Encoding FHIR Medical Data for Transformers

by

Trevor Yu

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Applied Science
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2025

© Trevor Yu 2025

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The open source Fast Healthcare Interoperability Resources (FHIR) data standard is becoming increasingly adopted as a format for representing and communicating medical data. FHIR represents various types of medical data as resources, which have a standardized JSON structure. FHIR boasts the advantage of interoperability and can be used for electronic medical record storage and, more recently, machine learning analytics. Recent trends in the machine learning field have been the development of large, foundation models that are trained on large volumes of unstructured data. Transformers are a deep neural network architecture for sequence modelling and have been used to build foundation models for natural language processing. Text is input to transformers as a sequence of tokens. Tokenization algorithms break text into discrete chunks, called tokens. Using language tokenizers on FHIR JSON data is inefficient, producing several hundred text tokens per resource. Patient records may contain several thousand resources, which overall exceeds the total number of tokens that most text transformers can handle. Additionally, discrete encoding of numeric and time data may not be appropriate for these continuous quantities.

In this thesis, I design a tokenization method that operates on data using the open source Health Level 7 FHIR standard. This method takes JSON returned from a FHIR server query and assigns tokens to chunks of JSON, based on FHIR data structures. The FHIR tokens can be used to train transformer models, and the methodology to train FHIR transformer models on sequence classification and masked language modelling tasks is presented. The performance of this method on the open source MIMIC-IV FHIR dataset is validated for length-of-stay prediction (LOS) and mortality prediction (MP) tasks. In addition, I explore methods for encoding numerical and time-delta values using continuous vector encodings rather than assigning discrete tokens to values. I also explore using compression methods to reduce the long sequence lengths.

Previous works using MIMIC-IV have reported their performance on the LOS and MP tasks using XGBoost models, which use bespoke feature encodings. The results show that the FHIR transformer can perform the LOS task better than an XGBoost model, but the transformer performs worse at the MP task. None of the continuous encoding methods perform significantly better than discrete encoding methods, but they are not worse either. Compression methods provide a performance improvement on long sequence lengths in both accuracy and inference speed. Since performance is task dependent, future research should validate the performance of this method on other datasets and tasks. MIMIC-IV is too small to see benefits of pre-training, but if a larger dataset can be obtained, the methodology developed in this work could be applied towards creating a large FHIR foundation models.

Acknowledgements

This research was sponsored by Smile Digital Health. I would like to thank my fellow graduate students and co-op students in the Medical AI Lab for their passion in advancing medical AI research and creating a friendly lab workplace. I would like to thank my thesis readers for their time and feedback. I would especially like to thank my supervisor, Bryan Tripp, for his guidance and insight through my degree. I would like to thank my friends for all the side quests, big and small, that have made my time during my degree all the more enjoyable. Lastly, I would like to thank my family for their love and support in pursuing my degree away from home.

Table of Contents

Author's Declaration ii

Abstract iii

Acknowledgements iv

List of Figures ix

List of Tables xiii

1 Introduction 1

1.1 Motivation 1

1.2 Contributions 3

2 Background 5

2.1 Transformer Architecture 5

2.2 Language Modelling 8

2.3 FHIR Data Model 9

2.3.1 FHIR datatypes 9

2.4 Medical Language Models 12

2.5 Med-BERT: pretrained contextualized embeddings on large-scale structured electronic health records for disease prediction 14

2.6 Scalable and accurate deep learning with electronic health records 16
2.7 Integrated multimodal artificial intelligence framework for healthcare applications 17
2.8 Spatial Semantic Pointers 19
2.9 Legendre Memory Unit 21

3 Data **24**
3.1 Datasets 24
3.1.1 MIMIC-IV FHIR Resource Types 26
3.1.2 Resource Bundling 27
3.1.3 Relevant Resource Fields 27
3.2 Tokenizing FHIR Resources 30
3.2.1 Elemental Data Type Representations 30
3.2.2 Tokenization Example 34
3.2.3 Post-tokenization Processing 35
3.3 HAIM MIMIC Dataset 35
3.4 Extended Context MIMIC Dataset 40

4 Transformer Model **43**
4.1 Transformer Encoder 43
4.2 Encoding Numeric Values 45
4.3 Discrete Numeric Encoding 46
4.3.1 Quantile Bins 46
4.3.2 Normalized Bins 47
4.4 Continuous Numeric Encoding 47
4.4.1 Linear Encoding 48
4.4.2 Multi-layer Perceptron 48
4.4.3 Gaussian Basis 48

4.4.4 SSP Encoding . . . 49
4.4.5 SSP Similarity Basis . . . 50
4.5 Encoding Timestamps . . . 51
4.5.1 Round to Integer . . . 51
4.5.2 Logspace Bins . . . 52
4.5.3 Percentile Bins . . . 53
4.5.4 Continuous Encoding . . . 53
4.6 Sequence Compression Layer . . . 53
4.6.1 Sum Compression . . . 54
4.6.2 LMU Compression . . . 55
4.6.3 Cross-attention Compression . . . 56
4.7 Classification Tasks . . . 57

5 Experimental Methods **61**
5.1 Metrics . . . 61
5.2 Training Hyperparameters . . . 62
5.3 Experimental Design . . . 63
5.3.1 Baseline Transformer on HAIM Dataset . . . 63
5.3.2 Numeric and Time Encoding . . . 64
5.3.3 Compression Methods on Extended Context Dataset . . . 65

6 Results **68**
6.1 Baseline Transformer on HAIM Dataset . . . 68
6.2 Continuous Numeric Encoding . . . 68
6.3 Continuous Time Encoding . . . 70
6.4 Combined Continuous Encoding . . . 72
6.5 Compression Layers . . . 73

7 Discussion **84**

7.1 Interpretation of Findings 84

7.2 Benefits of Methodology 85

7.3 Limitations of Methodology 87

7.4 Directions for Future Research 88

7.5 Conclusion 89

References **91**

APPENDICES **104**

A Pre-training **105**

A.1 Methodology 105

A.1.1 MLM with Continuous Encoding 106

A.1.2 MLM with Compression 106

A.2 Pre-training MIMIC Dataset 108

A.3 Experimental Methods 110

A.4 Results and Discussion 110

List of Figures

2.1 Example Observation resource in JSON format from the MIMIC FHIR dataset. This resource represents a glucose blood test. 10

2.2 Example Observation resource with the primitive datatypes highlighted. Numeric datatypes are highlighted in orange, strings are highlighted in cyan, timestamps are highlighted in yellow, and URIs are highlighted in green. . 11

2.3 Example Observation resource with some general-purpose datatypes highlighted. Coding datatypes are highlighted in yellow, Quantity datatypes are highlighted in cyan, and Reference datatypes are highlighted in green. . . . 13

3.1 Example Observation resource with the tokenizer's elemental datatypes highlighted. Coding datatypes are highlighted in green, numeric datatypes are highlighted in orange, Reference datatypes are highlighted in magenta, strings are highlighted in cyan, and timestamps are highlighted in yellow. . 36

3.2 Example Observation resource with the relevant fields bolded and the associated datatypes highlighted. Note there is less data highlighted here compared to Figure 3.1. Coding datatypes are highlighted in green, numeric datatypes are highlighted in orange, Reference datatypes are highlighted in magenta, strings are highlighted in cyan, and timestamps are highlighted in yellow. 37

3.3 Example Observation resource tokenized and the associated datatypes highlighted. Sentinel tokens for numeric data (60) and time data (61) would be replaced by representations based on the encoding method later in the model. Coding datatypes are highlighted in green, numeric datatypes are highlighted in orange, Reference datatypes are highlighted in magenta, strings are highlighted in cyan, and timestamps are highlighted in yellow. 38

3.4 Final tokenized representation of several resources tokenized. An example Patient (yellow), Encounter (cyan), Observation (green), and Procedure (magenta) resource are shown. The resource type IDs are used to create resource type embeddings, while the resource enumeration is used for compression methods to identify which subsequences of tokens belong to which resource. The numeric vals and codes are used later to determine statistics when making numeric encoding representations, described in section 4.2. Time vals are used when making time encoding representations, described in section 4.5. 39

4.1 Base transformer architecture based on the BERT model using Transformer ++ improvements. I use a hidden dimension D=768 and 12 layers. The batch size (B) and sequence length (L) are different depending on the task and dataset. 44

4.2 Distribution of normalized values after applying $x = x - \text{median}/\text{spread}$ . . 58

4.3 Continuous encoding basis functions and dot product similarity function $y = f(x) \cdot f(0)$ over the range $[-10, 10]$. 59

4.4 Diagram of the continuous encoding process. In this example, normalized numeric base values are passed through the continuous encoding layer that creates a sequence of vectors from a sequence of 1-dimensional values. These vector encodings are inserted back into the embedding vector sequence at the positions of the numeric sentinel tokens (token 60). A similar process can be applied if performing continuous encoding of time values. 59

4.5 Diagram of the resource compression process. The resource enumeration indicates which embedding vectors are grouped together. The compression function is applied independently to each group of vectors. 60

6.1 Test metrics box plots for the baseline transformer and XGBoost models on the HAIM dataset. 70

6.2 Test metrics box plots for numeric encoding transformer experiments on the HAIM dataset on the LOS task. N indicates the encoded value is the normalized measurement, while Q indicates the encoded value is the estimated quantile for that measurement. Only the Bin methods are discrete encodings, while the rest are continuous encodings. 76

6.3 Test metrics box plots for numeric encoding transformer experiments on the HAIM dataset on the MP task. N indicates the encoded value is the normalized measurement, while Q indicates the encoded value is the estimated quantile for that measurement. Only the Bin methods are discrete encodings, while the rest are continuous encodings. 77

6.4 Test metrics box plots for time encoding transformer experiments on the HAIM dataset on the LOS task. All methods operate on the log timedelta in hours. Methods to the left of the blue divider are discrete encodings, while those to the right are continuous encodings. 78

6.5 Test metrics box plots for time encoding transformer experiments on the HAIM dataset on the MP task. All methods operate on the log timedelta in hours. Methods to the left of the blue divider are discrete encodings, while those to the right are continuous encodings. 79

6.6 Test metrics box plots for all encoding methods on the HAIM dataset on the LOS task. Num refers to continuous numeric encoding with normalized values. Time refers to continuous time encoding with the log timedelta in hours. NumTime refers to the combined encoding where both numeric values and time values are encoded with the same method. Control uses Round to integer and Bins on normalized values. 80

6.7 Test metrics box plots for all encoding methods on the HAIM dataset on the MP task. Num refers to continuous numeric encoding with normalized values. Time refers to continuous time encoding with the log timedelta in hours. NumTime refers to the combined encoding where both numeric values and time values are encoded with the same method. Control uses Round to integer and Bins on normalized values. 81

6.8 Test metrics box plots for all compression methods on the extended context dataset for the LOS task. NC = no compression, LMU = Legendre memory unit, XA = cross attention. 82

6.9 Test metrics box plots for all compression methods on the extended context dataset for the MP task. NC = no compression, LMU = Legendre memory unit, XA = cross attention. 83

A.1 Graphical illustration of the MLM task using regression on continuous encodings. (1) shows the original sequence of input IDs and the associated numeric values that would be continuously encoded. (2) illustrates the masking process, where some continuous values are corrupted. From left to right, Mask removes the value from the continuous encoding sequence and inserts a `[MASK]` token (`104`) into the input IDs, Random replaces the value with a random value drawn from a uniform distribution, and Same performs no additional operations. The right-most continuous value is not masked. (3) shows the resulting input sequences after the masking process. Notably, the numeric values sequences contains fewer elements since some are masked. (4) represents the input vectors to the transformer backbone, where the darker vectors are those that have been continuously encoded. (5) represents the hidden state vectors output by the transformer backbone. The regression task is only performed for the three highlighted vectors that have been altered during the masking process. Language modelling classification would be performed on some of the other vectors. Finally, (6) shows the model predictions and the corresponding labels from (1) on which the regression loss is computed. 107

A.2 Graphical illustration of the MLM task when using compression. (1) shows the sequence of input vectors after the masking process and the associated token labels. The labels are in the form of "token category:index" where C = code, M = miscellaneous, N = numeric, T = time, U = unit, X = text. (2) shows the resulting sequence of vectors after compression is applied. This sequence is input to a transformer backbone and the final layer hidden state is shown in (3). Each of the hidden state vectors is passed through the six classification heads, shown in (4). The code and text heads are illustrated as larger as they have more vocabulary items compared to the others. (5) shows the resulting logits after each classification head and the MLM labels for each token category. The cross entropy loss is only computed over items that were masked, unused logits are shown in white. The final loss is the average of the individual category losses. Following the fourth resource, highlighted in blue, there are 4 different categories on which MLM predictions are made, demonstrating how multiple vectors from within a single resource can be predicted with this method. 109

List of Tables

3.1 Counts of each FHIR resource type in the MIMIC-IV FHIR *hosp* dataset . 25

3.2 Counts of each FHIR resource type in the MIMIC-IV FHIR *icu* dataset . . 25

3.3 External FHIR coding systems and their descriptions 32

3.4 Token counts for different code category types 34

3.5 Demographic, chart event, laboratory event, and procedure event features used in the HAIM dataset. CRRT = Continuous renal replacement therapy, EEG = Electroencephalogram, GCS = Glasgow Coma Scale, MCH = Mean corpuscular hemoglobin, MCHC = Mean corpuscular hemoglobin concentration, NIBP = Non-invasive blood pressure, PICC = Peripherally inserted central catheter, RDW = Red blood cell distribution width, SpO_2 = Peripheral oxygen saturation. 41

3.6 Record size filtering parameters for the Extended Context FHIR Dataset. . 42

4.1 Bin edges for "round to int" timestamp encoding method 52

4.2 Examples of how different raw values are processed before applying various encoding methods. The top two rows exemplify how continuous data is encoded using discrete binning. The bottom three rows exemplify the way quantile and normalize numeric base methods produce inputs for continuous encoding methods. interp = linear interpolation function, M = median, MAD = mean absolute deviation 54

5.1 XGBoost hyperparameters 64

5.2 Numeric encoding experimental setup. Exp. = Experimental condition. . . 65

5.3 Time encoding experimental setup. Exp. = Experimental condition, Q = quantile numeric base. 66

5.4 Combined continuous numeric and time encoding (numtime) experimental setup. Exp. = Experimental condition. 66
5.5 Compression methods experimental setup. Exp. = Experimental condition. 67

6.1 Test AUROC results for baseline transformer and XGBoost on HAIM dataset. Best scores are bolded. 69
6.2 Test AUROC for continuous numeric encoding for transformer for the LOS task on the HAIM dataset. Best scores are bolded. 69
6.3 Test AUROC for continuous numeric encoding for transformer for the MP task on the HAIM dataset. Best scores are bolded. 71
6.4 Test AUROC for continuous time encoding for transformer for the LOS task on the HAIM dataset. Best scores are bolded. 71
6.5 Test AUROC for continuous time encoding for transformer for the MP task on the HAIM dataset. Best scores are bolded. 72
6.6 Test AUROC for combined continuous numeric and time encoding for transformer for the LOS and MP tasks on the HAIM dataset. Best scores are bolded. 73
6.7 Test AUROC for compression methods for the LOS task on the extended context dataset. Best scores are bolded. 74
6.8 Test AUROC for compression methods for the MP task on the extended context dataset. Best scores are bolded. 75

Chapter 1

Introduction

1.1 Motivation

Electronic health records (EHRs) are a digital collection of a patient's medical history, typically maintained on healthcare provider's information systems. There are many kinds of data contained within EHRs such as patient demographics, administrative details about visits, records of vital signs, laboratory tests, assessments, medications, progress notes, specialist reports, records of interventions and surgeries, and past diagnoses. The collection of these data is primarily for provider records, however, patient EHR data can be important for sharing patient history with other providers, patient access to results, prior authorization of insurance claims, population-level health monitoring, and clinical decision support systems.

Many recent works have applied machine learning (ML) methods to EHR data for predictive health analytics, which aim to improve the efficiency of healthcare operations and improve accuracy of predicting clinical outcomes. Examples of such works include detection of pneumonia in chest X-rays [1], predicting the onset of diseases such as diabetes mellitus [2], early sepsis prediction in intensive care unit (ICU) settings [3], and predicting patient outcomes from hospital stays, such as mortality or prolonged stay durations [4]–[6]. Particularly, length-of-stay prediction and mortality prediction can assist hospitals in optimizing resource allocation for patient care [7], [8]. Identification and early intervention for high-patients can lead to improved health outcomes [9], [10].

Many of these health ML solutions that operate on EHR data focus on solving one specific task, performing selection of clinically relevant data, engineering statistical features,

optimizing performance through hyperparmeter tuning, and using classical ML models like decision trees, support vector machines, and shallow neural networks. Such systems built using the classical ML framework are often compatible with explainable artificial intelligence (XAI) methods, which can help improve end-user trust in the system [11]. Custom feature engineering can incorporate domain-specific knowledge into the modelling approach, and classical ML models are typically lightweight to deploy.

In domains such as vision and language processing, deep learning has become the dominant approach. Recent deep learning methods focus on training large foundation models on massive amounts of data [12]. Deep learning models are large neural networks that process inputs that are structured collections of vectors, rather than assuming individual features are particularly meaningful. For example, a classical machine learning approach might compute statistical features on the distribution of pixel colours in an image. In contrast, deep learning methods would treat an image as a spatial array of pixel vectors. Deep learning models can learn complex, abstract representations through composing models of several non-linear layers [13]. Optimal parameters of these layers can be obtained through backpropagation [14]. Deep learning models have been demonstrated to be much more capable at solving very complex tasks, such as the emergence of transformer models for language modelling [15]–[17]. However, deep learning models are much larger than classical ML models, requiring more compute and data to train. Furthermore, deep learning methods are often black-box models and it can be challenging to explain why a model makes a particular decision.

Interoperable health data systems promote the ease of developing software that can make use of EHR data for various applications. Central to building many modern interoperable health data systems is the Health Level 7 (HL7) organization's Fast Healthcare Interoperability Resources (FHIR) medical data standard. FHIR could serve as a consistent interface between the underlying data and many applications, such as EHRs, billing and insurance services, advanced analytics, and machine learning applications.

There is an increase in the rate of adoption of the FHIR standard in Canada, the US, and many countries globally, with several nations starting to introduce regulation that mandates the use of FHIR for healthcare data exchange [18]. In the US, the Centers for Medicare and Medicaid Services (CMS) issued the Interoperability and Patient Access final rule, which requires CMS payers, including Medicare and Medicaid, to use FHIR APIs for Patient Access, Provider Directory and Payer-to-Payer exchange [19]. With an increase in adoption of FHIR data standards, there will be more opportunities to apply machine learning solutions to data complying to this standard, rather than developing towards custom solutions for each database.

Classical ML models developed on EHR have limited interoperability. Firstly, the datasets from which they are developed are often private, so other developers cannot guarantee that the target datasets have the same data formats or even that the same features available. Since classical ML models are typically developed with specific applications and engineered features to suit that application, new models must be developed for each new task. In contrast, if a foundational deep learning model for EHR data can be developed that processes FHIR data, this model can be applied widely to any database that uses this standard and can be fine-tuned to specific tasks of interest. Furthermore, deep neural networks should be more capable at solving complex tasks.

Directly applying transformer neural networks to processing FHIR data poses some challenges. Firstly, transformer network inputs are sequences of discrete tokens. For text models, tokenization algorithms are developed based on frequently occurring characters in the text. This distribution is not the same as text in FHIR JSON representations, so a single FHIR resource is represented as several hundred text tokens. With a patient record containing thousands of resources, the sequence of tokens to represent a patient record exceeds the typical input sizes for transformers. Tokenized representations of FHIR data might be better defined using structured units of the FHIR standard as opposed to text characters. Medical data also contains many important continuous quantities, such as laboratory test results and timestamps. Applying discrete tokenization methods to these quantities may not be the most appropriate.

1.2 Contributions

In this thesis, I seek to contribute methodology towards creating foundation models that operate on FHIR data. I also explore experimental methods that might address potential weaknesses in the initial modelling approach. In this thesis, I contribute the following:

- Methodology to process FHIR data in the form of JSON into a sequence of tokens, which can be used as inputs for transformer models
- A comparison of the performance of this ML approach with a classical ML approach on the open source MIMIC-IV dataset
- Exploration of the use of continuous vector representation for numeric measurements and time data, rather than using discrete representations
- Exploration of methods to reduce sequence length for transformer sequence models by compressing subsequences of vectors based on the resource type

Chapter 2 covers the background on the neural network architecture and methods for continuous data encoding and compression. It also covers existing related works from which we build our methods. Chapter 3 covers the dataset and data processing. Chapter 4 details the modelling methods. Chapter 5 details the experimental plans and analysis methods. Chapter 6 reports the results obtained through experimentation. Chapter 7 discusses the findings, their impact, advantages, limitations, and future directions for this work. Finally, Appendix A discusses our pre-training methodology and discusses why we did not continue using it for the main thesis experiments.

Chapter 2

Background

In this chapter, I outline the foundational neural network architecture used in this work, the transformer. I describe the transformer architecture as it is used in BERT [16], further details of the masked language modelling (MLM) task [16], and recent changes to the architecture that are used in state-of-the-art language models [20]. Next, I discuss motivating works that apply machine learning to electronic health record (EHR) data. In [21], a BERT-like encoder model is applied to sequences of medical diagnostic codes. In [4], FHIR data is used for sequence modelling with LSTM networks. Finally, I cover the literature on spatial semantic pointers [22] and the Legendre memory unit (LMU) state-space neural network [23]–[25], which are components of our experimental methods that may be novel even to readers familiar with deep learning.

2.1 Transformer Architecture

Prior to 2017, the state of the art in sequence processing and natural language modelling was recurrent neural networks (RNNs), such as long-short term memory (LSTM) networks [26], [27]. However, RNNs require processing the sequence of input vectors autoregressively, such that generating the current hidden state h_t requires generating the previous hidden state, h_{t-1}. This sequential computation inherently limits the ability to accelerate RNN training through parallelization on GPU hardware, especially at longer sequence lengths.

The attention mechanism has previously been integrated with RNNs for language tasks [28]–[30] but in 2017, Vaswani *et al.* [15] introduced a novel sequence transduction architecture based entirely on attention. This "transformer" architecture has become the basis

for current state-of-the-art language models. This transformer architecture has advantages over RNNs since the attention operation can be parallelized effectively through matrix multiplication and reliably allows for long-range sequence dependencies, though with $\mathcal{O}(T^2)$ memory requirement, where T is the sequence length. The quadratic memory scaling means the maximum sequence length that can be processed is limited by the system's memory.

The original transformer architecture in [15] is a sequence transduction model, with an encoder to process input sequences and a decoder to produce an output sequence in a different output vocabulary. This thesis and other background works discussed focus on "Encoder only" architectures (like BERT [16]), where a single input sequence is processed for token-level or sequence-level classification tasks.

The transformer architecture is composed of three main components. The input layers transform raw inputs into a sequence of vectors. In typical NLP applications, inputs are a sequence of integers called "token IDs" that map each discrete token to a random embedding vectors. Other data modalities such as images, continuous signals, or even video game inputs have different methods for mapping inputs to vector sequences [31]–[33]. In Vaswani *et al.* [15], the input layers also used additive sinusoidal position embeddings to the input vector sequence. Without position embeddings, a transformer does not have representation of ordering, and the sequence would be treated as a bag of vectors.

The main body of the transformer [15] is composed of alternating layers of Multi-Head Self Attention (MHSA) blocks and Feed Forward Network (FFN) blocks. For a given input sequence element, the MHSA operator outputs a representation that is a weighted sum of all other elements in the sequence. The weighting is based on attention, which computes how compatible a given "query" vector is with all other "key" vectors in the sequence. Each key vector has a corresponding "value" vector, and the final representation for each query is created by performing a weighted sum of all value vectors based on the compatibility between the query and key. The "multi-head" component of MHSA performs this attention operation independently, then combines the result from each head into a final representation.

The MHSA block itself is composed of h attention heads. Each head has independent key, query, and value linear projections that operate on the input vector sequence, where each vector has dimension d. The dimension of the resulting key, query, and value vectors of each head is $d' = d/h$. The key, query, and value projections are shown in Equation 2.1 and operate on an input sequence, $\boldsymbol{X} \in \mathbb{R}^{T \times d}$. Scaled-dot-product attention, the compatibility function, is performed on each head. First, the dot product between the keys and queries is computed and scaled by the dimension of the head. Softmax is used to normalize the

compatibility function. This so-called "attention matrix" is then multiplied with the values and the results from all heads are concatenated and projected back to the original hidden size, d. Optionally, a mask may be used to prevent computing attention over certain positions in the sequence, such as padding tokens. Equation 2.2 shows the attention equation for a single head of dimension d' operating on a sequence of length T.

$$\begin{aligned} \boldsymbol{Q} &= \boldsymbol{W}_q \boldsymbol{X} \in \mathbb{R}^{T \times d'} \\ \boldsymbol{K} &= \boldsymbol{W}_k \boldsymbol{X} \in \mathbb{R}^{T \times d'} \\ \boldsymbol{V} &= \boldsymbol{W}_v \boldsymbol{X} \in \mathbb{R}^{T \times d'} \end{aligned} \tag{2.1}$$

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d'}}\right)\boldsymbol{V} \in \mathbb{R}^{T \times d'} \tag{2.2}$$

After the MHSA block, dropout is applied and a residual connection to the inputs is added [15], [34]. Following these transformation, LayerNorm [35] is applied to the hidden state before passing to the next layer.

In Vaswani *et al.* [15], the FFN layer is a simple network composed of a linear projecting to a larger latent space (*e.g.* $d \to 4d$), applying the GeLU non-linearity [36], and a linear projection back to the original hidden dimension. This transformation is applied to each sequence element in parallel, along the last dimension of the input tensor. Similarly to attention blocks, dropout is applied, followed by adding a residual connection, and finally applying LayerNorm before passing to the next block.

The final component of transformer architectures is an output layer, where the hidden representations are projected to the appropriate output shapes depending on the task. For language modelling, the parameters in the final linear layer are often tied to the embedding parameters such that the logits are of shape $Y \in \mathbb{R}^{B \times T \times V}$, where B is the batch size, T is the sequence length, and V is the vocabulary size. The loss is the cross entropy between the vocabulary distribution and the true label for each token in the sequence and batch. For sequence classification tasks, an aggregation operation is applied to the logits to combine the sequence of vectors into a single vector, and a final linear layer is applied to obtain the desired number of output classes.

Since the introduction of the transformer in 2017 [15], BERT (2018) [16] and GPT (2019) [17], further developments to the transformer architecture have resulted in empirical improvements in performance and convergence speed. This group of changes is sometimes called "Transformer++" recipe [37] and describes a set of changes made to generative

pre-trained transformer language models such as LLaMA [38], Qwen [20], Falcon [39] and others. The LayerNorm position after each block is changed to be before the processing layers of each block [40]. Instead of LayerNorm, RMSNorm [41] is used. Additive sinusoidal position embeddings are changed to multiplicative Rotary Position Encoding (RoPE) which is applied during each attention computation step [42]. RoPE has been shown to scale better to longer sequence lengths and even extrapolated to sequence lengths not seen in training [43]. Finally, the FFN layer uses gated linear units [44] with the swish activation function [45]. The FFN uses an expansion ratio of 8/3 instead of 4 as in [46]. These changes have been shown to improve the training stability of transformers [47].

2.2 Language Modelling

Text inputs for transformer models are broken into discrete chunks, called tokens. A token may represent an entire word, or a portion of a word. A tokenization algorithm such as byte-pair encoding [48] or WordPiece [49] is used to generate the vocabulary of text chunks that comprise the vocabulary for a language model. Each of the tokens in the vocabulary is mapped to an integer "input ID" and the sequence of these input IDs forms the inputs to a transformer. The transformer uses an embedding layer to assign a randomly initialized vector to each input ID. The embedding layer essentially functions as an index lookup from the input ID to a vector.

The Causal Language Modelling (CLM) task seeks to estimate the joint probability of an observed sequence of tokens [17]. The joint probability is factorized into a product of conditional probabilities of observing the current token given the previous tokens in the sequence, $P(T_n|T_1, T_2, ..., T_{n-1})$. The task objective is thus to maximize the likelihood of observing a given token, given the previous tokens, over all positions in the sequence [17]. The task is formulated as a multinomial classification problem over the vocabulary for each position in the sequence, given the context of all past tokens. CLM can be used to train a language for both generative and sequence classification language tasks.

Masked Language Modelling (MLM) is a sequence de-noising tasks where the goal is to estimate the probability distribution of a corrupted token given the context of all other tokens in the sequence, $P(T_k|T_1, T_2, ..., T_{k-1}, T_{k+1}, ..., T_T)$. This is also formulated as a multinomial classification task over the vocabulary but only for certain tokens in the sequence. The BERT model originally introduced this task and used a 15% masking probability, and allocating 80% of those masked tokens to use the `[MASK]` token, 10% to use a random token, and 10% to use the original token as inputs [16]. By keeping some inputs the same, the model cannot rely on the `[MASK]` token as a signal to make predictions, and

must learn a contextual representation for every token, even if its not corrupted. BERT predicts the loss only over masked tokens in the masked language modelling task [16]. MLM can be used to train a language model for classification, but not to generate text.

2.3 FHIR Data Model

The basic component of the FHIR data model is a "Resource". Resources can be used to exchange or store healthcare information that can be used in solving both clinical and administrative problems [50]. A FHIR resource belongs to a resource type, which is a definition that describes standard patterns of elements that compose the structure of the resource. These elements are further built from elemental datatypes defined by the FHIR standard, such as a CodeableConcept, Quantity, Period, or Reference. Resources also contain a unique identifier, metadata, and human-readable content. FHIR resources can be represented in several formats, including JSON, XML, and Turtle. Particularly, JSON is highly used as a data exchange format in web application programming interfaces (APIs).

A brief example of an Observation resource in JSON format is printed in Figure 2.2. This resource describes a glucose blood test result. This example will be used as an illustrative example throughout the Background and Methods sections.

2.3.1 FHIR datatypes

Like many data standards, FHIR defines a set of primitive datatypes which are the simplest types for individual elements [51]. Primitive datatypes in FHIR include integers, decimals, booleans, strings, dates, times, datetimes, uniform resource identifiers (URIs) for linking to other data outside the content of the resource, codes, which are symbols represented as a string from a given code system, and base-64 binary for representing complex media. Primitive datatypes can be elements of arrays or components of more complex structures such as general-purpose datatypes or resources. Figure 2.2 shows the example Observation resource with the primitive datatypes highlighted.

General-purpose datatypes are structures with named elements and associated values that are primitive datatypes or other general-purpose datatypes. General-purpose datatypes are more human-interpretable than the primitive datatypes as they are designed to represent units of medical data.

The Coding datatype represents a concept that is assigned a symbol from a defined "code system". The Coding datatype has a `system` URI that identifies the particular

```
{
'resourceType': 'Observation',
'id': '7e2317ea-17d7-5b14-8b6b-f003151e5f3e',
'extension': [{
    'url': 'http://mimic-fhir/StructureDefinition/mimic-lab-priority',
    'valueString': 'STAT'}],
'category': [{
    'coding': [{
        'system': 'http://terminology.hl7.org/CodeSystem/observation-category',
        'code': 'laboratory',
        'display': 'Laboratory'}]}],
'code': {
    'coding': [{
        'system': 'http://mimic-fhir/CodeSystem/mimic-d-labitems',
        'code': '50931',
        'display': 'Glucose'}]},
'subject': {'reference': 'Patient/54ee7569-2e21-57f3-8ffa-dbdf2dc859b9'},
'effectiveDateTime': '2147-01-13T15:24:00-05:00',
'issued': '2147-01-13T16:20:00-05:00',
'valueQuantity': {
    'value': 77,
    'unit': 'mg/dL',
    'system': 'http://mimic-fhir/CodeSystem/mimic-units',
    'code': 'mg/dL'},
'specimen': {'reference': 'Specimen/c7059b3e-4d5c-50f0-8f17-9ea7e1689bd4'},
'referenceRange': [{
    'low': {
        'value': 70,
        'unit': 'mg/dL',
        'system': 'http://mimic-fhir/CodeSystem/mimic-units',
        'code': 'mg/dL'},
    'high': {
        'value': 105,
        'unit': 'mg/dL',
        'system': 'http://mimic-fhir/CodeSystem/mimic-units',
        'code': 'mg/dL'}}]
}
```

Figure 2.1: Example Observation resource in JSON format from the MIMIC FHIR dataset. This resource represents a glucose blood test.

coding system, a `code`, which is a code symbol itself, and a string `display` which can be used to as a human-readable representation of the `code`.

The Quantity datatype represents an amount that can be measured. It has components

```
{
'resourceType': 'Observation',
'id': '7e2317ea-17d7-5b14-8b6b-f003151e5f3e',
'extension': [{
    'url': 'http://mimic-fhir/StructureDefinition/mimic-lab-priority',
    'valueString': 'STAT'}],
'category': [{
    'coding': [{
        'system': 'http://terminology.hl7.org/CodeSystem/observation-category',
        'code': 'laboratory',
        'display': 'Laboratory'}]}],
'code': {
    'coding': [{
        'system': 'http://mimic-fhir/CodeSystem/mimic-d-labitems',
        'code': '50931',
        'display': 'Glucose'}]},
'subject': {'reference': 'Patient/54ee7569-2e21-57f3-8ffa-dbdf2dc859b9'},
'effectiveDateTime': '2147-01-13T15:24:00-05:00',
'issued': '2147-01-13T16:20:00-05:00',
'valueQuantity': {
    'value': 77,
    'unit': 'mg/dL',
    'system': 'http://mimic-fhir/CodeSystem/mimic-units',
    'code': 'mg/dL'},
'specimen': {'reference': 'Specimen/c7059b3e-4d5c-50f0-8f17-9ea7e1689bd4'},
'referenceRange': [{
    'low': {
        'value': 70,
        'unit': 'mg/dL',
        'system': 'http://mimic-fhir/CodeSystem/mimic-units',
        'code': 'mg/dL'},
    'high': {
        'value': 105,
        'unit': 'mg/dL',
        'system': 'http://mimic-fhir/CodeSystem/mimic-units',
        'code': 'mg/dL'}}]
}
```

Figure 2.2: Example Observation resource with the primitive datatypes highlighted. Numeric datatypes are highlighted in orange, strings are highlighted in cyan, timestamps are highlighted in yellow, and URIs are highlighted in green.

such as `value`, which is a decimal that represents the numeric measurement, `unit`, which is a string representation of the unit, and `system` and `code` which represent the coded form

of the unit from a particular coding system.

The Ratio datatype represents an numeric amount that can be expressed as a ratio between a numerator and denominator. It has `numerator` and `denominator` components which are both Quantity datatypes.

The Period and Range datatypes are similar. They both have two elements representing upper and lower values. The difference is that Period has `start` and `end` components which are datetime datatypes, while Range has `high` and `low` components which are Quantity datatypes.

The CodeableConcept datatype is a common pattern in representing healthcare data. It represents a value using one or more references to a coding system such as an ontology or medical terminology, but may also be described by text. Common medical ontologies include the International Classification of Diseases (ICD) for disease codes [52], [53], Logical Observation Identifiers Names and Codes (LOINC) for laboratory codes [54], [55], RxNorm for clinical drugs [56], [57], and SNOMED Clinical Terms, which can represent a wide variety of medical terminology and has mappings to several other standard terminology systems [50], [58]. The CodeableConcept has a `coding` component that refers to one or more Coding datatypes and a `text` string.

The Reference datatype is a special-purpose datatype within FHIR that describes a link from the source resource, which contains the reference, to a target resource. References allow for the creation of a web of healthcare information and reduce the amount of repetition when describing high-level resources. References always link to the same kind of resource. For example, a `subject` element of a resource will always refer to a Patient resource. References have a `reference` component which is a string with the identifier of the target resource, and a `type` URI that describes what kind of resource it links to.

Figure 2.3 shows the example Observation resource with Coding, Quantity, and Reference datatypes highlighted. The `referenceRange` field is a Range datatype composed of multiple Quantity datatypes. The `code` field is also a CodeableConcept datatype, with the `coding` field containing a list of Coding datatypes.

2.4 Medical Language Models

The first use of transformer language models applied to healthcare data were ones processing sequences of medical codes, such as Med-BERT [21] and BEHRT [59]. These models used the MLM task to learn vector representations over a vocabulary of ICD-9 and ICD-10

```
{
'resourceType': 'Observation',
'id': '7e2317ea-17d7-5b14-8b6b-f003151e5f3e',
'extension': [{
    'url': 'http://mimic-fhir/StructureDefinition/mimic-lab-priority',
    'valueString': 'STAT'}],
'status': 'final',
'category': [{
    'coding': [{
        'system': 'http://terminology.hl7.org/CodeSystem/observation-category',
        'code': 'laboratory',
        'display': 'Laboratory'}]}],
'code': {
    'coding': [{
        'system': 'http://mimic-fhir/CodeSystem/mimic-d-labitems',
        'code': '50931',
        'display': 'Glucose'}]},
'subject': {'reference': 'Patient/54ee7569-2e21-57f3-8ffa-dbdf2dc859b9'},
'effectiveDateTime': '2147-01-13T15:24:00-05:00',
'issued': '2147-01-13T16:20:00-05:00',
'valueQuantity': {
    'value': 77,
    'unit': 'mg/dL',
    'system': 'http://mimic-fhir/CodeSystem/mimic-units',
    'code': 'mg/dL'},
'specimen': {'reference': 'Specimen/c7059b3e-4d5c-50f0-8f17-9ea7e1689bd4'},
'referenceRange': [{
    'low': {
        'value': 70,
        'unit': 'mg/dL',
        'system': 'http://mimic-fhir/CodeSystem/mimic-units',
        'code': 'mg/dL'},
    'high': {
        'value': 105,
        'unit': 'mg/dL',
        'system': 'http://mimic-fhir/CodeSystem/mimic-units',
        'code': 'mg/dL'}}]
}
```

Figure 2.3: Example Observation resource with some general-purpose datatypes highlighted. Coding datatypes are highlighted in yellow, Quantity datatypes are highlighted in cyan, and Reference datatypes are highlighted in green.

medical diagnostic codes, instead of natural text tokens. These medical code language models were further fine-tuned for tasks such as disease prediction, patient mortality pre-

diction, prolonged length-of-stay prediction, and unexpected readmission prediction. These models would be capable of processing singular types of FHIR resources, such as Condition resources, if the codes are from the same coding system, but cannot process other kinds of medical data from a patient's history, such as Observation resources that record patient vitals and laboratory results.

Another common use of transformer language models with healthcare data are those trained on standard NLP tasks that contain medical jargon or clinical conversations. Examples of such models include BioClinical BERT [60], MedGPT [61], Med-PaLM [62], and GPT-4 [63]. These models have demonstrated the ability to predict possible future diseases from clinical notes [61], generate clinical summaries from patient conversations [64], reason through clinical scenarios [62], and even achieve scores on the US Medical Licensing Exam that are within the pass range [65], [66]. Text models have also been shown to be capable of generating FHIR MedicationStatement resources from clinical text snippets [67].

However, these text models are not suitable for processing large volumes of FHIR data, as raw FHIR data in JSON format are not efficiently tokenized by standard text tokenization methods. The example resource in JSON format is converted into 436 tokens using the GPT-4o tokenizer. The standard text tokenizers generate a large number of tokens when encountering punctuation marks that create the JSON format and also in the URIs. Given GPT-4o's maximum context length of 128,000 tokens, the maximum number of resources this model could process in JSON format is just a few hundred. However, in the dataset this thesis uses, patent records may have thousands of FHIR resources. Text models are therefore unsuitable for processing entire patient histories of FHIR data.

2.5 Med-BERT: pretrained contextualized embeddings on large-scale structured electronic health records for disease prediction

Med-BERT [21] is a transformer encoder that was trained using the BERT modelling technique [16] applied to structured EHR data in the form of International Classification of Diseases (ICD) medical codes. Rasmy *et al.* construct the input data as a time-ordered sequence of diagnostic codes extracted from a patient's EHR record. Within a single patient visit, diagnostic codes are ordered by priority as indicated from their datasets. Similar to BERT, they use an additive segment embedding for each visit and a learned additive positional embedding. The transformer architecture used is essentially the same as

BERT, but with different hyperparameters for the model size (6 layers, hidden dimension of 192, 6 attention heads, feed-forward size of 64).

The Med-BERT pre-training method adapts the MLM and next sentence prediction (NSP) tasks from BERT [16] to suit their medical task. Instead of performing NSP, the sequence classification task during pre-training was a binary classification task to predict whether any of the visits in the history were a prolonged length of stay, greater than 7 days. With the MLM task, only one code was masked per patient, instead of applying a 15% probability to mask individual tokens independently. It should be noted that the average number of codes per patient (and thus tokens per sequence) as between 7 to 33, depending on the dataset. Thus, one token per sequence is a decrease in the number of tokens that could be masked per sequence at the larger sequence lengths compared to the original BERT methodology. The same probabilities from BERT [16] for assigning `[MASK]`, random, and unchanged tokens are used for Med-BERT.

The Med-BERT model is fine-tuned on disease prediction tasks, which are binary prediction tasks to predict whether a patient was diagnosed with pancreatic cancer or a patient with a diabetes diagnosis experienced heart failure among a cohort of negative controls. Med-BERT was demonstrated to be more effective than comparative deep learning techniques like the bi-directional gated recurrent unit (Bi-GRU) network used in RETAIN [68] or on networks based on non-contextual embeddings from clinical word2vec embeddings [69] on the fine-tuning tasks. Pre-training was also demonstrated to be effective for transfer learning, improving performance compared to training a randomly initialized Med-BERT architecture on the fine-tuning tasks [21].

The datasets used by Med-BERT for pre-training and fine-tuning, as well as pre-trained model weights are not accessible, due to data use policies. It is therefore not possible to directly compare the performance models developed in this thesis to Med-BERT and their tasks. However, their methodology for pre-training a transformer with MLM has been demonstrated to be effective for healthcare machine learning tasks and form the basis for our methodology.

The Med-BERT methodology could be improved by including temporal information. Med-BERT encodes visit embeddings as ordinal, but it does not distinguish whether there is a long temporal delay between multiple visits. This information may be useful in a medical context. This thesis aims to explore different time encoding methods for EHR data. Med-BERT notes that future work could include applying their methodology to medical codes representing sequences medications and laboratory tests. This thesis includes processing EHR data that includes these modalities. Med-BERT also notes the challenge of determining the level of a basic token for general EHR data and how to sequence codes

when there is no deterministic ordering like medical code priority. This thesis proposes to use the structure of FHIR resources to determine tokens and their ordering from EHR data.

2.6 Scalable and accurate deep learning with electronic health records

Rajkomar *et al.* present a deep learning-based prediction model that processes entire EHR records in FHIR format [4]. In this paper, the FHIR EHR data is converted into a sequence of tokens, from which a sequence of embedding vectors is derived as inputs to the deep learning models. The final model is an ensemble of a Long Short-Term Memory network (LSTM) [27], a feed-forward network that uses time-aware attention, and a boosted decision tree. The decision tree uses hand-crafted binary decision rules that compare a feature, as described by the token name, value, and time-delta, to various thresholds on the value, time-delta, or count within a patient's record and relevant features that could be derived from these values. The models are trained end-to-end on various predictive healthcare tasks, including in-patient mortality, unexpected readmission within 30 days, prolonged length of stay of at least 7 days, and predicting the primary and secondary diagnostic codes upon discharge. Similar to Med-BERT, the datasets and exact models are not available for public access, so direct comparison is not possible. However, their methodology for tokenization of FHIR EHR data is shown to be effective for use in time-series models such as the LSTM and this thesis uses the encoding methods as a basis for the methodology.

The FHIR EHR datasets used in [4] contain Patient, Encounter, Medication, Observation (including vital signs and nursing documentation), Composition (free-text notes), Condition, MedicationAdministration, MedicationOrder, ProcedureRequest, and Procedure resources. The elements of the FHIR resources are converted into tokens based on the FHIR datatypes. For codings, the `system` and `code` strings are concatenated and mapped to a single token. Text element strings are tokenized by splitting on white-space and punctuation and each vocabulary item is mapped to a token. Tokens representing codes, like medications, and for text must occur at least twice in the dataset to be kept. Infrequent tokens are hashed to a small set of out-of-vocabulary tokens.

Numeric values are represented in different ways depending on the model architecture. In one method, the name of the measurement, the value, and the unit are concatenated as strings and mapped to a token. For example, "Hemoglobin 15 g/dL" would be one token. The other method computes the quantile bin of the value for each type of measurement

and creates tokens by concatenating the string representation of the name, bin, and unit. For example, if 15 is the 75-th percentile for that measurement, the bin string would be "0.75". The percentiles are rounded into 20 buckets, i.e. every 5-th percentile is a bucket. Tokens from numeric observations have to have occurred at least 100 times in the dataset to be kept, or else they are hashed to the out-of-vocabulary tokens.

Timestamps are represented using different methods, depending on the model architecture. In general, all use logarithmically spaced bins based on the difference between the prediction time and the timestamp of the resource (henceforth called the "resource timedelta"). One model uses exponentially increasing bins of 1, 2, 4, 8, ... days and similarly for years for patient age. Another rounds the logarithm of the resource timedelta in seconds to the nearest integer. These bins are then mapped to tokens and concatenated to the rest of the input sequence.

As transformers have been demonstrated to be state-of-the-art in many other sequence modelling tasks, I do not seek to use LSTM networks or the other modelling methods presented in [4] in this thesis. However, the methods for creating a sequence of tokens based on the datatypes of the FHIR elements will serve as a set of baseline methods in this thesis. However, there are some aspects that could be further extended. Particularly, creating token representations for Reference datatype is not addressed by [4]. Some resources, like Conditions in MIMIC-IV FHIR, do not have timestamps and require references to other resources to determine their temporal placement in the EHR record. Other resources like MedicationRequest do not fully describe the medication composition and reference a Medication resource with details of all the components. Additionally, using discretization to represent continuous numeric and temporal data may not be the most suitable. In this thesis, I explore whether continuous representations of these quantities might improve performance. Lastly, [4] reports very large values for the median number of tokens at discharge for a patient, at 86,477 and 122,961 for their two datasets. As transformers encounter memory barriers when processing long sequence lengths, this is motivation to investigate methods for compressing the sequence length of tokenized EHR data.

2.7 Integrated multimodal artificial intelligence framework for healthcare applications

Soenksen *et al.* present a Holistic AI in Medicine (HAIM) framework that integrates multiple modalities for predictive health machine learning tasks [6]. In this paper, data from various modalities in MIMIC-IV [70]–[72] are processed with modality-specific feature ex-

tractors. Various combinations of modalities are used as inputs for extreme gradient-boosted trees (XGBoost) [73] to perform classification tasks for predicting chest pathology diagnoses, length of stay, and 48-hour patient mortality.

The HAIM framework uses tabular, time-series, medical images, and natural language as data modalities from MIMIC-IV. Tabular data includes patient demographics, such as their age, gender, ethnicity, marital status, and insurance. These features are all categorical and simply encoded as integers.

Time-series data consists of records of procedure events (such as insertion of a Foley catheter or performing peritoneal dialysis), lab events (such as blood glucose levels, blood sodium levels, or red blood cell count), and chart events (such as measurements of heart rate, blood pressure, or oxygen saturation). A time-series of numerical measurements from these events are collected for a time range relevant to the patient's hospital visit and 11 statistical features are computed for each item's time-series: min, max, mean, variance, mean difference, mean absolute difference, total absolute difference, difference between the beginning and end of the time-series, the number of peaks, and the slope of a linear trend line fitted to the time-series. A list of all the time-series events can be found in Table 3.5.

Text data consists of notes from electrocardiogram, echocardiogram, and radiology reports. Features are generated by passing the text through a transformer encoder model, BioClinical BERT, that has been pre-trained on biomedical text [60]. Of note, this text model has been pre-trained on notes from the MIMIC-III dataset, some of which may overlap with the notes from MIMIC-IV. Each different note type has an associated 768-dimensional vector embedding produced by the text model which are used as features.

Finally, image data consists of chest X-rays. Features are generated by passing each image through a pre-trained convolutional network (CNN). The CNN is called Densenet121-res224-chex [74], which is a DenseNet [75] pre-trained on the CheXpert dataset [76] which is a large, labelled chest X-ray dataset. The output probabilities for the 18 pathology classes used in the CheXpert dataset are used as features, as well as the 1024-dimensional embedding of the image encoder before the dense layer classifier.

Soenknsen *et al.* demonstrate that adding more modalities improves prediction performance, as measured by area under the receiver operating characteristic (AUROC). Furthermore, the authors find that vision features contribute the most to improved performance on chest pathology diagnosis, while time-series features contribute the most to length-of-stay prediction and mortality prediction.

Since the MIMIC-IV dataset is openly available, I seek to use the dataset and data processing techniques presented in [6] as a benchmark for performance. I aim to improve upon the performance of just using time-series and demographics features, as these data

are available in the MIMIC-IV FHIR dataset, while images and text are not. The inclusion of chest X-ray and report text modalities from MIMIC-IV are outside the scope of this thesis but could serve as a direction for future work. Due to this limitation, I focus on comparing the length-of-stay prediction and mortality prediction tasks.

The data processing techniques used by [6] also leave areas for refinement. Although the authors report the dataset has 34,537 items, there are only 7,279 unique hospitalizations. Because the authors test models that use X-ray images, a unique dataset item is created for each unique X-ray image, and multiple X-ray views may be taken per hospitalization. However, the tabular and time-series data are identical for different X-ray views and are only unique to the hospitalization. For my use case, the dataset provided in PhysioNet [70]–[72], [77] only effectively has 7,279 samples.

Furthermore, large amounts of time-series data and tabular data are excluded from the items selected by this study. There are many potentially relevant chart events (such as temperature, pain assessments, other physician observations), lab events (such as microbiology cultures), and procedure events (such as invasive intubation and imaging studies) that are not included in the ones chosen for HAIM feature engineering. However, custom feature engineering becomes unscalable if all available types of data are included.

2.8 Spatial Semantic Pointers

Spatial semantic pointers (SSPs) are based on ideas from the Vector-Symbolic Architecture (VSA) approach, which is a computing paradigm that relies on using high dimensional, random vectors to represent unique concepts. Formally, a VSA is a vector space, a set of algebraic operators, and a similarity measure [78]. VSAs can create and manipulate vector representations of concepts by combining vectors by using typically the bundling, binding, and unbinding algebraic operators. New, structured vector representations can be made by combining discrete concepts (symbols) to create new vector representations by the algebraic operations.

Bundling is used to create a vector that is similar to both input vectors, sometimes called "superimposition". Similarity is measured by the VSA's similarity metric. Bundling is typically represented by the "+" symbol. The resultant vector $\mathbf{x} = \mathbf{a} + \mathbf{b}$ can be thought as containing the concepts described by both $\mathbf{a}$ and $\mathbf{b}$.

Binding is used to create a vector that is connected to both input vectors. Binding typically uses the "$\otimes$" symbol. A common use case for binding is to create role-filler pairs, where one symbolic vector represents a role and the other represents the value of

a concept that fills that role. For example, one could create a vector representing the colour of an object using the role-filler pair $\mathbf{x} = \mathbf{COLOUR} \otimes \mathbf{RED}$. The resultant vector after the binding operator is dissimilar to both its inputs. However, one of the inputs can be retrieved through the unbinding operator with the inverse of the other input. In the previous example, I could retrieve an approximate representation of $\mathbf{RED} \approx (\mathbf{COLOR})^{-1} \otimes \mathbf{x}$.

Holographic Reduced Representations (HRRs) are a VSA first described by Plate [79] and is the VSA used by SSPs. HRR uses d-dimensional vectors drawn from $\mathbb{R}^d$ and uses the cosine distance as the similarity metric. The bundling operator is implemented through element-wise vector addition and the binding operator is implemented through circular convolution ($\circledast$). Of note, the circular convolution operation can be computed as the Hadamard product of the input vectors in the Fourier domain: $\mathbf{x} \circledast \mathbf{y} = \mathcal{F}^{-1}\{\mathcal{F}\{\mathbf{x}\} \odot \mathcal{F}\{\mathbf{y}\}\}$. This allows for an efficient computation of the binding operation on GPUs, which have optimized implementations of the fast Fourier transform (FFT) algorithm and vector multiplication.

Previous work has used HRR representations to create structured embedding vectors for medical code representations based on the SNOMED CT medical ontology [80].

While most uses of VSAs focus on representing discrete structures, SSPs are an extension that can be used to represent structure over a continuous space [22]. When integrated into the VSA structures, SSPs have been used for combining continuous quantities, such as spatial locations, with discrete objects for applications such as robotic navigation [81], [82].

When using the HRR operators for binding and bundling, SSPs can be seen as "fractional binding" of a vector representing a unit of a continuous quantity. For example, if a vector $\mathbf{x}$ represents the continuous quantity 1 meter, then I could represent 3 meters as $\mathbf{x}^3 = \mathbf{x} \circledast \mathbf{x} \circledast \mathbf{x}$ by binding $\mathbf{x}$ with itself a total of 3 times. Since the HRR binding operator, circular convolution, has formulation with the Fourier transform, I can express this repeated binding as an element-wise exponentiation in the complex domain.

$$\phi_{\mathbf{x}}(k) = \mathbf{x}^k = \mathcal{F}^{-1}\left\{\mathcal{F}\{\mathbf{x}\}^k\right\}, \; k \in \mathbb{R} \tag{2.3}$$

By extending k to the real domain, Equation 2.3 still holds and the resulting vector, $\phi_{\mathbf{x}}(k)$ can represent any continuous quantity. The resulting features of $\phi_{\mathbf{x}}(k)$ are similar to random Fourier features [83].

2.9 Legendre Memory Unit

State-space representations are a mathematical modelling technique that describe the temporal evolution of a continuous dynamical system through a first-order differential equation that relates state ($\mathbf{x} \in \mathbb{R}^m$), input ($\mathbf{u} \in \mathbb{R}^n$), and output ($\mathbf{y} \in \mathbb{R}^p$) variables. If a system is linear and time-invariant (LTI), then the dynamical system can be described by Equation 2.4, where $\dot{\mathbf{x}}$ is the time-derivative of the state. $\mathbf{A} \in \mathbb{R}^{m \times m}$ is the state matrix, which describes how the state informs its own change. $\mathbf{B} \in \mathbb{R}^{m \times n}$ is the input matrix, which informs how the input is integrated into the state. $\mathbf{C} \in \mathbb{R}^{p \times m}$ is the output matrix, which describes how the output is derived from the state. Finally, $\mathbf{D} \in \mathbb{R}^{p \times n}$ is the feed-through matrix, which describes how the input directly affects the output. $\mathbf{D}$ is set to the zero matrix in many systems.

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \\ \mathbf{y} &= \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t) \end{aligned} \tag{2.4}$$

Recent advances in deep learning integrate state-space representations into sequence processing neural networks, including the Legendre memory unit (LMU) [23], [24], structured state space sequence models (S4) [84], HiPPO [85] and Mamba [37], [86]. Such neural networks are called "state-space models" (SSMs) and have been used effectively in language modelling [25], [37]. At the core of SSMs, a discretized state-space dynamical system is used to compute representations of sequences of length T into a state of size m. This mechanism is an effective compression method if $m < T$.

In this thesis, I explore the use of the Legendre memory unit SSM as a sequence compression layer. The dynamical system used in the LMU is based on finding optimal LTI approximations of an ideal delay system [23]. The ideal delay system produces an output signal that is an exact copy of the input signal, but delayed by some amount of time, θ. Voelker and Eliasmith recognize that such a system would require infinite state variables, but they derive an optimal, LTI approximation of the system and give the state-space matrix formulations shown in Equation 2.5, given a delay window (θ), m-dimensional state ($\mathbf{x}(t)$), and 1-dimensional inputs ($u(t)$) and outputs ($y(t)$) [23].

$$
\begin{aligned}
A_{i,j} &= \frac{2i+1}{\theta}\begin{cases} -1, & i < j \\ (-1)^{i-j+1}, & i \geq j \end{cases}, \\
B_i &= \frac{(2i+1)(-1)^i}{\theta}, \\
C_i(\theta') &= (-1)^i \sum_{l=0}^{i} \binom{i}{l}\binom{i+l}{j}(-\frac{\theta'}{\theta})^l, \quad 0 \leq \theta' \leq \theta, \\
D &= 0, \ i,j \in [0, m-1]
\end{aligned}
\tag{2.5}
$$

Given a memory state $\mathbf{x}_t$, it is possible to decode a delayed version of the input signal from the state vector of the delay network, so long as the desired delay, θ', is within the delay window, θ. The matrix $\mathbf{C}$ can be constructed to decode $u(t-\theta') \approx \boldsymbol{C}(\theta')^\top \boldsymbol{x}_t$, $\theta' \in [0, \theta]$ using the formulation in Equation 2.5. Voelker and Eliasmith note that the functions used to generate $\boldsymbol{C}(\theta')$ are shifted Legendre polynomials, which are the namesake of the system.

The original LMU in [23] is a recurrent architecture that is a non-linear dynamical system that computes functions based on a recurrent hidden state, delay network memory state, and inputs. The delay network operates on a 1-D state that is a linear combination of the input, hidden state, and previous memory state. However, the recurrence relationships in this architecture is not conducive to parallelized GPU training which is typically used for training neural networks.

Further modifications to the LMU network in [24] drop the residual connections and recurrent hidden state, thereby relying only feed-forward computations based on the current inputs and the delay network memory state. Furthermore, the delay network is applied to each dimension in the input vector, to better represent high-dimensional inputs, such as are often used in deep learning.

The formulation for the parallelized LMU is given in Equations 2.6, 2.7, and 2.8. These equations use $\boldsymbol{x}_t \in \mathbb{R}^{1\times d_x}$ as the layer input, $\boldsymbol{u}_t \in \mathbb{R}^{1\times d_u}$ as the input to the delay network, $\boldsymbol{m}_t \in \mathbb{R}^{d_m \times d_u}$ as the memory state, and $\boldsymbol{y}_t \in \mathbb{R}^{1\times d_y}$ as the output. $\boldsymbol{W}_*$ and $\boldsymbol{b}_*$ are the weight matrix and bias vector of a linear layer, and $f(\cdot)$ is a non-linear activation function.

$$
\boldsymbol{u}_t = f(\boldsymbol{W}_u \boldsymbol{x}_t + \boldsymbol{b}_u) \tag{2.6}
$$

$$
\boldsymbol{m}_t = \bar{\boldsymbol{A}}\boldsymbol{m}_t + \bar{\boldsymbol{B}}\boldsymbol{u}_t \tag{2.7}
$$

$$
\boldsymbol{y}_t = f(\boldsymbol{W}_m \boldsymbol{m}_t + \boldsymbol{W}_x \boldsymbol{x}_t + \boldsymbol{b}_y) \tag{2.8}
$$

Additionally, $\bar{\boldsymbol{A}} \in \mathbb{R}^{d_m \times d_m}$ and $\bar{\boldsymbol{B}} \in \mathbb{R}^{d_m \times d_u}$ are discretized versions of the delay network state and input matrices, since neural networks process discrete sequences. Using the zero-order hold method with $dt = 1$, Equation 2.9 describes how the discretized matrices can be obtained from the formulations for $\boldsymbol{A}$ and $\boldsymbol{B}$ given in Equation 2.5.

$$\begin{aligned} \bar{\boldsymbol{A}} &= e^{\boldsymbol{A}} \\ \bar{\boldsymbol{B}} &= \boldsymbol{A}^{-1}(e^{\boldsymbol{A}} - \boldsymbol{I})\boldsymbol{B} \end{aligned} \tag{2.9}$$

To obtain efficient computation of the state at an arbitrary time step from Equation 2.7, Chilkuri and Eliasmith [24] note that LTI systems have a non-recurrent method of evaluating the state computation given in Equation 2.10.

$$\boldsymbol{m}_t = \sum_{j=1}^{t} \bar{\boldsymbol{A}}^{t-j} \bar{\boldsymbol{B}} \boldsymbol{u}_j \tag{2.10}$$

The state computation in Equation 2.10 can be transformed into a matrix multiplication if I pre-compute the powers of $k = 1..t$ for $\bar{\boldsymbol{A}}^k \bar{\boldsymbol{B}}$ as defined in Equation 2.11. Then, with the input matrix, $\boldsymbol{U} \in \mathbb{R}^{T \times d_u}$, whose computation does not require recurrent operations over different time-steps, I can directly compute the final memory state, m_T, with matrix multiplication as shown in Equation 2.13.

$$\boldsymbol{H} = [\bar{\boldsymbol{A}}^0 \bar{\boldsymbol{B}} \;\; \bar{\boldsymbol{A}}^1 \bar{\boldsymbol{B}} \;\; ... \;\; \bar{\boldsymbol{A}}^{T-1} \bar{\boldsymbol{B}}] \in \mathbb{R}^{d_m \times T} \tag{2.11}$$

$$\boldsymbol{U} = [\boldsymbol{u}_1 \; \boldsymbol{u}_2 \; ... \; \boldsymbol{u}_T] \in \mathbb{R}^{T \times d_u} \tag{2.12}$$

$$\boldsymbol{m}_T = \boldsymbol{H}\boldsymbol{U} \in \mathbb{R}^{d_m \times d_u} \tag{2.13}$$

Chapter 3

Data

In this chapter, I describe the MIMIC-IV FHIR dataset and methods used to prepare the dataset. I then describe the tokenization method to turn FHIR JSON data into sequences of tokens for a transformer model. Finally, I describe the datasets constructed from the MIMIC-IV dataset for my experiments.

3.1 Datasets

The MIMIC-IV v2.2 *hosp* and *icu* modules are the original source of data used [70]–[72]. The MIMIC-IV data is a retrospective collection of de-identified patient electronic health records (EHRs) from the Beth Israel Deaconess Medical Center in Boston, Massachusetts between the years 2008 and 2019. The *hosp* module contains data from the hospital-wide EHR system, including patient demographics, hospital admission timing, laboratory tests, microbiology culture tests, diagnostic codes, surgical procedure codes, drug prescription, drug dispenses, records of medications administered to patients, and medication mix compositions. The *icu* module contains records from the EHR systems in the intensive care unit (ICU) department, including physician charted observations, patient vitals, dates of medical events, and records of procedures performed in the ICU. The data from MIMIC-IV v2.2 was converted into FHIR format by Bennett *et al.* [87] and is available through PhysioNet [70], [88]. In this work, the MIMIC-IV FHIR dataset is further prepared for language modelling with transformers.

The MIMIC-IV FHIR data is stored in new-line delimited JSON (ndjson) files with each file storing all the FHIR resources from a specific resource type. This is the same format that would be produced by a FHIR server bulk export (`$export`) operation. Table 3.1

and Table 3.2 describe the resource types from the *hosp* and *icu* datasets that I use from MIMIC-IV FHIR, their corresponding files from the PhysioNet dataset, and the total number of resources of each type. The filenames are in the format of `Mimic<filename>.ndjson` but are abbreviated in the tables. Some resources cannot be loaded due to JSON decoding errors but these errors account for much less than 1% of the data.

Table 3.1: Counts of each FHIR resource type in the MIMIC-IV FHIR *hosp* dataset

FHIR Resource Type	**MIMIC-IV File**	**Number of Resources**
Patient	Patient	299,712
Encounter	Encounter	431,712
Organization	Organization	1
Location	Location	39
Condition	Condition	4,756,325
Procedure	Procedure	669,186
Observation	ObservationLabevents	118,171,367
	ObservationMicroOrg	289,928
	ObservationMicroSusc	1,107,278
	ObservationMicroTest	2,184,371
Specimen	Specimen	1,587,215
	SpecimenLab	13,376,689
Medication	Medication	19,689
	MedicationMix	6,338
MedicationRequest	MedicationRequest	15,416,897
MedicationDispense	MedicationDispense	12,689,767
MedicationAdministration	MedicationAdministration	27,754,178

Table 3.2: Counts of each FHIR resource type in the MIMIC-IV FHIR *icu* dataset

FHIR Resource Type	**MIMIC-IV File**	**Number of Resources**
Encounter	EncounterICU	73,181
Procedure	ProcedureICU	696,092
Observation	ObservationChartevents	313,645,032
	ObservationDatetimeevents	7,112,999
	ObservationOutputevents	4,234,967
MedicationAdministration	MedicationAdministrationICU	8,978,893

3.1.1 MIMIC-IV FHIR Resource Types

Patient resources contain demographic and administrative information about an individual. There are fields with information about a patient's sex, gender, race, ethnicity, and age (which can be derived from birth date) that may be informative in determining health outcomes. However, there are many fields in the Patient resources that serve a more administrative function, such as name, identifiers, and contact information. MIMIC-IV does not include personal health information and performs a de-identification process that replaces patient names with random identifier strings and offsets all dates that correspond to a patient record by an arbitrary amount, so birth dates and dates of services cannot be linked to real events [72]. However, all records associated with a patient's history maintain the correct relative times.

Encounter resources characterize the location and time during which healthcare was provided. For example, they include information like whether the setting was ambulatory care, at-home care, in-patient hospital care, *etc.* In MIMIC-IV FHIR, the Encounter resources track the locations of patient care through different areas of the hospital, such as an in-patient unit, an operating room, or an intensive care unit (ICU). The Encounter also records the admission and discharge times from each location.

Condition resources in MIMIC-IV FHIR describe a point-in-time medical diagnosis. These resources use the International Classification of Diseases (ICD) [52] version 9 or 10 codes to represent diagnoses. These diagnosis codes are typically assigned for billing purposes at the end of an Encounter. These resources do not have timestamps associated with them in MIMIC-IV FHIR, so I assign timestamps based on the end of the associated Encounter resources.

Procedure resources describe any actions performed on a patient or by a practitioner. They often describe surgical interventions, diagnostic procedures, assessments, therapies, and administrative procedures, such as informing a patient or their family about care plans.

Observation resources make up the vast majority of resources in the MIMIC-IV FHIR dataset. Observation resources record measurements and examinations of a patient. The MIMIC-IV FHIR dataset includes chart events (patient vitals, physician charted observations), datetime events (records of dates of notable medical events such as dialysis, insertion of a catheter), lab events (such as blood test results and other tests performed on patient specimens), microbiology culture tests (including which bacteria are susceptible to antibiotics), and output events (such as quantities of urine or drainage). Related to Observation resources are Specimen resources, which document the type and timing of samples collected from a patient. Specimens are linked to Observations via references; the Observation resource does not directly encode information contained in a Specimen resource. MIMIC-IV

FHIR does not use standardized terminology for Observations, such as LOINC [54], [55] for lab observations.

MIMIC-IV FHIR uses several resource types to describe the life cycle of patient interaction with medications. The Medication resource identifies the drug through a drug code and quantities of ingredients, if the Medication is composed of multiple ingredients (such as IV bags). Other resource types refer to medication resources to describe the actual drug. MedicationRequest resources describe an order for a medication and the dosage instructions. These are typically prescriptions given by physicians that are entered into the hospital's pharmacy system. MedicationDispense resources describe the event that a medication is dispensed by a pharmacy to fulfil a MedicationRequest. The dispensed medication may be different than the order. MIMIC-IV is a hospital dataset, so all MedicationDispense resources describe medications from the hospital pharmacy system and not from external pharmacies. Finally, MedicationAdministration resources describe the event that a patient consumes or receives a medication. This can include self-adminsistration of oral medication, but in the MIMIC-IV hospital context, also includes a lot of IV infusion events. These medication-related resouces often contain a Dosage structure that contains information about dosing, route, and timing.

3.1.2 Resource Bundling

I grouped resources by their corresponding patients to create individual files for each patient. This format resembles the Bundle of resources returned from a FHIR server `$everything` operation that queries all the resources that reference a given patient. To do this, I iterate through all resources in a MIMIC-IV FHIR file and append the resource to a separate ndjson file based on patient ID from the `subject` field. The Organization, Location, and Medication resources do not have `subject` fields, so they are compiled as shared resources between all patients.

3.1.3 Relevant Resource Fields

For each of the resource types in the MIMIC-IV FHIR dataset, a subset of the resource fields are chosen to be encoded. These fields represent important information in the resource that are medically relevant, such as medical codes, timestamps, and measurements. The relevant fields of each resource were determined by examining the mapping of original MIMIC-IV features to FHIR fields **git˙mimic˙mappings**. Fields noted in the MIMIC-FHIR documentation that always have the same value were excluded. Fields that represent identifiers

and metadata are also excluded. I also choose the timestamp for `effectiveDateTime` and exclude other timestamps, such as `issued` timestamps. Some fields have nested components that are relevant. A colon (`:`) separated path of field names was included in the configuration. The configuration file used for experiments is described in Listing 3.1.3.

```
{
    "Patient": [
        "gender",
        "birthDate",
        "maritalStatus",
        "communication:language",
        "extension"
    ],
    "Encounter": [
        "class_fhir",
        "priority",
        "type",
        "serviceType",
        "period:start",
        "period:end",
        "hospitalization:admitSource"
    ],
    "Condition": ["code", "encounter:reference:period:end"],
    "Procedure": [
        "category",
        "code",
        "encounter:reference:location:name",
        "performedDateTime",
        "performedPeriod:start",
        "performedPeriod:end",
        "bodySite"
    ],
    "Observation": [
        "category",
        "code",
        "effectiveDateTime",
        "valueQuantity",
        "valueCodeableConcept",
        "valueString",
        "valueDateTime",
        "interpretation",
        "note:text",
        "specimen:reference",
        "referenceRange",
        "hasMember:reference",
```

```
        "collection:collectedDateTime"
    ],
    "MedicationRequest": [
        "status",
        "medicationReference:reference",
        "authoredOn",
        "dosageInstruction"
    ],
    "MedicationDispense": [
        "medicationCodeableConcept",
        "effectiveDateTime",
        "quantity",
        "dosageInstruction"
    ],
    "MedicationAdministration": [
        "category",
        "medicationCodeableConcept",
        "effectiveDateTime",
        "dosage:text",
        "dosage:method",
        "dosage:dose"
    ],
    "Dosage": [
        "text",
        "route",
        "timing:code",
        "maxDosePerPeriod"
    ],
    "MedicationAdministrationDosage": [
        "text",
        "route",
        "doseQuantity",
        "rateQuantity",
        "timing:code",
        "maxDosePerPeriod"
    ]
}
```

Listing 3.1: FHIR tokenizer configuration file

3.2 Tokenizing FHIR Resources

For language model transformers, inputs are a sequence of integer IDs that represent text tokens. For the FHIR tokenization method, the aim is to create a sequence of input IDs that represents discrete chunks of medical data based on the FHIR data structures. I create a tokenization algorithm that generates tokens based on the component datatypes of a FHIR resource. I base the method on the JSON representation of FHIR as it is a format commonly used in web API payloads. The JSON bundle of FHIR resources represents all resources associated with a single patient and the tokenization algorithm produces a sequence of tokens that represents this entire patient EHR record.

First, the JSON bundle of FHIR resources is sorted by timestamp of the resources to ensure a consistent ordering. The JSON representation of each resource is parsed into a Python data structure from the `fhirclient` package. This enables specific processing of FHIR elements depending on their data type. Each resource is processed by extracting the relevant field names according to tokenizer configuration file. The configuration file contains a list of field names for each resource type. To tokenize a resource, each relevant field is searched for by traversing the nested attributes of the FHIR resource object in a recursive, depth-first manner until an "elemental data type" is reached. I use four elemental data types: coding, numeric, timestamps, and text. Each elemental data type is converted into tokens, as described in subsection 3.2.1. A field may have multiple tokens in its representation. The resulting tokens from each relevant field that was found in the resource are concatenated together to form a sequence of tokens for that resource. An enumeration representing what resource these tokens belong is also created. The final representation for a single patient's EHR record is composed of three sequences: FHIR tokens, resource enumeration, and resource type. Each of these sequences is created by concatenating the representations of each resource together. This is illustrated in Figure 3.4.

3.2.1 Elemental Data Type Representations

The coding elemental datatype is the same as the FHIR Coding datatype. To tokenize the coding datatype, the tokenizer simply looks up the system and code from a vocabulary of codes and assigns it a token ID. I ignore the `display` field as tends to be unique with the code and is redundant. Largely, the vocabulary is composed of medical codes such as ICD-9 and ICD-10 diagnostic and procedure codes, medication codes, and other hospital-specific codes for admission categories, laboratory items, specimen descriptions, and units of measurement. These vocabulary items are easily obtainable from the FHIR CodeSystem

and ValueSet JSON files available from the MIMIC Profiles GitHub repository associated with the MIMIC-IV FHIR project **git˙mimic˙profiles**. Vocabulary items are also created for code systems that are part of the HL7 standard FHIR specification and also for items in FHIR extensions found in the MIMIC-IV FHIR dataset. A list of the existing coding systems is reported in subsection 3.2.1.

The numeric datatype includes the integer and decimal primitive types in FHIR. The numeric representations in the MIMIC-IV FHIR dataset are limited to observation quantity values and reference ranges, medication dose quantities, and medication dispense maximum dose ratios. As I experiment with different numeric encoding methods, I separate out the numeric values and the associated coding for the value in a separate sequence, and insert a sentinel token into the FHIR token sequence. This sentinel token represents the fact that a numeric value would go in that location. In a production system, a single numeric encoding method could be used to directly encode numeric values, but for the purposes of experimentation, I separate the values and codes out to process later.

Timestamps include the datetime and date primitive datatypes in FHIR. To tokenize timestamps, I use a similar approach to Rajkomar *et al.* [4]. During tokenization, I pass a reference timestamp (t_0) as input and use it to compute the difference between a resource's timestamp (t) in hours, then take the log of this timedelta. The reference timestamp is more recent than the resource timestamp, as resources record medical information in the past. Rather than using just the natural log function, I use $\ln(1+x)$ to ensure that small timedeltas are still positive. I assume that small fractions of an hour do not need to be well differentiated for our tasks, and can be treated as values between 0 and 1 rather than being logarithmically spaced. For example, a 1 second difference and a 1 minute difference would be represented as 0.0002 and 0.0165 with the $\ln(1+x)$ function, but as -8.188 and -4.787 with $\ln(x)$. Henceforth, I refer to the value $\ln(1+(t_0-t))$ as the "log timedelta", where the difference between timestamps is in hours. Because I also experiment with time encoding methods, I separate out all the log timedeltas and insert sentinel tokens that represent time values into the FHIR token sequence. The sentinel tokens can be replaced later to insert the desired encoding method into those locations.

Text includes the string primitive datatype in FHIR. While strings are a component of the CodeableConcept datatype, I consider applying the text tokenization methods to fields that are intended to be strings alone, such as `valueString` or `note:text`. To tokenize text, the tokenizer for `bert-base-uncased` is used with slight modifications. Instead of using the `[CLS]` and `[SEP]` tokens at the beginning and end of strings, two of the unused tokens were assigned to represent beginning of sentence (`[BOS]`) and end of sentence (`[EOS]`) tokens, since the `[CLS]` and `[SEP]` are be used in the construction of sequences for the MLM task over all the tokens in the FHIR transformer vocabulary. The BERT tokenizer

Table 3.3: External FHIR coding systems and their descriptions

Code system URI	**Description and usage**
https://hl7.org/fhir/R4/codesystem-condition-category.json	Categories describing the context of a Condition resource
https://hl7.org/fhir/R4/valueset-languages.json	Languages in a Patient resource
https://hl7.org/fhir/codesystem-administrative-gender.json	Gender categories in a Patient resource
https://hl7.org/fhir/codesystem-medicationrequest-status.json	Status codes for MedicationRequest resources
https://hl7.org/fhir/us/core/STU4/ValueSet-birthsex.json	Codes for birth sex in a Patient resource
https://terminology.hl7.org/5.5.0/CodeSystem-v3-ActCode.json	Describes the type of Act an instance represents, such as whether an Encounter is an emergency or in-patient visit
https://terminology.hl7.org/5.5.0/CodeSystem-v3-MaritalStatus.json	Codes for martital status in a Patient resource
https://terminology.hl7.org/5.5.0/CodeSystem-v3-NullFlavor.json	Describes why values are not present in any resource
https://terminology.hl7.org/5.5.0/CodeSystem-v3-ObservationInterpretation.json	Describes basic interpretations of Observations, such as high, in-range, or susceptible
https://terminology.hl7.org/5.5.0/ValueSet-v3-ActPriority.json	Describes the urgency with which an Act was performed, such as routine or stat for lab Observations
urn:oid:2.16.840.1.113883.6.238	Codes for race and ethnicity used by the US CDC in a Patient resource

also contains the `[PAD]`, `[UNK]`, and `[MASK]` tokens which were reused by the MLM task. Overall, the BERT tokenizer adds 30,522 tokens to the vocabulary.

In total, the tokenizer has a vocabulary of 112,319 tokens. This count includes tokens reserved for potential tokenized representations of numeric and time data. A summary of which types of code systems contribute to token counts can be found in Table 3.2.1. I choose a final vocabulary size of 112,640 as it is a nearby multiple of 1024.

Beyond the Coding datatype, the tokenizer also handles other FHIR general-purpose datatypes, including Reference, CodeableConcept, Quantity, Ratio, Period, and Range. The tokenizer treats these general-purpose datatypes as being composed of one or more fields that are Coding, numeric, string, or timestamp datatypes, applies the relevant tokenization method, and returns a list of tokens.

To handle the Reference datatype, the tokenizer creates a hash map of all the resource IDs from the input bundle to easily look up references. Because the JSON bundle from the FHIR server `$everything` operator includes referenced resources, I can assume the tokenizer has access to these resources. If a value from a referenced field needs to be used, the configuration file must specify the full path from the parent resource, through the reference, to the referenced resource's field name. For example, the Condition resource in our dataset does not have a timestamp, so to obtain a timestamp, I use `encounter:reference:period:end` as the series of field names to obtain the end time of the associated Encounter resource. When the tokenizer hits the `encounter:reference` field value, which is a URI to a different resource, it will look up that URI in the hash map of other resource in the bundle, and continue searching for `period:end` in that resource. If a representation for the entire referenced resource is desired, the configuration file would list `<field name>:reference` and the tokenizer would assume all relevant fields of that resource type should be processed and returned.

The CodeableConcept datatype is composed of one or more Coding datatypes and a human-readable text description. I omit tokenizing the `text` field because it is typically unique to each associated coding. The representation for the CodeableConcept datatype is only composed of the tokens from the Coding datatypes in the `coding` field.

The Quantity datatype is represented by the token for the numeric `value` field and a single token for the `unit`, `system`, and `code` fields, which are essentially treated the same as a Coding datatype would be. The Range datatype is tokenized by concatenating the token representations for the `low` and `high` Quantity datatypes together. Similarly, the Period datatype is the concatenation of the `start` and `end` timestamp token representations.

The Ratio datatype is tokenized by computing the result of dividing the `numerator` by the `denominator` field values. The resulting value is tokenized similar to other numeric

Table 3.4: Token counts for different code category types

Code systems	**Number of tokens**
Medication codes	32,105
BERT text tokens	30,522
ICD diagnosis codes	27,913
ICD procedure codes	12,788
Observation codes	6,211
Act codes	1,302
Units	684
Miscellaneous	592
Time tokens	101
Numeric tokens	101
Total	**112,319**

data, and the item code of the Ratio is used as the associated code for the numeric value when performing normalization.

3.2.2 Tokenization Example

Figure 3.1 shows the example Observation resource with the elemental Coding, numeric timestamp, and timestamps highlighted for the resource's relevant fields. The Reference datatype is also highlighted. Figure 3.2 shows the same resource with the relevant field names bolded and the corresponding datatypes highlighted. There are less data highlighted here than in Figure 3.1, as some fields such as `id` and `issued` are not specified to be tokenized.

The `category` and `code` fields are both CodeableConcept datatypes composed of Coding datatypes. For each Coding, the relevant token is looked up from the vocabulary based on the `system` and `code` fields.

Assume the reference datetime is 20:00:00 on the same date and time zone as the value of the `effectiveDateTime` field. The timedelta in hours would be 20:00 - 15:24 = 4.6 hours. The value for the time representation would $\log(1 + 4.6) = 1.722$. The timestamp sentinel token is inserted into the token sequence, and the value 1.722 is added to a separate list of time values.

Each of the Quantity datatypes in the `valueQuantity`, `referenceRange:low` and `referenceRange:high` fields are tokenized the same manner. Following the `valueQuantity` field, the value 77 is added to a list of numeric values. The main coding of the Observation resource, which comes from the `code:coding:code` field, is added to a separate list of codes that are associated with each numeric value. In a production setting, one method of performing numeric encoding could be done at this point, so this separation would not be required. For example, I could have computed the percentile associated with the value 77 within the distribution of all glucose measurements, then assign it a corresponding token, as is done in Rajkomar *et al.* [4]. The token for mg/dL is also looked up in the vocabulary. The tokens for each Quantity are the numeric sentinel token and the unit token. Later, if token representations for numeric values are used, they would be inserted at the locations of the sentinels.

The URI from the Reference in the `specimen` field would be located within the tokenizer's hash map for all other resources. The tokenizer would then find the `type` field within this Specimen resource and return the token for that Coding.

Figure 3.3 shows the resulting tokens and returned representations as a result of the steps described above. Figure 3.4 shows the result of tokenizing a Patient, Encounter, Observation, and Procedure resource and how the additional resource type IDs and resource enumeration are created. The resource type IDs are used in an additive resource embedding. The resource enumeration is used for compression methods to identify which subsequence of tokens belong to which resource.

3.2.3 Post-tokenization Processing

All sequences are left padded to the maximum sequence length, since this is the padding format required by FlashAttention [89]. The maximum sequence length depends on the task and dataset. The token IDs are padded with the `[PAD]` token from the BERT tokenizer. The resource type encoding is padded with `0`'s, which represent padding, as the resource type encodings start from 1. The resource enumeration is padded with `-1`'s, which are ignored by the custom compression layer code.

3.3 HAIM MIMIC Dataset

In this thesis, I consider using only the demographics and time-series data specified in [6] and filtering the FHIR data to include those data. This FHIR data is intended to match

```
{
'resourceType': 'Observation',
'id': '7e2317ea-17d7-5b14-8b6b-f003151e5f3e',
'extension': [{
    'url': 'http://mimic-fhir/StructureDefinition/mimic-lab-priority',
    'valueString': 'STAT'}],
'category': [{
    'coding': [{
        'system': 'http://terminology.hl7.org/CodeSystem/observation-category',
        'code': 'laboratory',
        'display': 'Laboratory'}]}],
'code': {
    'coding': [{
        'system': 'http://mimic-fhir/CodeSystem/mimic-d-labitems',
        'code': '50931',
        'display': 'Glucose'}]},
'subject': {'reference': 'Patient/54ee7569-2e21-57f3-8ffa-dbdf2dc859b9'},
'effectiveDateTime': '2147-01-13T15:24:00-05:00',
'issued': '2147-01-13T16:20:00-05:00',
'valueQuantity': {
    'value': 77,
    'unit': 'mg/dL',
    'system': 'http://mimic-fhir/CodeSystem/mimic-units',
    'code': 'mg/dL'},
'specimen': {'reference': 'Specimen/c7059b3e-4d5c-50f0-8f17-9ea7e1689bd4'},
'referenceRange': [{
    'low': {
        'value': 70,
        'unit': 'mg/dL',
        'system': 'http://mimic-fhir/CodeSystem/mimic-units',
        'code': 'mg/dL'},
    'high': {
        'value': 105,
        'unit': 'mg/dL',
        'system': 'http://mimic-fhir/CodeSystem/mimic-units',
        'code': 'mg/dL'}}]
}
```


Figure 3.1: Example Observation resource with the tokenizer's elemental datatypes highlighted. Coding datatypes are highlighted in green, numeric datatypes are highlighted in orange, Reference datatypes are highlighted in magenta, strings are highlighted in cyan, and timestamps are highlighted in yellow.

with the XGBoost feature dataset on PhysioNet [70], [77]. However, there are duplicate entries in the HAIM PhysioNet dataset, since there are dataset items from the same encounter

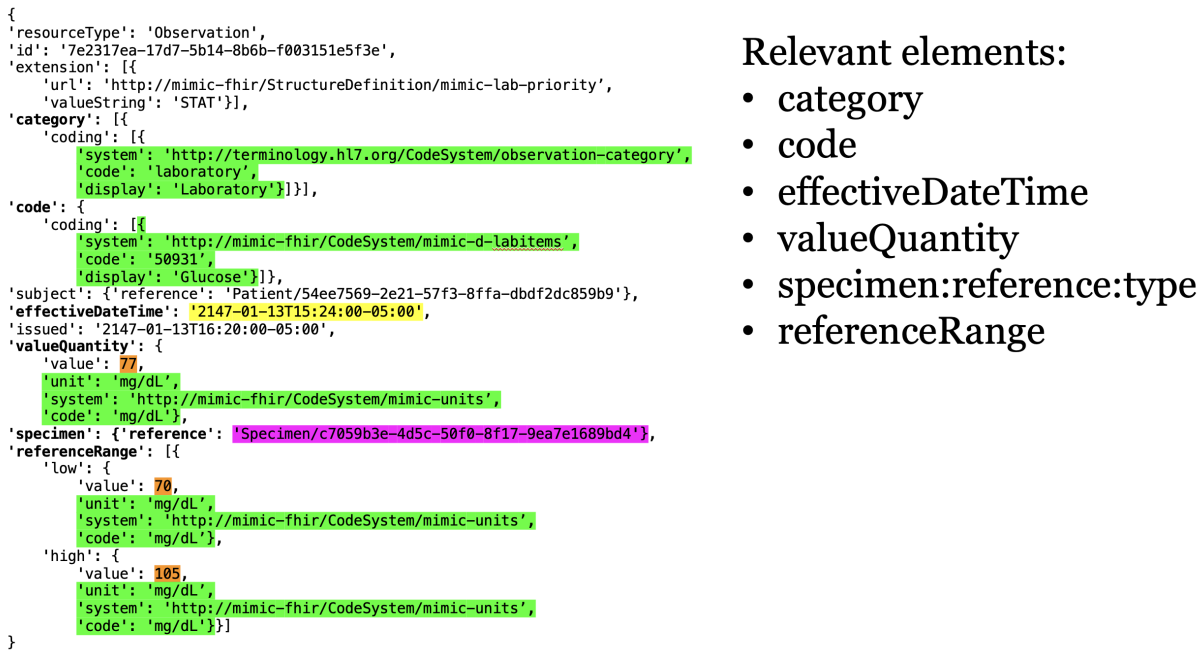


Figure 3.2: Example Observation resource with the relevant fields bolded and the associated datatypes highlighted. Note there is less data highlighted here compared to Figure 3.1. Coding datatypes are highlighted in green, numeric datatypes are highlighted in orange, Reference datatypes are highlighted in magenta, strings are highlighted in cyan, and timestamps are highlighted in yellow.

with identical time-series and demographic data, but different X-ray images. Removing duplicate entries results in a dataset that has 7k examples, which is fairly small for deep learning training purposes.

To address these concerns, I create a new dataset that filters patients using similar criteria as the HAIM MIMIC dataset, but with more relaxed patient selection. Notably, the code from the HAIM authors filters items based on the provided `hadm_id` field, which is missing for many relevant entries. Relaxing this requirement results in over 40k encounters instead of 7k.

This new HAIM MIMIC dataset is created by first finding the patients who have a chest X-ray recorded in the MIMIC CXR dataset that occurred during the hospital encounter. As in [6], the predictions are made at the time of the imaging study. Instead of taking information for the time-series items that are between the beginning of the encounter to



Figure 3.3: Example Observation resource tokenized and the associated datatypes highlighted. Sentinel tokens for numeric data (60) and time data (61) would be replaced by representations based on the encoding method later in the model. Coding datatypes are highlighted in green, numeric datatypes are highlighted in orange, Reference datatypes are highlighted in magenta, strings are highlighted in cyan, and timestamps are highlighted in yellow.

the time of the imaging, I take items from 12 hours prior to the encounter to the time of the imaging. This is because many of the relevant lab and chart events have timestamps that occur before the encounter start timestamp. This buffer allows more data to be included. I maintain the same list of demographic features and lab, chart, and procedure events as [6], which are listed in Table 3.5. Each relevant encounter's data is tokenized with the FHIR tokenizer and padded to a maximum length of 4096. Additionally, the same criteria are used to extract the same encounters from the base MIMIC-IV tables and the feature engineering of [6] is applied to create the dataset for XGBoost models.

The dataset splits in the HAIM MIMIC code are based on the generated “HAIM ID”

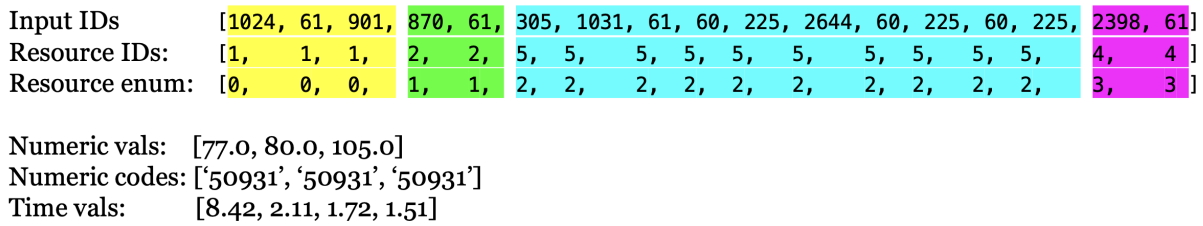


Figure 3.4: Final tokenized representation of several resources tokenized. An example Patient (yellow), Encounter (cyan), Observation (green), and Procedure (magenta) resource are shown. The resource type IDs are used to create resource type embeddings, while the resource enumeration is used for compression methods to identify which subsequences of tokens belong to which resource. The numeric vals and codes are used later to determine statistics when making numeric encoding representations, described in section 4.2. Time vals are used when making time encoding representations, described in section 4.5.

which is a combination of patient identifier, their encounter identifier, and the X-ray date. However, splitting based on this information may result in the same patient in both the training and test datasets. Instead, I split on the patient identifier, ensuring that if a patient has multiple records in the dataset, they all remain in either the training or test datasets.

Finally, I produce labels for the length-of-stay and mortality prediction tasks the same as [6]. For the length-of-stay prediction task, a positive label is assigned if the duration of the stay from the prediction time point to the end of the stay is less than 48 hours and the patient did not die within those 48 hours. The instance is assigned a negative label otherwise. This results in 25% of the data having a positive class label for length-of-stay prediction and a positive class weight of 3.0 to ensure positives and negatives are balanced. For the mortality prediction task, a positive label is assigned if the patient died within 48 hours of the prediction time point and a negative label is assigned otherwise. This results in 1.33% of the data having a positive class label for mortality prediction and a positive class weight of 73.0 to ensure positives and negatives are balanced. I use Equation 3.1 to compute this class weight.

$$\text{positive class weight} = \frac{N - N_+}{N_+} \tag{3.1}$$

3.4 Extended Context MIMIC Dataset

To test the compression methods, I create a dataset with very long sequence lengths, using data from multiple visits and many more types of resources compared to the HAIM MIMIC dataset. First, I filter for patients who have at least three visits to the hospital in their record, and that the most recent visit has a duration of at least 24 hours. I then take all the FHIR resources that correspond to the *hosp* and *icu* modules for each of those patients. The summary of resource types is listed in Table 3.1 and Table 3.2.

The Extended Context FHIR dataset is three replicates of the same patients' data but with different number of encounters. The 1 Encounter dataset uses the data from the most recent encounter, starting from 12 hours before the admission timestamp to 24 hours into the visit. The 2 Encounter dataset uses the same data as the 1 Encounter dataset and prepends the data from the second-most recent visit. The 3 Encounter dataset extends the 2 Encounter dataset by adding the third-most recent visit to the context.

Each of these three datasets is processed using the FHIR tokenizer using the configuration file provided in Listing 3.1.3. Each of these 1, 2, and 3 Encounter datasets are filtered to remove patient records which are too long or too small, based on the number of tokens and number of resources. The details of this filtering are presented in Table 3.6. Overall, less than 4% of the records are removed from each dataset and there are roughly 40k patients per dataset.

The prediction timestamp is always based on 24 hours into the last encounter and the task labels are also based on the final encounter. For the length-of-stay prediction task, a positive label is assigned if the duration of the stay is greater than 7 days from the admission time and a negative label is assigned otherwise. This results in a 15.8% positive class ratio, requiring a positive class weight of 5.3 to balance classes. For the mortality prediction task, a positive label is assigned if the patient is discharged with a status of "expired" and a negative label is assigned otherwise. This results in a 1.86% positive class ratio, requiring a positive class weight of 53.0 to balance classes.

Table 3.5: Demographic, chart event, laboratory event, and procedure event features used in the HAIM dataset. CRRT = Continuous renal replacement therapy, EEG = Electroencephalogram, GCS = Glasgow Coma Scale, MCH = Mean corpuscular hemoglobin, MCHC = Mean corpuscular hemoglobin concentration, NIBP = Non-invasive blood pressure, PICC = Peripherally inserted central catheter, RDW = Red blood cell distribution width, SpO_2 = Peripheral oxygen saturation.

Demographics	Chart events	Laboratory events	Procedure events
Age	Heart rate	Glucose	Foley catheter
Gender	NIBP - Systolic	Potassium	PICC line
Ethnicity	NIBP - Diastolic	Sodium	Intubation
Marital status	NIBP - Nominal	Chloride	Peritoneal dialysis
Language	Respiratory rate	Creatinine	Bronchoscopy
Insurance	SpO_2 by pulse oxymetry	Urea nitrogen	EEG
	Verbal GCS response	Bicarbonate	Dialysis CRRT
	Eye opening GCS response	Anion gap	Dialysis catheter
	Motor GCS response	Hemoglobin	Chest tube removed
		Hematocrit	Hemodialysis
		Magnesium	
		Platelet count	
		Phosphate	
		White blood cells	
		Total calcium	
		MCH	
		Red blood cells	
		MCHC	
		MCV	
		RDW	
		Platelet count	
		Neutrophils	
		Vancomycin	

Table 3.6: Record size filtering parameters for the Extended Context FHIR Dataset.

Number of Encounters	Min resources	Max resources	Max tokens	Ratio kept
1	3	2,048	16,384	39,595 / 40,541 = 97.66%
2	3	6,144	49,152	39,568 / 40541 = 97.60%
3	3	10,240	81,920	39,255 / 40,541 = 96.83%

Chapter 4

Transformer Model

In this chapter, I describe the transformer architecture used. I also describe the continuous encoding methods I apply to numeric and time data, and the compression layers.

4.1 Transformer Encoder

I adapt the architecture of the Qwen2 [20] model. This architecture uses the architecture of a "transformers ++" model with FlashAttention2 [90] and has public code available on HuggingFace. In principle, other models with similar architectures like LLaMA [38] could be used. At the time of investigation, a bi-directional encoder based on the Qwen2 architecture was the top of the Massive Text Embedding leader board [91], giving some validity to a similar use case in this thesis.

I use the size of the BERT-base model [16] when determining the architecture hyperparameters. This includes using a hidden dimension of $d = 768$, 12 layers, and 12 attention heads each with a head dimension $d_h = 64$. The embedding size is determined based on the vocabulary of the tokenized FHIR data and rounded up to a multiple of 1024. A vocabulary size of 112,640 is used. Since the vocabulary size is much larger than typically used in language models, I choose to maintain tied embeddings for the language modelling task, unlike in Qwen2 training where the authors opt for untied embeddings. This results in a large memory savings as the embeddings use approximately 87 million parameters.

I also use a resource type embedding layer with 10 items that represents the different resource types. The first vocabulary item, at index 0, is used for padding. The resource

type embedding vectors are added to the token vectors before passing the sequence through the transformer backbone. A diagram of the overall architecture is shown in Figure 4.1.

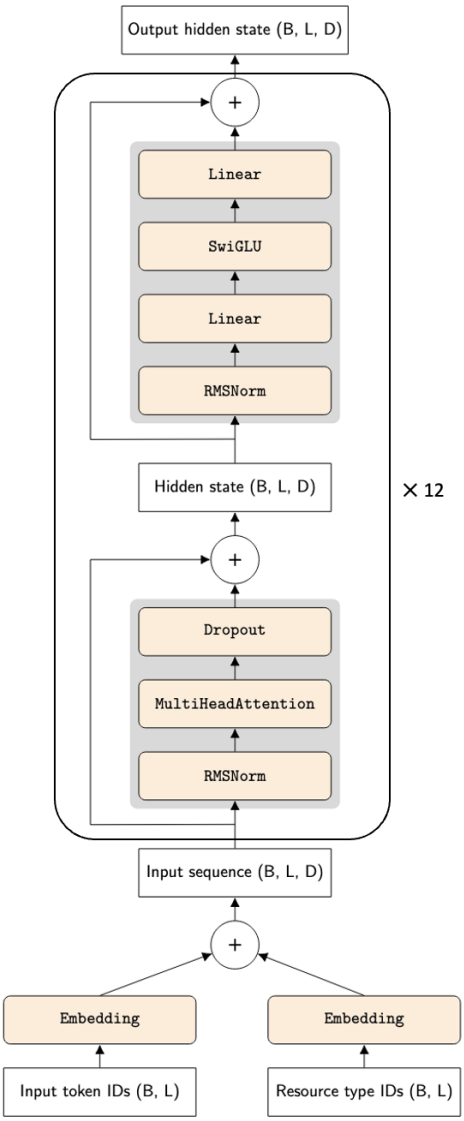


Figure 4.1: Base transformer architecture based on the BERT model using Transformer ++ improvements. I use a hidden dimension D=768 and 12 layers. The batch size (B) and sequence length (L) are different depending on the task and dataset.

4.2 Encoding Numeric Values

The numeric representations in the MIMIC-IV FHIR dataset are limited to observation measurements and reference ranges, medication dose quantities, and medication dispense maximum dose ratios. All these numeric data are positive real numbers ≥ 0.

Because raw measurement values can have vastly different ranges, standardization is a common technique applied to bring raw values into a unit variance and zero mean distribution by subtracting the mean and dividing by the standard deviation. This standardization must be done to every different type of measurement or quantity in the dataset.

The discrete encoding based on percentiles used in [4] informs our first method of transforming the raw numeric values from the FHIR records. The "Quantile numeric base" uses the entire distribution of values in the dataset to estimate the probability that all other values in the distribution are less than the given value. The quantile of a value within its distribution can be obtained from the cumulative distribution function (CDF), which computes the probability that all values in the distribution are less than or equal to the given value. Because the CDF outputs a probability between 0 and 1, this is an effective method of normalizing raw numeric values to an acceptable range. The CDF for each type of numeric measurement is estimated by computing the 100 percentiles and using those to linearly interpolating the CDF from any input quantile.

$$\bar{x} = P(X \leq x) \in [0, 1] \tag{4.1}$$

I also explore a more typical method of normalizing raw values by performing a form of standardization. The distributions of many of the measurements are heavily skewed, typically with a long tail for large outliers. Outliers can cause the mean and standard distribution to be poor estimates of centre and spread of the distribution. Instead, I normalize the data by subtracting the median and dividing by the mean absolute deviation from the mean (MAD). I investigate other estimates for spread by normalizing lab measurements between the 5th percentile to 95th percentile inclusive by subtracting by the mean and dividing by various formulations for deviation. I exclude the extrema as they can be several thousand times greater than the median and make visualizing the distribution difficult. I find that mean absolute deviation from the mean had the highest R^2 value on the quantile-quantile probability plot and the shape of the distribution best fit a standard normal distribution, which is plotted in red in Figure 4.2.

To use this normalization method, the median and MAD need to be computed for each of the different lab items, patient vitals, and other numeric measurement types in

the dataset. I compute these statistics directly from the MIMIC-IV tables rather than the FHIR dataset.

$$\begin{aligned} \text{MAD} &= \frac{1}{n}\sum_{i=1}^{n}|x_i - \text{med}(X)| \\ \bar{x} &= \frac{x - \text{mean}(X)}{\text{MAD}} \end{aligned} \tag{4.2}$$

The following sections describe the methods to encode numeric values. I apply a discrete approach, by converting the quantile values or the normalized values into bins which are each assigned a random embedding vector. I also test various continuous encoding methods that transform a 1-dimensional quantile or normalized value into a d-dimensional vector.

4.3 Discrete Numeric Encoding

When performing discrete numeric encoding, the quantile numeric base or normalized numeric base values are assigned a bin, which is assigned a token ID. The series of token IDs for all numeric values is then inserted back into the rest of the token IDs by replacing the numeric value sentinel tokens. Following this, the token ID sequence is passed to the embedding layer as normal. If I were not experimenting with numeric encoding methods, this process of computing the numeric base and assigning a bin could be done in the tokenizer.

4.3.1 Quantile Bins

The number of bins used to encode the quantile values depends on the number of unique values in the dataset for the item code. If there are more than 100 unique values, 20 bins are used for quantiles, which corresponds to every fifth percentile. Otherwise, only 5 bins are used, so each bin corresponds to every 20th percentile.

For medication doses and medication maximum dose ratios, each unique medication generally has <5 unique values over which the vast majority of doses are concentrated. Using 5 quantile bins would not be appropriate, as there are often one or two unique dosage values that account for more than 90% of the data per medication. Across all medications, there are less than 2500 unique values. Therefore, percentile bins across all possible values

are used to discretize numbers for medication dose quantities and medication dispense maximum dose ratios. I use the distribution of unique values to compute the percentiles to expand the range and avoid repeat bins when there are high-frequency values, since there are some unique values that are highly overrepresented in the dataset and they may occupy multiple percentiles.

Overall, 100 tokens are assigned to represent the 100 percentile bins that are possible. For the 5 and 20 quantile bins, the tokens chosen are the percentiles at the midpoints of those bins. For example, for a quantile bin representing the 20th to 40th percentile, I choose a token for the 30th percentile.

One potential issue with encoding numeric data with discrete bins is that these bins do not reveal how much bigger a measurement is compared to another. Furthermore, there is no guarantee the network is learning that certain bins are larger than others.

4.3.2 Normalized Bins

To quantize the range of values resulting from normalization, I create 20 bins between the range $[-5, 5]$ and each bin is mapped to a single token. For the medication dose quantities and medication dispense maximum dose ratios, the range of possible values is logarithmically distributed, so I instead just take the log of the values as the input value to the discrete and continuous encoding.

4.4 Continuous Numeric Encoding

When performing continuous numeric encoding, the sequence of quantile or normalized numeric base values are passed through a separate continuous encoding layer to produce a sequence of vectors. This process is illustrated in Figure 4.4. The continuous encoding layer performs a function $f : \mathbb{R} \to \mathbb{R}^d$ that transforms each 1-dimensional real input value to a d-dimensional vector. In machine learning, it is common to use a set of d continuous basis functions $f_i : \mathbb{R} \to \mathbb{R}$ and evaluate each basis function at the input value. The vector of all basis function outputs would form the vector representation.

I consider the properties of the vector encoding method itself and also its behaviour in the dot product attention function. If a given key vector, $\boldsymbol{k} = f(x_0)$ represents some numeric value, I consider how the similarity score between $\boldsymbol{k}$ and all other query vectors on some range, $\boldsymbol{q} = f(x),\ x \in [a, b]$, looks like. That is, the similarity function of vector

encoding function, $f : \mathbb{R} \to \mathbb{R}^d$, over range $[a, b]$ is $\text{sim}(a, b, x_0) = f(x) \cdot f(x_0),\ x \in [a, b]$. I plot a lower-dimensional example of the basis functions I consider and the dot product similarity over the range $[-10, 10]$ with the vector representation, $\boldsymbol{k} = f(0)$ in Figure 4.3 and discuss each method in the subsections below.

4.4.1 Linear Encoding

One potential method to transform a scalar numeric input into a vector is to do a linear projection from 1-dimension into d-dimensions. However, this is essentially representing the input as a scaled version of some random, high-dimensional vector. This is not an effective use of the high-dimensional vector space to represent a diverse range of inputs. Linear projection is also problematic for attention, since it guarantees all numeric values would have pre-softmax dot product similarity scores that are a linear function of the original value. This could force the attention operation to combine information only from the largest or smallest numeric values in the sequence.

4.4.2 Multi-layer Perceptron

While linear projection would have issues, a multi-layer perceptron (MLP) is another standard deep learning method to encode inputs into a vector space. MLPs operate as learnable basis functions and have the benefit of being updated by backpropagation to learn optimal parameters. I use a MLP with $d = 768$ hidden neurons that use the same swish [45] activation function as the transformer architecture.

4.4.3 Gaussian Basis

Gaussian basis encoding could be seen as a smoother one-hot encoding for values over a finite range. I evaluate the continuous input value through a series of Gaussian basis functions that are shifted and scaled to cover a specific range. With normalized numeric base, I represent the range $[-10, 10]$, and with quantile numeric base, I represent the range $[0, 1]$. The range specifies how to evenly space the centres of d Gaussians and the standard deviation is set to $(\text{high} - \text{low})/d$. The means or standard deviation of the Gaussian regression are not updated through backpropagation. The Gaussian basis similarity function looks like a Gaussian, so attention scores are high when the input values are close. One potential downside of this method is that when using the normalize numeric base, many outliers could land outside the $[-10, 10]$ range, where their representations will

be the zero vector. I chose the range $[-10, 10]$ because normalized measurements between the 5th and 95th percentile fall between $[-6, 6]$ and expanding this will accommodate small outliers. However, there are some measurements with maximum values several thousand times greater than the 95th percentile.

4.4.4 SSP Encoding

I also consider using SSPs to encode continuous values. SSP representations will ensure the output vectors of two nearby inputs will be highly similar, as measured by the cosine similarity function. For a given real input, k, the cosine similarity between the SSP encoding of k and the SSP encoding of nearby values, $q \in [k - \Delta, k + \Delta]$ looks like a peak near k and decreases to small, oscillating values far from k. The width of this high similarity peak can be controlled by scaling inputs before applying the SSP encoding. Computing cosine similarity includes computing the dot product, so it has a high correlation with attention scores. Unlike the Gaussian basis encoding, SSPs have non-zero representations for all $k \in \mathbb{R}$.

The encoding of the continuous value $k \in \mathbb{R}$ with the SSPs using the basis vector $\mathbf{x}$ is given in Equation 4.3. To improve stability and ensure that the operation can be computed efficiently on GPUs, there are some restrictions on the structure of $\mathbf{x}$.

$$\phi_{\mathbf{x}}(k) = \mathbf{x}^k = \mathcal{F}^{-1}\left\{\mathcal{F}\{\mathbf{x}\}^k\right\}, \; k \in \mathbb{R} \tag{4.3}$$

Firstly, to ensure that element-wise exponentiation does not change the magnitude of the resulting vector when $|k|$ is large, I should choose elements of $\mathbf{X} = \mathcal{F}\{\mathbf{x}\}$, $\mathbf{X} \in \mathbb{C}^d$ to be unitary, i.e. $|\mathbf{X}_i^k| = 1$. When the inverse Fourier transform is applied, this ensures that $\phi_{\mathbf{x}}(k)$ will be unit length. To meet this requirement, I choose $\mathbf{X} = e^{2\pi j \cdot \boldsymbol{\theta}}$, $\boldsymbol{\theta} \in \mathbb{R}^d$.

Secondly, since neural network input vectors are expected to be real, I can apply the conjugate symmetry of real-valued FFTs and reduce the size of the complex vectors by half, which reduces memory usage. If $\mathbf{x}$ is real, then $\mathbf{X}$ is even-symmetric, such that $\mathbf{X}_{d-i} = \mathbf{X}_i^*$. Additionally, the first component, representing the zero-frequency term must be real, $\text{Im}(\mathbf{X}_1) = 0$.

Combining these requirements, $\boldsymbol{\theta} \in \mathbb{R}^{d//2}$ must have the first element representing the zero-frequency term, which must have both a magnitude of 1 and be real. Choosing $\boldsymbol{\theta}_1 = 0$ satisfies these requirements, such that $e^{2\pi j \boldsymbol{\theta}_1} = 1$. Putting this into the SSP equation, I can parametrize the operation with $\boldsymbol{\theta} \in \mathbb{R}^{d//2}$ and operate on continuous input $k \in \mathbb{R}$ in 4.4. The SSP operator represents shifting the phase of each Fourier frequency component

by an amount θk and mixing the components to obtain a vector representation. For a neural network, I can initialize $\boldsymbol{\theta}$ from a random distribution and learn optimal parameters through backpropagation.

$$\phi(k|\boldsymbol{\theta}) = \mathcal{F}^{-1}\left\{e^{2\pi j(\boldsymbol{\theta} \odot k)}\right\} \tag{4.4}$$

Furthermore, this operation can be effectively run in torch using a few lines of code. PyTorch has the real FFT operation. To ensure circular convolution is computed, `n=D` must be set.

```
phis = x @ (theta / h)
phis = torch.fft.irfft(torch.exp(2.j * torch.pi * phis), n=D)
```

Listing 4.1: PyTorch code to compute SSP encoding.

4.4.5 SSP Similarity Basis

The similarity function of the SSPs closely resembles the sinc function except it is periodic. This function is shaped similarly to a Gaussian with a peak at the centre, but has oscillations that dampen to 0. These functions can form a set of basis functions that are equally spaced between a given period, which can be set based on the range of values. This set of basis functions is obtained when using the SSP FFT formulation in 4.4 by setting $\boldsymbol{\theta}$ to d linearly spaced points between $[0, 1]$, where d is half the SSP dimension, due to conjugate symmetry.

With SSP similarity function as basis functions, the dot product similarity function between a key vector and other queries is also the same sinc shaped function. One concern about the periodicity of these functions is that there is no difference between values x and $x+T$ where T is the period. This may be problematic for representing extrema. That is, x and $rem(x, T)$ are the same. When using the normalize numeric base, most standardized values are distributed around 0. It is unlikely that a large extrema would have a remainder of 0, but this does not guarantee the representation of an outlier will be greater than the median or less than the median in a meaningful way. This would not be an issue with the quantile numeric base, as all values are guaranteed to be in the range $[0, 1]$.

Using a random $\boldsymbol{\theta}$ is referred to as "SSP Encoding" and using the linear spaced $\boldsymbol{\theta}$ is referred to "SSP Similarity Basis" due to the equivalence to encoding using the SSP similarity function as the basis.

4.5 Encoding Timestamps

One of the elements that lacked investigation in previous works was the method for encoding time between events in a medical record. I describe the methods used to encode time, specifically the delta time between a reference timestamp, t_0, and other resource timestamps, t, in the following section.

I adapt the methods from Rajkomar *et al.* for our representation of timestamp values [4]. Rajkomar *et al.* compute the difference in seconds between a reference timestamp and the timestamp of most recent element in the record. They then round the logarithm of this timedelta to the nearest integer.

Our HAIM dataset does not have timedeltas in the range of seconds to minutes and the minimal timedeltas for the HAIM dataset are in the range of hours. I therefore compute the timedelta in units of hours for our time representations. I also take the logarithm of the timedelta with $\text{log1p(x)} = \ln(1 + x)$ to ensure a timedelta of 0 can be encoded. There are not small fractions of an hour in our dataset, but extending to a real-world scenario, using a base timedelta in seconds could be done to deal with the distortion at small values for the log1p function. The absolute value of the timedelta is taken to prevent taking log of negative values, however, the sign of the timedelta is recorded and there are separate bins for both positive and negative deltas. A negative timedelta may occur if the a resource has a timestamp in the future of the reference resource, for example, if the discharge time is used as the reference but the recorded timestamp of a resource was entered by a doctor after a patient was discharged. For the purposes of our prediction tasks, we do not include resources with timestamps in the future of the reference timestamp, but when including the entire patient history for pre-training, future timestamps may occur.

4.5.1 Round to Integer

This method is adopted from [4] and simply rounds the log timedelta to the nearest integer. Equation 4.5 describes how a resource timestamp, t, is converted to a token ID with the Round to Integer encoding method, including the steps to taking the logarithm. The T_{offset} is an integer offset that describes where in the vocabulary the time token representations start.

$$\text{token ID}(t) = T_{\text{offset}} + \text{sgn}(t_0 - t) \cdot \text{nint}(\ln(1 + |t_0 - t|)) \tag{4.5}$$

The times corresponding to each token are represented in the table below. Based on the binning of this representation, there is a lack of resolution in smaller time ranges, where

Table 4.1: Bin edges for "round to int" timestamp encoding method

Bin Number	Upper edge
0	0 h
1	1.72 h
2	6.39 h
3	19.09 h
4	2.23 d
5	6.14 d
6	16.77 d
7	45.65 d
8	124.16 d
9	337.16 d
10	2.51 y
11	6.83 y
12	18.58 y
13	50.50 y
14	137.28 y

I expect that within a visit those time differences matter. Most of the distribution of timestamps that would occur within a single encounter are between bins 3-5, or 19 hours to 6 days. With only three bins representing this time range, there is a significant lack of resolution. There is also a lack of resolution on large timedeltas, which would represent ages. Particularly, this implies that patients are between 6-18 years old, 18-50 years old, and 50-137 years old. Disease morphology can be significantly different for ageing patients and there is a lack of resolution among the adult ages.

To address the shortcomings of the method used in Rajkomar *et al.*, I propose the following methods in the subsequence subsections to encode timedeltas.

4.5.2 Logspace Bins

The first proposed method for timedelta encoding is to simply increase the resolution of the bins. The largest integer encoding log timedeltas in the dataset would be 14. Instead of relying on integer resolution for discretizing the log timedeltas, I create 100 bins in the logarithmic range $[0, 14]$. This encoding method has a finer resolution for ages at the upper

end and for resources recent to the reference timestamp. Practically, this can be achieved by using the `np.digitize` function with an array of bins.

4.5.3 Percentile Bins

The next proposed method is to use the statistics of the timedelta distribution to determine the bins. This would maintain a high resolution around the most commonly used timestamps. Similar to logspace bins, I choose 100 bins and use the percentile statistics to determine bin ranges. I sample the distribution of timedeltas from 10,000 records in the dataset and compute the log timedeltas of all resources in those records to the end point of the most recent encounter. When including frequency of each timedelta to compute the percentiles, there are certain timedeltas that are overwhelmingly represented and therefore take up many bins, such as 0 representing when resources have the same timestamp as the reference. The bins at the upper end representing ages are sparse. To address this, I compute the percentiles based on the distribution of unique timedeltas, which expands the upper range while still maintaining a greater resolution around common timedeltas. Once the bins are computed, `np.digitize` can be used to create token IDs.

4.5.4 Continuous Encoding

In addition to the discrete encoding methods described above, I also encode the log timedelta using the same continuous methods used for numeric encoding, namely Gaussian basis (subsection 4.4.3), MLP encoding (subsection 4.4.2), SSP encoding (subsection 4.4.4), and SSP similarity basis (subsection 4.4.5).

4.6 Sequence Compression Layer

To address the challenge of modelling long sequence lengths, even with our improved FHIR tokenization, I sought to test methods that compress a sequence of vectors representing one resource into a single vector. Thus, the sequence length of a representation for a patient record would be the number of resources in that record. For our custom FHIR tokenizer, this is approximately an 9× reduction in sequence length, while for text, this could result in around a 300× reduction in sequence length.

The resource enumeration is used by the compression functions to determine which subsequence of embedding vectors belong to a resource. I apply a compression function to

Table 4.2: Examples of how different raw values are processed before applying various encoding methods. The top two rows exemplify how continuous data is encoded using discrete binning. The bottom three rows exemplify the way quantile and normalize numeric base methods produce inputs for continuous encoding methods. interp = linear interpolation function, M = median, MAD = mean absolute deviation

Example item code	**Method**	**Function parameters**	**Raw value**	**Resultant value**
Creatinine	Quantile Bins	Bins edges: [0.5, 0.7, 0.9, 1.1, 1.8]	0.98	60th %ile → token 160
Timedelta	Round to Integer	N/A	3.57	time bin 4 → token 204
Glucose	Quantile	CDF estimate: x = [0.0, 63, ..., 350, 500] y = [0.0, 0.01, ..., 0.99, 1.0]	150	interp(150, x, y) = 0.7966
Blood pressure	Normalize	Median: 117 MAD: 18.8	120	(x - Median) / MAD = 0.159
Drug dose	Normalize (log)	N/A	0.1	ln(0.1) = -2.30

each subsequence of vectors independently. Our input is thus $X = (X^1, X^2, ..., X^i, ..., X^N)$ where each X^i is composed of a sequence of vectors, $(x_1, x_2, ..., x_{k^i})$ that correspond to resource $i = 1..N$. Our compression function, $\mathcal{F} : \mathbb{R}^{k^i \times d} \rightarrow \mathbb{R}^d$ is applied to each such X^i and the resulting compressed vectors are concatenated to create a final compressed sequence $X_c \in \mathbb{R}^{N \times d}$. I add the resource type embedding and perform padding to the sequence of compressed vectors to a maximum sequence length before the compressed sequence is processed by the transformer. The high level process is illustrated in Figure 4.5. In the following subsections, I detail the methodology of the three compression functions I consider: sum compression, LMU compression, and cross-attention compression.

4.6.1 Sum Compression

For sum compression, I define the compression function to be the sum across the sequence dimension of X_i, shown in Equation 4.6. This can be thought as a pooling operator over the subsequence of vectors corresponding to one resource. Taking the sum or mean of

a sequence is a common pooling technique in language modelling. In practice, this sum compression can be easily implemented with the `torch.scatter_reduce` operator. An index tensor derived from the resource enumeration is an additional input to the function. This index indicates which input vectors are included in the sum reduction, and the result's index position. For example, the first five elements of the index could be $[0, 0, 0, 1, 1, ...]$. This indicates the first three input vectors are summed and the result is placed into index 0 of the output, and the next two input vectors are summed and the result is placed into index 1 of the output. The compressed sequence can be padded to a maximum length after the reduction.

$$\mathcal{F}(X^i, k^i) = \sum_{j=1}^{k^i} X_j^i \tag{4.6}$$

4.6.2 LMU Compression

For LMU compression, I define the compression function to be a linear projection of the final state of the LMU memory state after processing k_i vectors. To obtain the input $\boldsymbol{U}$ matrix, the input sequence hidden dimension is projected down to d_m. Then with $\boldsymbol{H}$ defined in Equation 2.11, final memory state at $T = k_i$ is computed using Equation 2.13. Because the memory state is multi-dimensional, I flatten before applying the linear projection to obtain the final compressed representation.

The hyperparameters for d_u, d_m and d_y are chosen based on the dimensions used in multi-headed attention. Similar to how state-space models are used in language modelling, I treat the state-dimension, as the number of heads, so I set $d_m = 12$ for our model. I treat the input dimension to be similar to the head dimension, so I set $d_u = 64$. The final linear projection is back to the original embedding size, so I use $d_y = 768$.

$$\begin{aligned} \boldsymbol{U} &= \boldsymbol{X}\boldsymbol{W}_u \in \mathbb{R}^{k_i \times d_u} \\ \boldsymbol{M} &= \boldsymbol{H}\boldsymbol{U} \in \mathbb{R}^{d_m \times d_u} \\ \boldsymbol{Y} &= \text{flatten}(\boldsymbol{M})\boldsymbol{W}_y \in \mathbb{R}^{1 \times d_m d_u} \end{aligned} \tag{4.7}$$

To create compressed representations for all resources in a record, I need to iterate through all the sub-sequences of vectors for each resource, which may be different lengths. However, since GPUs operate best with batched matrix transformations, I create batches

of resources of set sequence lengths and left pad all resources smaller than those desired sequence lengths with 0 vectors. By using left padding, the LMU essentially sees 0 vectors as old inputs, and its most recent inputs will be the relevant sequence elements.

I find almost all resources are less than 64 elements long, with a majority being less than 16 elements long. I choose bin edges of $[0, 8, 16, 32, 64, \max(\text{lengths})]$ to make batches of resources whose representations have those lengths. I assume each resource can be processed independently, which fits with how GPUs execute batched operations.

4.6.3 Cross-attention Compression

Cross-attention compression is inspired by the Perceiver IO architecture [33], which uses cross-attention to encode a long sequence length into a shorter length latent space. Unlike self-attention, cross-attention uses a different sequence for the queries than the keys and values. To obtain shorter output sequences, the query inputs should have a smaller sequence length.

For cross-attention compression, I define the compression function as the resulting vector after the multi-headed cross-attention operation with keys and values of the sequence elements in $\boldsymbol{X}_i$ and a single query vector that represents the resource type. I choose this query vector so that different resource types might learn different kinds of elements to select through the attention operation.

The single-head cross-attention operation is shown in Equation 4.8. I set $d_h = 64$ to be the head dimension of the main transformer and use 12 heads. The query, x_{resource} is the resource type embedding vector.

$$
\begin{aligned}
\boldsymbol{Q} &= \boldsymbol{W}_q x_{\text{resource}} \in \mathbb{R}^{1 \times d_h} \\
\boldsymbol{K} &= \boldsymbol{W}_k X_i \in \mathbb{R}^{T \times d_h} \\
\boldsymbol{V} &= \boldsymbol{W}_v X_i \in \mathbb{R}^{T \times d_h} \\
\boldsymbol{Y} &= \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}}\right)\boldsymbol{V} \in \mathbb{R}^{1 \times d_h}
\end{aligned}
\tag{4.8}
$$

Similarly to LMU compression, resources are processed in batches and sub-sequences are padded to maximum lengths according to the length of resources in that batch. I use attention masks to ensure only non-padded elements are included in the attention operation.

4.7 Classification Tasks

Both the LOS and MP tasks are binary classification tasks. I use the binary cross-entropy loss function, but with a weight, $w_+ > 1$. The weight is applied to the minority positive class examples and increases the effect they have on the loss term, particularly on misclassifying positive samples. Equation 3.1 describes how to obtain the positive class weight and Equation 4.9 describes the equation for the loss term that incorporates this weight.

$$\begin{gathered} \mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} L(x_i, y_i) \\ L(x, y) = w \cdot y \cdot \log(x) + (1 - y) \cdot \log(1 - x), \; w = \begin{cases} w_+, & y = 1 \\ 1, & y = 0 \end{cases} \end{gathered} \tag{4.9}$$

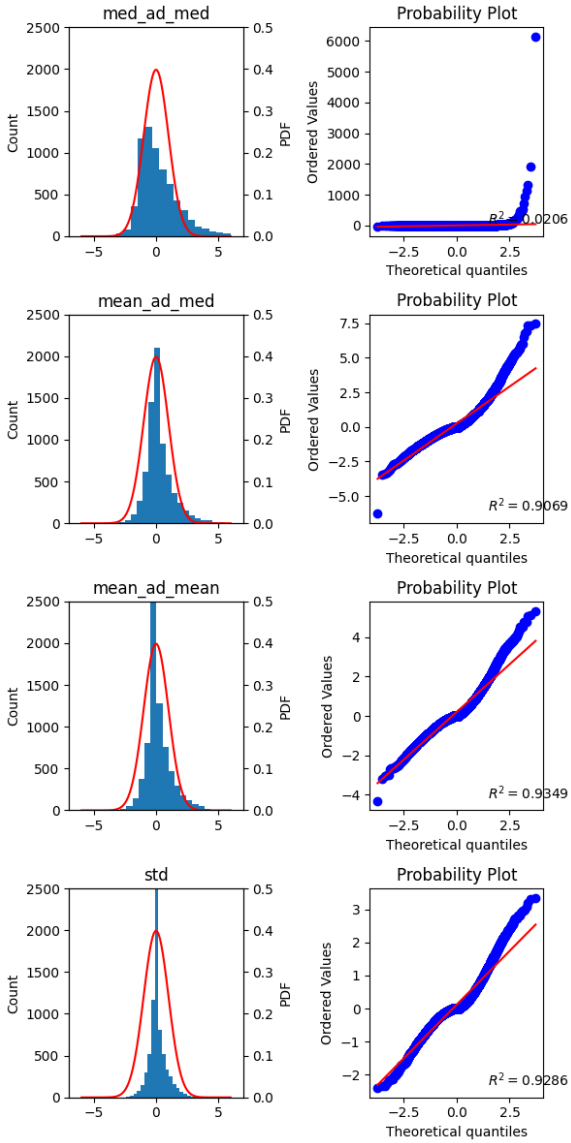


Figure 4.2: Distribution of normalized values after applying $x = x - \text{median}/\text{spread}$ to lab event measurements between the 5th and 95th percentile. The different estimates for spread are med_ad_med = median absolute deviation from the median, mean_ad_med = mean absolute deviation from the median, mean_ad_mean = mean absolute deviation from the mean, std = standard deviation. The red curve is the standard Gaussian distribution. Distributions more closely shaped to the standard Gaussian and have a higher R^2 value on the probability plot are more Gaussian.

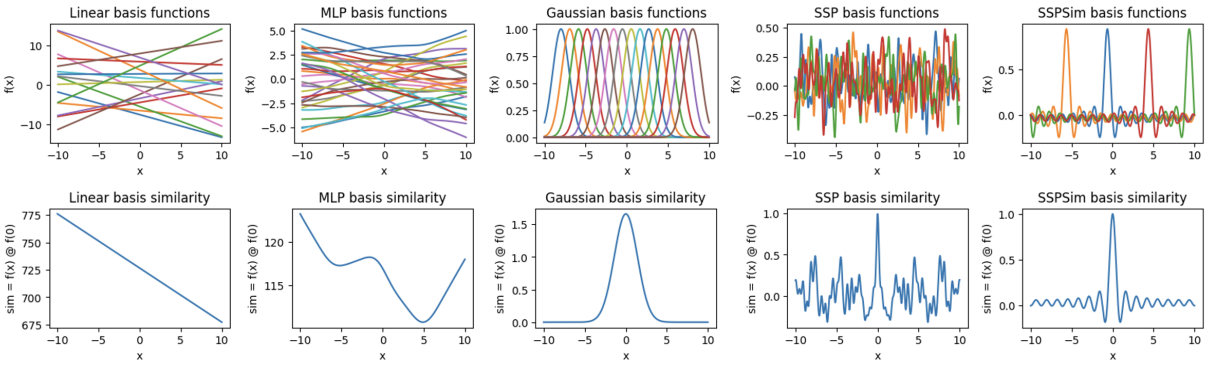


Figure 4.3: Continuous encoding basis functions and dot product similarity function $y = f(x) \cdot f(0)$ over the range $[-10, 10]$.

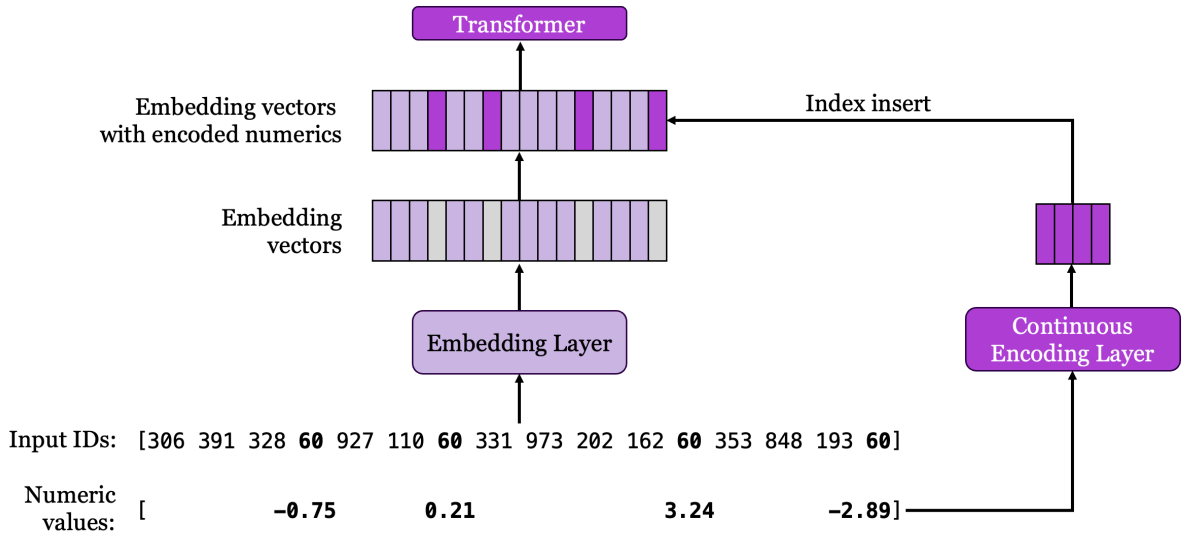


Figure 4.4: Diagram of the continuous encoding process. In this example, normalized numeric base values are passed through the continuous encoding layer that creates a sequence of vectors from a sequence of 1-dimensional values. These vector encodings are inserted back into the embedding vector sequence at the positions of the numeric sentinel tokens (token 60). A similar process can be applied if performing continuous encoding of time values.

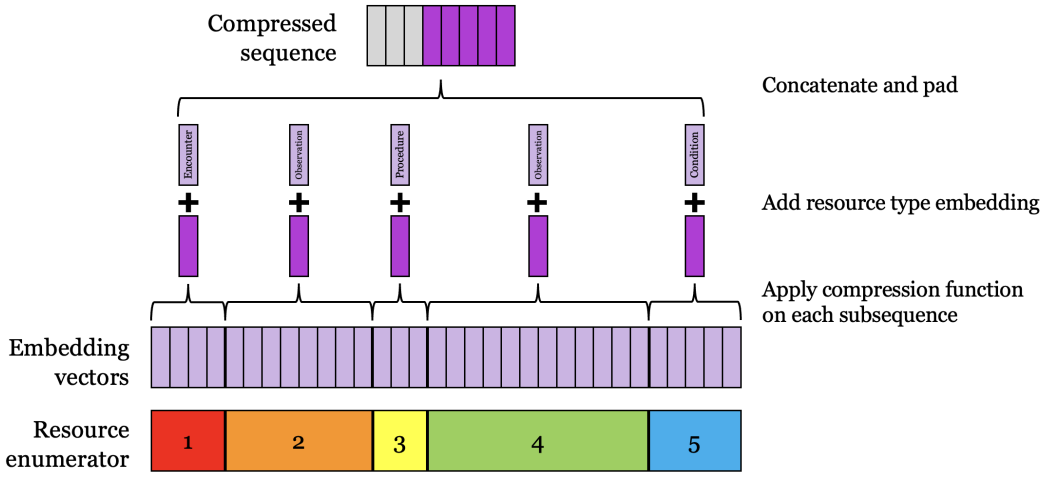


Figure 4.5: Diagram of the resource compression process. The resource enumeration indicates which embedding vectors are grouped together. The compression function is applied independently to each group of vectors.

Chapter 5

Experimental Methods

In this chapter, I describe metrics for evaluating the different proposed methodology, the training hyperparameters for experimentation, and the experimental design.

5.1 Metrics

The binary classification tasks for length of stay prediction and mortality prediction are imbalanced. Therefore, different test metrics may have different interpretations compared to balanced classification problems. For all four metrics, a higher score is better.

Previous medical AI literature uses area under the receiver operating curve (AUROC) as the main metric for evaluation [4], [6], [21]. The AUROC metric is computed as the area under a curve for which the x-axis is the false positive rate and the y-axis is the true positive rate, which come from different classifier thresholds. The AUROC represents the probability that a positive example will be ranked higher than a negative example, regardless of the decision threshold. A no-skill classifier would have an AUROC score of 0.5 [92]. However, AUROC can be optimistic if the classes are imbalanced [93].

I also report on the average precision score (APS). The APS computes the weighted average of the precision obtained at different decision thresholds. APS approximates the area under the precision-recall curve, but is less optimistic. Area under the precision-recall curve is suggested as an alternative metric for imbalanced datasets, as interpretation of the receiver-operator curve can be misleading when the negative class is dominant [94]. This is because the false positive rate used in AUROC is the ratio of false positives to all negative predictions. The model may be able to predict the majority negative class

well, and achieve a low false positive rate and thus high AUROC. However, this does not guarantee good performance at predicting the minority positive class. A no-skill classifier would have an APS that is equal to the fraction of the positive class.

Lastly, I report the unweighted accuracy and F1-scores. Accuracy, which is the ratio of correctly classified examples to the total number of examples, can be overly optimistic for imbalanced classification problems, because a no-skill classifier can guess the majority class to obtain a good score. Furthermore, improving performance on the minority class may result in a reduction in accuracy. F1-score is the harmonic mean between the precision and recall. However, performance may be optimized by choosing a different decision threshold as informed by the receiver operating curve and precision recall curves. Therefore, these metrics are less reliable indicators of performance as they represent performance at only one decision threshold. For all the results, I report accuracy and F1-score on a decision threshold where the positive class is chosen if the class probability is ≥ 0.5.

For comparing the transformer to XGBoost models, I use the independent T-test to assess whether the mean scores of two groups have identical means. For all other grouped comparisons, I use one-way ANOVA to determine whether the means of all groups tested are the same. If I reject the ANOVA null hypothesis that all groups have the same mean, I use Dunnett's multiple comparison test to compare the means of the experimental methods against the mean of the control method to determine whether any experimental methods are significantly different [95]. I use two-tailed tests in all cases and use a significance value of $\alpha = 0.05$ to reject the null hypothesis.

5.2 Training Hyperparameters

I train directly on the binary classification tasks and do not perform any pre-training. I initially attempted to pre-train transformer models on the MLM task, but this did not result in meaningful improvements. A further discussion of pre-training can be found in Appendix A.

I use the AdamW [96], [97] optimizer with the hyperparameters $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$, and weight decay $\lambda = 0.1$ [38]. I use a cosine decay learning rate schedule with 10% warm-up steps [16] to a peak learning rate of 3×10^{-4} and decay to 10% of the peak learning rate [20]. I train with a total batch size of 64 over 10 epochs, which is approximately 6.3k steps for both the HAIM dataset and the Extended Context dataset. I perform evaluations every 100 steps and save the best checkpoint based on the validation AUROC. The best checkpoint is evaluated on the held-out test set. I train 5 replicates

of the model with different initialization seeds for the parameters and setting the dataset splits for training, validation, and testing.

I use bfloat16 mixed-precision training, but allow for float32 for attention activations and RMSNorm computations in the Qwen2 architecture [20]. I also use float32 for FFT computations in SSP layers. I train on a single node with 4 Nvidia A100 40GB GPUs to accelerate training. Distributed training is accomplished using the Huggingface `transformers` and `accelerate` libraries to enable PyTorch Fully Sharded Data Parallel [98] with sharded gradient operations to decrease training runtime and GPU VRAM usage. I also accelerate the attention computations with FlashAttention [89].

5.3 Experimental Design

I organize the experiments into three subsections. Firstly, I compare a baseline transformer model with the XGBoost model from [6] on the HAIM dataset. Next, I investigate variations of encoding methods for numeric and time data, including continuous encodings on the HAIM dataset. Lastly, I investigate whether using compression methods will improve the memory, runtime, and performance on the Extended Context dataset.

5.3.1 Baseline Transformer on HAIM Dataset

For this experiment, I compare the performance of a baseline transformer model using tokenized FHIR data as inputs to the XGBoost model developed in [77] that uses tabular and time-series features derived from the MIMIC-IV dataset directly. The baseline transformer uses the architecture and training recipe described previously, and uses the encoding methods that most closely match with those used in [4], namely Quantile Bins numeric encoding and Round to Integer time encoding.

The XGBoost model uses a single feature vector per patient visit that encodes the tabular demographic data and time-series data from the chart events, lab events, and procedure events tables in MIMIC-IV using the same methodology as Soenksen *et al.* [6], [77]. The demographic features listed in Table 3.5 are encoded to integers, with age simply as the age of the patient at prediction time in years, and the other categorical features being assigned an integer representing the category. 11 time-series features are generated per item, namely the minimum, maximum, mean, variance, mean difference, mean absolute difference, maximum absolute difference, total absolute difference, difference between the last and first elements, number of peaks, and linear trend. These features were computed

on just the sequence of values and the actual time difference between values was not taken into account for features such as the linear trend. A total of 611 features are generated, which are concatenated into a single feature vector that is input to the XGBoost model. It is possible features are missing, in which case they are filled with a default value of 0. The labels are generated identically to the HAIM FHIR dataset which are based on [6].

I average the performance over 5 trials with a different seed each, with a single trial consisting of the following. A train and test dataset are generated by splitting the dataset based on the patient identifier using the seed. The training set is used to perform a grid search over a small set of hyperparameters listed in Table 5.1 with 5-fold cross validation, which is optimized on the validation AUROC. The positive class weight is used for the training loss. Following the hyperparameter search, the best model is evaluated on the held-out test dataset to generate result metrics. The metrics are averaged over the 5 seeds.

Table 5.1: XGBoost hyperparameters

Parameter	Values
Max depth	[5, 6, 7, 8]
Number of estimators	[200, 300]
Learning rate	[0.1, 0.3, 0.5]

5.3.2 Numeric and Time Encoding

I split the experiments into three comparisons, all operate on the length-of-stay prediction (LOS) and mortality prediction (MP) tasks on the HAIM dataset. The first set of tests examines whether varying the numeric encoding method improves performance. I test the discrete bin encoding method and the continuous Gaussian basis, MLP, SSP, and SSP similarity basis methods. These methods are tested on both the quantile numeric base and normalized numeric base as inputs to the encoding method. The configurations tested are listed in Table 5.2.

The second set of tests examines whether varying the time encoding method improves performance. I test the discrete Round to Integer, Logspace Bins, and Quantile Bins encoding methods, and the continuous Gaussian basis, MLP, SSP, and SSP similarity basis methods. All these methods operate on the log timedelta in hours. The configurations tested are listed in Table 5.3.

Table 5.2: Numeric encoding experimental setup. Exp. = Experimental condition.

	Numeric base	Numeric method	Time method
Control		Bin	
Exp.		Gaussian basis	
Exp.	Quantile	MLP	Round to integer
Exp.		SSP	
Exp.		SSP similarity basis	
Control		Bin	
Exp.		Gaussian basis	
Exp.	Normalized	MLP	Round to integer
Exp.		SSP	
Exp.		SSP similarity basis	

Lastly, I look at using the same continuous encoding methods for both numeric and time encodings, which I refer to as "numtime" encoding. The numtime methods are compared to the control that uses discrete methods which match the encoding methods used in [4], namely using numeric Bin encoding and Round to integer. However, I use the normalized numeric base because this is the most sensible base for doing continuous encoding methods. The configurations tested are listed in Table 5.4.

5.3.3 Compression Methods on Extended Context Dataset

For the compression method experiments, I test the sum, LMU, and cross attention compression methods against the non-compression control. The compression methods operate as a layer after the embedding and continuous encoding layers. All compressed sequences also have the attention mask created to match the compressed sequences. I use the Bin numeric encoding method with quantile base and the Round to integer time encoding method for all models.

I test all these models on the datasets with 1, 2, and 3 encounters, except for non-compression on the 3 encounter dataset because the model runs out of memory. The configurations tested are listed in Table 5.5.

Table 5.3: Time encoding experimental setup. Exp. = Experimental condition, Q = quantile numeric base.

	Numeric method	Time method
Control	Bin Q	Round to integer
Exp.		Logspace bins
Exp.		Quantile bins
Exp.		MLP
Exp.		SSP
Exp.		SSP similarity basis

Table 5.4: Combined continuous numeric and time encoding (numtime) experimental setup. Exp. = Experimental condition.

	Numeric method (Normalized base)	Time method
Control	Bin	Round to integer
Exp.	Gaussian basis	Gaussian basis
Exp.	MLP	MLP
Exp.	SSP	SSP
Exp.	SSP similarity basis	SSP similarity basis

Table 5.5: Compression methods experimental setup. Exp. = Experimental condition.

	Number of encounters	Compression method
Control	1	No compression
Exp.		Sum
Exp.		LMU
Exp.		Cross attention
Control	2	No compression
Exp.		Sum
Exp.		LMU
Exp.		Cross attention
Exp.	3	Sum
Exp.		LMU
Exp.		Cross attention

Chapter 6

Results

All the box plots in this chapter show the mean using a green cross, while the median is indicated with an orange horizontal line.

6.1 Baseline Transformer on HAIM Dataset

Comparing the baseline transformer model to the XGBoost model used in [6], I find the performance on the different tasks is mixed. Table 6.1 shows the mean and standard deviation values of the test AUROC and Figure 6.1 shows the box plots of all metrics comparing the baseline transformer and the XGBoost model.

For the LOS task, I find that the transformer has a significantly greater test AUROC compared to the XGBoost model (t-test, $p < 0.001$). The transformer also has a significantly greater accuracy ($p = 0.0026$), unweighted F1-score ($p < 0.001$), and average precision score ($p < 0.001$). However, for the MP task, the XGBoost model has significantly greater test AUROC compared to the transformer (t-test, $p < 0.001$). The XGBoost model also has a significantly greater average precision score ($p < 0.001$).

6.2 Continuous Numeric Encoding

Comparing the baseline transformer model that uses Round to Integer discrete time encoding and Quantile Bins discrete numeric encoding, I find that none of the experimental methods are significantly able to out-perform the baseline. The F-test shows that there

Table 6.1: Test AUROC results for baseline transformer and XGBoost on HAIM dataset. Best scores are bolded.

Task	Method	AUROC Mean	AUROC Std
LOS	Transformer	**0.819**	0.0022
	XGBoost	0.780	0.0046
MP	Transformer	0.805	0.0107
	XGBoost	**0.855**	0.0059

are no means that are significantly different than one another for either task. There is no evidence that using Normalized or Quantile base for the encoding is better than the other.

The means and standard deviations for the test AUROC scores for each time encoding method tested are presented in Table 6.2 for the LOS task and Table 6.3 for the MP task. The box plots for all the test metrics are presented in Figure 6.2 for the LOS task and Figure 6.2 for the MP task.

Table 6.2: Test AUROC for continuous numeric encoding for transformer for the LOS task on the HAIM dataset. Best scores are bolded.

Numeric base	Method	AUROC mean	AUROC std
Normalized	Bins	**0.8203**	0.00339
	Gaussian basis	0.8156	0.00232
	MLP	0.8135	0.00158
	SSP	0.8148	0.00569
	SSP similarity basis	0.8160	0.00389
Quantile	Bins	**0.8193**	0.00206
	Gaussian basis	0.8162	0.00394
	MLP	0.8156	0.00080
	SSP	0.8162	0.00482
	SSP similarity basis	0.8168	0.00353

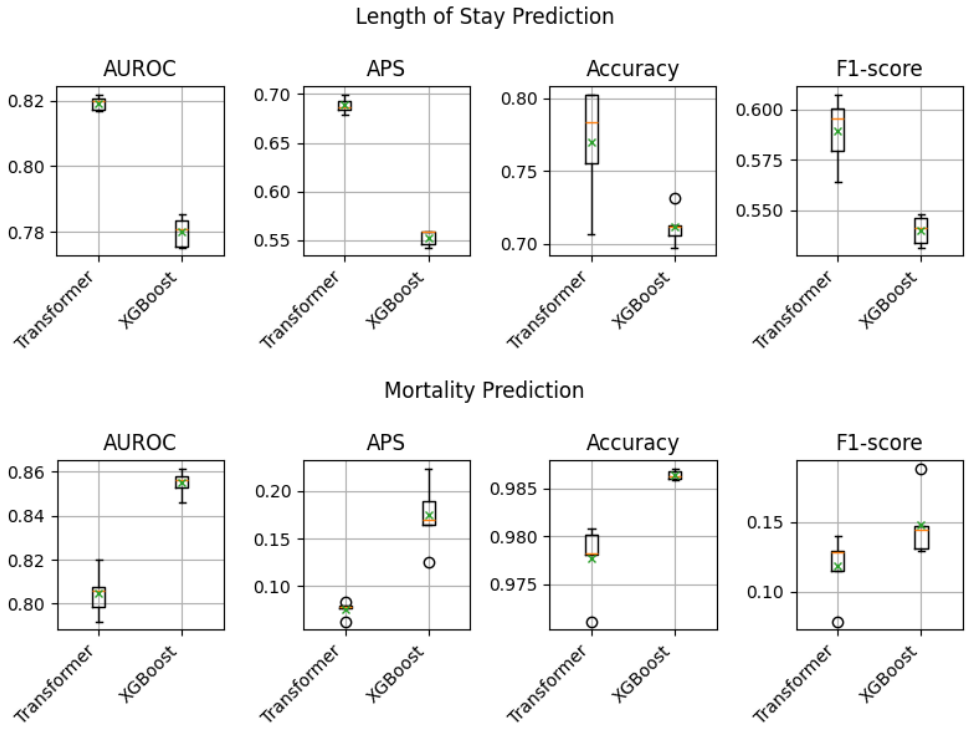


Figure 6.1: Test metrics box plots for the baseline transformer and XGBoost models on the HAIM dataset.

6.3 Continuous Time Encoding

I find that none of the experimental time encoding methods are significantly able to outperform the baseline method of Round to Integer.

On the LOS task, the F-test for AUROC shows that there are significant differences between means of all the methods tested ($p < 0.001$). Dunnett's test comparing the mean of each experimental method to the Round to integer control shows the means of the Gaussian basis ($p = 0.001$) and SSP similarity basis ($p = 0.002$) methods are significantly different from the control.

There are also significant differences within the mean F-1 scores (F-test, $p = 0.018$), with Dunnett's test showing the means of the Gaussian basis ($p = 0.031$) and SSP similarity basis ($p = 0.040$) methods are significantly different than the Round to integer control. There are no other significant results for the other metrics in the LOS task. Furthermore,

Table 6.3: Test AUROC for continuous numeric encoding for transformer for the MP task on the HAIM dataset. Best scores are bolded.

Numeric base	Method	AUROC mean	AUROC std
Normalized	Bins	0.8041	0.01329
	Gaussian basis	0.7927	0.00706
	MLP	0.7956	0.02103
	SSP	**0.8071**	0.01338
	SSP similarity basis	0.8059	0.01972
Quantile	Bins	0.8048	0.01070
	Gaussian basis	0.8043	0.01569
	MLP	0.8023	0.01298
	SSP	**0.8061**	0.01202
	SSP similarity basis	0.7964	0.00917

there are no significant statistical results for any metrics for the MP task. The means and standard deviations for the test AUROC scores for each time encoding method tested are presented in Table 6.4 for the LOS task and Table 6.5 for the MP task. The box plots for all the test metrics are presented in Figure 6.4 for the LOS task and Figure 6.5 for the MP task.

Table 6.4: Test AUROC for continuous time encoding for transformer for the LOS task on the HAIM dataset. Best scores are bolded.

	Method	AUROC mean	AUROC std
Discrete	Round to integer	**0.8180**	0.00221
	Logspace bins	0.8176	0.00533
	Percentile bins	0.8134	0.00349
Continuous	Gaussian basis	0.8068	0.00342
	MLP	0.8137	0.00410
	SSP	**0.8170**	0.00181
	SSP similarity basis	0.8063	0.00497

Table 6.5: Test AUROC for continuous time encoding for transformer for the MP task on the HAIM dataset. Best scores are bolded.

	Method	AUROC mean	AUROC std
Discrete	Round to integer	0.7784	0.02569
	Logspace bins	0.7995	0.01062
	Percentile bins	**0.8083**	0.01930
Continuous	Gaussian basis	0.7760	0.02171
	MLP	0.7895	0.00347
	SSP	**0.8033**	0.00719
	SSP similarity basis	0.7807	0.02370

6.4 Combined Continuous Encoding

I find that when using combined numeric and time encoding, none of the configurations significantly outperform the control, though some methods are significantly worse than the control.

On the LOS task, the F-test for AUROC shows there are significant differences between the means of all the combined numeric encoding methods and the control ($p < 0.001$) and Dunett's test shows the means for Gaussian basis ($p < 0.001$), MLP encoding ($p = 0.006$), and SSP similarity basis ($p < 0.001$) are all significantly different than the control. I also find that the SSP similarity basis encoding method does significantly worse than the control for the F1-score (Dunnett's test, $p = 0.029$) and APS (Dunnett's test, $p = 0.003$) metrics.

On the MP task, the F-test for AUROC shows there are significant differences between the means of all the combined numeric encoding methods and the control ($p = 0.022$) and Dunett's test shows the mean for Gaussian basis is significantly different than the control ($p = 0.03$).

The means and standard deviations for the test AUROC scores for the combined numeric and time encoding methods are presented in Table 6.6 for both tasks. The box plots for all the test metrics are presented in Figure 6.6 for the LOS task and in Figure 6.7 for the MP task. The combined continuous encodings are plotted along with the corresponding continuous numeric-only and time-only encodings for comparison. I find that there is no clear pattern that using combined encodings is always better than using only continuous numeric or only continuous time.

Table 6.6: Test AUROC for combined continuous numeric and time encoding for transformer for the LOS and MP tasks on the HAIM dataset. Best scores are bolded.

Task	Method	AUROC mean	AUROC std
LOS	Control	**0.8180**	0.00221
	Gaussian basis	0.8075	0.00329
	MLP	0.8116	0.00505
	SSP	0.8177	0.00397
	SSP similarity basis	0.8005	0.00242
MP	Control	0.7784	0.02569
	Gaussian basis	0.7721	0.01243
	MLP	0.7875	0.02037
	SSP	**0.8055**	0.02031
	SSP similarity basis	0.7787	0.01943

6.5 Compression Layers

For both LOS and MP tasks on the extended context dataset, I find that certain compression methods significantly improve performance compared to the non-compression controls.

First, I compare the different compression methods against the non-compression control for different number of encounters on the LOS task. Generally, the mean AUROC of all compression methods is higher that the corresponding control model for the same number of encounters, and Dunnett's test shows this difference in means is significant for 1 encounter sum compression ($p = 0.021$), 2 encounter sum compression ($p = 0.003$), and 2 encounter LMU compression ($p = 0.034$). While there is no 3 encounter non-compression control, the mean metric scores for the LOS task tend to be similar or better than the their corresponding 2 encounter models.

Dunnett's test shows significant improvements over the control for the APS metric using sum compression in 1 encounter ($p = 0.031$), and in F1-score using all compression methods in 2 encounters ($p < 0.001$ for all methods). However, I also find the accuracy metric of several compression models is significantly worse than the control, for LMU 1 encounter ($p = 0.006$), and for all methods with 2 encounters ($p < 0.01$ for all methods).

For the MP task, using compression is better compared to the non-compression controls for the same number of encounters. The mean AUROC of the compression methods is generally higher than the control for both 1 and 2 encounters. This difference is significant

Table 6.7: Test AUROC for compression methods for the LOS task on the extended context dataset. Best scores are bolded.

Number of encounters	Method	AUROC mean	AUROC std
1	No compression	0.8159	0.00545
	Sum	**0.8257**	0.00234
	LMU	0.8220	0.00558
	Cross attention	0.8200	0.00625
2	No compression	0.8166	0.00446
	Sum	**0.8321**	0.00725
	LMU	0.8274	0.00595
	Cross attention	0.8239	0.00646
3	No compression	–	–
	Sum	**0.8295**	0.00674
	LMU	0.8284	0.00890
	Cross attention	0.8266	0.00670

for 1 encounter LMU ($p = 0.009$), 2 encounter LMU ($p = 0.048$), and 2 encounter sum ($p = 0.042$).

There are significant improvements over the control for the APS metric using LMU compression for 1 encounter ($p = 0.005$), and in F1-score for all compression methods in 2 encounters ($p < 0.001$ for all methods). However, the accuracy metric of LMU with 2 encounters is significantly lower than the control ($p = 0.005$).

Table 6.8: Test AUROC for compression methods for the MP task on the extended context dataset. Best scores are bolded.

Number encounters	Method	AUROC mean	AUROC std
1	No compression	0.9057	0.01177
	Sum	0.9147	0.01739
	LMU	**0.9326**	0.00525
	Cross attention	0.9189	0.01189
2	No compression	0.8712	0.02045
	Sum	**0.9014**	0.01358
	LMU	0.9006	0.01406
	Cross attention	0.8767	0.02157
3	No compression	–	–
	Sum	**0.9012**	0.00888
	LMU	0.8983	0.01236
	Cross attention	0.8870	0.01483

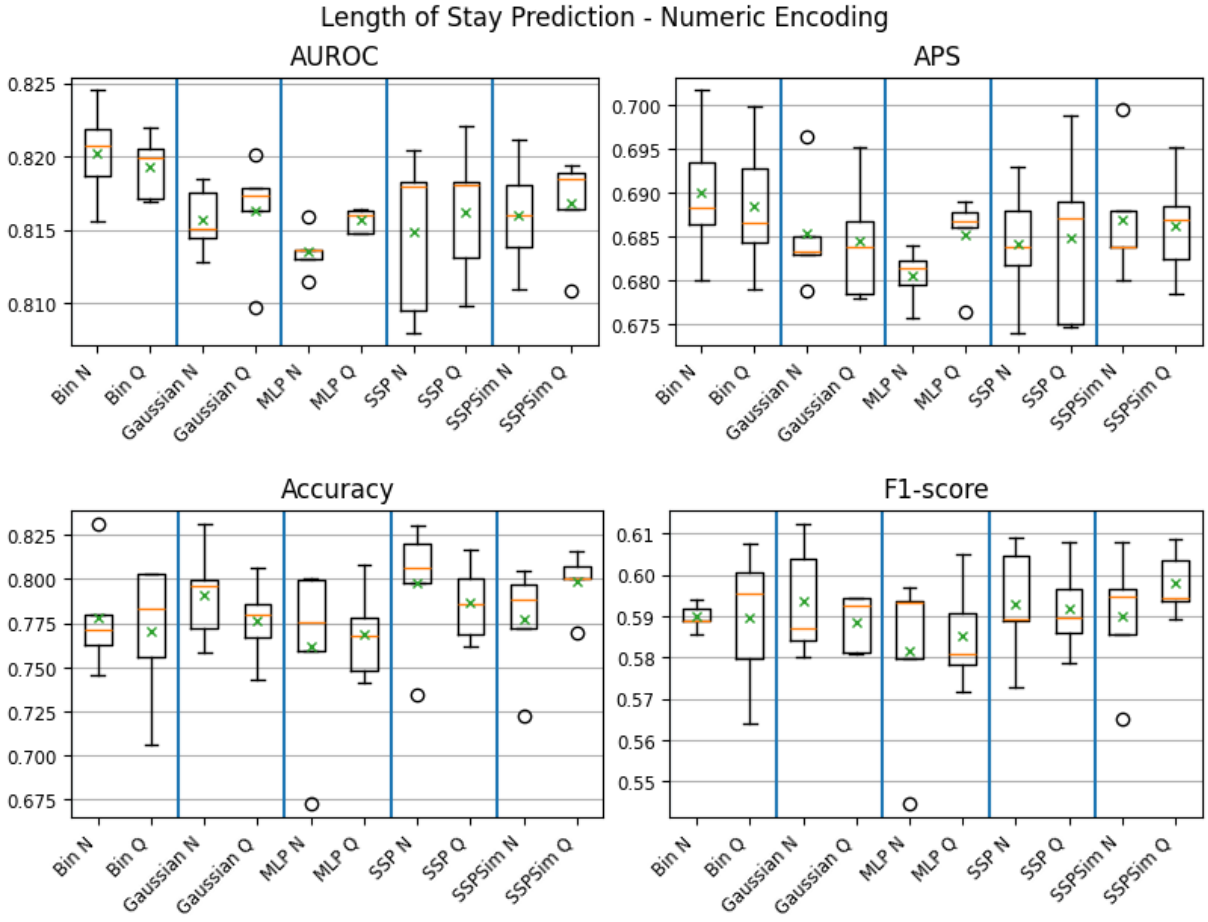


Figure 6.2: Test metrics box plots for numeric encoding transformer experiments on the HAIM dataset on the LOS task. N indicates the encoded value is the normalized measurement, while Q indicates the encoded value is the estimated quantile for that measurement. Only the Bin methods are discrete encodings, while the rest are continuous encodings.

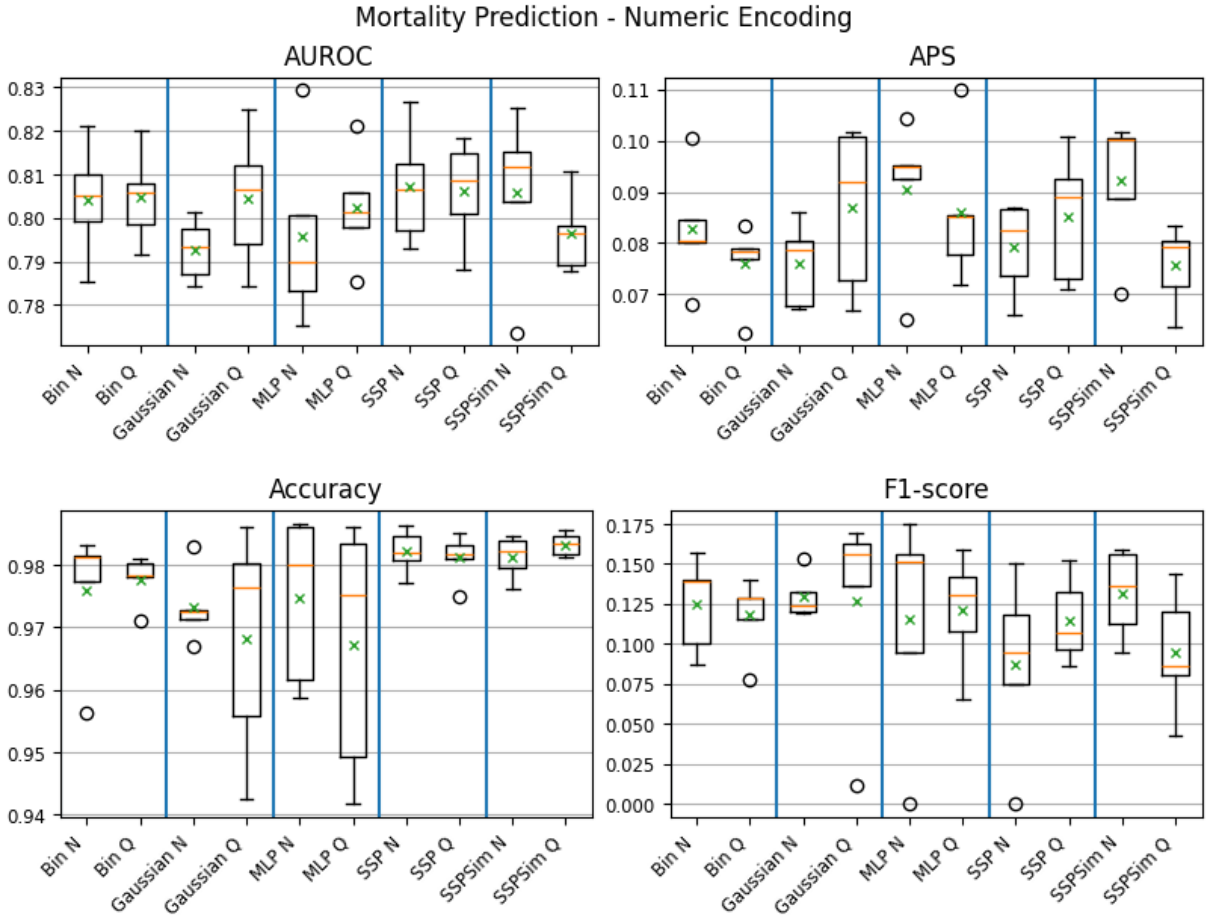


Figure 6.3: Test metrics box plots for numeric encoding transformer experiments on the HAIM dataset on the MP task. N indicates the encoded value is the normalized measurement, while Q indicates the encoded value is the estimated quantile for that measurement. Only the Bin methods are discrete encodings, while the rest are continuous encodings.

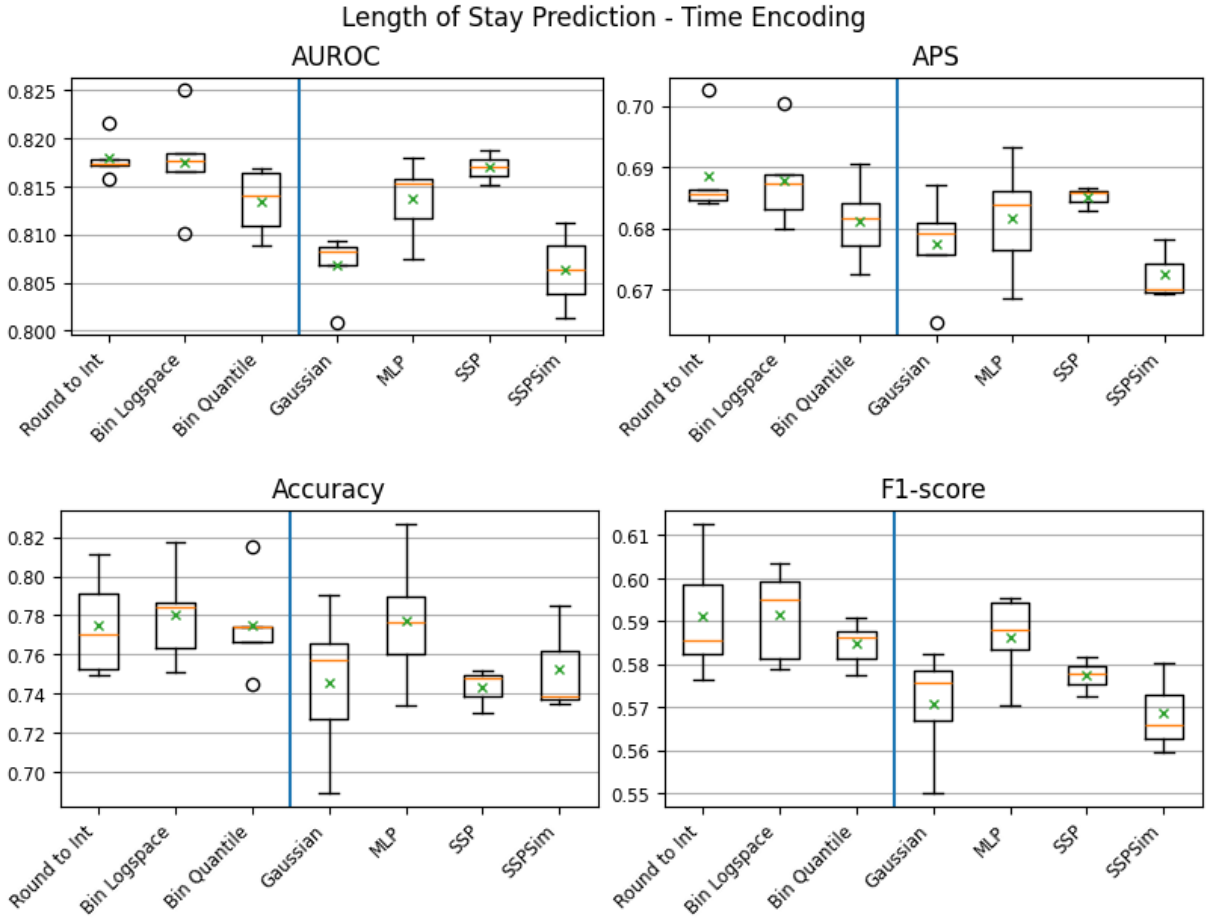


Figure 6.4: Test metrics box plots for time encoding transformer experiments on the HAIM dataset on the LOS task. All methods operate on the log timedelta in hours. Methods to the left of the blue divider are discrete encodings, while those to the right are continuous encodings.

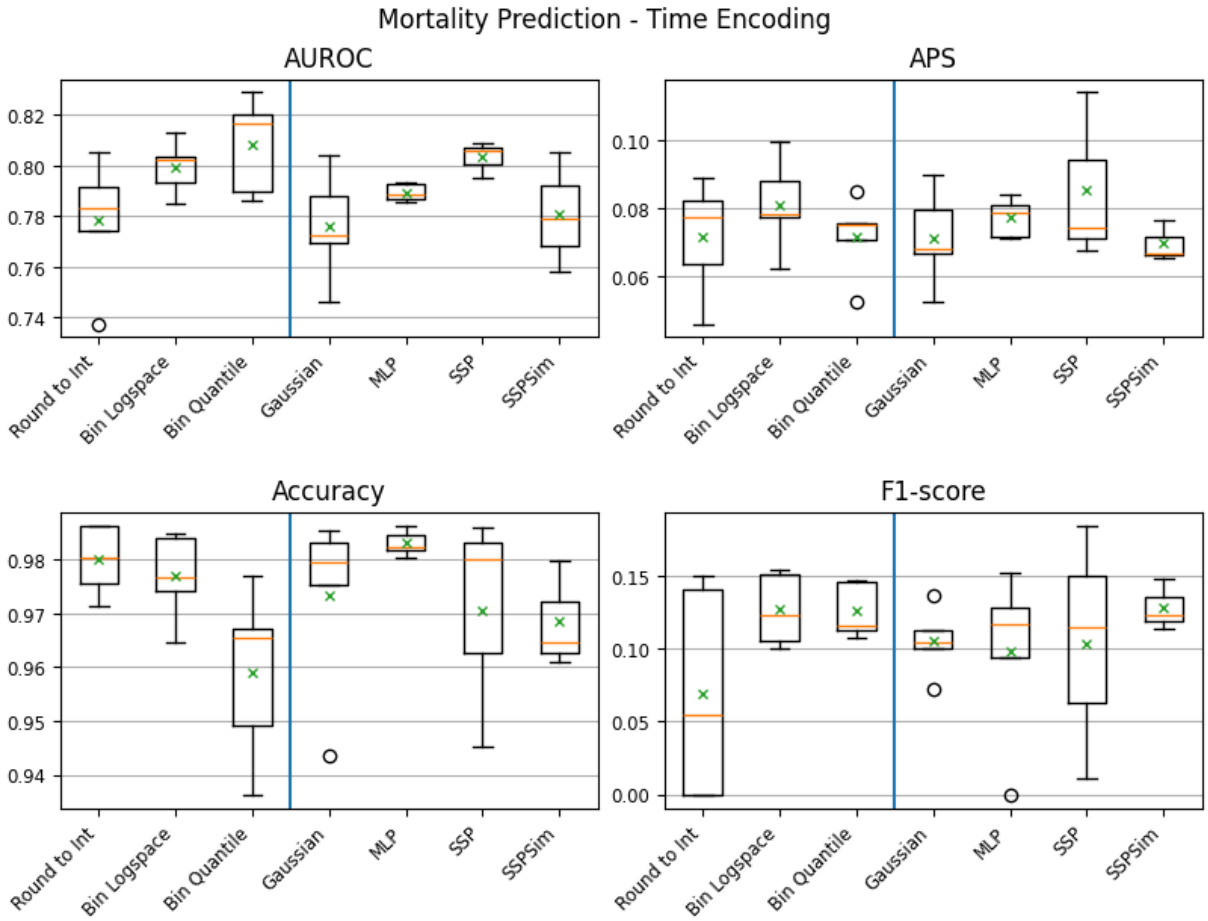


Figure 6.5: Test metrics box plots for time encoding transformer experiments on the HAIM dataset on the MP task. All methods operate on the log timedelta in hours. Methods to the left of the blue divider are discrete encodings, while those to the right are continuous encodings.

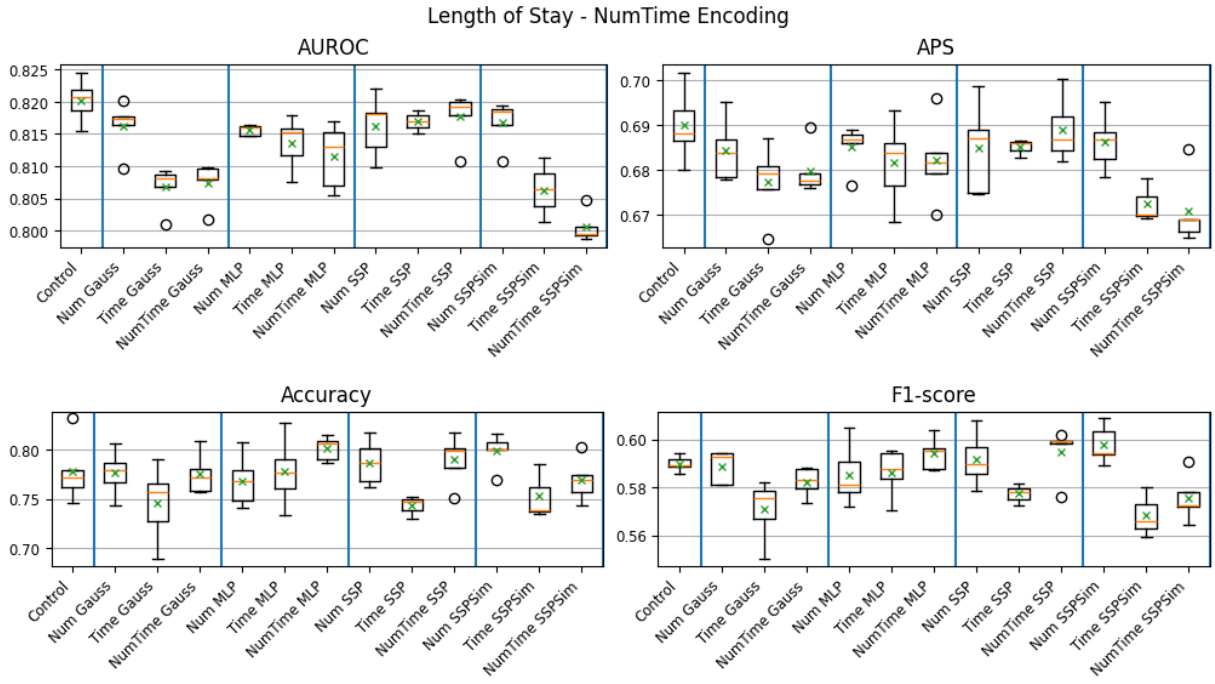


Figure 6.6: Test metrics box plots for all encoding methods on the HAIM dataset on the LOS task. Num refers to continuous numeric encoding with normalized values. Time refers to continuous time encoding with the log timedelta in hours. NumTime refers to the combined encoding where both numeric values and time values are encoded with the same method. Control uses Round to integer and Bins on normalized values.

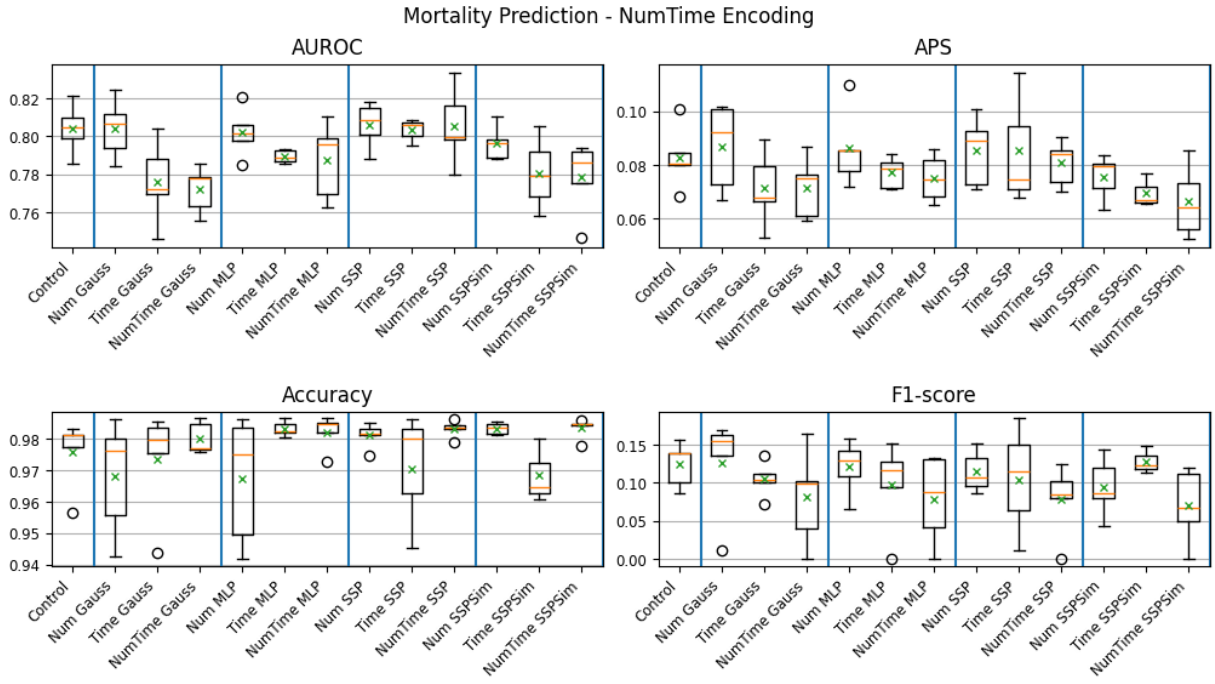


Figure 6.7: Test metrics box plots for all encoding methods on the HAIM dataset on the MP task. Num refers to continuous numeric encoding with normalized values. Time refers to continuous time encoding with the log timedelta in hours. NumTime refers to the combined encoding where both numeric values and time values are encoded with the same method. Control uses Round to integer and Bins on normalized values.

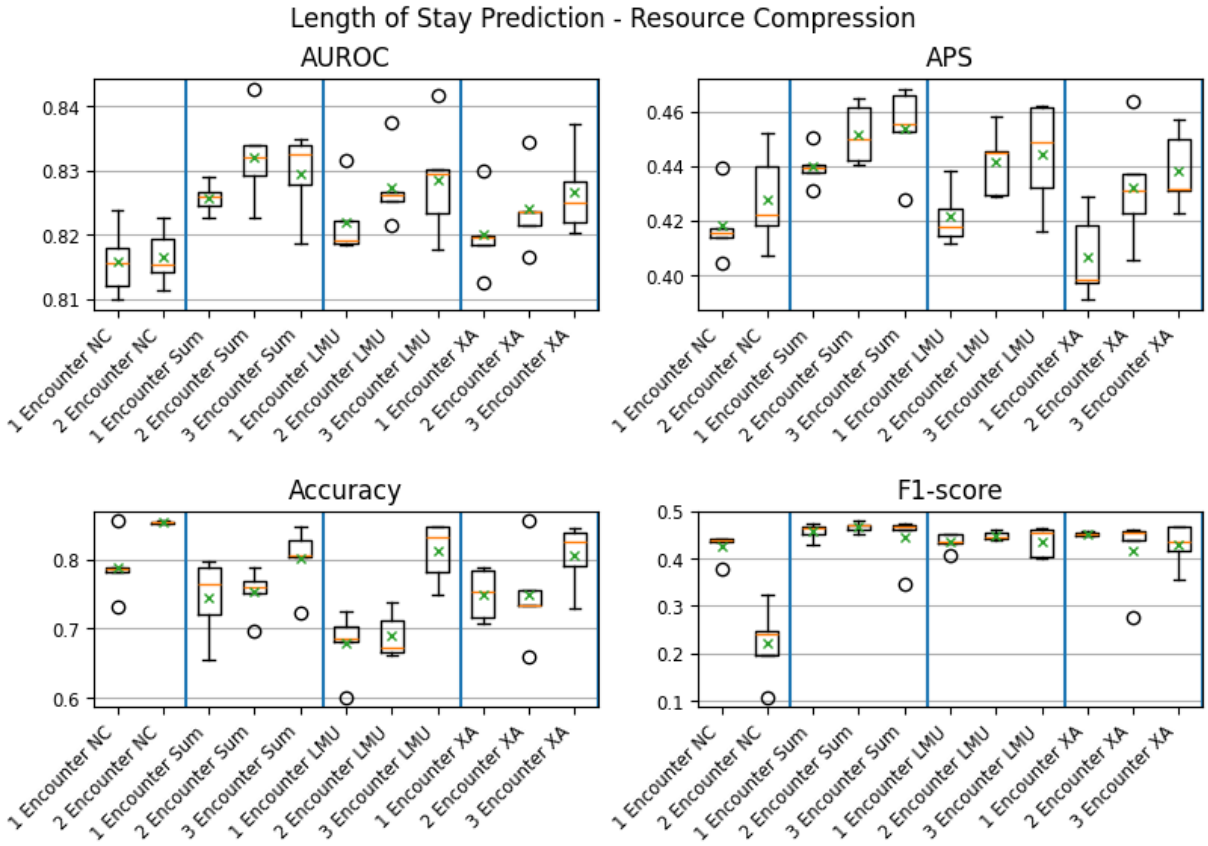


Figure 6.8: Test metrics box plots for all compression methods on the extended context dataset for the LOS task. NC = no compression, LMU = Legendre memory unit, XA = cross attention.

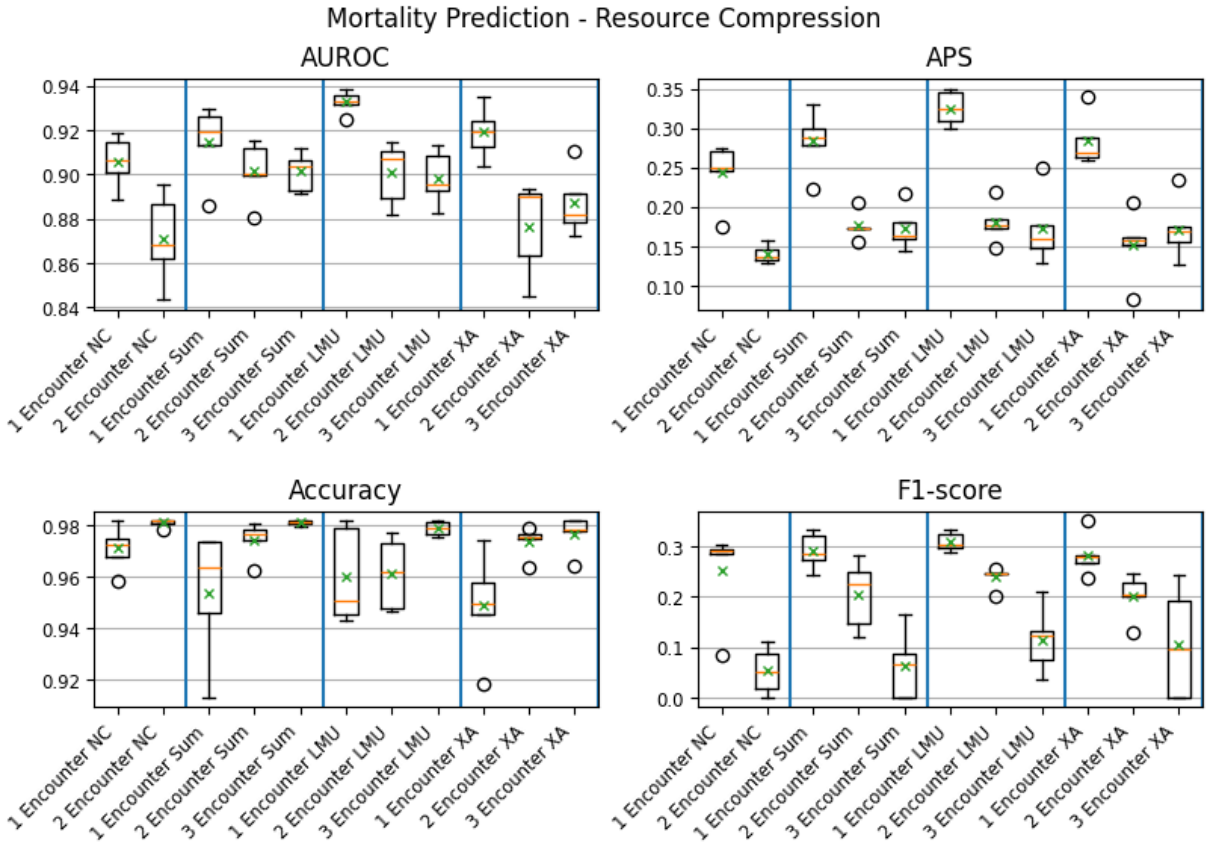


Figure 6.9: Test metrics box plots for all compression methods on the extended context dataset for the MP task. NC = no compression, LMU = Legendre memory unit, XA = cross attention.

Chapter 7

Discussion

7.1 Interpretation of Findings

It is inconclusive to determine whether the transformer performs better than the XGBoost model based on the results of the LOS and MP tasks on the HAIM dataset. While it is promising that the transformer is capable at performing significantly better than the XGBoost model at the LOS task, the transformer performs significantly worse at the MP task. In a production setting, it would be preferable to deploy a lower-cost model like XGBoost, which does not require expensive compute hardware like GPUs to run inference, if the performance of the XGBoost model is comparable or better. Further investigation should be done on additional tasks, such as disease prediction or unexpected readmission prediction, to compare the sequence modelling paradigm of the transformer to the time-series features with XGBoost.

I did not find evidence that the use of continuous encoding methods results in significantly improved performance compared to the baseline discrete encoding methods for numeric and time data. It is possible that the tasks I tested do not require high-precision representations of the numeric and time data, and the presence of a high or low value at a recent time in the record is informative enough to achieve adequate performance. For example, an abnormally high white blood cell count might indicate the onset of an infection that might prolong a hospital stay. However, there could exist other tasks where a more precise representation of numeric and time values are important, such as forecasting a patient's vitals. The use of continuous encoding could also be important if encoded FHIR data is used as inputs to conversational language models. Effectively being able

to decode information from a representation of a patient record may require more precise representations of numeric measurements than large bins.

Across both the LOS and MP tasks, I find that compression techniques improve performance. This suggests that encoding a single FHIR resource as a sequence of vectors is sparser than necessary, and using attention to compute relationships between vectors within the same resource may not be necessary. At its simplest, a resource will almost always have a timestamp and a code, which represent very distinct pieces of information. Observation resources may further contain numeric measurements, additional codes such as categories or specimen types. Through the Sum compression operation, these disparate pieces of information are combined into a single representation with no information about the order of the vectors.

From a VSA perspective using the Sum compression operator, each of these components represent a concept for that part of a resource. These concept vectors are bundled together, forming a vector that represents a union of all components. As an example, an Observation resource could be thought of as the union on concept vectors for "`two_hours_ago` + `glucose` + `80th_percentile_measurement` + `blood_sample` + `lab_measurement`". In the VSA paradigm, a bundle of concepts will have a high vector similarity to all component concepts [78]. HRR uses cosine similarity [79] which contains a dot product. It is possible for linear layers in a transformer to learn representations that are similar to components in the concept bundle and have high output scores, essentially detecting the presence of certain vectors in the bundle.

Assuming task performance is the same, when memory and runtime are considered for the various compression methods, Sum compression would be the most ideal. The runtime execution and memory usage of Sum compression are smaller than the LMU and Cross Attention compression methods. This is because the `torch.scatter_reduce` operation used to implement the Sum compression method is highly optimized for GPUs if the correct indexing data structure is created. In contrast, the LMU and Cross Attention operations must be computed per subsequence in a loop, which is much slower, and these methods use additional memory for activations for the matrix operations they use. Without significantly improved task performance, the simpler Sum method is recommended for further use in developing other prototypes that use compression.

7.2 Benefits of Methodology

Processing a patient's medical history in FHIR lends itself well to a sequence modelling approach, since the data is already structured as timestamped units of information. Com-

pared to classical ML feature engineering approaches, it is much easier to include additional data and scale this approach to massive datasets and incorporate more types of EHR data. The HAIM dataset selection of relevant events includes only 31% of the chart events, 63% of the laboratory events, and 9% of the procedure events available in the dataset for those patients. Each new event adds 11 new features under the HAIM XGBoost data processing framework, and there are a total of 1040 unused event types between those three tables. Additionally, many of these events are relevant for only a few patients, making the feature vector very sparse. While feature selection and engineering may work for smaller datasets, it becomes infeasible to scale to entire patient records where there are thousands of possible relevant events that might exist and unique processing methods need to be defined for each new modality. For example, the HAIM framework does not incorporate medications, past medical diagnoses, and surgical procedures, but these data may be extremely useful for different prediction tasks. Feature encoding methods for these medical data would need to be engineered into the system to use them.

In contrast, the simple datatype encodings of the FHIR tokenizer can scale to easily incorporating large amounts of new data from various resource formats because it recognizes that the data are structured from simple components and uses a complex model to learn relationships between these features. The tokenization framework assigns a random vector to each unit of data and the model learns sequential relationships between these vectors. This is essentially the approach that modern language modelling uses and has had immense success. Language modelling has changed from generating statistical features over the frequency of words and their grammatical roles into representing a sentence as a sequence of random vectors for a transformer neural network.

This work aims to promote the view that medical records should be viewed as a sequence of vectors representing a larger variety of medical data, rather than a collection of time-series features computed over select medical signals. Future work should be done to apply this framework on more complex tasks where classical machine learning methods run against dataset size limitations or where mechanical understanding of the problem is limited such that complex learned features may be more useful.

Our tokenizer is built to be compatible with the FHIR data standard. Particularly, the input to the tokenizer is a FHIR Bundle of resources associated with an individual patient record, which can be obtained from the `$everything` operation. Our training dataset was constructed from the outputs of the `$export` operation on a FHIR server containing the MIMIC-IV FHIR resources. Provided the medical database is in FHIR format, our tokenization method would apply to the dataset.

7.3 Limitations of Methodology

The translation of existing EHR databases into FHIR data is not trivial, especially the standardization of structure and terminology of arbitrary medical data. But there is motivation for medical data systems to use FHIR, and many existing products already use FHIR data formats. The ONC Cures Act Final Rule calls on the US healthcare industry to adopt a standard API that allows patients and providers to securely access and share healthcare data, and the framework for such interoperability uses the US CORE FHIR standard [99]. As the adoption of FHIR data standards increases, the use of our methodology becomes more relevant when developing medical AI models.

Our encoding method still requires some information that is specific to the database to function. First, a mapping between codes and token IDs is required for the tokenizer. However, FHIR servers already have CodeSystem files which provide the coding system names and set of codes available. These can be easily converted to the desired token ID mapping, which was done when creating our MIMIC-IV FHIR tokenizer. There are several standard coding systems in use, such as ICD, OMOP, RxNorm, and others, reducing the need to define custom maps for each new dataset. Additionally, if a pre-trained model were to be created, the vector embeddings for these common coding systems could be shared between models. The other required component is estimations of the statistical distributions of numeric measurements, either through the median and median absolute deviation for the normalized numeric base, or an empirical CDF for the quantile numeric base. Both of these can be estimated from a sample of records, though this is best accomplished with underlying access to the database tables, rather than through the FHIR API. These statistics may also be able to be shared if collected once at a representative population level. The tokenizer configuration also requires the list of fields in a resource that are relevant to tokenize. This is a form of data selection and still requires some exploratory data analysis to determine the optimal configuration. For example, some fields might always be present, such as an item status marked as "completed", so it would be redundant to include that field.

The MIMIC-IV dataset itself has some limitations. Firstly, MIMIC-IV uses custom codes for many of the data systems. The only standard coding system used is ICD-9 and ICD-10 for surgical procedures and diagnostic codes. This limits the interoperability of a model trained on this dataset, since embeddings developed for the custom coding systems will not be applicable to other coding systems. The MIMIC-IV dataset also anonymizes the exact dates by shifting them an arbitrary amount into the future. This shift is maintained the same for each patient record, such that the relative timedeltas between items is the same. However, this prevents us from testing methods to encode the absolute

date, since any results would not be representative of a real-world dataset. The absolute date could capture patterns in wide-spread health events, such as global pandemics, or seasonal patterns, such as influenza season.

Due to the inclusion of every possible token defined in the coding system, there are tokens in the vocabulary that will be unused or very rarely used. This completeness helps limit the number of out-of-vocabulary tokens, but infrequently seen tokens during training may not learn good representations during training compared to frequently seen tokens. The vocabulary size of the model is therefore very large and potentially inefficient due to unused or rare tokens. Further work could be done into investigating methods that incorporate medical semantic knowledge into the creation of the embedding vector representations, as in [80], which could improve rare token representations.

The tokenization method also does not explicitly include information about what field it represents. For example, the second code in an Observation might always be a numeric value, but it could have come from a `valueQuantity`, `valueRatio`, or `valueRange` field. It is possible that adding a field name embedding or using a graph neural network approach for encoding could resolve this issue. However, based on the compression results, it seems that aggregating all the vectors in a single resource subsequence improves performance, so this may not be a concern. The presence of the item code token might also resolve these ambiguities.

There are still resource types that our model cannot process effectively, such as images. While our tokenizer would be able to skip fields which would contain that data, the FHIR server `$everything` operation would still pull these data from the server, which could result in unnecessarily large payload sizes for the FHIR Bundle. This may make the `$everything` operation less desirable to use during inference, as the request times to retrieve large amounts of unnecessary data may be extremely long in a production setting.

7.4 Directions for Future Research

In addition to the suggestions for further research mentioned in the previous sections, I outline three directions for future research that I believe are most important: pre-training on a large EHR dataset, validating on further classification tasks, and using FHIR encodings with text models.

Due to the dataset size, it was not possible to effectively test whether pre-training a foundation model with language modelling tasks would result in improved performance. If a larger FHIR dataset that uses many standard coding systems can be obtained, a FHIR

foundational model could be developed using a language modelling pre-training approach. This would help validate the deep sequence modelling approach to EHR data. Further discussion of pre-training can be found in Appendix A.

There are other classification tasks that our method could be tested on to further validate its performance. Some examples include disease prediction tasks or unexpected readmission prediction. Particularly, it would be desirable to test performance on challenging tasks that do not already have successful classical ML approaches, as these may represent areas for deep learning models to improve upon current methods.

Beyond performing additional predictive tasks, transformer encodings could also be used to build systems that enable conversational access with FHIR resources. Firstly, transformer encoders can be used to create vector embeddings that represent documents. These vectors can be searched against an embedded query by selecting documents with high vector similarity. Document embedding is an important part of the retrieval augmented generation pipeline. While it is possible to use a text encoder to create document embeddings for FHIR resources, future work should be done to compare our encoding method with text encoders for retrieval tasks. FHIR embeddings could also be used as inputs to generative text models as a form of multimodal modelling with text and encoded EHR data. Future investigations could look at methods that use a combination of FHIR embeddings and natural language as inputs for tasks such as summarizing a patient's EHR history.

7.5 Conclusion

In this thesis, I present methodology to transform FHIR JSON data into a sequence of tokens, leveraging the compositional structure and datatypes in the FHIR standard to create meaningful units of medical data. The tokenization method is designed to be compatible with the existing `$everything` FHIR query, aligning with goals of FHIR interoperability. I train transformer models on the sequence classification tasks for length-of-stay prediction and mortality prediction and demonstrate it is possible for this model to out-perform an XGBoost model that uses bespoke data processing techniques on the open-source MIMIC-IV FHIR dataset. However, performance is task dependent. I find that the continuous encoding methods proposed did not improve performance, though they may have more benefits for other tasks when embeddings are used for conversational tasks with language decoders. The compression methods result in an improvement in accuracy and inference speed on long sequence lengths and would be beneficial for processing large medical histories. Lastly, pre-training transformer encoder models on the masked language modelling

task did not result in improvements on downstream task performance, but I detail the pre-training methodology so that future studies will be able to attempt training on a larger dataset and work towards creating foundation models for FHIR medical data.

References

[1] P. Rajpurkar, J. Irvin, K. Zhu, *et al.*, *CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning*, en, Nov. 2017. [Online]. Available: https://arxiv.org/abs/1711.05225v3 (visited on 10/04/2024).

[2] V. Jaiswal, A. Negi, and T. Pal, "A review on current advances in machine learning based diabetes prediction," *Primary Care Diabetes*, vol. 15, no. 3, pp. 435–443, Jun. 2021, ISSN: 1751-9918. DOI: 10.1016/j.pcd.2021.02.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S175199182100019X (visited on 03/04/2025).

[3] M. Moor, B. Rieck, M. Horn, C. R. Jutzeler, and K. Borgwardt, "Early Prediction of Sepsis in the ICU Using Machine Learning: A Systematic Review," English, *Frontiers in Medicine*, vol. 8, May 2021, Publisher: Frontiers, ISSN: 2296-858X. DOI: 10.3389/fmed.2021.607952. [Online]. Available: https://www.frontiersin.org/journals/medicine/articles/10.3389/fmed.2021.607952/full (visited on 03/04/2025).

[4] A. Rajkomar, E. Oren, K. Chen, *et al.*, "Scalable and accurate deep learning with electronic health records," en, *npj Digital Medicine*, vol. 1, no. 1, pp. 1–10, May 2018, Number: 1 Publisher: Nature Publishing Group, ISSN: 2398-6352. DOI: 10.1038/s41746-018-0029-1. [Online]. Available: https://www.nature.com/articles/s41746-018-0029-1 (visited on 06/22/2023).

[5] R. Bopche, L. T. Gustad, J. E. Afset, B. Ehrnström, J. K. Damås, and Ø. Nytrø, "In-hospital mortality, readmission, and prolonged length of stay risk prediction leveraging historical electronic patient records," *JAMIA Open*, vol. 7, no. 3, ooae074, Jul. 2024, ISSN: 2574-2531. DOI: 10.1093/jamiaopen/ooae074. [Online]. Available: https://doi.org/10.1093/jamiaopen/ooae074 (visited on 02/19/2025).

[6] L. R. Soenksen, Y. Ma, C. Zeng, *et al.*, "Integrated multimodal artificial intelligence framework for healthcare applications," en, *npj Digital Medicine*, vol. 5, no. 1, p. 149, Sep. 2022, ISSN: 2398-6352. DOI: 10.1038/s41746-022-00689-4. [Online]. Available: https://www.nature.com/articles/s41746-022-00689-4 (visited on 01/22/2025).

[7] H. Baek, M. Cho, S. Kim, H. Hwang, M. Song, and S. Yoo, "Analysis of length of hospital stay using electronic health records: A statistical and data mining approach," en, *PLOS ONE*, vol. 13, no. 4, e0195901, Apr. 2018, Publisher: Public Library of Science, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0195901. [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0195901 (visited on 03/04/2025).

[8] S. Hirji, S. McGurk, S. Kiehm, *et al.*, "Utility of 90-Day Mortality vs 30-Day Mortality as a Quality Metric for Transcatheter and Surgical Aortic Valve Replacement Outcomes," *JAMA Cardiology*, vol. 5, no. 2, pp. 156–165, Feb. 2020, ISSN: 2380-6583. DOI: 10.1001/jamacardio.2019.4657. [Online]. Available: https://doi.org/10.1001/jamacardio.2019.4657 (visited on 03/04/2025).

[9] A. Bottle, S. Middleton, C. J. Kalkman, E. H. Livingston, and P. Aylin, "Global Comparators Project: International Comparison of Hospital Outcomes Using Administrative Data," en, *Health Services Research*, vol. 48, no. 6pt1, pp. 2081–2100, 2013, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/1475-6773.12074, ISSN: 1475-6773. DOI: 10.1111/1475-6773.12074. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/1475-6773.12074 (visited on 03/04/2025).

[10] A. Krasowski, J. Krois, A. Kuhlmey, H. Meyer-Lueckel, and F. Schwendicke, "Predicting mortality in the very old: A machine learning analysis on claims data," en, *Scientific Reports*, vol. 12, no. 1, p. 17 464, Oct. 2022, Publisher: Nature Publishing Group, ISSN: 2045-2322. DOI: 10.1038/s41598-022-21373-3. [Online]. Available: https://www.nature.com/articles/s41598-022-21373-3 (visited on 03/04/2025).

[11] J. Amann, A. Blasimme, E. Vayena, D. Frey, V. I. Madai, and the Precise4Q consortium, "Explainability for artificial intelligence in healthcare: A multidisciplinary perspective," *BMC Medical Informatics and Decision Making*, vol. 20, no. 1, p. 310, Nov. 2020, ISSN: 1472-6947. DOI: 10.1186/s12911-020-01332-6. [Online]. Available: https://doi.org/10.1186/s12911-020-01332-6 (visited on 12/03/2023).

[12] R. Bommasani, D. A. Hudson, E. Adeli, *et al.*, *On the Opportunities and Risks of Foundation Models*, en, Aug. 2021. [Online]. Available: https://arxiv.org/abs/2108.07258v3 (visited on 03/12/2025).

[13] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," en, *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, Publisher: Nature Publishing Group, ISSN: 1476-4687. DOI: 10.1038/nature14539. [Online]. Available: https://www.nature.com/articles/nature14539 (visited on 03/12/2025).

[14] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," en, *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986, Publisher: Nature Publishing Group, ISSN: 1476-4687. DOI: 10.1038/323533a0. [Online]. Available: https://www.nature.com/articles/323533a0 (visited on 03/12/2025).

[15] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, *Attention Is All You Need*, arXiv:1706.03762 [cs], Dec. 2017. [Online]. Available: http://arxiv.org/abs/1706.03762 (visited on 06/22/2023).

[16] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, arXiv:1810.04805 [cs] version: 1, Oct. 2018. [Online]. Available: http://arxiv.org/abs/1810.04805 (visited on 06/22/2023).

[17] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving Language Understanding by Generative Pre-Training," en,

[18] H. International and Firely, "2024 State of FHIR Survey Results," Tech. Rep., May 2024. [Online]. Available: https://www.hl7.org/documentcenter/public/white-papers/2024%20StateofFHIRSurveyResults_final.pdf (visited on 10/04/2024).

[19] *Interoperability and Patient Access final rule*, May 2020. [Online]. Available: https://www.govinfo.gov/content/pkg/FR-2020-05-01/pdf/2020-05050.pdf (visited on 03/04/2025).

[20] J. Bai, S. Bai, Y. Chu, *et al.*, *Qwen Technical Report*, en, arXiv:2309.16609 [cs], Sep. 2023. [Online]. Available: http://arxiv.org/abs/2309.16609 (visited on 06/17/2024).

[21] L. Rasmy, Y. Xiang, Z. Xie, C. Tao, and D. Zhi, "Med-BERT: Pretrained contextualized embeddings on large-scale structured electronic health records for disease prediction," en, *npj Digital Medicine*, vol. 4, no. 1, pp. 1–13, May 2021, Number: 1 Publisher: Nature Publishing Group, ISSN: 2398-6352. DOI: 10.1038/s41746-021-00455-y. [Online]. Available: https://www.nature.com/articles/s41746-021-00455-y (visited on 06/22/2023).

[22] B. Komer, T. C. Stewart, A. R. Voelker, and C. Eliasmith, "A neural representation of continuous space using fractional binding," en,

[23] A. Voelker, I. Kajić, and C. Eliasmith, "Legendre Memory Units: Continuous-Time Representation in Recurrent Neural Networks," in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/952285b9b7e7a1be5aa7849f32ffff05-Abstract.html (visited on 11/23/2023).

[24] N. Chilkuri and C. Eliasmith, *Parallelizing Legendre Memory Unit Training*, arXiv:2102.11417 [cs], May 2021. [Online]. Available: http://arxiv.org/abs/2102.11417 (visited on 06/22/2023).

[25] N. Chilkuri, E. Hunsberger, A. Voelker, G. Malik, and C. Eliasmith, *Language Modeling using LMUs: 10x Better Data Efficiency or Improved Scaling Compared to Transformers*, arXiv:2110.02402 [cs], Oct. 2021. [Online]. Available: http://arxiv.org/abs/2110.02402 (visited on 10/19/2023).

[26] M. E. Peters, M. Neumann, M. Iyyer, *et al.*, *Deep contextualized word representations*, arXiv:1802.05365 [cs], Mar. 2018. DOI: 10.48550/arXiv.1802.05365. [Online]. Available: http://arxiv.org/abs/1802.05365 (visited on 11/23/2023).

[27] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735 (visited on 11/19/2024).

[28] Y. Kim, C. Denton, L. Hoang, and A. M. Rush, *Structured Attention Networks*, arXiv:1702.00887 [cs], Feb. 2017. DOI: 10.48550/arXiv.1702.00887. [Online]. Available: http://arxiv.org/abs/1702.00887 (visited on 03/03/2025).

[29] X. Wang, Z. Tu, L. Wang, and S. Shi, "Self-Attention with Structural Position Representations," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 1403–1409. DOI: 10.18653/v1/D19-1145. [Online]. Available: https://aclanthology.org/D19-1145 (visited on 06/22/2023).

[30] D. Bahdanau, K. Cho, and Y. Bengio, *Neural Machine Translation by Jointly Learning to Align and Translate*, arXiv:1409.0473 [cs], May 2016. DOI: 10.48550/arXiv.1409.0473. [Online]. Available: http://arxiv.org/abs/1409.0473 (visited on 03/03/2025).

[31] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in Vision: A Survey," *ACM Computing Surveys*, vol. 54, no. 10s, pp. 1–41, Jan. 2022, arXiv:2101.01169 [cs], ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3505244. [Online]. Available: http://arxiv.org/abs/2101.01169 (visited on 06/22/2023).

[32] C. Yang, M. B. Westover, and J. Sun, *BIOT: Cross-data Biosignal Learning in the Wild*, arXiv:2305.10351 [eess], May 2023. DOI: 10.48550/arXiv.2305.10351. [Online]. Available: http://arxiv.org/abs/2305.10351 (visited on 03/03/2025).

[33] A. Jaegle, S. Borgeaud, J.-B. Alayrac, *et al.*, *Perceiver IO: A General Architecture for Structured Inputs & Outputs*, arXiv:2107.14795 [cs, eess], Mar. 2022. [Online]. Available: http://arxiv.org/abs/2107.14795 (visited on 06/22/2023).

[34] K. He, X. Zhang, S. Ren, and J. Sun, *Deep Residual Learning for Image Recognition*, arXiv:1512.03385 [cs], Dec. 2015. DOI: 10.48550/arXiv.1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385 (visited on 03/03/2025).

[35] J. L. Ba, J. R. Kiros, and G. E. Hinton, *Layer Normalization*, arXiv:1607.06450 [stat], Jul. 2016. DOI: 10.48550/arXiv.1607.06450. [Online]. Available: http://arxiv.org/abs/1607.06450 (visited on 03/03/2025).

[36] D. Hendrycks and K. Gimpel, *Gaussian Error Linear Units (GELUs)*, arXiv:1606.08415 [cs], Jun. 2023. [Online]. Available: http://arxiv.org/abs/1606.08415 (visited on 11/23/2023).

[37] A. Gu and T. Dao, *Mamba: Linear-Time Sequence Modeling with Selective State Spaces*, arXiv:2312.00752 [cs], May 2024. DOI: 10.48550/arXiv.2312.00752. [Online]. Available: http://arxiv.org/abs/2312.00752 (visited on 01/30/2025).

[38] H. Touvron, T. Lavril, G. Izacard, *et al.*, *LLaMA: Open and Efficient Foundation Language Models*, en, arXiv:2302.13971 [cs], Feb. 2023. [Online]. Available: http://arxiv.org/abs/2302.13971 (visited on 06/20/2024).

[39] E. Almazrouei, H. Alobeidli, A. Alshamsi, *et al.*, *The Falcon Series of Open Language Models*, arXiv:2311.16867 [cs], Nov. 2023. DOI: 10.48550/arXiv.2311.16867. [Online]. Available: http://arxiv.org/abs/2311.16867 (visited on 03/03/2025).

[40] R. Xiong, Y. Yang, D. He, *et al.*, *On Layer Normalization in the Transformer Architecture*, arXiv:2002.04745 [cs, stat], Jun. 2020. [Online]. Available: http://arxiv.org/abs/2002.04745 (visited on 06/22/2023).

[41] B. Zhang and R. Sennrich, *Root Mean Square Layer Normalization*, arXiv:1910.07467 [cs], Oct. 2019. DOI: 10.48550/arXiv.1910.07467. [Online]. Available: http://arxiv.org/abs/1910.07467 (visited on 03/03/2025).

[42] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu, *RoFormer: Enhanced Transformer with Rotary Position Embedding*, arXiv:2104.09864 [cs], Aug. 2022. [Online]. Available: http://arxiv.org/abs/2104.09864 (visited on 06/22/2023).

[43] X. Liu, H. Yan, S. Zhang, C. An, X. Qiu, and D. Lin, *Scaling Laws of RoPE-based Extrapolation*, arXiv:2310.05209 [cs], Mar. 2024. DOI: 10.48550/arXiv.2310.05209 . [Online]. Available: http://arxiv.org/abs/2310.05209 (visited on 03/03/2025).

[44] N. Shazeer, *GLU Variants Improve Transformer*, arXiv:2002.05202 [cs, stat], Feb. 2020. [Online]. Available: http://arxiv.org/abs/2002.05202 (visited on 06/22/2023).

[45] P. Ramachandran, B. Zoph, and Q. V. Le, *Searching for Activation Functions*, arXiv:1710.05941 [cs] version: 2, Oct. 2017. DOI: 10.48550/arXiv.1710.05941. [Online]. Available: http://arxiv.org/abs/1710.05941 (visited on 03/03/2025).

[46] A. Chowdhery, S. Narang, J. Devlin, *et al.*, *PaLM: Scaling Language Modeling with Pathways*, arXiv:2204.02311 [cs], Oct. 2022. [Online]. Available: http://arxiv.org/abs/2204.02311 (visited on 06/22/2023).

[47] O. Rybakov, M. Chrzanowski, P. Dykas, J. Xue, and B. Lanir, *Methods of improving LLM training stability*, arXiv:2410.16682 [cs], Oct. 2024. DOI: 10.48550/arXiv.2410.16682. [Online]. Available: http://arxiv.org/abs/2410.16682 (visited on 03/03/2025).

[48] R. Sennrich, B. Haddow, and A. Birch, *Neural Machine Translation of Rare Words with Subword Units*, arXiv:1508.07909 [cs] version: 5, Jun. 2016. DOI: 10.48550/arXiv.1508.07909. [Online]. Available: http://arxiv.org/abs/1508.07909 (visited on 03/02/2025).

[49] Y. Wu, M. Schuster, Z. Chen, *et al.*, *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*, arXiv:1609.08144 [cs] version: 2, Oct. 2016. DOI: 10.48550/arXiv.1609.08144. [Online]. Available: http://arxiv.org/abs/1609.08144 (visited on 03/02/2025).

[50] *Overview of SNOMED CT*, eng, Product, Program, and Project Descriptions, Publisher: U.S. National Library of Medicine. [Online]. Available: https://www.nlm.nih.gov/healthit/snomedct/snomed_overview.html (visited on 03/07/2025).

[51] *Datatypes - FHIR v6.0.0-ballot2*. [Online]. Available: https://build.fhir.org/datatypes.html (visited on 03/10/2025).

[52] W. H. Organization, *ICD-10 : International statistical classification of diseases and related health problems : Tenth revision*, Edition: 2nd ed Pages: Spanish version, 1st edition published by PAHO as Publicación Científica 544, 2004.

[53] J. E. Harrison, S. Weber, R. Jakob, and C. G. Chute, "ICD-11: An international classification of diseases for the twenty-first century," eng, *BMC medical informatics and decision making*, vol. 21, no. Suppl 6, p. 206, Nov. 2021, ISSN: 1472-6947. DOI: 10.1186/s12911-021-01534-6.

[54] A. W. Forrey, C. J. McDonald, G. DeMoor, *et al.*, "Logical observation identifier names and codes (LOINC) database: A public use set of codes and names for electronic reporting of clinical laboratory test results," eng, *Clinical Chemistry*, vol. 42, no. 1, pp. 81–90, Jan. 1996, ISSN: 0009-9147.

[55] C. J. McDonald, S. M. Huff, J. G. Suico, *et al.*, "LOINC, a universal standard for identifying laboratory observations: A 5-year update," eng, *Clinical Chemistry*, vol. 49, no. 4, pp. 624–633, Apr. 2003, ISSN: 0009-9147. DOI: 10.1373/49.4.624.

[56] S. Liu, W. Ma, R. Moore, V. Ganesan, and S. Nelson, "RxNorm: Prescription for electronic drug information exchange," *IT Professional*, vol. 7, no. 5, pp. 17–23, Sep. 2005, Conference Name: IT Professional, ISSN: 1941-045X. DOI: 10.1109/MITP.2005.122. [Online]. Available: https://ieeexplore.ieee.org/document/1516084 (visited on 03/07/2025).

[57] S. J. Nelson, K. Zeng, J. Kilbourne, T. Powell, and R. Moore, "Normalized names for clinical drugs: RxNorm at 6years," *Journal of the American Medical Informatics Association : JAMIA*, vol. 18, no. 4, pp. 441–448, 2011, ISSN: 1067-5027. DOI: 10.1136/amiajnl-2011-000116. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3128404/ (visited on 03/07/2025).

[58] R. Vuokko, A. Vakkuri, and S. Palojoki, "Systematized Nomenclature of Medicine–Clinical Terminology (SNOMED CT) Clinical Use Cases in the Context of Electronic Health Record Systems: Systematic Literature Review," *JMIR Medical Informatics*, vol. 11, e43750, Feb. 2023, ISSN: 2291-9694. DOI: 10.2196/43750. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9941898/ (visited on 03/07/2025).

[59] Y. Li, S. Rao, J. R. A. Solares, *et al.*, "BEHRT: Transformer for Electronic Health Records," en, *Scientific Reports*, vol. 10, no. 1, p. 7155, Apr. 2020, Number: 1 Publisher: Nature Publishing Group, ISSN: 2045-2322. DOI: 10.1038/s41598-020-62922-y. [Online]. Available: https://www.nature.com/articles/s41598-020-62922-y (visited on 06/22/2023).

[60] E. Alsentzer, J. R. Murphy, W. Boag, *et al.*, *Publicly Available Clinical BERT Embeddings*, arXiv:1904.03323 [cs], Jun. 2019. [Online]. Available: http://arxiv.org/abs/1904.03323 (visited on 06/22/2023).

[61] Z. Kraljevic, A. Shek, D. Bean, R. Bendayan, J. Teo, and R. Dobson, *MedGPT: Medical Concept Prediction from Clinical Narratives*, arXiv:2107.03134 [cs], Jul. 2021. [Online]. Available: http://arxiv.org/abs/2107.03134 (visited on 06/22/2023).

[62] K. Singhal, S. Azizi, T. Tu, *et al.*, *Large Language Models Encode Clinical Knowledge*, arXiv:2212.13138 [cs], Dec. 2022. [Online]. Available: http://arxiv.org/abs/2212.13138 (visited on 06/22/2023).

[63] D. Brin, V. Sorin, A. Vaid, *et al.*, "Comparing ChatGPT and GPT-4 performance in USMLE soft skill assessments," en, *Scientific Reports*, vol. 13, no. 1, p. 16 492, Oct. 2023, Publisher: Nature Publishing Group, ISSN: 2045-2322. DOI: 10.1038/s41598-023-43436-9. [Online]. Available: https://www.nature.com/articles/s41598-023-43436-9 (visited on 03/08/2025).

[64] V. Nair, E. Schumacher, and A. Kannan, *Generating medically-accurate summaries of patient-provider dialogue: A multi-stage approach using large language models*, arXiv:2305.05982 [cs], May 2023. DOI: 10.48550/arXiv.2305.05982. [Online]. Available: http://arxiv.org/abs/2305.05982 (visited on 03/08/2025).

[65] V. Yaneva, P. Baldwin, D. P. Jurich, K. Swygert, and B. E. Clauser, "Examining ChatGPT Performance on USMLE Sample Items and Implications for Assessment," en-US, *Academic Medicine*, vol. 99, no. 2, p. 192, Feb. 2024, ISSN: 1040-2446. DOI: 10.1097/ACM.0000000000005549. [Online]. Available: https://journals.lww.com/academicmedicine/fulltext/2024/02000/examining_chatgpt_performance_on_usmle_sample.24.aspx (visited on 03/01/2025).

[66] T. H. Kung, M. Cheatham, A. Medenilla, *et al.*, "Performance of ChatGPT on USMLE: Potential for AI-assisted medical education using large language models," *PLOS Digital Health*, vol. 2, no. 2, e0000198, Feb. 2023, ISSN: 2767-3170. DOI: 10.1371/journal.pdig.0000198. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9931230/ (visited on 03/01/2025).

[67] Y. Li, H. Wang, H. Z. Yerebakan, Y. Shinagawa, and Y. Luo, "FHIR-GPT Enhances Health Interoperability with Large Language Models," en,

[68] E. Choi, M. T. Bahadori, J. A. Kulas, A. Schuetz, W. F. Stewart, and J. Sun, *RETAIN: An Interpretable Predictive Model for Healthcare using Reverse Time Attention Mechanism*, arXiv:1608.05745, Feb. 2017. [Online]. Available: http://arxiv.org/abs/1608.05745 (visited on 10/22/2024).

[69] Y. Xiang, J. Xu, Y. Si, *et al.*, "Time-sensitive clinical concept embeddings learned from large electronic health records," *BMC Medical Informatics and Decision Making*, vol. 19, no. 2, p. 58, Apr. 2019, ISSN: 1472-6947. DOI: 10.1186/s12911-019-0766-3. [Online]. Available: https://doi.org/10.1186/s12911-019-0766-3 (visited on 10/22/2024).

[70] A. L. Goldberger, L. A. N. Amaral, L. Glass, *et al.*, "PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, e215–e220, 2000 (June 13), Circulation Electronic Pages: http://circ.ahajournals.org/content/101/23/e215.full PMID:1085218; doi: 10.1161/01.CIR.101.23.e215.

[71] A. E. W. Johnson, L. Bulgarelli, L. Shen, *et al.*, "MIMIC-IV, a freely accessible electronic health record dataset," en, *Scientific Data*, vol. 10, no. 1, p. 1, Jan. 2023, ISSN: 2052-4463. DOI: 10.1038/s41597-022-01899-x. [Online]. Available: https://www.nature.com/articles/s41597-022-01899-x (visited on 08/13/2024).

[72] A. Johnson, L. Bulgarelli, T. Pollard, S. Horng, L. A. Celi, and R. Mark, *MIMIC-IV*. DOI: 10.13026/6MM1-EK67. [Online]. Available: https://physionet.org/content/mimiciv/2.2/ (visited on 08/13/2024).

[73] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, arXiv:1603.02754 [cs], Aug. 2016, pp. 785–794. DOI: 10.1145/2939672.2939785. [Online]. Available: http://arxiv.org/abs/1603.02754 (visited on 01/22/2025).

[74] J. P. Cohen, J. D. Viviano, P. Bertin, *et al.*, *TorchXRayVision: A library of chest X-ray datasets and models*, arXiv:2111.00595 [eess], Oct. 2021. DOI: 10.48550/arXiv.2111.00595. [Online]. Available: http://arxiv.org/abs/2111.00595 (visited on 01/22/2025).

[75] G. Huang, Z. Liu, L. v. d. Maaten, and K. Q. Weinberger, *Densely Connected Convolutional Networks*, arXiv:1608.06993 [cs], Jan. 2018. DOI: 10.48550/arXiv.1608.06993. [Online]. Available: http://arxiv.org/abs/1608.06993 (visited on 01/22/2025).

[76] J. Irvin, P. Rajpurkar, M. Ko, *et al.*, *CheXpert: A Large Chest Radiograph Dataset with Uncertainty Labels and Expert Comparison*, arXiv:1901.07031 [cs], Jan. 2019. DOI: 10.48550/arXiv.1901.07031. [Online]. Available: http://arxiv.org/abs/1901.07031 (visited on 01/22/2025).

[77] L. R. Soenksen, Y. Ma, C. Zeng, *et al.*, *Code for generating the HAIM multimodal dataset of MIMIC-IV clinical data and x-rays*, Aug. 2022. DOI: 10.13026/3F8D-QE93. [Online]. Available: https://physionet.org/content/haim-multimodal/1.0.1/ (visited on 01/28/2025).

[78] K. Schlegel, P. Neubert, and P. Protzel, "A comparison of Vector Symbolic Architectures," en, *Artificial Intelligence Review*, vol. 55, no. 6, pp. 4523–4555, Aug. 2022, arXiv:2001.11797 [cs], ISSN: 0269-2821, 1573-7462. DOI: 10.1007/s10462-021-10110-3. [Online]. Available: http://arxiv.org/abs/2001.11797 (visited on 09/27/2024).

[79] T. Plate, "Holographic reduced representations," *IEEE Transactions on Neural Networks*, vol. 6, no. 3, pp. 623–641, May 1995, Conference Name: IEEE Transactions on Neural Networks, ISSN: 1941-0093. DOI: 10.1109/72.377968. [Online]. Available: https://ieeexplore.ieee.org/document/377968 (visited on 02/07/2024).

[80] B. Hu, T. Yu, T. Tuinstra, *et al.*, "Encoding Medical Ontologies With Holographic Reduced Representations for Transformers," in *KiL'24: Workshop on Knowledge-infused Learning*, Barcelona, Spain: Association for Computing Machinery, Aug. 2024.

[81] N. S.-Y. Dumont, "Accurate representation for spatial cognition using grid cells," en,

[82] N. S.-Y. Dumont, P. M. Furlong, J. Orchard, and C. Eliasmith, "Exploiting semantic information in a spiking neural SLAM system," *Frontiers in Neuroscience*, vol. 17, p. 1 190 515, Jul. 2023, ISSN: 1662-4548. DOI: 10.3389/fnins.2023.1190515. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10354246/ (visited on 01/30/2024).

[83] P. M. Furlong and C. Eliasmith, "Fractional Binding in Vector Symbolic Architectures as Quasi-Probability Statements," en,

[84] A. Gu, K. Goel, and C. Ré, *Efficiently Modeling Long Sequences with Structured State Spaces*, arXiv:2111.00396 [cs], Aug. 2022. DOI: 10.48550/arXiv.2111.00396. [Online]. Available: http://arxiv.org/abs/2111.00396 (visited on 10/09/2024).

[85] A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Re, *HiPPO: Recurrent Memory with Optimal Polynomial Projections*, arXiv:2008.07669 [cs], Oct. 2020. DOI: 10.48550/arXiv.2008.07669. [Online]. Available: http://arxiv.org/abs/2008.07669 (visited on 01/30/2025).

[86] T. Dao and A. Gu, *Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality*, en, arXiv:2405.21060 [cs], May 2024. [Online]. Available: http://arxiv.org/abs/2405.21060 (visited on 08/14/2024).

[87] A. M. Bennett, H. Ulrich, P. van Damme, J. Wiedekopf, and A. E. W. Johnson, "MIMIC-IV on FHIR: Converting a decade of in-patient data into an exchangeable, interoperable format," *Journal of the American Medical Informatics Association*, vol. 30, no. 4, pp. 718–725, Apr. 2023, ISSN: 1527-974X. DOI: 10.1093/jamia/ocad002. [Online]. Available: https://doi.org/10.1093/jamia/ocad002 (visited on 10/17/2023).

[88] A. Bennett, J. Wiedekopf, H. Ulrich, P. van Damme, and A. Johnson, *MIMIC-IV on FHIR*. DOI: 10.13026/CQT2-0B27. [Online]. Available: https://physionet.org/content/mimic-iv-fhir/1.0/ (visited on 08/13/2024).

[89] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*, arXiv:2205.14135 [cs], Jun. 2022. [Online]. Available: http://arxiv.org/abs/2205.14135 (visited on 06/22/2023).

[90] T. Dao, *FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning*, en, arXiv:2307.08691 [cs], Jul. 2023. [Online]. Available: http://arxiv.org/abs/2307.08691 (visited on 06/13/2024).

[91] N. Muennighoff, N. Tazi, L. Magne, and N. Reimers, *MTEB: Massive Text Embedding Benchmark*, arXiv:2210.07316 [cs], Mar. 2023. DOI: 10.48550/arXiv.2210.07316. [Online]. Available: http://arxiv.org/abs/2210.07316 (visited on 03/02/2025).

[92] N. Japkowicz, "Assessment metrics for imbalanced learning," in *Imbalanced learning*, Section: 8 tex.eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118646106.ch8, John Wiley & Sons, Ltd, 2013, pp. 187–206, ISBN: 978-1-118-64610-6. DOI: https://doi.org/10.1002/9781118646106.ch8. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118646106.ch8.

[93] J. Davis and M. Goadrich, "The relationship between Precision-Recall and ROC curves," en, in *Proceedings of the 23rd international conference on Machine learning - ICML '06*, Pittsburgh, Pennsylvania: ACM Press, 2006, pp. 233–240, ISBN: 978-1-59593-383-6. DOI: 10.1145/1143844.1143874. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1143844.1143874 (visited on 03/05/2025).

[94] T. Saito and M. Rehmsmeier, "The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets," *PLoS ONE*, vol. 10, no. 3, e0118432, Mar. 2015, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0118432. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4349800/ (visited on 03/05/2025).

[95] C. W. Dunnett, "A Multiple Comparison Procedure for Comparing Several Treatments with a Control," EN, *Journal of the American Statistical Association*, Dec. 1955, Publisher: Taylor & Francis Group. [Online]. Available: https://amstat.tandfonline.com/doi/abs/10.1080/01621459.1955.10501294 (visited on 03/05/2025).

[96] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, arXiv:1412.6980, Jan. 2017. DOI: 10.48550/arXiv.1412.6980. [Online]. Available: http://arxiv.org/abs/1412.6980 (visited on 10/29/2024).

[97] I. Loshchilov and F. Hutter, *Decoupled Weight Decay Regularization*, arXiv:1711.05101, Jan. 2019. DOI: 10.48550/arXiv.1711.05101. [Online]. Available: http://arxiv.org/abs/1711.05101 (visited on 10/29/2024).

[98] Y. Zhao, A. Gu, R. Varma, *et al.*, *PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel*, arXiv:2304.11277, Sep. 2023. [Online]. Available: http://arxiv.org/abs/2304.11277 (visited on 10/29/2024).

[99] *21st Century Cures Act: Interoperability, Information Blocking, and the ONC Health IT Certification Program*, en, Jun. 2020. [Online]. Available: https://www.federalregister.gov/documents/2020/05/01/2020-07419/21st-century-cures-act-interoperability-information-blocking-and-the-onc-health-it-certification (visited on 03/10/2025).

[100] Y. Liu, M. Ott, N. Goyal, *et al.*, *RoBERTa: A Robustly Optimized BERT Pretraining Approach*, en, arXiv:1907.11692 [cs], Jul. 2019. [Online]. Available: http://arxiv.org/abs/1907.11692 (visited on 06/17/2024).

[101] P. J. Huber, "Robust Estimation of a Location Parameter," *The Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73–101, Mar. 1964, Publisher: Institute of Mathematical Statistics, ISSN: 0003-4851, 2168-8990. DOI: 10.1214/aoms/1177703732. [Online]. Available: https://projecteuclid.org/journals/annals-of-mathematical-statistics/volume-35/issue-1/Robust-Estimation-of-a-Location-Parameter/10.1214/aoms/1177703732.full (visited on 03/09/2025).

[102] A. Wettig, T. Gao, Z. Zhong, and D. Chen, *Should You Mask 15% in Masked Language Modeling?* arXiv:2202.08005 [cs], Feb. 2023. DOI: 10.48550/arXiv.2202.08005. [Online]. Available: http://arxiv.org/abs/2202.08005 (visited on 03/02/2025).

[103] M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy, "SpanBERT: Improving Pre-training by Representing and Predicting Spans," *Transactions of the Association for Computational Linguistics*, vol. 8, M. Johnson, B. Roark, and A. Nenkova, Eds., pp. 64–77, 2020, Place: Cambridge, MA Publisher: MIT Press. DOI: 10.1162/tacl_a_00300. [Online]. Available: https://aclanthology.org/2020.tacl-1.5/ (visited on 03/05/2025).

[104] Y. Levine, B. Lenz, O. Lieber, *et al.*, *PMI-Masking: Principled masking of correlated spans*, arXiv:2010.01825 [cs], Oct. 2020. DOI: 10.48550/arXiv.2010.01825. [Online]. Available: http://arxiv.org/abs/2010.01825 (visited on 03/05/2025).

[105] J. Hoffmann, S. Borgeaud, A. Mensch, *et al.*, *Training Compute-Optimal Large Language Models*, arXiv:2203.15556 [cs], Mar. 2022. DOI: 10.48550/arXiv.2203.15556. [Online]. Available: http://arxiv.org/abs/2203.15556 (visited on 03/03/2025).

APPENDICES

Appendix A

Pre-training

I initially intended to pre-train all our models using the masked language modelling task, then finetune the pre-trained model on several downstream tasks. However, it became apparent that this approach did not benefit the performance on downstream tasks and was not worth the compute time to run pre-training. In this chapter, I report on the methodology and discuss how pre-training a FHIR foundation model might be a valuable direction for future research.

A.1 Methodology

In this section, I describe the methodology to apply the masked language modelling (MLM) pre-training task to our model. In particular, modifications need to be made when including the continuous encoding (section 4.4) and sequence compression (section 4.6) methodology.

I use the MLM formulation as was used in BERT [16] but without any sequence-level task as recommended by [100]. I use the same masking probabilities as [16]: 15% probability of a token being masked, 80% probability of a masked token using the `[MASK]` token, 10% probability of a masked token using a random token, and 10% probability of a masked token using the original token for labels. Applying the standard MLM loss as a multinomial classification problem over the masked tokens is trivial when discrete numeric and time encodings are used without compression. This formulation is what the control models use with Quantile Bins numeric encoding and Round to Integer time encoding, as the targets for these quantities are simply the token IDs.

A.1.1 MLM with Continuous Encoding

To apply MLM to sequences of vectors where some are derived from continuous encoding, classification labels cannot created for the continuously encoded vectors. Instead, I split the modelling task into a classification component and regression component. The classification remains the same as the standard MLM task over any tokens, such as medical codes.

For numeric or time values that use continuous encoding, I instead predict the normalized value that was passed into the continuous encoding layer for that token and average the loss over all such continuously encoded tokens. The notions of masking and random replacement change how the inputs are constructed for the continuous encoding. To construct the regression labels, 15% of the continuous tokens are sampled to use as MLM targets. Next, a corruption process is performed, with the probabilities for "mask", "replace", and "same" used in BERT [16]. If a sequence element is masked, I insert a `[MASK]` token into the input ID sequence and remove its value from the continuous encoder input sequence. If a sequence element is replaced, I sample a random numeric input value from a uniform distribution and replace the true value in the continuous encoder inputs. I choose to use a uniform distribution because BERT MLM uses a uniform probability for selecting its replacement tokens from the vocabulary as well. For normalize numeric base, I use $\mathcal{U}(-6, 6)$ and for quantile numeric base, I use $\mathcal{U}(0, 1)$. If a sequence element is the same as the original, I make no changes to its input representation, but use the value as a target in the loss. This process is illustrated in Figure A.1.

I tried using mean squared error (MSE) loss for this regression task, but the effect of large outliers was profound, especially when using the normalize numeric base and could lead to large loss spikes. Instead, I used the Huber loss function [101] to reduce the effect of outliers on the loss term. The final model loss is the sum of the Huber regression loss and the cross entropy classification loss over all masked tokens.

A.1.2 MLM with Compression

Masking tokens before applying compression presents a challenge because each compressed vector may have multiple masked tokens in the subsequence, and therefore multiple labels. I considered masking only a single token per subsequence, but this would result in much less than 15% of the original sequence being masked and an inefficient use of data. Instead, I recognize that each resource subsequence typically contains several independent categories of tokens. I define six categories: code tokens (such as ICD diagnosis codes, observation and procedure item codes, and medication codes), numeric tokens, time tokens, text tokens

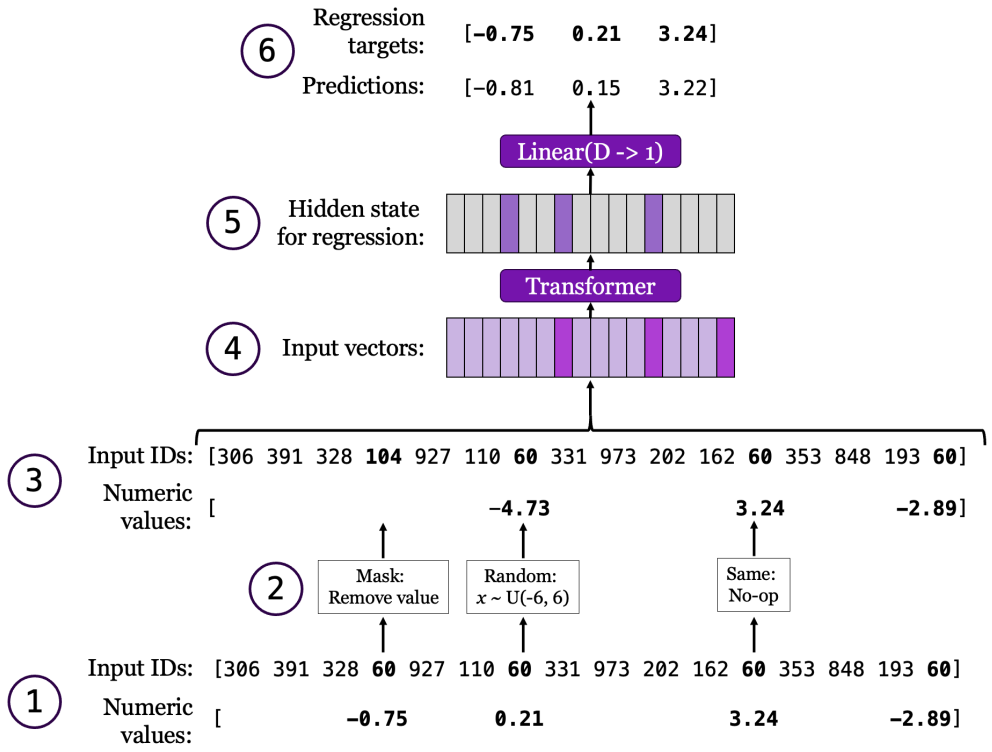


Figure A.1: Graphical illustration of the MLM task using regression on continuous encodings. (1) shows the original sequence of input IDs and the associated numeric values that would be continuously encoded. (2) illustrates the masking process, where some continuous values are corrupted. From left to right, Mask removes the value from the continuous encoding sequence and inserts a `[MASK]` token (`104`) into the input IDs, Random replaces the value with a random value drawn from a uniform distribution, and Same performs no additional operations. The right-most continuous value is not masked. (3) shows the resulting input sequences after the masking process. Notably, the numeric values sequences contains fewer elements since some are masked. (4) represents the input vectors to the transformer backbone, where the darker vectors are those that have been continuously encoded. (5) represents the hidden state vectors output by the transformer backbone. The regression task is only performed for the three highlighted vectors that have been altered during the masking process. Language modelling classification would be performed on some of the other vectors. Finally, (6) shows the model predictions and the corresponding labels from (1) on which the regression loss is computed.

(the BERT vocabulary), unit tokens (from the units of measurement code system), and miscellaneous tokens (all other tokens, including statuses and patient demographics). For each of these categories, I independently assign a 25% probability of the category being masked within the resource. If the category is masked, I pick a single token of that category within the resource subsequence at random and apply the mask/replace/same treatment to that token. By using a 25% probability of a category being masked, approximately 15% of the uncompressed sequence elements are masked in our dataset.

To compute the loss, I have a separate classification head for each category that predicts classes for vocabulary items within that category. The loss is only computed if that category was masked. The cross-entropy losses over all categories are averaged per compressed element, and the average loss over all compressed sequence elements is used as the final MLM loss. The overall process is illustrated in Figure A.2.

In principle, the continuous loss for numeric and time encoding could also be combined with these methods and a regression loss would be used on that head instead of a classification loss. I do not test this in our methods, but compression and continuous encoding should be compatible for pre-training.

A.2 Pre-training MIMIC Dataset

To test our model with MLM pre-training, I construct a dataset with an entire patient's history in the MIMIC-IV FHIR dataset, but only using resources from the *hosp* dataset. After filtering for patients with more than 3 resources in their record and less than 8192 tokens, there are 129,975 patients in our dataset. I separate out 24,000 patients for our fine-tuning dataset and 5,000 for our test dataset. The remaining 100,975 are used for pre-training.

To construct the pre-training dataset, I merge tokenized patient sequences together by concatenating two sequences such that the total length is as close to 8192 as possible. This is done to maximize the number of tokens which can be masked, as MLM already only uses 15% of tokens in the sequence to contribute to the loss. I insert `[SEP]` tokens between different patient sequences similar to how BERT separates two different sentences [16]. After two rounds of this merging procedure, it is possible that up to four short patient records could be represented in one sequence. There are a total of 39,042 sequences after merging. Of this dataset, 1000 sequences are reserved for validation during the pre-training stage.

To construct fine-tuning and test datasets, only one patient record is used per sequence

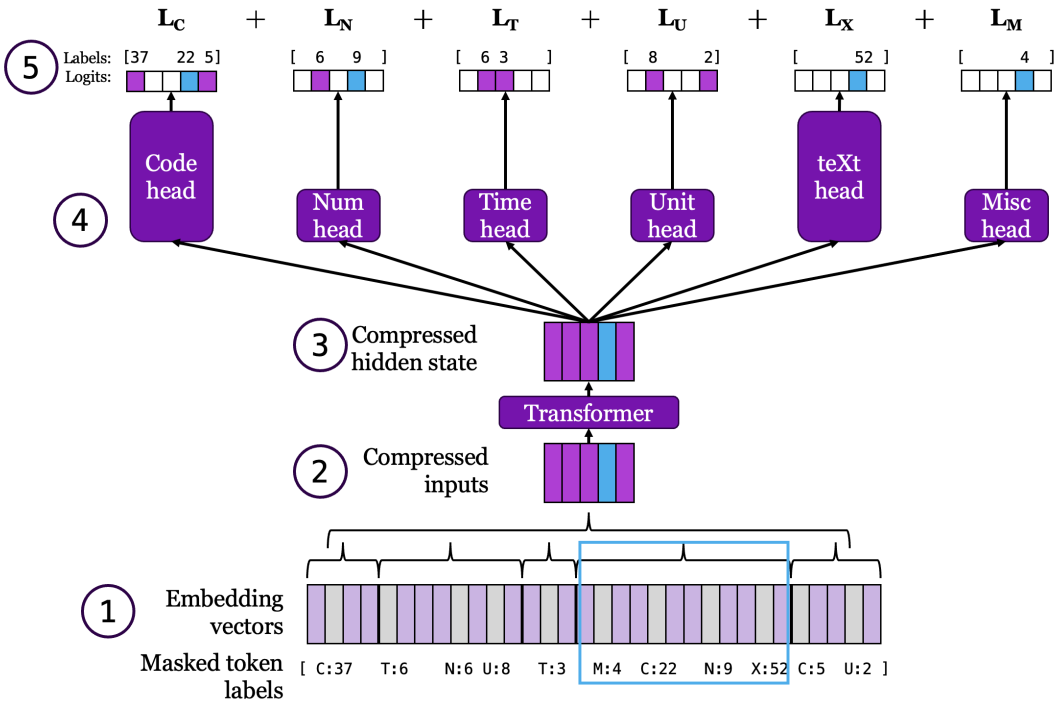


Figure A.2: Graphical illustration of the MLM task when using compression. (1) shows the sequence of input vectors after the masking process and the associated token labels. The labels are in the form of "token category:index" where C = code, M = miscellaneous, N = numeric, T = time, U = unit, X = text. (2) shows the resulting sequence of vectors after compression is applied. This sequence is input to a transformer backbone and the final layer hidden state is shown in (3). Each of the hidden state vectors is passed through the six classification heads, shown in (4). The code and text heads are illustrated as larger as they have more vocabulary items compared to the others. (5) shows the resulting logits after each classification head and the MLM labels for each token category. The cross entropy loss is only computed over items that were masked, unused logits are shown in white. The final loss is the average of the individual category losses. Following the fourth resource, highlighted in blue, there are 4 different categories on which MLM predictions are made, demonstrating how multiple vectors from within a single resource can be predicted with this method.

when performing sequence-level classification tasks. Each patient may have multiple Encounters on which the tasks can apply. Each Encounter becomes an item with a different reference timestamp for predictions and only resources prior to the reference timestamp

are kept when creating inputs. I use 24 hours into the encounter as the prediction timestamp. The LOS and MP tasks are formulated similarly to the Extended context dataset in section 3.4. Our fine-tuning dataset has an 11% positive LOS class with a positive class weight of 8.1, and a 1.65% positive MP class, with a positive class weight of 60.

The 24,000 fine-tuning patients produce 33,984 sequences and the 5,000 testing patients produce 7,365 sequences after creating separate sequences for the different prediction times. During fine-tuning, I reserve 4,000 sequences (not patients) for validation and model selection.

A.3 Experimental Methods

I pre-train our model with sequence lengths of 8192, a total batch size size of 32, and a total of 16,000 steps. This amounts to approximately 4.2B tokens or 13 epochs through the pre-training data. The rest of the training setup is similar to section 5.2. I pre-train three replicates using different seeds for data splits and parameter initialization.

Following pre-training, I fine-tune each model on the LOS and MP tasks following the methods in section 5.2, except the parameters are initialized from the pre-trained model and I only fine-tune for 5 epochs. Each seed used for pre-training uses a different seed for the classifier head parameter initialization, and dataset splitting for fine-tuning. I also train models directly on the fine-tuning tasks, without pre-training, for 10 epochs. Following fine-tuning, the best model based on validation AUROC is evaluated on the held-out test dataset.

A.4 Results and Discussion

I found overall the main benefit of pre-training was from slightly faster convergence on the fine-tuning tasks, where I could reliably achieve convergence within 5 epochs instead of 10 without pre-training. However, the time required to pre-train each model was over 30 hours on 2 Nvidia RTX 3090s, which is well over the time required for an extra 5 epochs on the fine-tuning dataset. I found no significant differences between the mean AUROC scores from models that were just fine-tuned on the LOS and MP tasks and the models that had MLM pre-training.

I validated that our methods worked to pre-train models using continuous encoding and compression separately, though I did not pre-train any models that used both methods

simultaneously. However, in principle, these methods would be compatible together. I found that pre-training with continuous numeric encodings did not significantly improve the mean AUROC compared to models that used the control Quantile bins encoding method. Pre-training was more stable using the Huber loss for regression rather than MSE, with fewer loss spikes. However, models using MSE loss did not perform significantly worse than the control models.

The models that used the compression methods performed well in pre-training, achieving over 92% validation classification accuracy in the MLM predictions across all prediction instances. I observed that certain categories were more challenging than others. The numeric token category was the most difficult, with a validation accuracy of 77%. The code category had the next lowest accuracy of 90%. Time, text, and miscellaneous tokens had approximately 95% accuracy, and the unit tokens had over a 99.9% accuracy. The ease of predicting the text and miscellaneous tokens may come from the fact there are not many instances and the most common ones cover a large portion of the data. For example, many of the text usages in the dataset are just the words "Normal" or "In range" as a text comment about an Observation. Similarly, the miscellaneous tokens are often statuses or patient demographic information which have a small subset of possible values that the model may easily be able to predict given the context. For example, tokens for race and ethnicity would be highly correlated, and unless both are masked, a model may be able to determine the other easily. Units appeared to be nearly perfectly predicted. However, when looking at the structure of the token sequence for Observation resources, the unit is repeated for the `valueQuantity` and `referenceRange` fields, so if only one of those is masked, then its value would be already present in the compressed vector from a different field. For time tokens, the high accuracy might be owed to the fact that many resource nearby in the sequence would share the same time token. Thus, a masked time token could be easily determined by the choosing the token that is used in neighbouring resources. Similar to the other methods, there was no significant performance gain by pre-training with compressed MLM. However, the compressed training with Sum compression was much faster than non-compressed pre-training (18 hours versus 30 hours).

Future work with MLM on FHIR data should examine adjusting the task to make MLM more effective. In this work, I only experimented with the BERT 15% masking ratio, but literature suggests higher masking rates such as 40% can be more effective for BERT-large sized models performance on downstream tasks [102]. Other masking strategies such as span-masking [103] or PMI-masking [104] may help address highly correlated token spans. There may also be FHIR-data specific strategies such as masking all time tokens in resources that share the same token, or encoding datatypes like ReferenceRange as a single token to reduce obvious correlations.

Comparisons to other literature suggests I did not have enough data in MIMIC-IV FHIR to effectively perform pre-training. MedBERT has access to 22 million patients for pre-training [21] while I use only 100 thousand patients for pre-training. In both my work and MedBERT, fine-tuning is performed on a dataset of around 30 thousand patients.

BERT was originally trained with 128 billion tokens over 40 epochs [16]. In contrast, our model is on the order of 100M parameters, and does 13 epochs through our training dataset to achieve 4.2 billion tokens. BERT fine-tuning tasks are also on the order of tens of thousands of examples.

Hoffmann *et al.* suggest that a 400M-parameter text transformer model trained with CLM would be optimally trained with respect to a fixed compute budget using one epoch of 8 billion tokens [105]. MLM only uses 15% of the tokens to compute the loss, whereas CLM uses all tokens in the sequence to compute the loss. However, tokenized FHIR data is different than text data, so using text scaling laws is not directly applicable to FHIR models. Additionally, scaling laws do not inform how downstream fine-tuning task performance is affected. However, the procedure suggests that for a model of 100M, a ball park of billions of tokens for one epoch is reasonable.

Overall, I estimate that I would require at least 10× to 100× the amount of data that I currently use in order to make pre-training effective. I would want at least 4 billion tokens for one epoch from millions of patients, compared to the 325 million tokens for one epoch from 100 thousand patients as I tested. If such a FHIR dataset can be obtained, future work should done to determine whether pre-training is beneficial for improving downstream performance on medical classification tasks using FHIR data. Such a dataset could also be used to determine scaling laws for compute-optimal FHIR models.