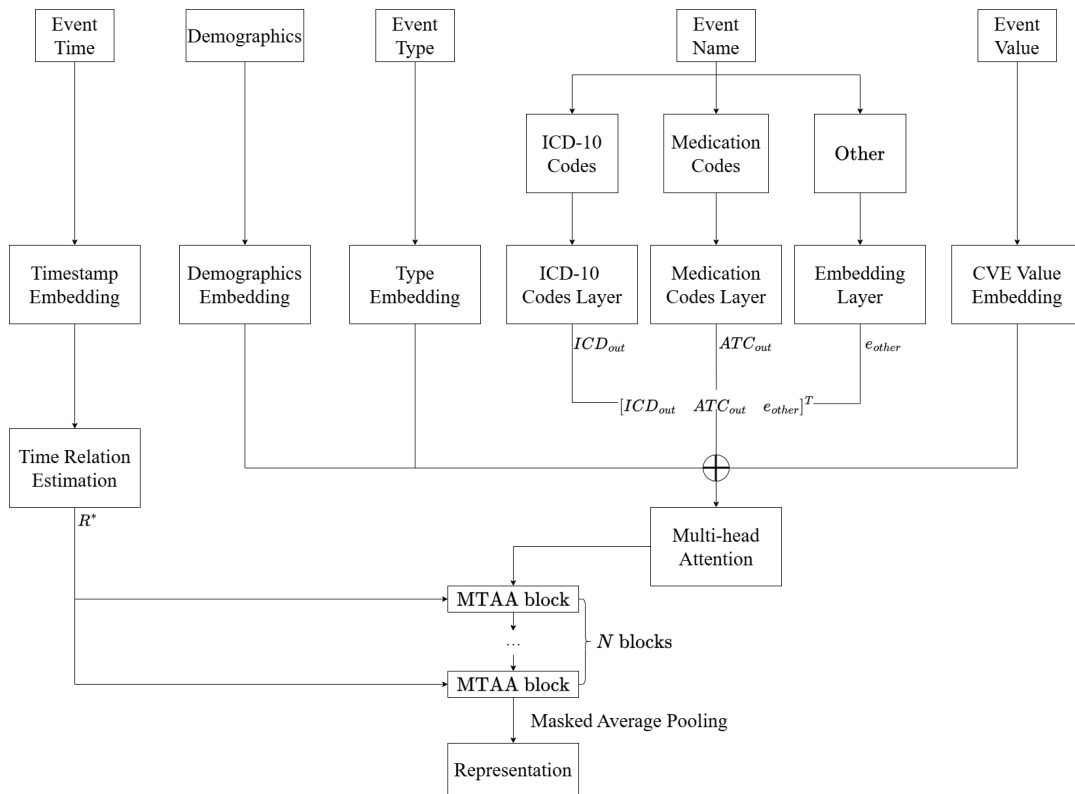


Figure 27: Final model overview

them to be merged into one. All of these event features are added together before passing through the encoder module. Each component of the final model is described in further details in the following parts.

5.4.1 Pretrain on ICD-10 Codes

As described previously, ICD-10 codes are hierarchically structured. For example, all the codes starting with **C0_** should be more similar to themselves than to **C1_** codes. Similarly, codes of group **C_** should be more relevant within group than between groups, i.e., groups **A_** or **B_**. Using contrastive learning on the whole event sequence cannot guarantee this desired similarity level, hence we developed a separate pretrained model for ICD-10 codes. As we have learned that these codes can be divided into levels, or layers, we make use of layered embeddings for each component of the codes so that this pretrained model can learn the similarity between layers instead of the whole codes. The implementation is similar to what was described during the third attempt as shown in Figure 25. For each character of a code, we assign a learnable embedding layer and later sum up all three layers. Then we proceed to train the model using contrastive learning framework in a manner that each code will be modified M times, creating M different views of the same codes. These M augmentations serve as the labels in the training process, with a view to obtaining an hierarchical relevance between codes. The results obtained have shown that this strategy works very well, meaning that average within-group similarity is much higher than average between-group similarity, retaining the natural hierarchical structures of ICD-10 codes.

(general idea of this paragraph content first, then into details) Given a code i from the set of \mathcal{V} unique ICD-10 codes as input, we divide the code into three parts (i_1, i_2, i_3) corresponding to each character and pass to three layers of embedding vectors. The first layer learns the first-level or most general category embeddings of the code. Given i_1 , it learns the matrix

$$ICD_1 \in \mathbb{R}^{26 \times d}$$

where 26 is the total possible cases of alphabetic characters and d is the embedding dimension so that the output is

$$ICD \text{ Layer } 1(i_1) = ICD_1[i_1] \in \mathbb{R}^d$$

Then each case of i_1 has the corresponding learnable embedding in higher space, allowing for further exploration during training and optimization. Similarly with the second layer, we have the matrix

$$ICD_2 \in \mathbb{R}^{10 \times d}$$

But now, instead of 26 alphabetic characters, there are only 10 possible single-digit numbers. The output of this layer is also a learnable embedding:

$$ICD \text{ Layer } 2(i_2) = ICD_2[i_2] \in \mathbb{R}^d$$

This makes sure every number is assigned to an embedding vector. The third layer is the same as the second layer because it also involves only a single-digit number. Hence, this layer returns:

$$ICD \text{ Layer } 3(i_3) = ICD_3[i_3] \in \mathbb{R}^d$$

The final output of this pretrained ICD model is a sum of all the above embeddings when they all have the same embedding dimension d :

$$ICD_{out} = ICD_1[i_1] + ICD_2[i_2] + ICD_3[i_3] \text{ and } ICD_{out} \in \mathbb{R}^d$$

5.4.2 Pretrain on ATC Codes

ATC codes have more characters than ICD-10 codes in general, but the first three characters follow the same structure, with the fourth one being a more specific category and the remaining are the active ingredient in the medication. We can already see that there is a hierarchy existing within these ATC codes, but with more layers. Therefore, inspired by the experiment with ICD-10 codes, we separately train the ATC codes with five layers of embedding, four of which correspond to the first four characters of the codes, and the last layer is for what is left in the code (active ingredient). The visualization of the structure is presented in Figure 28 as below.

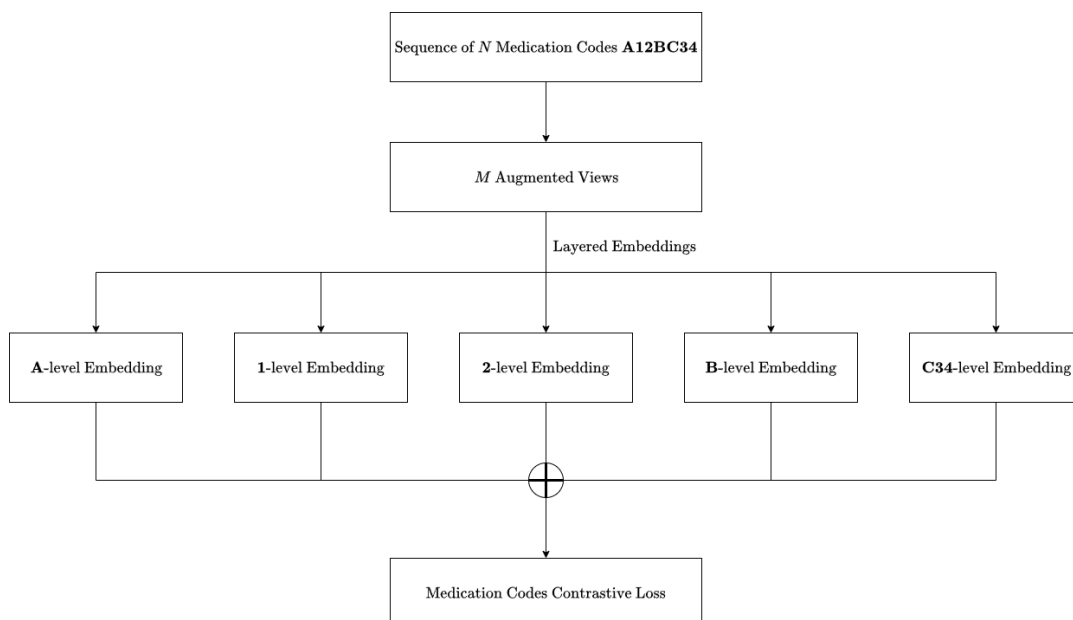


Figure 28: ATC Codes Model

Similar to ICD-10 codes, we want the model to recognize the hierarchical patterns within the ATC codes. By utilizing this layered embeddings structure, this goal can be achieved. For an ATC code like **A12BC34**, we divide into 5 subparts: **A**, **1**, **2** (similar to ICD-10 codes for the first three), **B**, and whatever is left after that, in this case **C34**. Each of these will be passed through an embedding layer and summed up for a projection layer. This prebuilt model is also trained using contrastive learning, meaning the model should be able to identify whether the codes are similar based on their subparts. The contrastive learning is conducted similarly, where each code is learned against M modifications, allowing the model to have more "imagination" about the similarity and difference between the codes. The results obtained are satisfactory in the sense that the model is now able to maintain the hierarchy and distinguish

between the codes.

As explained above in the ICD-10 codes pretrained model, the actual ATC codes are not passed directly through the model but assigned a learnable embeddings that can be optimized during the training to find out the relationship between and within features. Instead of three layers like ICD-10 codes, we assign five layers for ATC codes handling five components of an ATC code $(a_1, a_2, a_3, a_4, a_5)$ as introduced above. The first layer can be represented as

$$\text{ATC Layer 1}(a_1) = \text{ATC}_1[a_1] \in \mathbb{R}^d$$

with a learnable matrix $\text{ATC}_1 \in \mathbb{R}^{26 \times d}$ since we have an alphabetic character here, so there will be 26 options in total. The second and third layers are similar to each other due to the fact that they only carry one single digit, so we have:

$$\text{ATC Layer } j(a_j) = \text{ATC}_j[a_j] \in \mathbb{R}^d$$

with learnable matrices $\text{ATC}_j \in \mathbb{R}^{10 \times d}$ where $j = 2, 3$ because all of them are numbers from 0 to 9. The fourth layer is similar to the first one such that it has an alphabetical character, leading us to:

$$\text{ATC Layer 4}(a_4) = \text{ATC}_4[a_4] \in \mathbb{R}^d$$

with a learnable matrix $\text{ATC}_4 \in \mathbb{R}^{26 \times d}$. The fifth layer, also the final one, includes more information and is also less strict about the hierarchy compared to the first four. Thus, we group all the last 3 characters together and embed them using a simple lookup table. We can only provide a rough and general embedding layers mapping each instance of this layer to an embedding vector:

$$\text{ATC Layer 5}(a_5) = \text{ATC}_5[a_5] \in \mathbb{R}^d$$

but now with an unfixed-length matrix $\text{ATC}_5 \in \mathbb{R}^{\text{ATC vocab size} \times d}$

Given these, the final output of the ATC pretrained model can be obtained by summing them up:

$$\text{ATC}_{out} = \text{ATC}_1[a_1] + \text{ATC}_2[a_2] + \text{ATC}_3[a_3] + \text{ATC}_4[a_4] + \text{ATC}_5[a_5]$$

and $\text{ATC}_{out} \in \mathbb{R}^d$ as we use the same embedding dimension for all of these sublayers.

5.4.3 TAAT-based Encoder Module

The encoder module of this contrastive learning framework is the modified and modernized version of TAAT model [39] introduced previously in Section 4.3. First of all, we embed all of our variables through their corresponding embedding layers. For examples, categories (genders, subgroups, cancer, types, names including laboratory tests, procedure codes, and ‘measurements’) will go through normal lookup table embeddings, while continuous values (ages, values) will go through CVE (similar to STRaTS [30]). For special structures such as ICD-10 and medication codes, they

are passed through designated multi-layer embeddings, ensuring their hierarchy to remain after training. By utilizing embedding technique, we convert multimodal data including text-based codes and numerical values to a unified machine-readable format. This process can be formulated as follows:

$$\mathbf{e}_{j,gender} = Gender[g] \in \mathbb{R}^d$$

with g being either ‘Male’ or ‘Female’ and a learnable matrix $Gender \in \mathbb{R}^{2 \times d}$

$$\mathbf{e}_{j,subgroup} = Subgroup[s] \in \mathbb{R}^d$$

with s being a record of patient subgroup and a learnable matrix $Subgroup \in \mathbb{R}^{\text{Total subgroups} \times d}$ as the number of subgroups is purely dependent on the data itself. Similarly, for cancers and event types, as well as ‘measurements’ in event names:

$$\mathbf{e}_{j,cancer} = Cancer[c] \in \mathbb{R}^d$$

with c being a record of cancer and a learnable matrix $Cancer \in \mathbb{R}^{\text{Total cancers} \times d}$.

$$\mathbf{e}_{j,type} = Type[t] \in \mathbb{R}^d$$

with t being a type of the event and a learnable matrix $Type \in \mathbb{R}^{\text{Total types} \times d}$.

$$\mathbf{e}_{j,other} = Name[n] \in \mathbb{R}^d$$

with n being a laboratory test name, or a procedure code, or the ‘measurement’ text itself and a learnable matrix $Name \in \mathbb{R}^{\text{Total names} \times d}$ so that it can cover all possible instances of these event names. This embedding will be merged with the following ICD-10 codes and ATC codes embedding to achieve the similar shape as other embeddings.

For the ages and event values of the patient, they will go through CVE, which essentially is a feed forward network so that the numerical values are projected to another higher-dimensional space for exploring and learning. This is similar to STRaTS CVE. Hence, we have:

$$\mathbf{e}_{j,age} = CVE_{age}[ag] \in \mathbb{R}^d$$

and

$$\mathbf{e}_{j,value} = CVE_{value}[v] \in \mathbb{R}^d$$

Besides these straight-forward features, we also mention ICD-10 codes and ATC codes to be pretrained separately. For each i of ICD-10 codes and each a of ATC codes extracted from the event names, we obtain:

$$ICD_{j,out} = ICD_1[i_1] + ICD_2[i_2] + ICD_3[i_3] \text{ and } ICD_{out} \in \mathbb{R}^d$$

and

$$ATC_{j,out} = ATC_1[a_1] + ATC_2[a_2] + ATC_3[a_3] + ATC_4[a_4] + ATC_5[a_5] \in \mathbb{R}^d$$

The above embeddings are applied to a single event j of a patient at a time. When we are given a series of event features as inputs with shape $L \times d$ where L is the length of the patient's records, the embedding shape then becomes $\mathbb{R}^{L \times d}$, with the special case of $e_{other}, ICD_{out}, ATC_{out}$ when these are put together to obtain the full length of the trajectory. Then, the final form of event names embeddings when concatenated would be:

$$\mathbf{e}_{name} = [e_{other} \quad ICD_{out} \quad ATC_{out}]^T \in \mathbb{R}^{L \times d}$$

Together, we obtain the final embedding vectors of all features per patient included by summing up these embeddings to prepare for the next layer of Multi-Head Attention (MHA):

$$\mathbf{e} = \mathbf{e}_{age} + \mathbf{e}_{gender} + \mathbf{e}_{subgroup} + \mathbf{e}_{cancer} + \mathbf{e}_{type} + \mathbf{e}_{name} + \mathbf{e}_{value} \in \mathbb{R}^{L \times d}$$

The purpose of this MHA is to learn the different types of information in the features with regard to their position, or timestamps. Time is treated as positional embedding in this case, and it is also broken down into multiple levels to retain the relativity over sparse time series without the need to impute or interpolate. Before joining with the MHA outputs, time first goes through a Time Related Estimation (TRE) layer to extract information such as time gap between events, temporal decay of older event, and the intuition that closer events are more relevant. TRE gives the model a better understanding of the relationship between events considering the time they occurred, so a cluster of events might be more useful to look at rather than just a sequence of full single events. TRE is built upon the time delta, or the difference in time, between each event and the beginning event. This time delta tells the model about the relative position or distance between events. According to [TAAT model] with our modifications, given a series of timestamp (of a single patient) $[t_i^1, t_i^2, \dots, t_i^L]$ where L is the length of the patient's trajectory, we have the time delta as:

$$\Delta t_i = [\Delta t_i^1, \Delta t_i^2, \dots, \Delta t_i^L]$$

$$\Delta t_i^j = t_i^j - t_i^1, j = 1, 2, \dots, L$$

Furthermore, in order to minimize the sparsity between timestamps, the time delta is divided into a pyramid structure including multiple levels ϕ of time. For our cases, $\phi \in [1, 2, 3, 4]$, each corresponding to year - month - day - hour. The reason for this choice of structure instead of using smaller or bigger levels is that normally, the patient's records can span through years and many of them can happen during the same day. This number of layers can be adjusted by changing the value of ϕ , giving us different relation estimation between timestamps. When we have successfully derived the indices of time layers, we pass them through their corresponding embeddings that are more compatible with the model and easier for the learning process:

$$Z_i^T = V_T(\mathbf{t}_i) = [v_T(\Delta t_i^1), \dots, v_T(\Delta t_i^L)]^T \in \mathbb{R}^{L \times d}$$

with

$$v_T(\Delta t) = \sum_{j=1}^{\phi} v_j(\Delta^j t)$$

where $j = 1, 2, 3, 4$ corresponds to year, month, day, hour in our model. When the time is properly embedded, the temporal information within the sequence of events is calculated using self-attention mechanism as follows:

$$\text{Query: } Q_T = Z_i^T W_T^Q \in \mathbb{R}^{L \times d'}$$

$$\text{Key: } K_T = Z_i^T W_T^K \in \mathbb{R}^{L \times d'}$$

We then have two learnable projection matrices W_T^Q and W_T^K that can be used to calculate the temporal information during training phase:

$$\mathbb{R}^* = \frac{Q_T K_T^T}{\phi \sqrt{d}}$$

where the numerator $Q_T K_T^T$ learns how each timestamp related to others and the denominator is the scaling factor. The projection dimension d' is set to be equal to d in our model.

This TRE is integrated into MHA to create Multi-head Time-Aware Attention (MTAA), where the model goes through the learning process and tries to understand the relevance between time-events and events-events. This MTAA is able to capture the sparsity between events as well as how each event is related to others in the trajectory, or simply the semantic and temporal information of a series of events. It is composed of multiple Time-Aware Attention (TAA) run simultaneously in parallel. The details of TAA head when given the TRE matrix \mathbb{R}^* and the output $Z_i^{(n-1)} \in \mathbb{R}^{L \times d}$ of the $(n-1)^{th}$ layer is explained in the following:

$$\text{Query: } Q_i^{(n)} = Z_i^{(n-1)} W_Z^{(n),Q} \in \mathbb{R}^{L \times d'}$$

$$\text{Key: } K_i^{(n)} = Z_i^{(n-1)} W_Z^{(n),K} \in \mathbb{R}^{L \times d'}$$

$$\text{Value: } V_i^{(n)} = Z_i^{(n-1)} W_Z^{(n),V} \in \mathbb{R}^{L \times d'}$$

The attention score is then calculated using these matrices with the learnable projection ones including $W_Z^{(n),Q}$, $W_Z^{(n),K}$, $W_Z^{(n),V}$:

$$TAA = \text{softmax}\left(\frac{Q_i^{(n)} K_i^{(n)T}}{\sqrt{d'}} + \mathbb{R}^*\right) V_i^{(n)}$$

Essentially, what this does is that it computes how much attention each event should pay to others with regard to time, capturing both semantic and temporal information between events. The final score after going through MTAA is passed to a Feed Forward network and Layer Normalization as a core step to ensure the training stability and to add expressive power to the model.

Afterwards, the output goes through a projection layer to obtain latent representation of each patient trajectory, typically in a higher dimension to explore more complicated information corresponding to each time-event. The high-dimension latent representation are passed through the pooling layer to make sure that in the end, each record only has one representation vector that can be explored in lower space. This pooled latent vector represents a whole sequence of events, including the semantic information between events and the similarity compared to other patients.

5.5 Evaluation

5.5.1 ICD-10 Codes Embedding Vectors

The hierarchical structure of ICD-10 codes can be spotted easily when a human sees them, but it is not the same easy task for a machine to recognize. There has not been a study before about learning the relationship of ICD-10 codes considering their hierarchy, so this work also reveals a novel way to train a model to understand this structure in codes. This subsections discusses mostly the embedding vectors of ICD-10 codes, with some mention to the medication codes as their structures are quite similar.

The following results are extracted from the pretrained ICD-10 codes model on the codes from the training dataset. The corresponding weight for each layer is chosen to be 0.5, 0.3, 0.2, indicating the importance of each layer. The outermost layer, or the character at the beginning of the code, is the most important as it tells which part of the body the disease belongs to. The following digits are of the same importance, with the second digit having a little more weight than the third one. To assess the quality of the trained embedding vectors, we use cosine similar metrics and design a so-called "ground truth" tree graph with an artificial root connecting the first character of all the codes to determine how many steps needed to go from one code to another. If the two codes have the same first and second digit/character, they will be 1 step far. If they are only within the same first category, that will be 2 steps away. If the codes are different from the start, it means they are not within the same node, hence that is 3 steps between them. Using this logic to build a tree graph, we obtain the ground truth hierarchy for the ICD-10 codes. We use this and the trained distance from embedding vectors to study how well the model learns the relationship between codes.

As can be seen from Figure 29, when given two sets of codes starting with A0_ and A1_, the obtained results demonstrate very well their cosine similarity. When the codes share the same first and second category, it is almost likely that their similarity is very high, approximately 0.9 for all cases. When they only share the first category, the similarity drops significantly, and this can be seen from the lighter color in the graph. Even though they are still positively similar, the relationship is not as strong compared to when they share both categories. Figure 30 showcases another scenario when the codes are from completely different category. The visualization includes a lot of bland color moving around 0, indicating that these codes are not at all similar to each other. The behaviors expressed in these figures are ideal and close to what we expect from the hierarchical structures of ICD-10 codes. If the model was not trained with the weight, it would give the last digit of the code more importance than others, resulting in an undesired similarity: even if the codes are different right from the start, they could score high when they share the same last digit. This is because the way we design contrastive learning is to randomly change the last digit, making it think that this is the importance factor when there is no weight assigned to each layer, leading to a wrong understanding of the model as visualized in Figure 31. Therefore, to balance out the effect of the last digit and correct the importance of other layers, we decide to assign the weight in a way that the first layer has the most and the last layer has

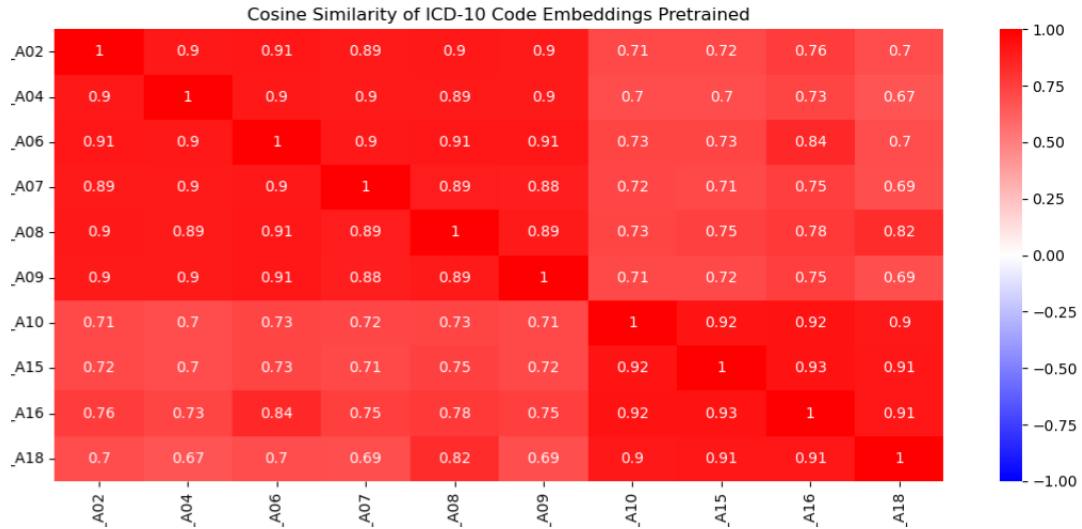


Figure 29: Samples of ICD-10 Codes in the same group

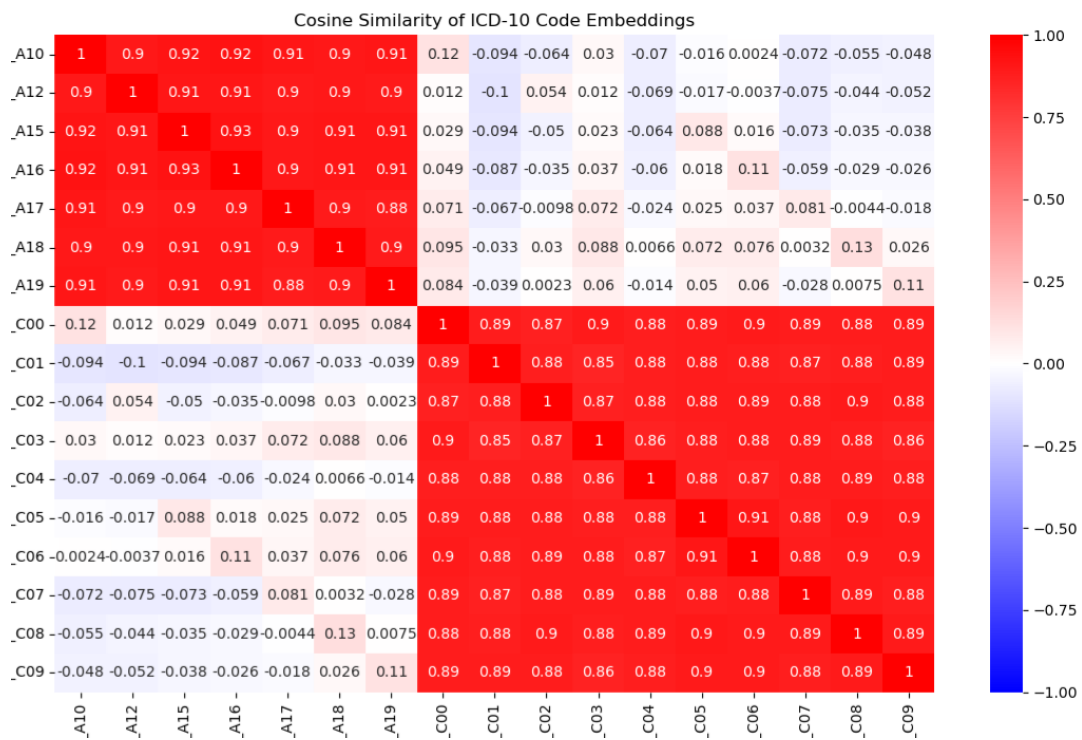


Figure 30: Samples of ICD-10 Codes in two different groups

the least. This ensures the proper hierarchy and gives us a better result of the ICD-10 codes embedding vectors as shown the figures above. As we mentioned about the pseudo ground truth tree manually constructed, we also use it as a way to measure the performance of the model. To be more specific, we calculate the distance of the learned embeddings and plot it against the codes. Since we have over 1000 codes, it would be difficult to put them all in one graph; therefore, we will pick a subset to

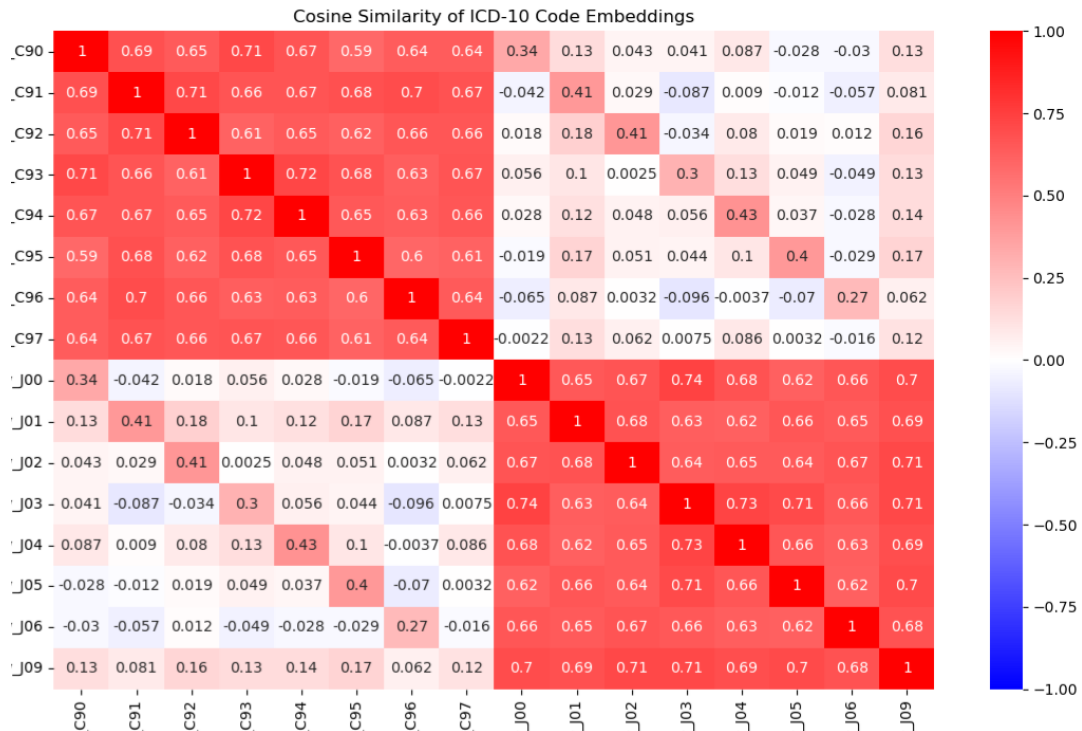


Figure 31: Samples of ICD-10 Codes in two different groups without weight

visualize, which happens to be the code starting with A. As can be seen in Figure 32, the pattern is very clear: when the codes are similar, their distance is 0; and this gets further when they gradually diverge along their code. Compared to the ground

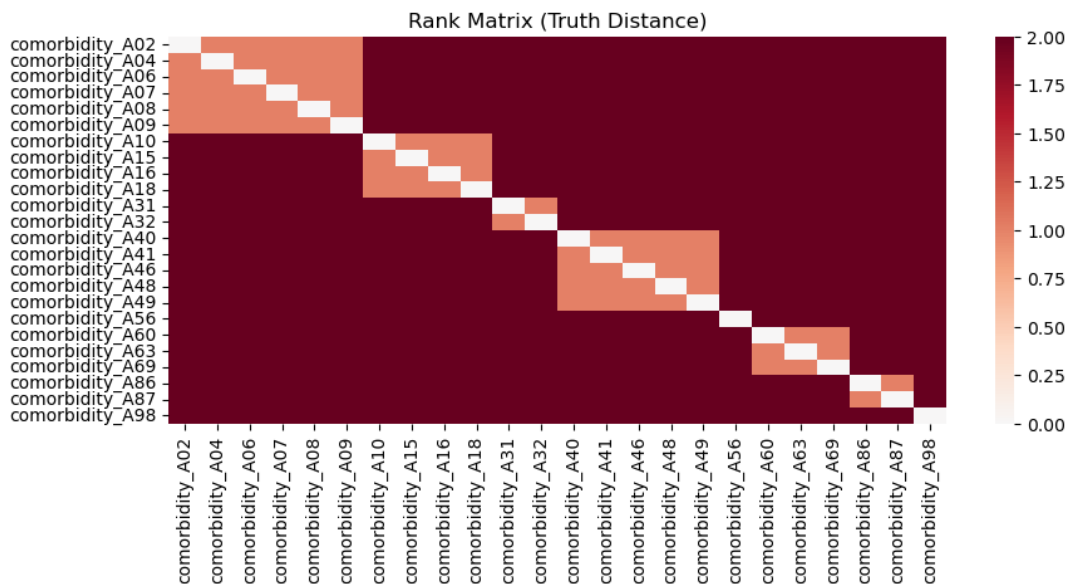


Figure 32: True Distance in Group A

truth data, the learned embeddings presented in Figure 33 show a resemblance in the

trend, even though the distance values are different depending on how we define the labels. To be even more specific, we calculate the difference in these two and plot them in Figure 34. And once again, the results are very desirable when we see the same pattern as expected from the hierarchy here. Building upon these codes of A

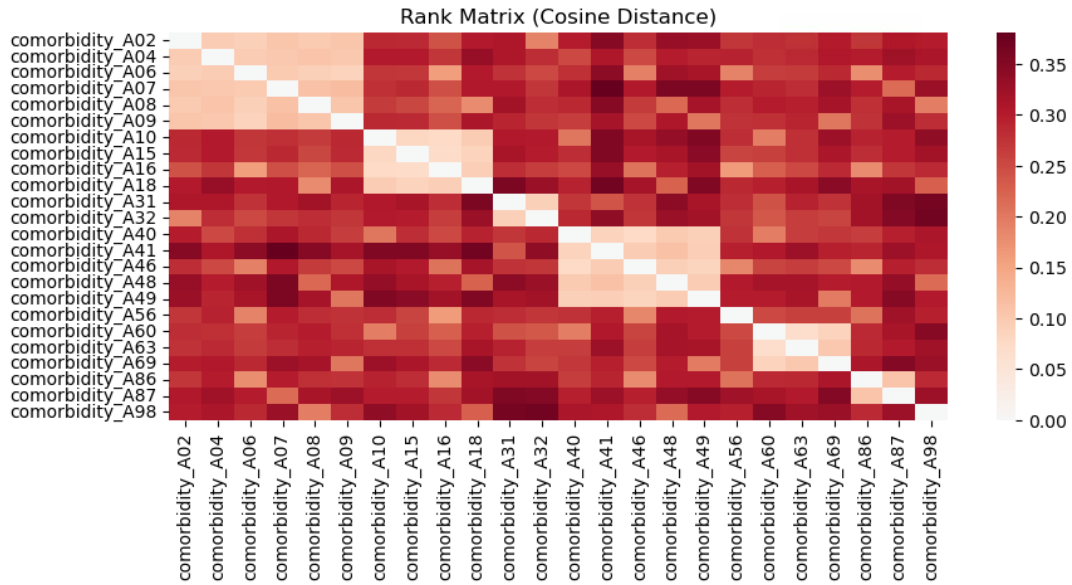


Figure 33: Cosine Distance in Group A

chapter, we may extend our understanding to Figures 35, 36, and 37 where all the codes are plotted, and again, we observe the similar profile of the codes. All of these show that the embeddings vector of the ICD-10 codes can resemble their true nature of levels. Another experiment to test with this pretrained model is the dendrogram of the embedding vectors, as presented in Figure 38. The problem here again is that we have too many codes to put in; but the main idea is that each color here presents a chapter of codes (the first character). Based on this dendrogram [45], the hierarchy of the ICD-10 codes are replicated very well by the learned embeddings, clearly connecting the codes within the same group first. Therefore, the chapters are also clearly clustered. And even though we have some unnecessary links on the top, we can still see that the process of merging codes into one group shows the exact behaviour of hierarchical structure we desire. A subset A, B, E of these codes are on display in the Figure 39 for a closer look on how the embeddings are grouped in a dendrogram. As can be clearly seen from the graph, each code is perfectly connected with others in the group, whether that is the first level (first digit) or second level (second digit). Additionally, to further validate the performance of ICD-10 code model, we use the Normalized Mutual Information NMI [42] as another metrics score. Given a ground truth dataset and a learned one, this score ranges from 0 to 1, with 0 being total difference and 1 being complete match between the two datasets. We have two parallel sets, one containing all the codes in order and the other holding the corresponding embedding vectors. We analyze the NMI in two ways: the first level and the second level. For the first level, we extract only the first digit of the true codes, and cluster the embeddings

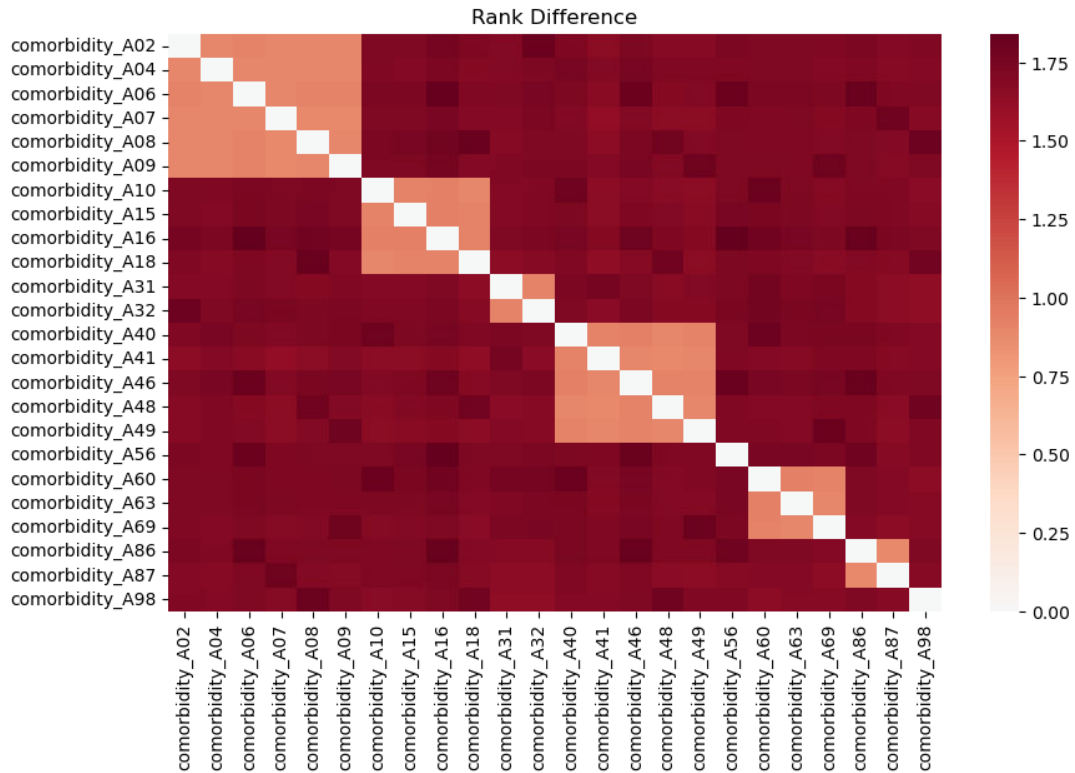


Figure 34: Difference between True and Cosine Distances in Group A

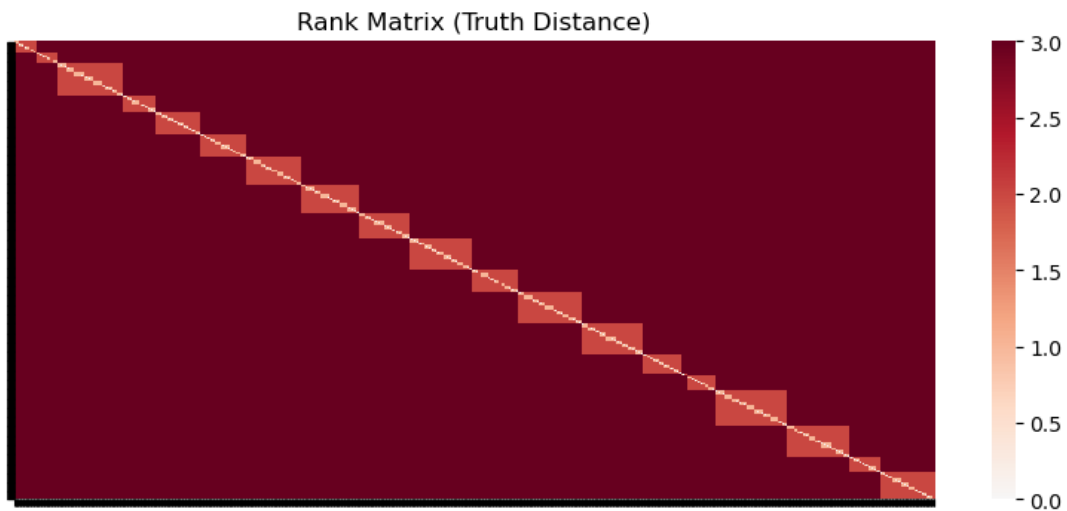


Figure 35: True Distance in All Groups

based on the number of such unique characters. Similarly, for the second level, we extract the first and second digits to be the labels and use their number to guide the clustering process. The results for both of the cases are 1.0, indicating the exact match between the true labels and the learned embeddings. This outcome once again shows that the model has recognized the pattern of ICD-10 codes very well and is able to

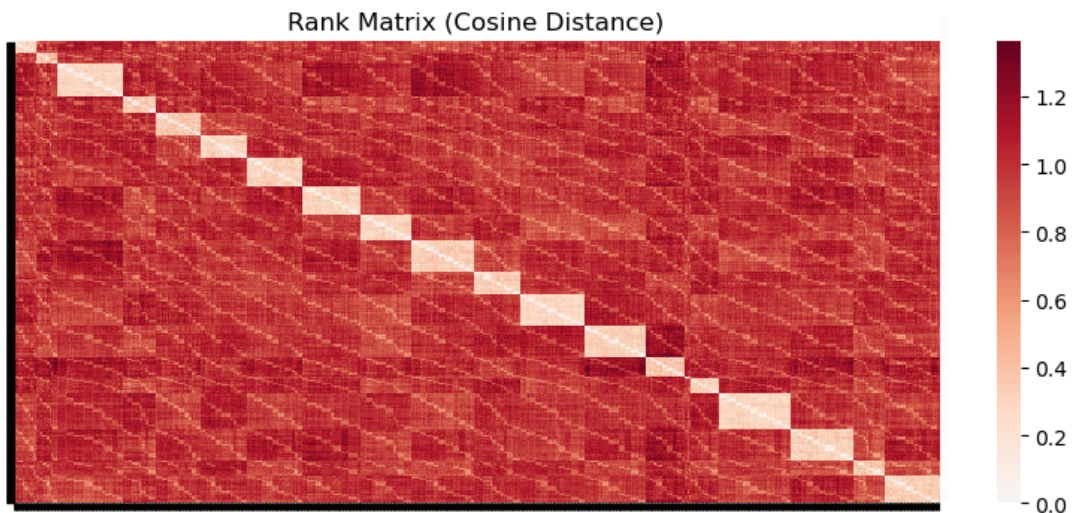


Figure 36: Cosine Distance in All Groups

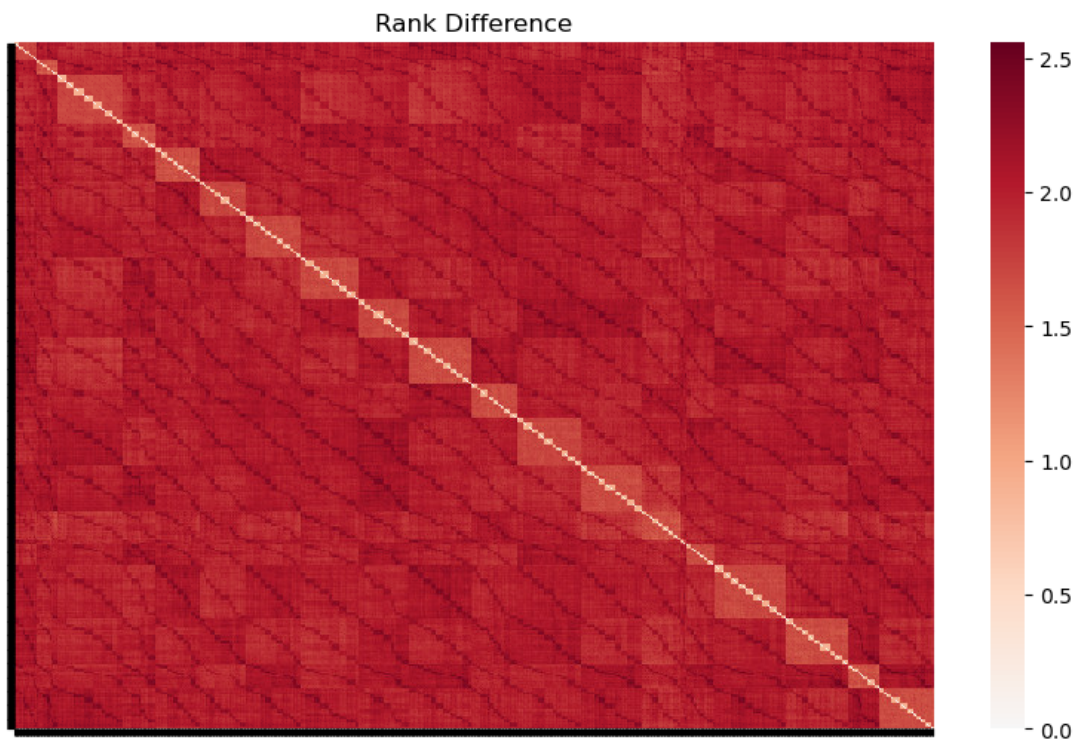


Figure 37: Difference between True and Cosine Distances in All Groups

reconstruct such hierarchical structure.

Using the same logic for medication codes pretrained model, we also achieve satisfying results which can clearly distinguish the hierarchical structure within these codes. This can be seen from Figure 40 where a subset of medication codes is calculated with cosine similarity and plotted against each other.

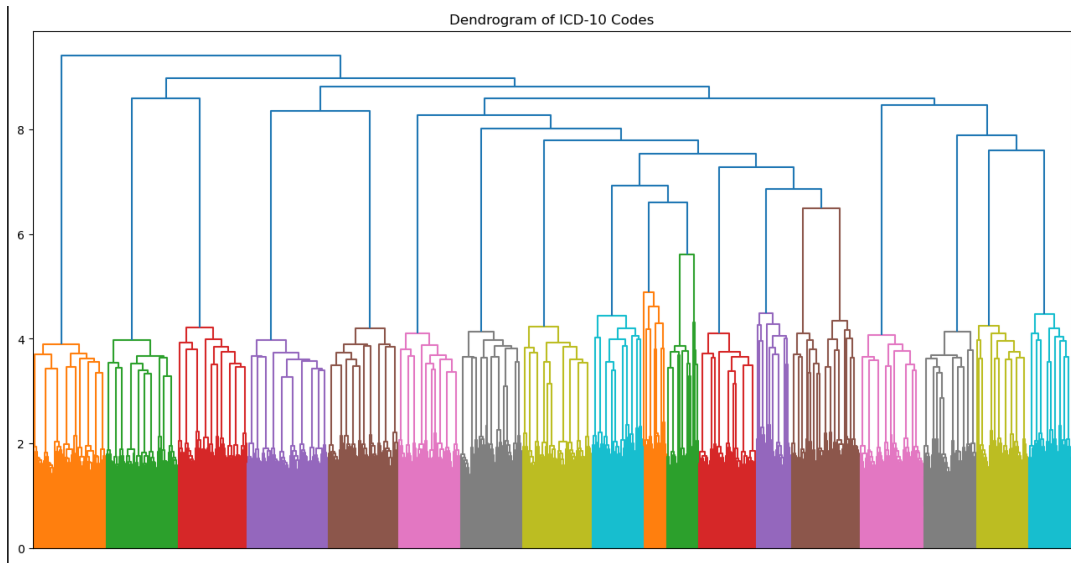


Figure 38: Dendrogram of All Groups

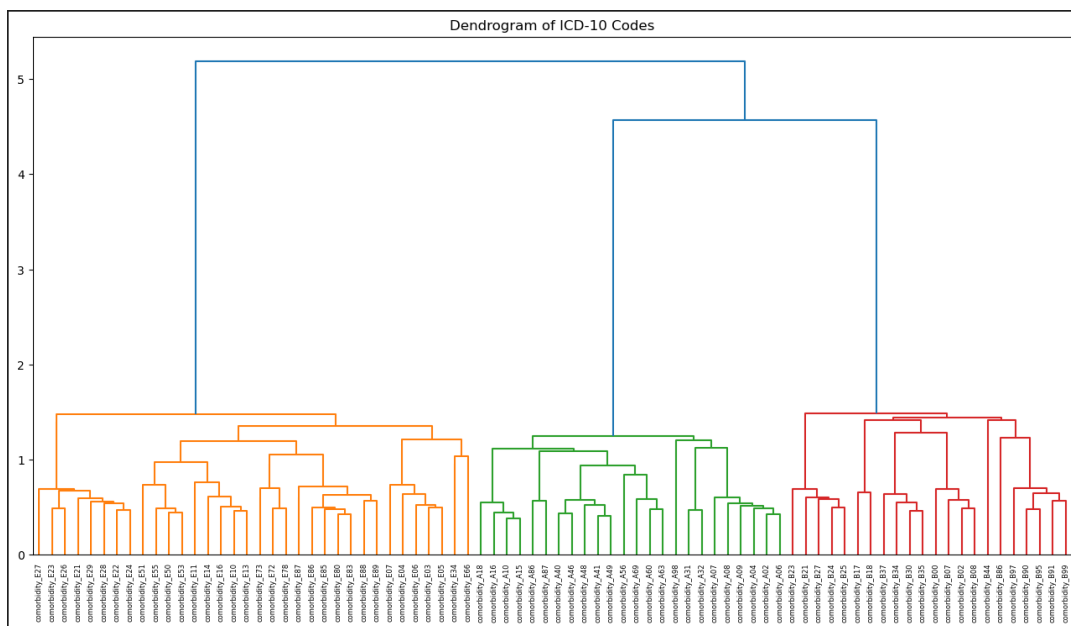


Figure 39: Dendrogram of 3 Groups

5.5.2 Representation Clustering

To see how the learned latent representations of the patients perform, we pass the test dataset through the completely trained model to obtain the embedding vectors in high dimension with full semantic and temporal information. After that, we conduct the dimension reduction using UMAP method [46], which is a novel scaling technique compatible with the complex of real-world data, to obtain the data in low dimension, to be more specific, in 2D space. This dimension reduction allows us to visualize the data and perform patient clustering based on the latent representations. We explore

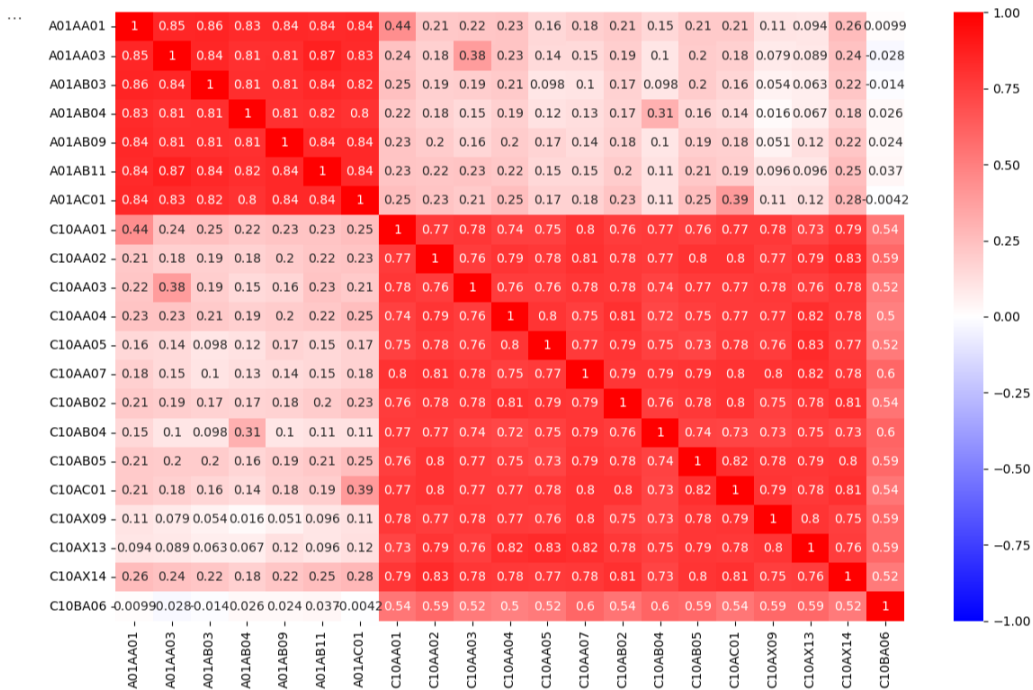


Figure 40: Samples of medication codes

various clustering methods to see how well the clusters align within and between each other, as well as how satisfying the final clustering results are.

First of all, we choose an arbitrary number of clusters that is not too small nor too big and perform the K-Means clustering [47]. K-Means clustering bases its clustering decision on centroids (average points), hence grouping each point to the closest centroid. We set the number of clusters to be 6 so that there is a just enough number of clusters to do some enrichments. By using this approach, the resulting clustering contours are presented in Figure 41. It is clearly that the K-Means algorithm has decided and grouped the patients very clearly, with only some small overlaps between clusters. With the patients divided into 6 parts, we can look into each of them and see what are the most common diagnosis codes, as visualized in Figure 42 with multiple plot each representing a cluster. As can be seen, the patients are clustered with regard to the codes very well, with each of these clusters profiling a prominent set of diagnosis. Figure 43 presents the mean value of several laboratory tests results among the clusters. We can observe the difference of these tests values across the clusters. At this point, we can see that the model has learned the semantics information of the patients' trajectory through the training process and now is able to tell apart similar and non-similar patients.

To further assess the clustering quality, we choose to analyze the average cosine similarity between patients within a cluster, together with the similarity between centroids of all clusters. The centroid is defined to be the average of all the reduced patients' embedding vectors, as this can capture and summarize the information more condensed. Using the same setting as described above, we achieve the following within-cluster similarity scores in Table 3 and between-cluster centroid similarity in

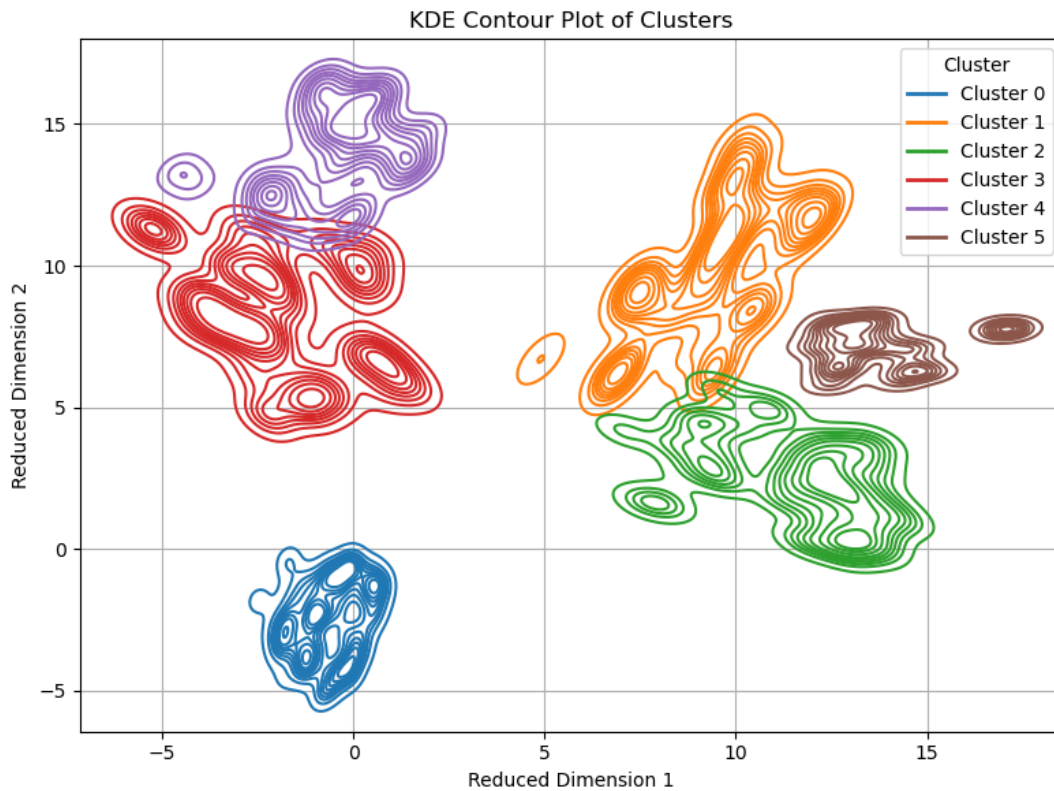


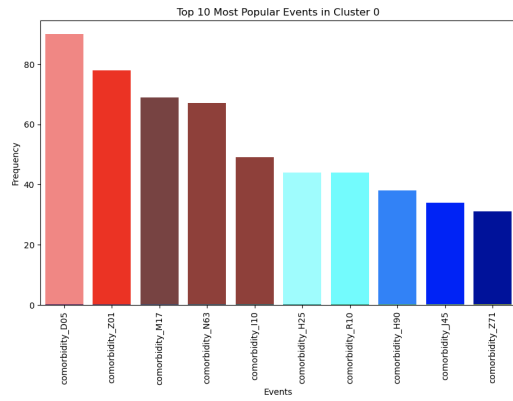
Figure 41: Contour of KMeans with 6 Clusters

Figure 44.

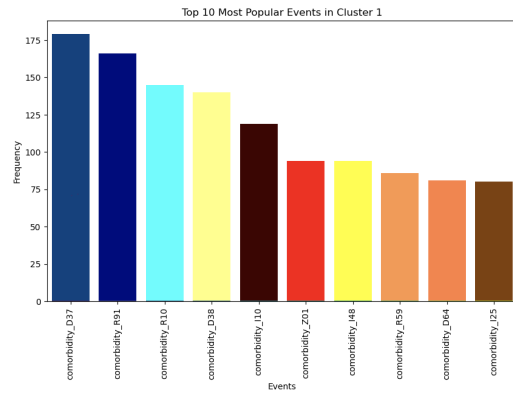
Cluster	Value
0	0.80
1	0.97
2	0.97
3	0.99
4	0.99
5	0.99

Table 3: Average Similarity within Clusters of KMeans with 6 Clusters

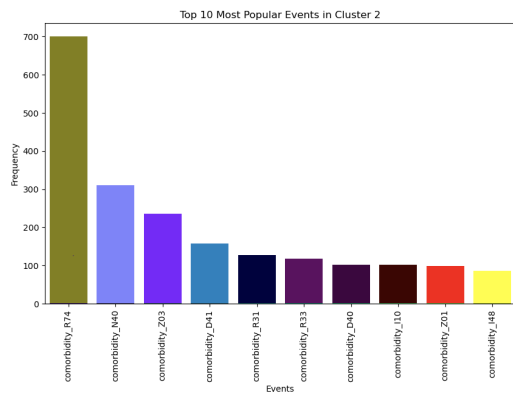
The intra-cluster similarities are very high, indicating that the model has done a good job on pulling similar patients close to each others. However, when we look at the inter-cluster similarity, it seems that there are some clusters are too similar, which could mean that we are not dividing the patients optimally. Then, the next step is to try K-Means clustering with a smaller number of clusters, which we now choose to be 4. Table 4 explains what kind of results this setting is giving us. Once again, the intra-cluster similarity is quite decent, with the majority of them reaching 0.95 or above. Looking at the inter-cluster scores in Figure 45, we notice that the situation where multiple clusters share high similarity no longer exists, with only just one case



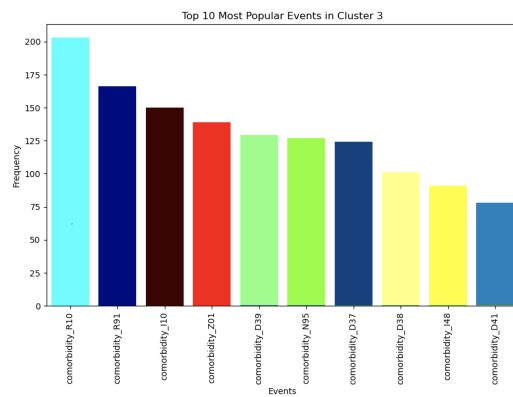
(a) Cluster 0



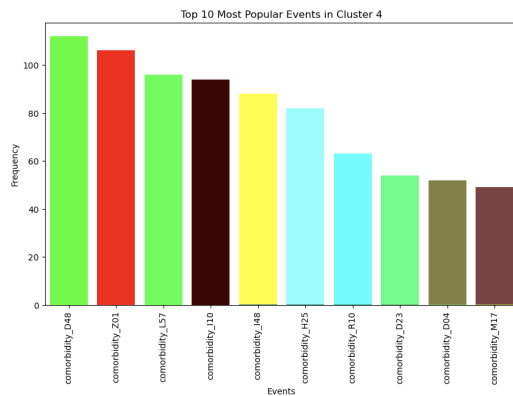
(b) Cluster 1



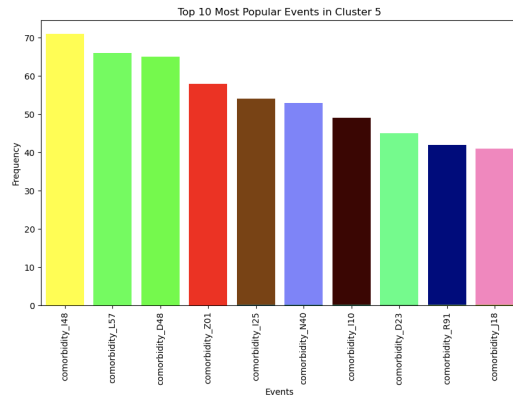
(c) Cluster 2



(d) Cluster 3



(e) Cluster 4



(f) Cluster 5

Figure 42: Most common diagnosis codes in each cluster.

of Cluster 1 and 3 having a bit higher score than others. This means that the clustering has now divided the patients more effectively, reducing the risk of leaking important patients' information from one cluster to another.

Now that we know 4 is a reasonable choice of cluster numbers, we move forward with another method of clustering specializing on hierarchy - Agglomerative Clustering [48]. Agglomerative clustering features the hierarchy in partitioning, which can possibly

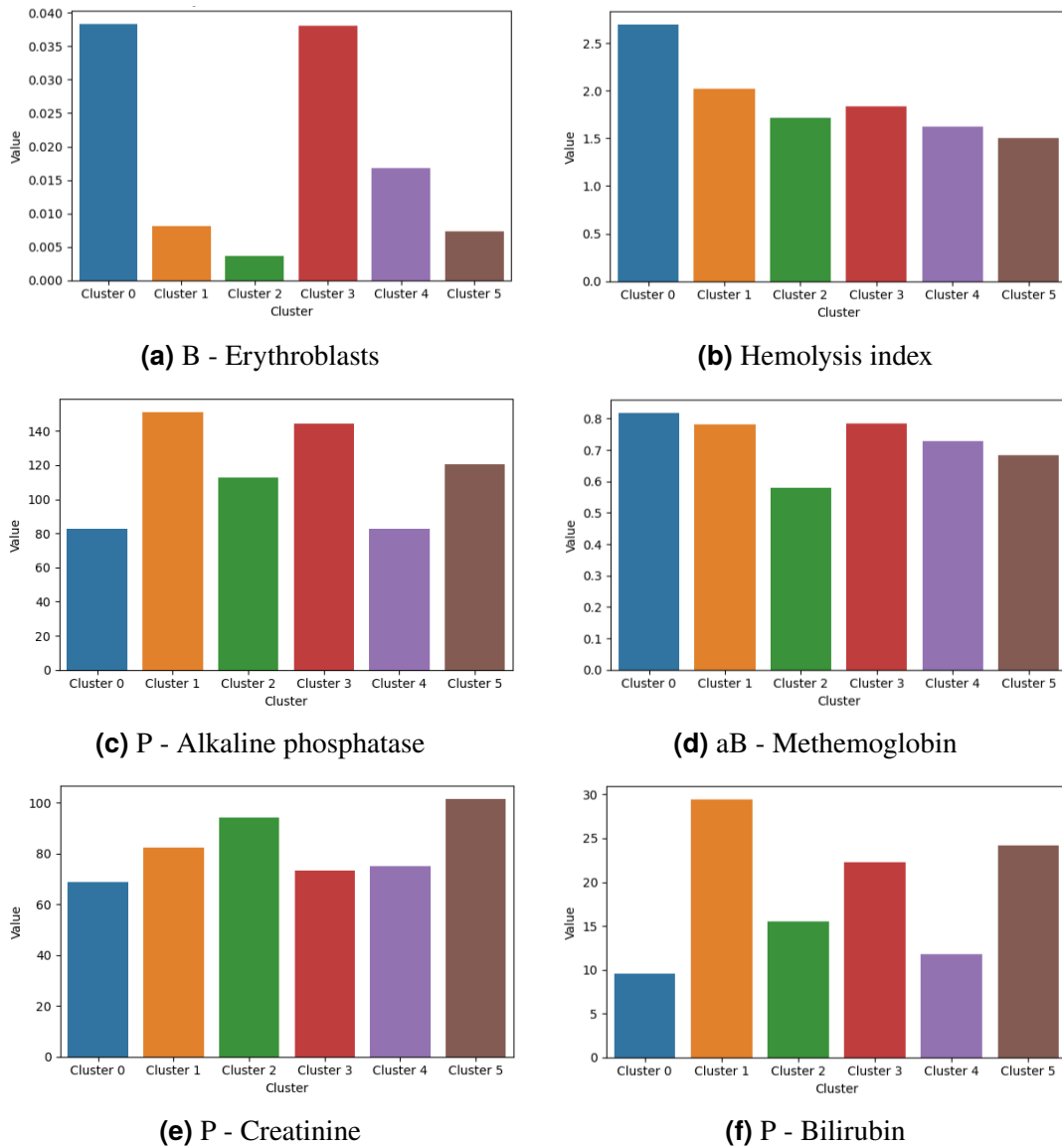


Figure 43: Mean values of various lab results.

Cluster	Value
0	0.77
1	0.99
2	0.96
3	0.98

Table 4: Average Similarity within Clusters of KMeans with 4 Clusters

find the patterns and sub-patterns between patients trajectories. Figure 46 tells us that this approach leads to a slightly different way of clustering the patients. However, both these methods succeed in ensuring the similarity within the cluster, with Agglomerative

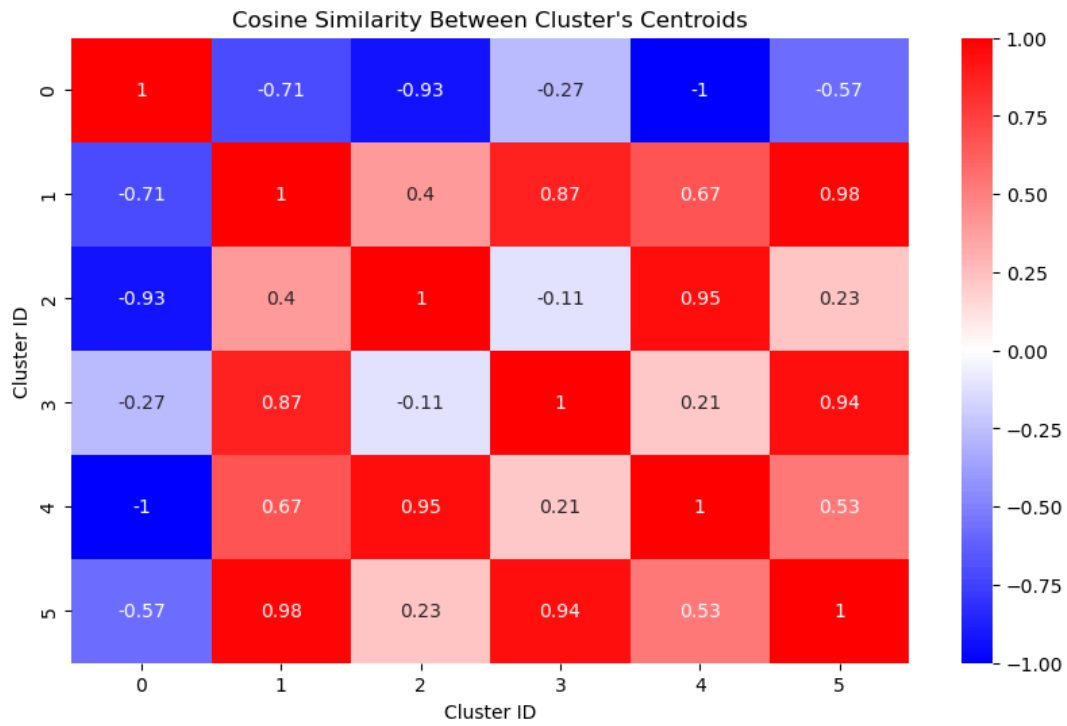


Figure 44: Similarity between Clusters of KMeans with 6 Clusters

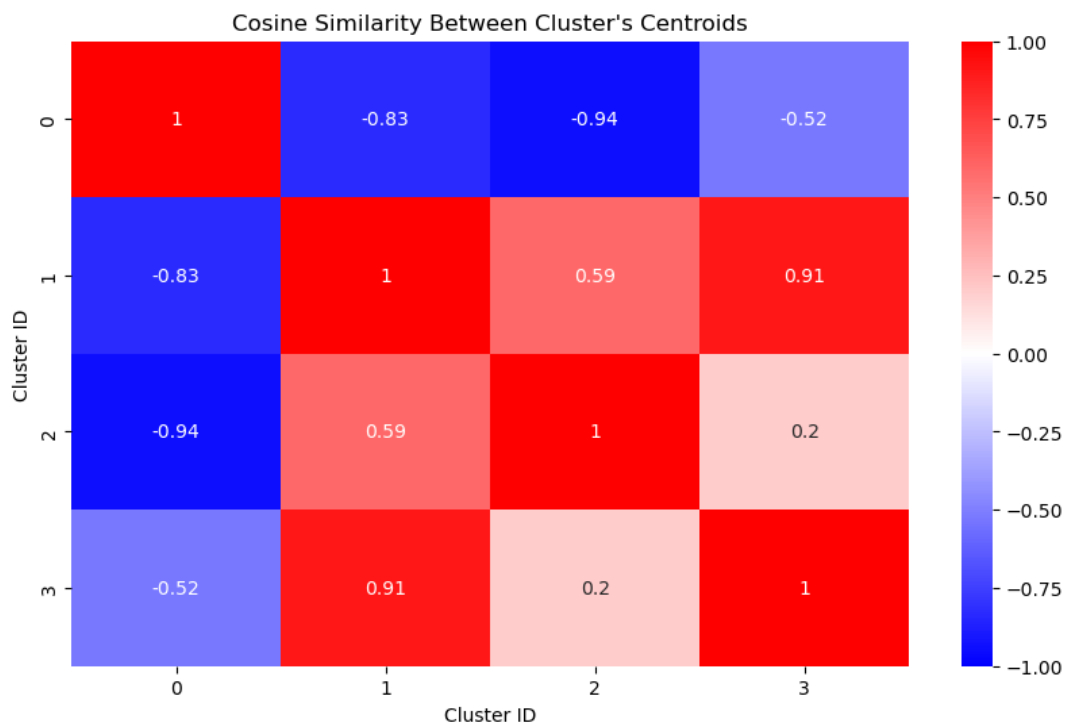


Figure 45: Similarity between Clusters of KMeans with 4 Clusters

Cluster	Value
0	0.94
1	0.98
2	0.84
3	0.99

Table 5: Average Similarity within Clusters of Agglomerative with 4 Clusters

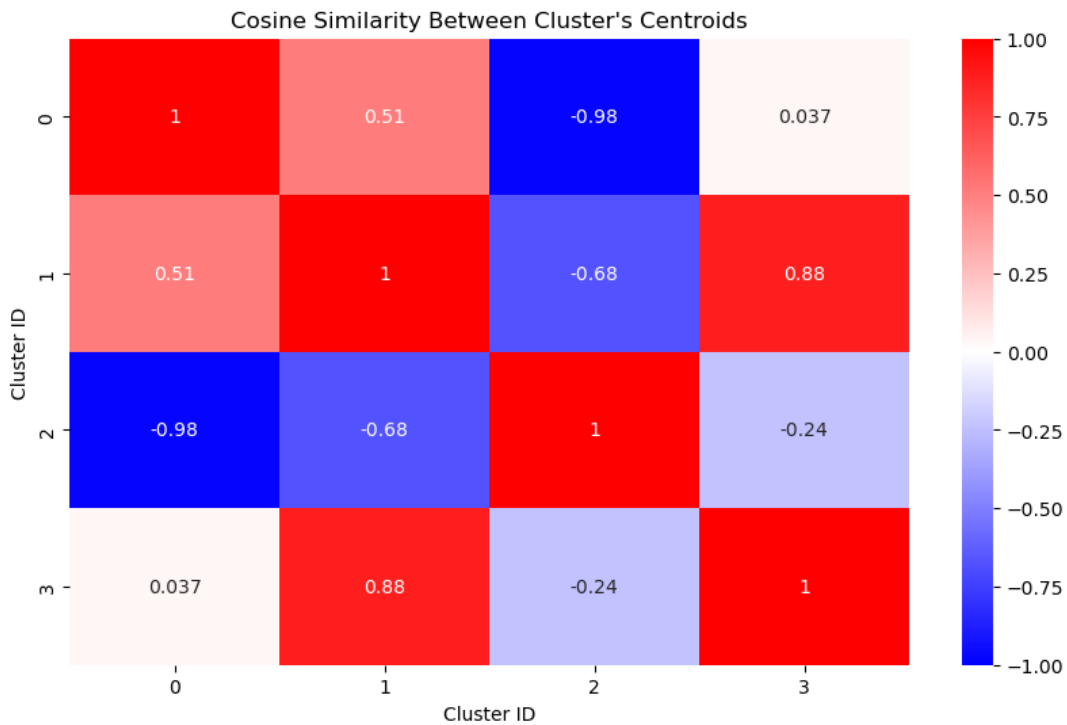


Figure 46: Similarity between Clusters of Agglomerative with 4 Clusters

one achieving even a higher score in general compared to K-Means one as presented in Table 5. Again, we see that, with 4 clusters, the inter-cluster similarity looks appropriate with no pair of clusters sharing a high metrics score.

With these first-hand latent representations from contrastive learning, we can further utilize them in downstream tasks such as survival analysis and time-to-event prediction. The simple metrics used to evaluate these clustered embedding vectors show that the model is learning very well to separate patients who are similar and who are different. With more finetuning tasks and objectives, we believe the patients' latent representations can perform decently with regard to temporal and semantic information in each profile.

6 Conclusion

This final chapter concludes the results as well as consolidates the research goals that we want to achieve. There are also future directions discussed should there be further instance of this model in the future.

6.1 Discussion

In this paper, we have performed multiple experiments with the aim of answering our research questions regarding contrastive learning on EHR data. For a multimodal dataset, we have discussed different augmentation methods for various modalities and validated that there are many ways to perform augmentation as all the noise introduced would eventually be helpful for contrastivity. A novelty with regard to the EHR data that we pay attention to is the hierarchical structure within ICD-10 and medication codes that have yet to be studied in detail previously. The pretrained models for learning this kind of structure have been proved to work perfectly when the codes can be clustered according to their group, and the cosine similarity between the codes also tell which group or subgroup they belong to. Additionally, to solve the problem of imputation or interpolation with sparse and irregular data, we utilize the TAAT model [39] which learns the timestamps in pyramid levels to achieve the temporal attention. Its transformer-based architecture also helps capture the semantic information from sequence data, and this applies very well to patient trajectories where there are many events with different types and values. Last but not least, we provide a framework following SimCLR [11] to put these all together to perform contrastive learning using EHR data, allowing for learning the personalized representations from multimodal records of patients. The preliminary results show that this is a promising approach to further solve the problem of generalization in machine learning within the medical field.

6.2 Limitations and Future Directions

Despite the prospects and results this work brings back, there are several places for improvement considering the future directions. One important aspect is that how to increase the capacity of the model when learning the contrastivity between patients, meaning that how we can provide more inputs within a batch for it. One solution that ModernBERT [26] has implemented is the unpadding method, but unfortunately, we cannot make it to work this time. If that can be applied successfully, we will be able to have twice the number of inputs per batch compared to what we have now.

Another thing that would prove useful is to assess the model in further downstream tasks such as survival analysis, predictive coding, time-to-event prediction. For now, we only analyze the latent representations of patients obtained from the model and perform some simple clustering methods without further utilizing its potential.

When it comes to such clustering, we notice one point during our examination at the clusters: the demographics information is quite dominant in every cluster. Our current implementation of the architecture makes the model pay a lot of attention to

the demographics information, as this is repeated for every timestamp in the sequence. This action boosts its appearance considerably, somewhat confusing the model that this is the most important to learn. Should the model be trained again, we can try other approaches of integrating this piece of information more subtly.

In conclusion, we believe our work has some unique and meaningful contributions to the current state of contrastive learning in medical field with promising results that have been validated in this paper. Besides, there is still room for further enhancement should there be future instances of the framework and model. With proper utilization and adjustments, the model would prove to be helpful in an actual setting of a hospital with regard to the personalization of each patient.

7 References

- [1] G. Wysocki, S. Orkisz, K. Balawender, M. Golberg, and A. Żytkowski, “The human body-not only a biological entity”, *Translational Research in Anatomy*, vol. 34, p. 100 270, 2024.
- [2] J. T. Yurkovich, Q. Tian, N. D. Price, and L. Hood, “A systems approach to clinical oncology uses deep phenotyping to deliver personalized care”, *Nature Reviews Clinical Oncology*, vol. 17, no. 3, pp. 183–194, 2020.
- [3] E. Choi, M. T. Bahadori, A. Schuetz, W. F. Stewart, and J. Sun, “Doctor ai: Predicting clinical events via recurrent neural networks”, in *Machine learning for healthcare conference*, PMLR, 2016, pp. 301–318.
- [4] E. Choi et al., “Multi-layer representation learning for medical concepts”, in *proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1495–1504.
- [5] Y. Li et al., “Behrt: Transformer for electronic health records”, *Scientific reports*, vol. 10, no. 1, p. 7155, 2020.
- [6] A. Vaswani et al., “Attention is all you need”, *Advances in neural information processing systems*, vol. 30, 2017.
- [7] T. Cai, F. Huang, R. Nakada, L. Zhang, and D. Zhou, “Contrastive learning on multimodal analysis of electronic health records”, *arXiv preprint arXiv:2403.14926*, 2024.
- [8] S. Ketabi and D. Ramachandram, “Bridging electronic health records and clinical texts: Contrastive learning for enhanced clinical tasks”, *arXiv preprint arXiv:2505.17643*, 2025.
- [9] E. Choi, M. T. Bahadori, J. Sun, J. Kulas, A. Schuetz, and W. Stewart, “Retain: An interpretable predictive model for healthcare using reverse time attention mechanism”, *Advances in neural information processing systems*, vol. 29, 2016.
- [10] Y. Tang, Y. Zhang, and J. Li, “A time series driven model for early sepsis prediction based on transformer module”, *BMC Medical Research Methodology*, vol. 24, no. 1, p. 23, 2024.
- [11] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, “A simple framework for contrastive learning of visual representations”, in *International conference on machine learning*, PmLR, 2020, pp. 1597–1607.
- [12] T. Seymour, D. Frantsvog, and T. Graeber, “Electronic health records (ehr)”, *American Journal of Health Sciences*, vol. 3, no. 3, p. 201, 2012.
- [13] W. H. Organization, *International statistical classification of diseases and related health problems 10th revision*, 2019. [Online]. Available: <https://icd.who.int/browse10/2019/en>.
- [14] W. H. Organization, *Anatomical therapeutic chemical (atc) classification*. [Online]. Available: <https://www.who.int/tools/atc-ddd-toolkit/atc-classification>.

- [15] R. Hadsell, S. Chopra, and Y. LeCun, “Dimensionality reduction by learning an invariant mapping”, in *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR’06)*, IEEE, vol. 2, 2006, pp. 1735–1742.
- [16] A. Mumuni and F. Mumuni, “Data augmentation: A comprehensive survey of modern approaches”, *Array*, vol. 16, p. 100 258, 2022.
- [17] Z. Wang et al., “A comprehensive survey on data augmentation”, *arXiv preprint arXiv:2405.09591*, 2024.
- [18] E. Hoffer and N. Ailon, “Deep metric learning using triplet network”, in *International workshop on similarity-based pattern recognition*, Springer, 2015, pp. 84–92.
- [19] K. Sohn, “Improved deep metric learning with multi-class n-pair loss objective”, *Advances in neural information processing systems*, vol. 29, 2016.
- [20] A. v. d. Oord, Y. Li, and O. Vinyals, “Representation learning with contrastive predictive coding”, *arXiv preprint arXiv:1807.03748*, 2018.
- [21] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation”, Tech. Rep., 1985.
- [22] Y. LeCun et al., “Backpropagation applied to handwritten zip code recognition”, *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.
- [23] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding”, in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.
- [24] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al., “Improving language understanding by generative pre-training”, 2018.
- [25] A. Dosovitskiy et al., “An image is worth 16x16 words: Transformers for image recognition at scale”, *arXiv preprint arXiv:2010.11929*, 2020.
- [26] B. Warner et al., “Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference”, *arXiv preprint arXiv:2412.13663*, 2024.
- [27] N. Shazeer, “Glu variants improve transformer”, *arXiv preprint arXiv:2002.05202*, 2020.
- [28] T. Dao, “Flashattention-2: Faster attention with better parallelism and work partitioning”, *arXiv preprint arXiv:2307.08691*, 2023.
- [29] J. Zeng et al., “Boosting distributed training performance of the unpadded bert model”, *arXiv preprint arXiv:2208.08124*, 2022.
- [30] S. Tipirneni and C. K. Reddy, “Self-supervised transformer for sparse and irregularly sampled multivariate clinical time-series”, *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 16, no. 6, pp. 1–17, 2022.

- [31] A. E. Johnson et al., “Mimic-iii, a freely accessible critical care database”, *Scientific data*, vol. 3, no. 1, pp. 1–9, 2016.
- [32] I. Silva, G. Moody, D. J. Scott, L. A. Celi, and R. G. Mark, “Predicting in-hospital mortality of icu patients: The physionet/computing in cardiology challenge 2012”, in *2012 computing in cardiology*, IEEE, 2012, pp. 245–248.
- [33] S. Hochreiter and J. Schmidhuber, “Long short-term memory”, *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [34] C. Zang and F. Wang, “Scehr: Supervised contrastive learning for clinical risk prediction using electronic health records”, in *Proceedings. IEEE International Conference on Data Mining*, vol. 2021, 2021, p. 857.
- [35] Y. Ma et al., “Global contrastive training for multimodal electronic health records with language supervision”, *arXiv preprint arXiv:2404.06723*, 2024.
- [36] U. of Florida Health, *Uf health*. [Online]. Available: <https://ufhealth.org/>.
- [37] Y. Wu, J. Zhang, X. Chen, X. Yao, and Z. Chen, “Contrastive learning with large language models for medical code prediction”, *Expert Systems with Applications*, vol. 277, p. 127 241, 2025.
- [38] J. Achiam et al., “Gpt-4 technical report”, *arXiv preprint arXiv:2303.08774*, 2023.
- [39] L. Deng et al., “Time-aware attention-based transformer (taat) for cloud computing system failure prediction”, in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, Association for Computing Machinery, 2024.
- [40] P. Micikevicius et al., “Mixed precision training”, *CoRR*, 2017.
- [41] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.
- [42] A. Strehl and J. Ghosh, “Cluster ensembles—a knowledge reuse framework for combining multiple partitions”, *Journal of machine learning research*, vol. 3, no. Dec, pp. 583–617, 2002.
- [43] Findata, *Data permit hus/355/2025*, Finnish Social and Health Data Permit Authority Findata.
- [44] W. I. D. Mining, “Data mining: Concepts and techniques”, *Morgan Kaufmann*, vol. 10, no. 559-569, p. 4, 2006.
- [45] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009.
- [46] L. McInnes, J. Healy, and J. Melville, “Umap: Uniform manifold approximation and projection for dimension reduction”, *arXiv preprint arXiv:1802.03426*, 2018.

- [47] J. B. McQueen, “Some methods of classification and analysis of multivariate observations”, in *Proc. of 5th Berkeley Symposium on Math. Stat. and Prob.*, 1967, pp. 281–297.
- [48] A. K. Jain and R. C. Dubes, *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.