COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS PHYSICS AND INFORMATICS

PREDICTION OF HEALTH STATUS DETERIORATION

Master thesis

2025

Bc. Marián Kravec

COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS PHYSICS AND INFORMATICS

PREDICTION OF HEALTH STATUS DETERIORATION

Master thesis

Study program:	Applied informatics
Branch of study:	Applied informatics
Department:	Department of Applied Informatics
Supervisor:	MSc. František Dráček
Consultant:	

Bratislava, 2025 Bc. Marián Kravec

58379976

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. Marián Kravec
Študijný program:	aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor:	informatika
Typ záverečnej práce:	diplomová
Jazyk záverečnej práce:	anglický
Sekundárny jazyk:	slovenský

Názov: Prediction of Health Status Deterioration
Predikcia zhoršenia zdravotného stavu

Anotácia: V súčastnosti sa sektor zdravotníctva na Slovensku vyznačuje nizkou mierou využita dostupnych zdravotníckych dát. V rámci tejto prace je cieľom ukázať, že z existujúcjich dát je možné predikovať vyvoj dalšieho zdravotného stavu pacienta, poprípade odhadnúť vývoj budúcich nákladov za účelom lepšieho plánovania prerozdelenia financí v rámci sektoru.

Cieľ: Práca bude rozdelená na dve časti, v prvej študent urobí teoretické zhrnutie existujúchich metód spracovania dát a metód strojového učenia, ktoré sa budú dať potenciálne aplikovať na daný problém. V druhej časti navrhne a aplikuje predičkný model.

Literatúra: T. Sk, L. M. G, L. R. K and R. R. J, "Health Status Prediction using ML Techniques," 2022 6th International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 2022, pp. 1191-1196, doi: 10.1109/ICCMC53470.2022.9753766.

Jödicke, A.M., Zellweger, U., Tomka, I.T. et al. Prediction of health care expenditure increase: how does pharmacotherapy contribute?. BMC Health Serv Res 19, 953 (2019). https://doi.org/10.1186/s12913-019-4616-x

Vedúci:	MSc. František Dráček
Katedra:	FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry:	doc. RNDr. Tatiana Jajcayová, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 05.10.2023

Dátum schválenia: 05.10.2023

prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

[illegible]

.... student

.... vedúci práce

I hereby declare that I have written this thesis by myself, only with help of referenced literature, under the careful supervision of my thesis advisor.

. . . .

Bratislava, 2025 Bc. Marián Kravec

Acknowledgment

I would like to express my sincere gratitude to all those who supported me throughout the journey of completing this diploma thesis.

First and foremost, I am deeply thankful to my supervisor, MSc. František Dráček, for his invaluable guidance, encouragement, and insightful feedback during every stage of my research. His expertise and patience have been instrumental in shaping this work.

I would also like to thank the Faculty of Mathematics, Physics and Informatics at Comenius University for providing an inspiring academic environment and access to state-of-the-art research facilities. The faculty's commitment to excellence in mathematics, physics, and informatics has greatly enriched my academic experience.

Special thanks go to my colleagues and friends for their support, stimulating discussions, and for making this journey memorable.

Finally, I am profoundly grateful to my family for their unwavering love, patience, and motivation. Their belief in me has been my greatest source of strength.

Abstract

Currently, the healthcare sector in Slovakia is characterized by a low rate of utilization of available healthcare data. The aim of this work is to show that from existing data it is possible to predict the development of the patient's further health status, or to estimate the development of future costs in order to better plan the redistribution of finances within the sector.

Keywords: cost prediction, patient future, transformers

Abstrakt

V súčastnosti sa sektor zdravotníctva na Slovensku vyznačuje nízkou mierou využitia dostupných zdravotníckych dát. V rámci tejto prace je cieľom ukázať, že z existujúcich dát je možné predikovať vývoj ďalšieho zdravotného stavu pacienta, poprípade odhadnúť vývoj budúcich nákladov za účelom lepšieho plánovania prerozdelenia financií v rámci sektoru.

Kľúčové slová: predikcia ceny, budúcnosť pacienta, transformery

Contents

List of Figures

List of Tables

Terminology

Terms

Abbreviations

- **CPT** - Current Procedural Terminology.
- **EHR** - Electronic Health Records.
- **ICD-10-CM** - International Classification of Diseases, Tenth Revision, Clinical Modification.
- **MKCH-10** - International Classification of Diseases, Tenth Revision (Medzinárodná klasifikácia chorôb).
- **ATC** - Anatomical Therapeutic Chemical.
- **LLM** - Large language model.
- **LaBSE** - Language-agnostic BERT sentence embedding model.
- **BERT** - Bidirectional encoder representations from transformers.
- **KNN** - K-nearest neighbors algorithm.
- **FFNN** - Feed forward neural network.
- **MLP** - Mutlilayer perceptron.
- **MLM** - Masked Language Model.
- **NSP** - Next Sentence Prediction.
- **TLM** - Translation Language Model.
- **PCA** - Principal component analysis.
- **RNN** - Recurrent neural network
- **SRN** - Simple recurrent network
- **GRU** - Gated recurrent unit RNN
- **LSTM** - Long short-term memory RNN

Motivation

State governments and insurance companies collect and store vast amounts of medical data, including patient histories, treatment records, prescriptions, and billing information. This data can be used for various purposes, such as detecting fraudulent activities, tracking contagious disease outbreaks, or allocating future supplies of purchased medication.

Another interesting application is predicting a patient's future health, which allows for forecasts of potential diseases they may develop and the treatments they might require. Unfortunately, due to the large number of significant factors not captured in medical records, this task is almost impossible to accomplish accurately.

That's why, in this study, we focus on a simpler problem: predicting the expected cost for a patient in the next year. Specifically, we aim to estimate the total anticipated costs of medications and medical procedures provided to a patient over a one-year period. In general, our approach involves two main tasks. First, we embed patient records into numerical vectors. Second, we train a model to predict each patient's future costs based on their previous records.

In the very first chapter, we briefly introduce our goal, the challenges we encountered, and the methods we used. The second chapter provides a quick overview of studies that address similar problems. The third chapter is dedicated to introducing the data we used. In the fourth chapter, we present the models and techniques employed for embedding records and for the prediction task. The fifth chapter is a brief section on the technical aspects of software design, including the programming languages and libraries we used. After that, in the sixth chapter, we describe how we implemented solutions to our two main tasks. The seventh chapter focuses on preliminary research, specifically the testing of embedding methods and various model parameters. Finally, in the eighth chapter, we discuss the results of the final model.

Chapter 1

Introduction

In this chapter, we briefly introduce our goal, the challenges we encountered, and the methods we used.

The main goal of our study is to develop software capable of analyzing patients' historical records-specifically, the medications prescribed to them and the medical procedures they have undergone-in order to predict the cost category each patient will belong to in the following year. In other words, we aim to estimate the expected expenses for each patient over the next year based on their prior medical data.

To achieve the desired outcome, we faced three primary challenges. First, we needed to transform patients' historical records into numerical vectors suitable as input for machine learning models. Second, we had to simulate patients' potential futures by predicting both likely medication prescriptions and medical procedures they might undergo in the subsequent year. Finally, we required a method to calculate the expected costs of these projected treatments and aggregate them into an annual cost estimate for each patient.

For record transformation (referred to as embedding later in the study), we focused on embedding medications, diagnoses, and medical procedures. We employed one of two methods depending on whether the data contained structured codes. When structured codes were available, we split the code into its hierarchical components, embedded each part individually, and then combined the results to create a comprehensive representation. For data without structured codes or meaningful organization, we utilized a machine learning-based language model to generate embeddings directly from textual descriptions.

To generate future medical records, we evaluated several sequential data models, including LSTM (Long Short-Term Memory) networks and decoder-only Transformer architectures, as patients' future medications and procedures exhibit strong temporal dependencies on their historical records, with recent events carrying greater predictive weight.

To estimate the cost category of each medical record, we primarily utilized a Multi-layer Perceptron (MLP) model. This neural network takes the numerical representation of a record (generated through our embedding process) as input and predicts its likely cost category. While the MLP served as our core approach, we also explored alternative models such as Gradient Boosting and Ridge Regression for comparative analysis.

After resolving the three key challenges we integrated these components into a unified software solution. This final system sequentially processes patient histories through each stage to generate annual expense forecasts.

Chapter 2

Similar studies

A necessary prerequisite for predicting a patient's future is embedding patient records into numerical representations. This process is relatively straightforward for attributes that are already numeric, such as age, or for attributes with structured codes, like diagnoses. However, with medical procedures, a challenge arises: the codes associated with procedures lack a hierarchical structure. As a result, two similar procedures might have completely different codes. To address this, we needed a method to embed procedures so that similar procedures receive similar embeddings. One effective approach is to group medical procedures into clusters, assigning similar embeddings to procedures within the same cluster and dissimilar embeddings to those in different clusters.

Lorenzi et al. from Duke University in Durham developed a novel algorithm called Predictive Hierarchical Clustering [26]. This algorithm was designed for agglomerative clustering of surgical CPT codes. It uses a one-pass, bottom-up approach that utilizes EHR data-specifically, 317 predictors such as lab values and patient history, while excluding CPT information-from 3 723 252 patients and 3 132 CPT codes, where each patient has one main surgical CPT code. For each CPT code, they create a tree containing patients with the corresponding code. Afterwards, at each iteration, the algorithm considers merging all pairs of existing trees. To compare two trees, it uses two hypotheses: the first hypothesis states that the data in both trees are generated from the same model, while the second states that the data in each tree are generated from models with different parameters. The final value is a weighted average of the probabilities of these two hypotheses, considering the data in the trees, where the weight is the probability of the first hypothesis which formula is shown in Eq. 2.1.

$$p(D_k|T_k) = p(H_1^k)p(D_k|H_1^k) + (1 - p(H_1^k))p(D_i|T_i)p(D_j|T_j) \tag{2.1}$$

Where D_k is the set of data in the merged tree (formed by merging T_i and T_j), T_k is the merged tree, H_1^k is the first hypothesis, and D_i and D_j are the data in trees

T_i and T_j. During experimental testing, they compared the results of their approach to clustering CPT codes into 16 clinical groups. To evaluate performance, they used both their clustering method and clinical clustering to predict additional procedures for patients in a validation group, then computed the area under the receiver operating characteristic curve (AUROC) and the area under the precision-recall curve (AUROC). Their model achieved a few percentage points of improvement in these metrics compared to clinical clustering.

In the end, we decided not to use this approach in our work, as it cannot reliably distinguish whether two procedures are considered similar due to their actual similarity or simply because they frequently occur together as part of the determination or treatment of a specific diagnosis. This distinction is important, since procedures that only appear together may have vastly different costs associated with them, which could pose issues for the prediction model.

The main goal of our study is to predict the future cost for each patient, specifically how much money will be spent on drugs and procedures for a given individual.

One similar study in this regard is by Caballer-Tarazona et al. [12], which aimed to predict patients' future healthcare costs using an "Aggregated Clinical Risk Group 3" (ACRG3) variable derived from standardized Clinical Risk Groups (CRGs). The ACRG3 variable consists of two components: the first assigns patients to one of nine grouped CRGs, and the second categorizes them into one of six severity levels. The authors also tested whether adding demographic variables like age or sex improved model performance.

To address the high proportion of zero-cost observations, they developed a two-part model:

1. First part: A logit model estimating the probability of incurring non-zero costs.
2. Second part: Either a log-linear ordinary least squares (OLS) regression or a generalized linear model (GLM) with a log link to predict expected costs for patients with positive expenditures.

The final predicted cost was calculated as the product of the probability from the first part and the conditional expectation from the second part. Their models achieved adjusted R^2 values of 46.4%–49.4%, which they noted as comparable to results from studies using alternative patient classification systems.

This approach was ultimately not viable for us, as it relies on specific attributes such as CRG and patient severity level, which we did not have available.

Another study focused on future patient cost prediction is by Mohammad Amin Morid et al. [30], which compares multiple models for predicting a patient's future cost category using medical and pharmacy claims data.

For each patient, their dataset included 21 features, all related to the amounts spent on that patient. Specifically, these features included values such as the total historical cost for the patient, the total cost in the last six months, the number of months with costs above the patient's average, and the linear trend of cost over time.

The authors aimed to predict into which of five cost categories a patient would fall in the future. To achieve this, they compared several predictive models, including Linear Regression, Decision Tree, and Artificial Neural Network (ANN). For Linear Regression, they also tested versions with regularization, such as Lasso and Ridge models. In the case of Decision Tree models, they evaluated improved ensemble methods based on decision trees, including Random Forest, Bagging, and Gradient Boosting techniques.

In their testing, the most successful model turned out to be Gradient Boosting, which was able to predict the correct cost category in almost 95% of cases. However, for the highest cost bucket-which was expected to contain only 2% of the population-the model was successful in just 37.5% of cases. The second and third best models were the ANN and Ridge models, both of which achieved very similar results, with around 91.5% overall accuracy in category assignment and over 50% correct classification in each cost category. In conclusion, they determined that Gradient Boosting generally provides the best performance for cost prediction models, while ANN offers superior results specifically for higher-cost patients.

This study gave us hope that our data might contain the necessary information for accurate prediction, as most of the features in their study were computed using information about the cost of drugs or procedures-data that is present for each record in our dataset, which also contains much more detail about the causes of those costs.

Chapter 3

Medical data

Our models were developed for a database of health insurance claims maintained by the Slovak National Health Information Center (NCZI). We focused primarily on datasets related to drug prescriptions and medical procedures administered by outpatient healthcare providers.

To train and verify the model, we used completely artificial data with a structure matching that of the NCZI database. The data were generated in a way that simulates realistic patient records.

The data we have can be split into two categories: patient records and a mapping table.

3.1 Patients records

Patients records are split into two dataset:

- Records of medical procedures from outpatient healthcare
- Records of prescribed drugs

In total, we have data on 173 355 patients. Each patient has at least 70 records, resulting in a dataset with 133 679 705 records overall.

3.1.1 Records of medical procedures from ambulatory health care

Each row of this dataset corresponds to a single patient record, specifically representing one medical procedure performed on that patient. Each record consists of the following variables:

- Date of the procedure - date when procedure was performed.
- Code of the patient - identification code unique for the patient.
- Age of the patient - age of the patient at the time of procedure.
- Gender of the patient - gender patient had assigned in the system.
- Code of the diagnosis - identification code unique for the diagnosis for which procedure was prescribed.
- Code of the procedure - identification code unique for the medical procedure.
- Cost of the procedure - cost associated with performing of the procedure.

For our prediction model, we use most of this information. The date of the procedure, combined with the patient's age, generates timestamp data used to chronologically order all records for a patient and serves as one dimension of the record embedding. The patient identification code enables the aggregation of all records for a single individual. Diagnosis and procedure codes are mapped to their corresponding numerical vectors and embedded into the record's vector representation (see 5.1). Finally, cost is encoded into a cost category and serves as the target variable for training the record cost prediction models.

3.1.2 Records of prescribed medicines

Similarly to the dataset containing procedures, each row in this dataset corresponds to a single patient record, in this case representing one drug prescription for a specific patient. Each record consists of the following variables:

- Date of the prescription - date when drug was prescribed.
- Code of the patient - identification code unique for the patient.
- Age of the patient - age of the patient at the time of procedure.
- Gender of the patient - gender patient had assigned in the system.
- Code of the diagnosis - identification code unique for the diagnosis for which drug was prescribed.
- Code of the drug - identification code unique for the drug.
- Cost of the drug - cost associated with performing of the procedure.

The use of these variables is also similar to that for procedures, with the only difference being that, instead of encoding the procedure into the record embedding, we encode the drug.

3.2 Mappings

In addition to the patient records dataset, we utilize three mapping files to translate identification codes from the raw patient records into their corresponding embedding vectors. These mapping datasets are:

- Diagnosis mapping
- Drug mapping
- Medical procedure mapping

These datasets are publicly available dimension tables maintained and used by NCZI. In general, all three mapping tables have a comparable structure consisting of three main attributes. The first is an internal identification number used by NCZI to join these mappings to patient records. The second attribute is the official code; in the case of drugs and diagnoses, this is an internationally standardized structured code, while medical procedures use a code specific to Slovakia, which unfortunately lacks any meaningful structure. The final important part is the textual description.

We used these dimension tables to generate embedding vectors for drugs, diagnoses, and medical procedures. The process of this embedding is explained in Sec. 7.1.

Chapter 4

Theory - models

During our research, we utilized several different machine learning models. In this chapter, we introduce these models from a more theoretical point of view.

4.1 Multilayer perceptron

A multilayer perceptron is a feed-forward neural network, meaning that data flows in a single direction and neurons do not form cycles. This network consists of fully connected, sometimes called dense, layers with non-linear activation functions, as shown in Fig. 4.1. We can see that this model can be split into three parts: the input layer, which loads the data; the hidden layers, which extract the desired information using linear transformations and activation functions; and finally, the output layer, which applies a final linear transformation followed by an activation function to produce the output. Each layer can be described using the formula shown in Eq. 4.1, where h_i is the resulting vector of the i-th layer, $act()$ is a non-linear activation function, W_i is the weight matrix of the i-th layer, h_{i-1} is the resulting vector from the previous layer, and b_i is the bias vector.

$$h_i = act(W_i h_{i-1} + b_i), \tag{4.1}$$

4.2 Recurrent neural network

In general, a recurrent neural network is a type of neural network that, in some way, uses results from previous steps to improve its predictions. These models are used for ordered data, where each subsequent step depends on more than just the immediately preceding one.

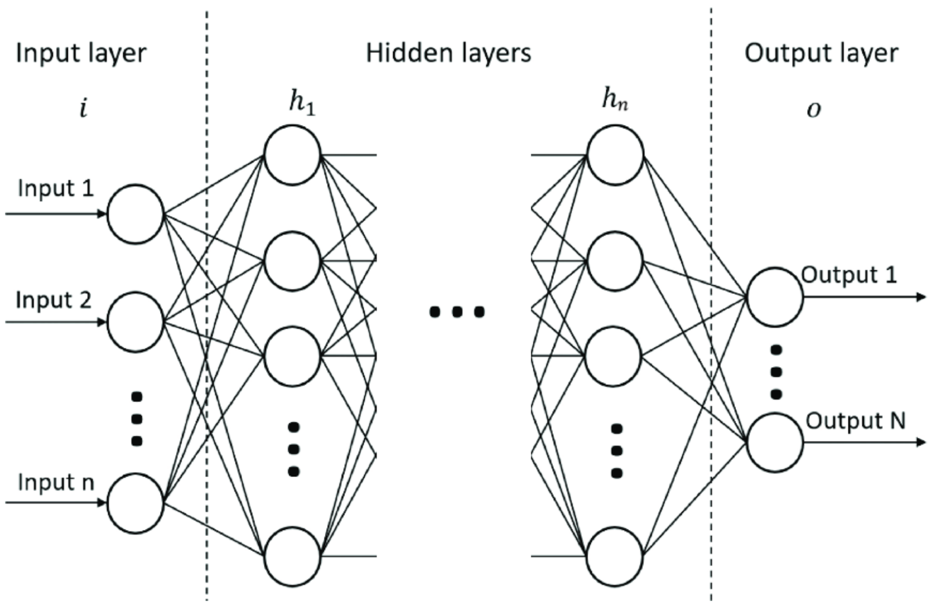


Figure 4.1: Architecture of multilayer perceptron [23].

4.2.1 Elman RNN

The Elman RNN, also known as a simple recurrent network (SRN), is a type of recurrent neural network that utilizes the results of the hidden layer before activation in the previous step as an additional input in the next one. We can see this architecture in Fig. 4.2, where on the left we observe how forward propagation occurs and on the right how it is unrolled over time. Here, the middle layer, which is the hidden layer of the model, receives two inputs: the input vector x_t, where t denotes the step (usually a time step), which is multiplied by a matrix of weights W_i, and the vector h_{t-1}, which is the result of this layer (also called the context vector) from the previous step, multiplied by a different matrix of weights denoted as W. These two results are summed together, and the result is passed to the activation layer, which generates the new vector h_t [20]. This entire process is summarized in Eq. 4.2, where b_i and b are bias vectors.

$$h_t = act(x_t W_i^T + b_i + h_{t-1} W^T + b). \tag{4.2}$$

Usually, this result is directly used as the output or passed through a single fully-connected feed-forward layer. In a multi-layered version of this network, the output of the first hidden layer is fed into the next hidden layer, along with the output of that specific layer from the previous step. Both inputs are multiplied by weight matrices specific to that layer, summed together, and then passed through an activation function.

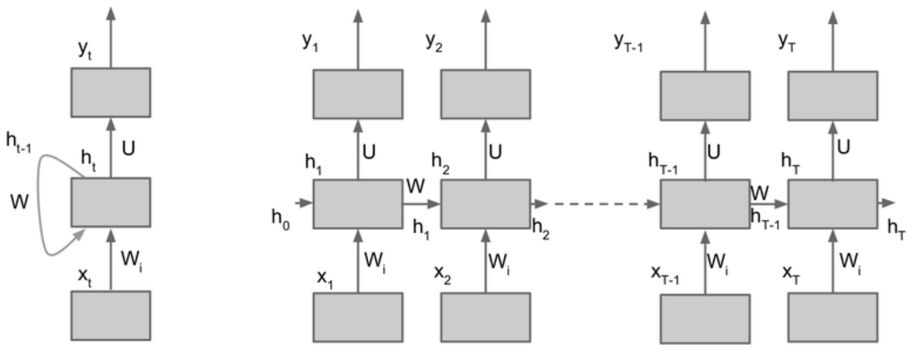


Figure 4.2: Architecture of Elman RNN [8].

4.2.2 Gated Recurrent Unit

A Gated Recurrent Unit (GRU) is a more complex RNN variant compared to the Elman RNN. Developed as a simplification of the even more intricate LSTM model, the GRU primarily consists of three interacting components: the reset gate, update gate, and candidate hidden state computation. These components collaboratively regulate information flow to produce the final prediction.

The structure of a GRU's hidden layer is illustrated in Fig. 4.3. In this diagram:

- Sigmoid: Represents a fully connected feed-forward layer with a sigmoid activation function.
- Tanh: Represents a fully connected feed-forward layer with a hyperbolic tangent activation function.

The reset gate, located on the left side of the diagram, uses the input and previous hidden state vectors to modify the previous hidden state vector, which is then fed into the candidate hidden state computation. This process helps capture short-term dependencies in time series by selectively removing information from the previous hidden state vector.

The candidate hidden state computation is similar to the Elman RNN but with two key differences: first, the inputted hidden layer is modified by the reset gate's result; second, the resulting vector from this part serves only as a candidate that undergoes further calculation. This candidate vector primarily contains current and short-term past information due to the specific vectors that form its input.

The update gate, similar to the reset gate, uses the concatenation of the input and previous hidden state vector as its input. However, its results are used to modify both the previous hidden state and the candidate hidden state, creating a final hidden state that is a weighted average of these two vectors, with weights derived from this gate. This architecture helps capture long-term dependencies in time series by reintroducing relevant information from the previous hidden state vector into the final hidden state, combining it with the candidate hidden state vector that contains mostly current and short-term information. The entire architecture is depicted in Fig. 4.3.

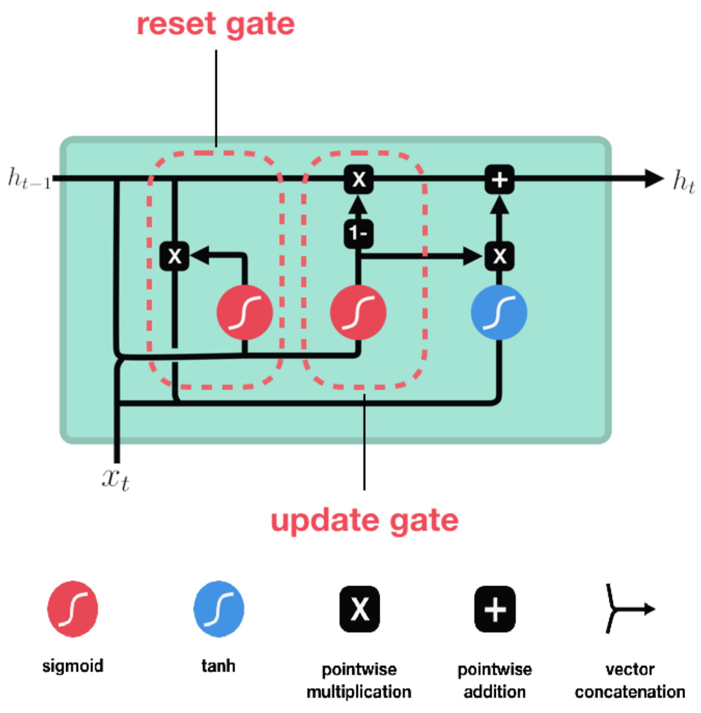


Figure 4.3: Architecture of GRU hidden layer.

The multi-layered version of this architecture is achieved by stacking hidden layers, where the hidden state vector of one layer becomes the input vector for the next.

4.2.3 Long Short-Term Memory

As mentioned earlier, the LSTM is a more complex predecessor of the GRU. Developed to address the vanishing gradient problem that limits long-term memory in models like the Elman RNN, the LSTM introduces a memory vector, often called a memory cell, designed to maintain information over longer periods. As shown in Fig. 4.4, this model consists of three gates and a candidate memory computation. The legend in this diagram has the same meaning as in the GRU architecture diagram.

All three gates and the candidate memory computation share the same input, which consists of the input vector and the previous hidden state vector. In each case, this input passes through a fully connected feed-forward layer and then into an activation function. For the gates, this activation function is the sigmoid function, which bounds all values within the range (0,1), while the candidate memory calculation uses the hyperbolic tangent as its activation function.

The forget gate is responsible for removing unnecessary information from the memory vector by performing an element-wise multiplication of its result with the previous memory vector.

The input gate, together with the candidate memory vector computation, is designed to introduce new information into the memory cell after adjustments made by the forget gate. First, the model computes the candidate memory vector; this vector is then adjusted by the input gate's result and added to the modified previous memory vector, resulting in a new memory vector.

Finally, the output gate determines how much each part of the memory should contribute to the new hidden state vector. This process is illustrated in Fig. 4.4.

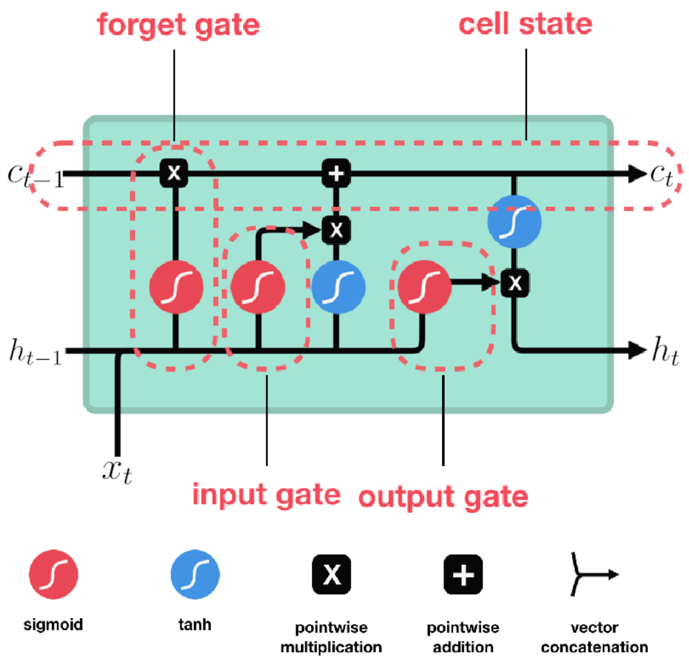


Figure 4.4: Architecture of LSTM hidden layer.

4.3 Transformer

The Transformer is a deep learning model architecture introduced in 2017 by Vaswani et al. [40] that is especially good at handling sequences, such as text. This architecture consists of an encoder and a decoder, as shown in Fig. 4.5. The encoder in this model is meant to create a contextualized representation of the input, while the decoder takes this contextualized representation and combines it with previous outputs to predict the next output. In addition to these two main blocks, this model usually also contains a tokenizer, an embedding layer, and positional encoding to prepare the input data, as well as a fully connected feed-forward layer with a softmax activation function to turn the result of the decoder into a probability distribution over possible tokens.

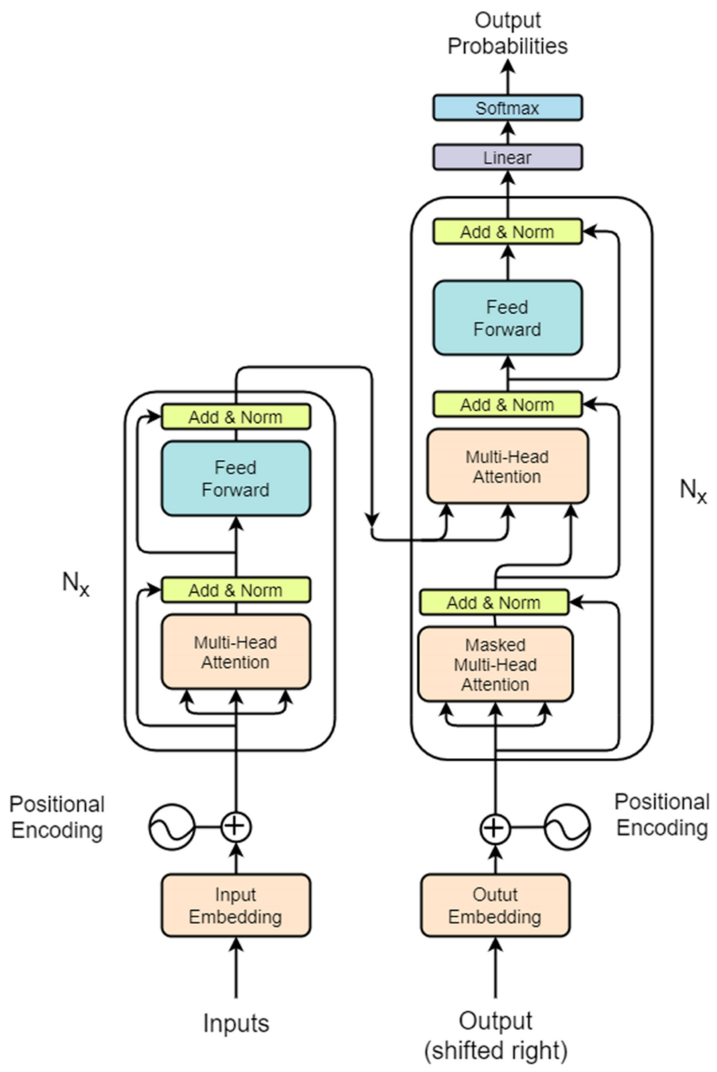


Figure 4.5: Architecture of one encoder-decoder block in transformer model from original "Attention is all you need" paper [40].

4.3.1 Tokenizer, Embedding layer and Positional encoding

The first part of the Transformer model is usually input preparation, which may vary depending on the type of data used. This part consists of three components:

- Tokenizer: This layer splits the input into tokens. For example, if the input is textual, tokens might be words or syllables.
- Embedding layer: The task of this layer is to transform each input token into a numerical vector of the same length.
- Positional encoding: The final preparation layer adds a vector to each token's embedding, encoding its position relative to other tokens.

In most cases, the Tokenizer and Embedding layer are trained separately and do not change during training of the Transformer, while the Positional Encoding is trained alongside the rest of the Transformer.

4.3.2 Encoder

Encoder architecture consists of multiple attention blocks, where each block consists of multi-headed self-attention and a multi-layered feed-forward neural network. After each of these steps, the embeddings from the step input are added to the output, and the result is layer-normalized. Adding the step input embedding creates residual paths that help mainly with the vanishing gradient problem, while layer normalization ensures that the results neither explode nor vanish, and also brings a bit of additional non-linearity to the model. First, input embeddings are split into parts, each of which goes into a separate self-attention head.

The architecture of self-attention is shown in Fig. 4.6, where we can see that each input token is multiplied by key, query, and value matrices to obtain their key, query, and value vectors. After that, each query is multiplied with each key to create the attention matrix, which encodes how much each token influences the others. This matrix then goes into a column-wise softmax function to normalize it, and finally, it is multiplied by the matrix of value vectors to create new embeddings of the tokens that should contain not only the original information but also the influence information.

The results of each self-attention head are then concatenated back into new embeddings that have the same dimensions as the original ones. After adding the residual connection and normalization, the results go into a multi-layered feed-forward neural network to further extract information from the embeddings.

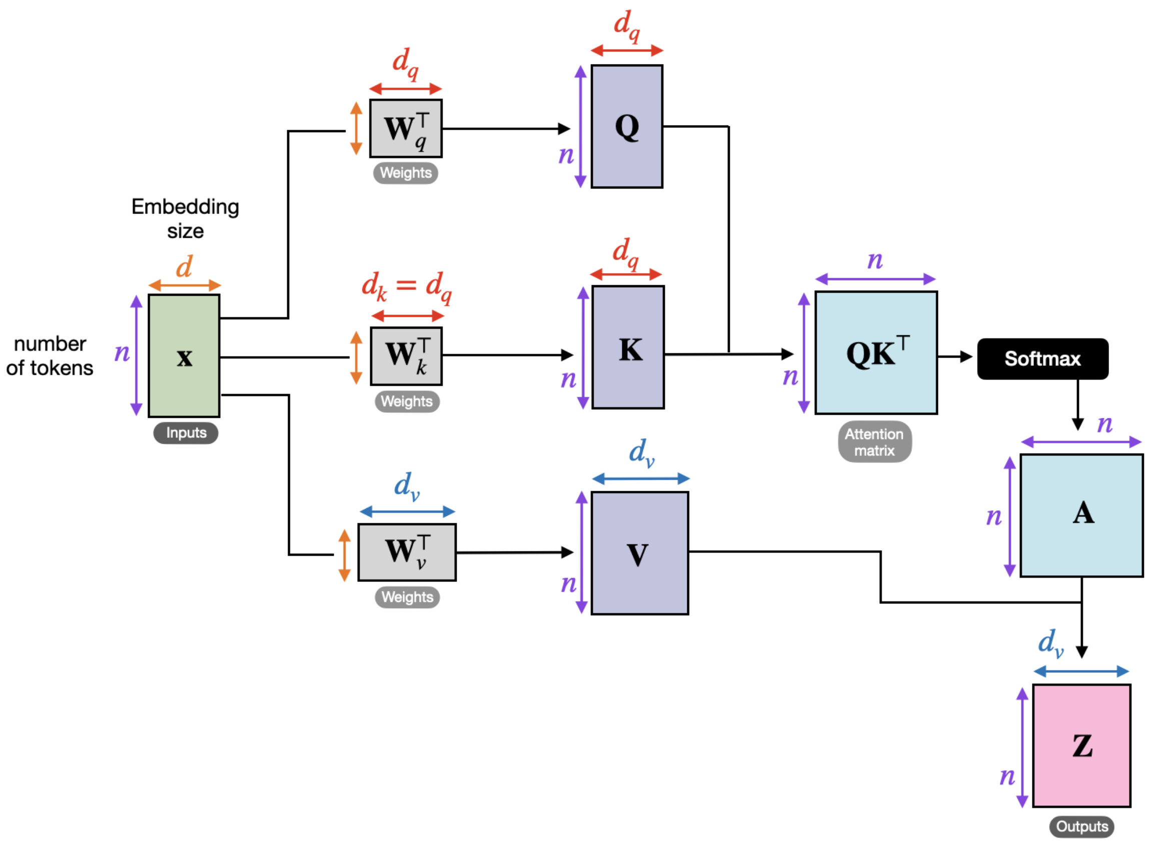


Figure 4.6: Architecture of Self-attention mechanism.

4.3.3 Decoder

The architecture of the decoder, similarly to the encoder, consists of multiple attention blocks; however, in this case, each attention block consists of three parts instead of two.

The first part is masked multi-headed self-attention, which is similar to unmasked self-attention, with the only difference being the application of masking to the attention matrix before the softmax function. This ensures that words at certain positions do not affect words at other specific positions. The decoder uses what is called causal or look-ahead masking, which prevents future tokens from affecting past ones. In our case, this guarantees that a record cannot be influenced by records that occur later, that is, in the future from its perspective.

The second part is multi-headed cross-attention, which takes two lists of token embeddings and computes the effect of tokens in the first list on tokens in the second list. In this case, the key and value vectors are computed from the contextualized representation produced by the encoder, while the query vectors are computed from the results of self-attention in the decoder. Other than this, the remaining architecture is the same as unmasked self-attention.

The final part of the decoder attention block is a multi-layered feed-forward neural network that further extracts information from the embeddings.

4.3.4 Fully-connected Feed-forward layer

After that, the results of the decoder pass into the final fully-connected feed-forward layer, which changes the dimensionality of the decoder output from the embedding size to the vocabulary size and applies a softmax activation to the result. This setup is designed to produce a probability distribution over all possible tokens for the last token.

In our case, we modified this final step. We encountered a problem where most records (our tokens) were unique, creating an extremely large vocabulary. This issue arose partially due to the many possible combinations of diseases, drugs, and medical procedures encoded in each record, and partially due to timestamps adding uniqueness, as it is unlikely for two patients to receive the same drug for the same disease at the same age.

To address this, we removed the softmax activation and altered the fully-connected feed-forward layer so that the result maintains the embedding dimension. We then added a function that splits the result, finds the closest embedding for each part, and concatenates these embeddings together, yielding a specific new embedding prediction instead of a probability distribution.

4.3.5 Decoder-only Transformer

The Decoder-only version of the Transformer model is a simplified variant that entirely excludes the Encoder part and removes the Cross-Attention mechanism from the Decoder. This model is typically used when generating subsequent tokens without relying on a stable contextual input that would normally be processed by the Encoder.

This simplification makes the architecture more akin to a standard RNN in terms of input-output behavior, where the model generates sequences step-by-step without explicit cross-contextual dependencies.

4.4 Language-agnostic BERT Sentence Embedding

The Language-agnostic BERT Sentence Embedding model, also known as LaBSE, is a model trained with the main goal of generating similar representations for pairs of sentences that have the same meaning and are only translations of each other in two different languages [4].

The architecture of the LaBSE model consists of four parts [7]:

1. Encoder-only transformer (BERT model)
2. Pooling layer
3. Dense layer
4. Normalization layer

4.4.1 Encoder-only Transformer (BERT model)

The first and most important part of the LaBSE model is the transformer, a deep learning architecture. More specifically, LaBSE uses BERT (Bidirectional Encoder Representations from Transformers), an encoder-only transformer architecture. This means the model lacks the decoder found in the standard Transformer (typically used for prediction tasks), allowing BERT to focus solely on extracting contextual information from input text.

The architecture of the standard BERT model includes:

1. Tokenizer layer
2. Embedding layer
3. Encoder
4. Task layer

Tokenizer layer

The first layer is the tokenizer, which takes the input text and splits it into tokens. In the case of the BERT model, this is called the PieceWise tokenizer, which splits text into subwords, something similar to syllables. The PieceWise tokenizer has advantages compared to other tokenizers that use either words or characters. Compared to character-wise tokenization, subwords contain more information than individual characters. Compared to word tokenization, there are far fewer subwords than words, and subwords are more similar across multiple languages, resulting in a much smaller vocabulary. This is especially important for multilingual models. After splitting, this layer assigns an integer number to each unique token. The LaBSE model's vocabulary distinguishes around 500,000 different tokens.

Embedding layer

After that comes the embedding layer, which assigns a real-number vector to each token. Specifically, the BERT model computes three distinct embeddings, sums them, and normalizes the result to produce the final embedding:

- Token type embedding: The base embedding where each token in the vocabulary is assigned a unique vector.
- Positional embedding: Encodes the token's position within the sequence, providing contextual information about its location.
- Segment type embedding: Indicates which segment (typically a sentence) the token belongs to, crucial for processing multi-sentence input.

Encoder

The third and most important layer is the encoder. This is the layer in which contextual information is mined from the text. The architecture of this layer is the same as the encoder described in Sec. 4.3.2. In the BERT variant used by LaBSE, the encoder contains 12 attention blocks.

Task layer

The training process of the BERT model usually consists of two tasks on which the model is trained at the same time. The first is Masked Language Modeling (MLM), where 15% of input tokens are masked, meaning they are either replaced by a mask placeholder or by a random different token. The masked input then goes into the model, and the resulting tokens at the positions of the masked tokens in the input are compared to the correct tokens before masking. This creates an error that is back-propagated through the model, updating its parameters [18]. The other task usually used is called Next Sentence Prediction (NSP). In this task, the model receives input that starts with a special classification token and then two spans of text separated by a special separator token. The task of the model is to determine whether these two spans of text can appear one after another, or more precisely, whether they appeared consecutively in the training corpus. This information is encoded in the first token of the result using two special tokens, either "is next" or "not next." Similarly, the difference between the expected and resulting first token creates an error that is back-propagated through the model [39].

However, in some BERT-based models like LaBSE, the second task is replaced with Translation Language Modeling (TLM). This task is an extension of MLM, in which the model receives two concatenated sentences instead of one, where the second sentence is a translation of the first in another language. The rest of the task is the same as in MLM: the whole input is masked, and the model is tasked with predicting the masked tokens [17].

The task layer is used primarily only during pre-training and is omitted when the model is used for a different task, as many use cases do not need tokens in the results but instead use embeddings from the encoding layer as a form of text encoding, which is then used in task-specific layers for fine-tuning.

4.4.2 Pooling layer

After the BERT model returns embeddings for all input tokens, these embeddings need to be aggregated into a single vector that represents the embedding of the entire input. This aggregation is typically performed using a pooling layer. In the case of LaBSE, this pooling is done simply by taking the embedding of the first token, which is the embedding of the special classification token added to the beginning of the BERT input.

4.4.3 Dense layer

The next layer is a standard feed-forward dense layer using a hyperbolic tangent activation function. The number of input and output neurons is the same, and the layer includes an additional bias neuron.

4.4.4 Normalization layer

The final layer is normalization, whose task is to normalize the resulting vector by dividing it by its L_2 norm, ensuring the final vector has an L_2 norm equal to one.

4.5 Word2vec model

Word2vec is a neural network-based method for generating word embeddings, which are dense vector representations of words that capture their semantic meaning and relationships. In other words, properties like the distance between two embeddings contain underlying information about those words, such as their similarity. There are two main approaches to implementing Word2vec:

- Continuous bag-of-words (CBOW)
- Skip-gram

4.5.1 CBOW approach

A model using the CBOW approach receives a sequence of words called the context, with one word missing, and tries to predict the missing target word as output. The model initially assigns one-hot encoding to each word in its dictionary. During training, each word from the context is first converted into its one-hot encoded embedding, which is then multiplied by a weight matrix to obtain its lower-dimensional dense embedding. The dense embeddings of all words in the context are then averaged, and the resulting embedding goes into a hidden layer, which transforms the vector back into the dimension of the vocabulary. Finally, a softmax function is applied to get the probability of each word in the vocabulary being predicted as the missing word. We can see this architecture in Fig. 4.7. Training is usually done using a fixed context window moving along the training text.

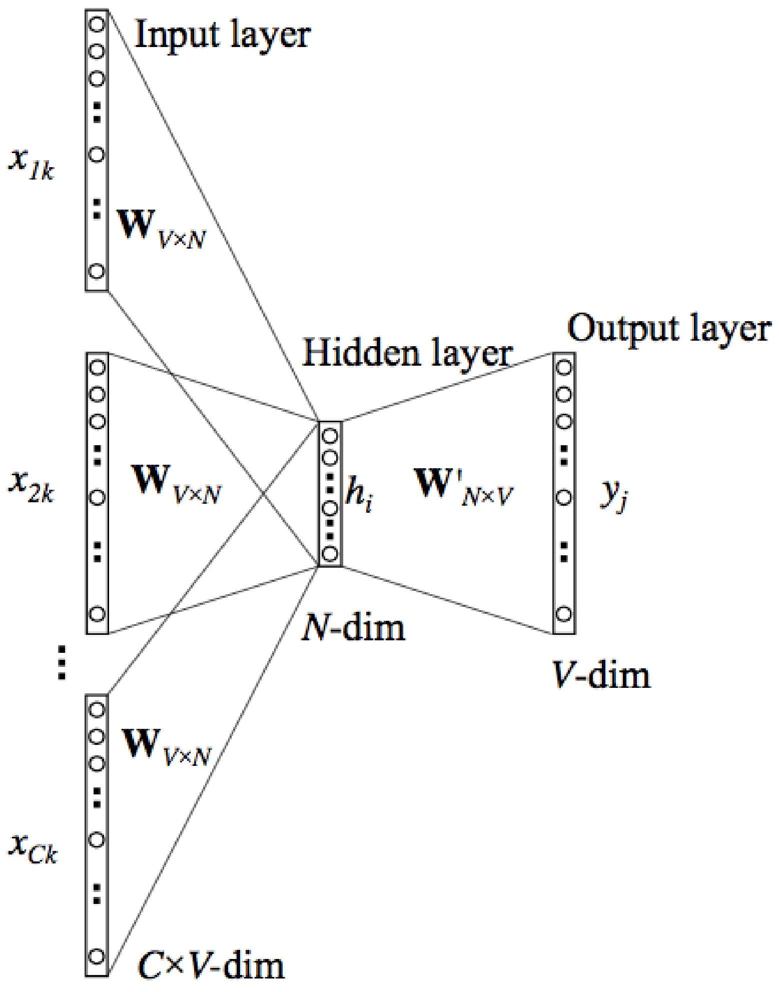


Figure 4.7: Architecture of NN to train CBOW implementation of Word2vec model [25].

4.5.2 Skip-gram approach

The Skip-gram approach works in the opposite way, the model receives a target word and tries to predict the surrounding context words. Similar to CBOW, the input word is first one-hot encoded and then multiplied by a weight matrix to transform it into a dense embedding. This embedding is passed to the next layer, which transforms it back into the vocabulary dimension. A softmax function is then applied to generate a probability distribution over potential context words.

The Skip-gram's loss function is the sum of the negative log-likelihoods of all context words. This architecture is illustrated in Fig. 4.8.

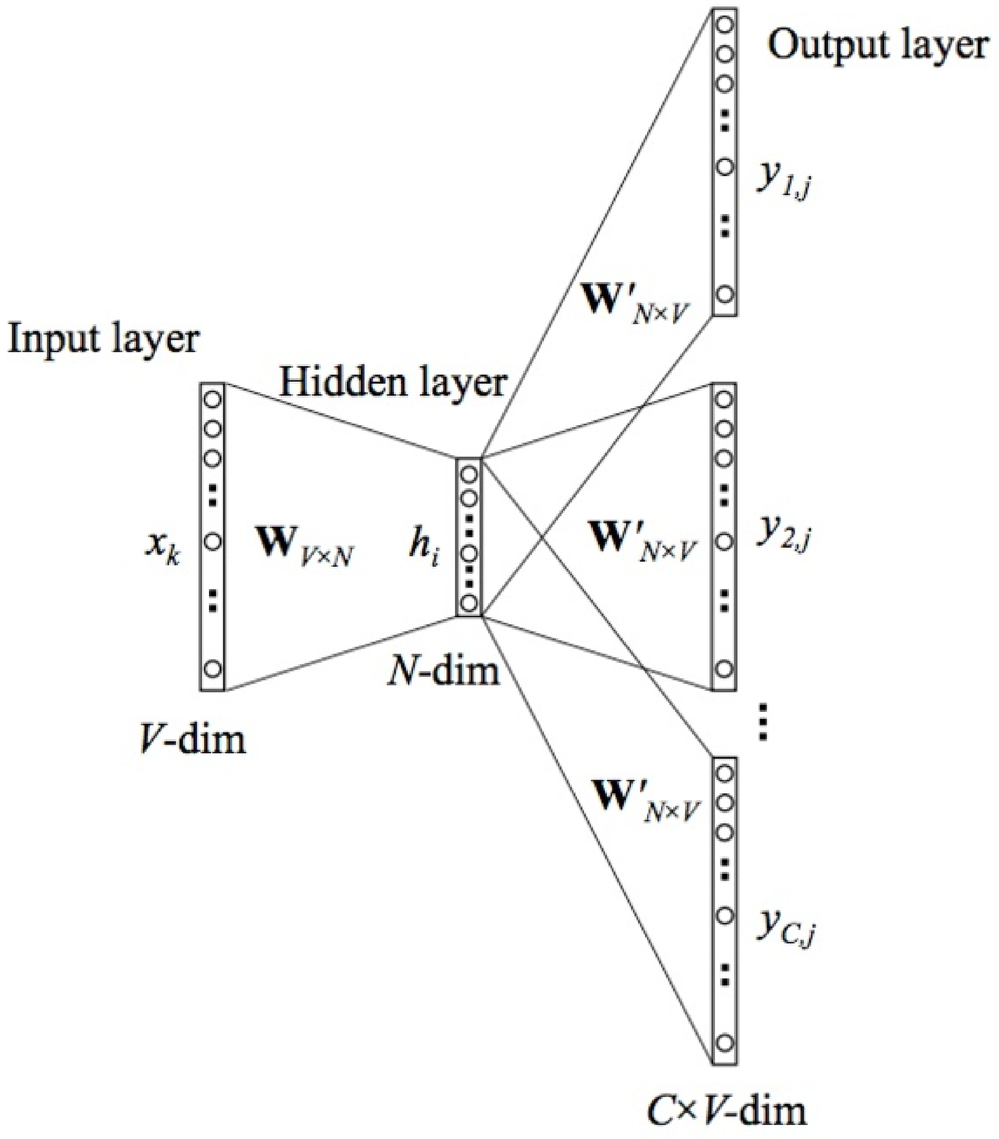


Figure 4.8: Architecture of NN to train Skip-gram implementation of Word2vec model [28]

Chapter 5

Proposed Methods

The task of predicting the future cost of a patient can be split into multiple sub-tasks, which follow each other. The sub-tasks are:

1. Embed the patient's history into numerical vectors
2. Compute the expected number of records the patient will have in the next year
3. Predict future records for the patient
4. Predict the cost of each future record
5. Compute the total cost of the patient for the next year

5.1 Embedding of Patient

The first sub-task in predicting a patient's future costs is to embed each patient record into a numerical vector that can be interpreted by a neural network. Our main goal was to create embeddings that retain similarity information-meaning that records with similar diagnoses, drugs, and medical procedures receive similar embeddings. We define similarity using the Euclidean distance between embedding vectors. Preserving similarity is important because it simplifies the prediction task: the model only needs to predict a closely related record, rather than the exact one.

For each patient record, we embed four distinct attributes. The first, and simplest to implement, is the timestamp, which is calculated using numerical and date values. The remaining three attributes, which are diagnosis, medical procedure, and prescribed drug, are more complex to embed.

5.1.1 Timestamp

The attribute we refer to as the timestamp of a patient's record can be more precisely described as an approximation of the patient's age at the time of either a medical procedure or drug prescription. This timestamp is calculated using two available pieces of information: the patient's age in years and the date of the record. The specific procedures we use to compute this timestamp are detailed in section 7.1.1. In general, we identify the first record, compute its timestamp based on the patient's age as an approximate age in days at that point, and then calculate each subsequent timestamp as the timestamp of the first record plus the difference in record dates. In this way, each timestamp provides an approximation of the patient's age while also conveying the order of records and the relative timeframe between any two records.

5.1.2 Diagnosis embedding

The base diagnosis information we embedded was the ICD-10-CM code for the disease. ICD-10-CM stands for "International Classification of Diseases, Tenth Revision, Clinical Modification," which is used to code and classify medical diagnoses [13]. In Slovakia, this classification system is known as MKCH-10-SK (Medzinárodná klasifikácia chorôb) [6].

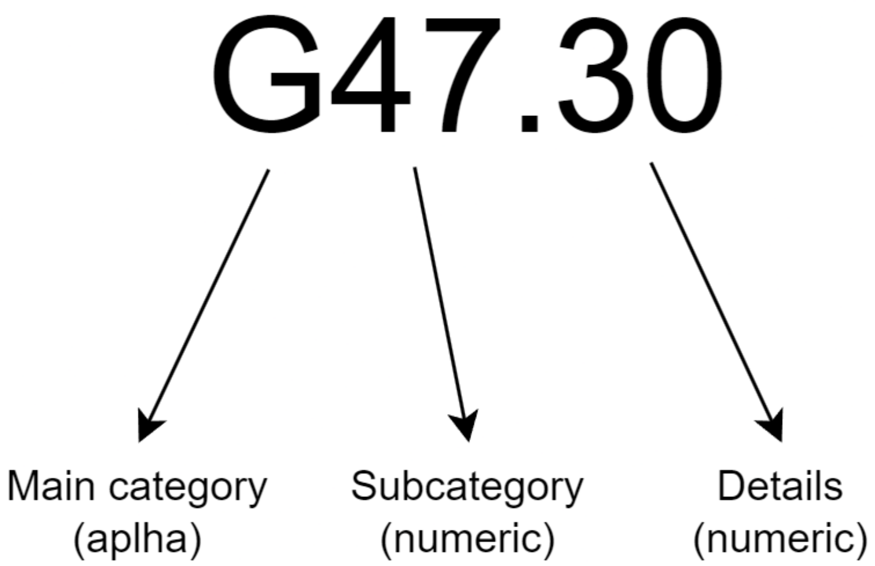


Figure 5.1: Structure of MKCH-10 code.

This code consists of three parts, as shown in Fig. 5.1. The first part is a letter that encodes the main categories of diseases, also known as chapters. For example, codes starting with G represent diseases of the nervous system. Following this, there are two numeric characters that further specify the subcategory of the disease, such as codes from G40 to G47, which are episodic and paroxysmal disorders, with G47 specifically being sleep disorders. We observe that episodic and paroxysmal disorders only extend up to G47, meaning that theoretically, subgroups G48 and G49 could exist with a 4 in the second position but would not belong to the same G4 subcategory as G40 or G47 . Fortunately, this is not the case; when a higher-level subcategory (like G4) does not have 10 lower-level subcategories (like G47), these subcategories do not exist at all. If there are more than 10 lower-level subcategories, they are assigned multiple consecutive higher-level subcategories, such as disorders of other endocrine glands, which span from E20 to E35. The code then contains a dot, after which there are characters that further describe the details of the disease, such as etiology, anatomic site, and severity. According to the official documentation, ICD-10-CM codes can be up to 7 characters long, meaning that after the first three characters specifying the category, there can be up to 4 additional alphanumeric characters to further specify the disease [14]. However, the version used in Slovak healthcare database, MKCH-10, contains only up to two numeric characters to specify the disease [38]. These details are organized such that the first position conveys higher-level information than the second; for example, G47.3 is sleep apnea, and G47.30 is primary central sleep apnea.

To embed this code, we first split it into its three parts and separately embed each part. We then concatenated the embeddings of each part to obtain the final disease embedding. For the main category, we generated random vectors from a uniform distribution for each letter in the English alphabet (representing all main categories). We chose random vectors to ensure relatively similar distances between any two main categories, as there are no inherent relationships among these categories. We also considered using one-hot encoding, which would make the distance between every pair of categories identical, but opted against it because this approach would constrain the vector length.

In embedding the second part, which is the subcategory, we used a different approach. For each number between 00 and 99 (all possible values of this part), we assigned a linearly spaced number within a chosen interval. This value was then repeated multiple times to create a vector. This repetition was implemented to encode the importance of this part of the code. This approach has both advantages and disadvantages. The advantage is that we can be sure that closely related disease subgroups, like G46 and G47, would receive similar embeddings since their subgroup codes are close on the number line. However, there are also two disadvantages. The first is that G49 and G50 would be similarly close as G46 and G47, but fortunately, in most cases, either the X9 code does not exist at all, creating a gap, or if the X9 code exists it belongs to a category that goes past X as a higher-level subgroup. Another disadvantage is that the distance between two higher-level subgroups can vary quite dramatically, even though in reality there might not be a reason for such a difference. For example, using this approach, the higher-level subgroup G40–G47 is much closer to subgroup G50–G59 than to G80–G83.

Finally, to embed details, we used the same approach as for subgroups, since these codes function similarly. The only difference was that not all codes included second-level details. In such cases, we added "5" as a proxy value to minimize the average distance from all potential codes sharing the same first-level detail code (with second-level information) while maximizing the average distance to codes with different first-level details.

Once all parts were embedded, we created the final embedding by concatenating them. To encode the importance of each part, we assigned different vector lengths. This works because each value in the vector has the same mean and variance. We encoded the importance hierarchically: the main category received the longest embedding, subgroups a medium length, and details the shortest embedding.

5.1.3 Drug embedding

Similarly to diagnosis, to embed drug information we used the international code associated with each drug. In the case of drugs, this was the Anatomical Therapeutic Chemical (ATC) classification system. Like the MKCH-10 code, the ATC code can be split into multiple parts, with each subsequent part providing more detailed information. The ATC code consists of five parts or levels.

Structure of the ATC code:

- First level: A single letter representing one of 14 main anatomical or pharmacological groups (e.g., "C" for cardiovascular system). These groups are shown in Fig. 5.2.
- Second level: A two-digit number specifying a pharmacological or therapeutic subgroup (e.g., "C03" for diuretics).
- Third and fourth levels: Each uses a single letter to further classify pharmacological, therapeutic, or chemical subgroups (e.g., "C03C" for high-ceiling diuretics and "C03CA" for sulfonamide derivatives).
- Fifth level: A two-digit number identifying the specific chemical substance (e.g., "C03CA01" for furosemide).

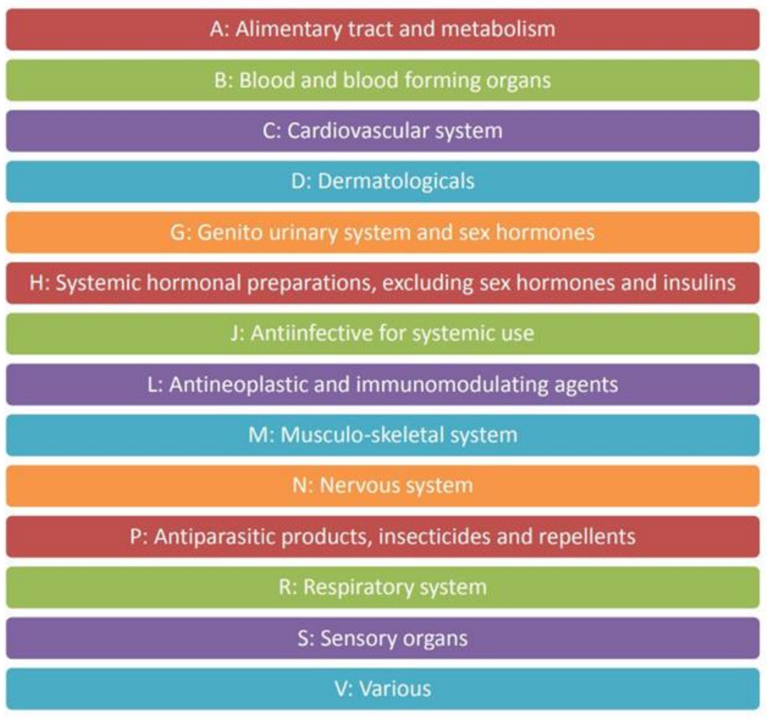

A: Alimentary tract and metabolism
B: Blood and blood forming organs
C: Cardiovascular system
D: Dermatologicals
G: Genito urinary system and sex hormones
H: Systemic hormonal preparations, excluding sex hormones and insulins
J: Antiinfective for systemic use
L: Antineoplastic and immunomodulating agents
M: Musculo-skeletal system
N: Nervous system
P: Antiparasitic products, insecticides and repellents
R: Respiratory system
S: Sensory organs
V: Various

Figure 5.2: Fourteen main anatomical or pharmacological groups and their corresponding first level ATC code [1].

Embedding was performed in a manner very similar to the diagnosis embedding: each level was embedded separately, and the final embedding was created by concatenating them. In this case, each level was embedded using a random vector drawn from a uniform distribution. We chose random vectors because none of the ATC levels contain internal subgroupings analogous to the hierarchical subgroups seen in diagnosis codes (see 5.1.2 for subgroup code details). To encode the importance of each level, we again varied the lengths of the random vectors. This approach ensures that the similarity between codes depends more on matches in the lower, more important levels than in the higher ones.

5.1.4 Medical procedure embedding

The final component to embed was medical procedures. In this case, no structured code is implemented within the Slovak healthcare system. To address this, we embedded textual descriptions of procedures using two approaches:

- Multilingual Large Language Model (LLM): We selected the LaBSE model (section 4.4), trained on multiple languages including Slovak. LaBSE generates similar embeddings for semantically equivalent sentences across languages, which aligns with medical terminology often being international.

- Slovak-specific Word2Vec: A Word2Vec model (section 4.5) trained exclusively on Slovak textual data.

For both methods, we reduced the dimensionality of the resulting embeddings using Principal Component Analysis (PCA) to eliminate low-variance dimensions encoding minimal information.

To construct the final record embedding, we concatenated all four components (timestamp, diagnosis, medical procedure, and prescribed drug). Since timestamp and diagnosis are always available, we substituted missing drug or procedure data with zero vectors of matching length. This neutral substitution preserves compatibility between datasets while maintaining centered embeddings.

5.2 Prediction of future number of records

Our task is to determine how much a patient will cost in the future, specifically in the next year. Since our approach predicts future records, assigns costs to them, and sums these into a total cost, we need to estimate how many records to generate to simulate the next year. For this task, we tried two different approaches.

The first approach was to predict the approximate number of records using linear regression, which takes the counts of records from previous years and predicts the next year's count. The second approach used a stopping criterion based on timestamp information available in each input and generated record. This method stops generating records once the difference between the last timestamp from the patient's data and the last generated timestamp surpasses a one-year threshold.

Each method has its own disadvantages. In the case of linear regression, the number of records per year can vary significantly. Even if we expect an increase [10], regression might fail to capture this trend, especially if the patient's data includes only a few years of historical records. A potential issue with the second approach is its reliance on how well the model learned that timestamps should always increase.

5.3 Future record prediction

This task is both the most important and the most challenging to train. Our goal is to develop a model capable of predicting a patient's potential next record based on their previous ones. Generally, this task is nearly impossible with the amount of information available, as numerous factors influence whether, when, and what new disease a patient might contract, how their current state will evolve, and what specific actions a doctor will take.

Fortunately, our ultimate goal is not to predict a patient's specific future but to estimate the likely total cost of their future records. We anticipate that even if we cannot predict exact outcomes, we can still estimate the overall cost.

To achieve this prediction, we experimented with multiple models. The first three models we tested were Recurrent Neural Networks (RNNs): multi-layer Elman RNN, multi-layer Gated Recurrent Unit (GRU) RNN, and multi-layer Long Short-Term Memory (LSTM) RNN. The architecture of these models is detailed in Sec. 4.2. The last model we tried was a Transformer, specifically a Decoder-only Transformer model, explained in Sec. 4.3.

5.4 Record Cost Prediction

The next step in total cost prediction involves estimating the cost of individual records. This is necessary because the future records we predict lack cost information, requiring an additional model to determine it. We adopted this approach to leverage the entire record for prediction, rather than relying solely on a single cost dimension that could have been added. For this task, we selected a standard Multilayer Perceptron (MLP), a multi-layered, fully-connected feed-forward neural network, whose architecture is detailed in Sec. 4.1.

We also trained a Gradient Boosting model and a Ridge regression model for comparative analysis. These were chosen to represent two distinct model families: decision trees (Gradient Boosting) and linear regression (Ridge). We selected these specific representatives because they employ more sophisticated techniques than base models in their respective groups and have demonstrated performance comparable to, if not better than, artificial neural networks in similar studies, such as the work by Mohammad Amin Morid et al. [30] (discussed in Chap. 2).

5.5 Prediction of future cost of patient

This last task is our main goal: to predict the cost of a patient in the next year. To achieve this, we utilize the results from all our previous tasks. First, we embed the patient's records. Then, we predict the records that the patient could receive in the next year. For each predicted record, we predict its cost category. Finally, all costs are summed and transformed into a patient cost category.

Chapter 6

Software Design

This chapter is dedicated to introducing the software used to create, train, validate, and use the machine learning model described in this thesis. The entire code was written in Python, more specifically in `Python 3.11.4`. We chose this language for its ease of use and wide selection of libraries for data processing and machine learning. All scripts are available in the GitHub repository at `https://github.com/MarianK-py/diploma_thesis_code`.

The code can be divided into three parts:

1. Embedding – code to create embedding mapping files
2. Model training and validation – code to set up, train, validate, and save prediction models; this needs to be run once
3. Predictor – code to load trained models and predict the future cost of inputted patients

The code for model training and validation, and the code for prediction, use the same technologies, which is why they are described in a single section. Now we will introduce the libraries and pre-trained models used in our code: first those used in general, and then those specific to each part mentioned above.

6.1 General

Some of the libraries were used in all parts of the code to maintain coherence in the technologies used.

In this category belongs the `Pandas 2.2.1` library, a fast, powerful, flexible, and easy-to-use open-source data analysis and manipulation tool [33], which was used to load, manipulate, and save all datasets. Another is the `Numpy 1.26.4` library, an open-source project that enables numerical computing with Python [22]. We used it for computations such as random number generation, calculation of mean and standard deviation for data normalization, and many other tasks.

6.2 Embedding

For embedding, we used a couple of additional libraries since we utilized several algorithms and pre-trained models. More specifically, the libraries and specific algorithms and models we used are the following:

- `Scikit-learn 1.2.2` – simple and efficient tools for predictive data analysis [35]. We specifically used the function to compute PCA in order to decrease the dimensionality of medical procedure embeddings, and the K-means clustering function to check whether the embedding has the desired property.
- `NLTK 3.8.1` – this abbreviation stands for Natural Language Toolkit, it's a library for building Python programs to work with human language data [11]. In our case, we used the tokenizer function to split descriptions of medical procedures into tokens, in our case, words.
- `Simplemma 1.1.2` – provides a simple and multilingual approach to finding base forms or lemmata [9]. We used it to lemmatize our tokenized text since the lemmatizer provided by this library includes Slovak and Czech languages.
- `SentenceTransformer 2.2.2` – a go-to Python module for accessing, using, and training state-of-the-art text and image embedding models [37], which allowed us to easily load the LaBSE model from Hugging Face.
- `Gensim 4.3.3` – a library for topic modelling, document indexing, and similarity retrieval with large corpora [36], which we used to load the Word2vec model trained specifically for the Slovak language.

6.3 Model training, validation and prediction

For training, validation, and using models for predictions, we primarily used `PyTorch 2.6.0`, which is an optimized tensor library for deep learning using GPUs and CPUs [34]. This library provided us with all the required components, such as linear, non-linear, and specialized layers, to assemble both simpler neural networks like the MLP (used for prediction of the cost category of a record) and more complex neural networks like LSTM or Transformer (used to predict future records). In addition to the building blocks for the networks themselves, PyTorch also offers other components needed for model training, such as optimizers and loss functions, with the possibility to modify them for our specific requirements, as we did for the loss function used in future record prediction.

Another library used here was `Scikit-learn 1.2.2`, from which we used the linear regression model. We considered this as one possible way to determine how many future records we should generate for a patient to estimate their expected amount for the next year.

Chapter 7

Implementation

This chapter focuses on how we implemented solutions for our problems proposed in Chap. 5, including details such as the dimensions assigned to each part of the code embedding and which hyperparameters we attempted to optimize.

7.1 Embedding of Patient

The first sub-task in predicting a patient's future costs is to embed each patient record into a numerical vector that can be interpreted by a neural network. For each patient record, four types of information are embedded. The first and easiest to implement is the timestamp, which is computed using numerical and date values. The other three, which are more complex, are the diagnosis, medical procedure, and prescribed drug.

7.1.1 Timestamp

To compute the timestamp, we first gathered all records for a single patient and identified the one with the earliest date. This record served as a pivot for calculating all timestamps for the patient. To compute the timestamp for this initial record, we took the patient's age in years, added half a year, and then multiplied by 365 to obtain an approximate age in days, which served as the timestamp. We added half a year to improve the approximation, as we only have the age in years and do not know whether the patient's birthday occurred one or eleven months ago; however, we assumed it to be on average six months, or half a year. For all subsequent records, we calculated the difference in days between the record date and the date of the first record, and then added this difference to the timestamp from the first record to create the timestamp for each subsequent record.

7.1.2 Diagnosis embedding

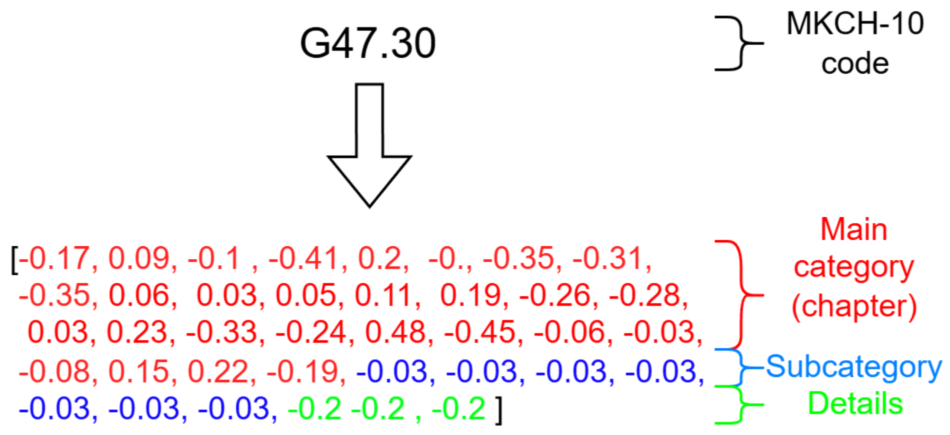
As discussed in Sec. 5.1.2, the embedding of diagnoses is based on the MKCH-10-SK (ICD-10-CM) code of the disease. We split this code into three parts, embed each part independently, and finally concatenate the results.

To embed the main category, we generated a vector containing random numbers using a uniform distribution for each letter of the English alphabet. We chose to sample these random numbers from the interval [-0.5, 0.5]. This interval was chosen mostly arbitrarily, as we planned to pass the resulting embedding into a normalization function once it was complete.

For the subcategory and details, we linearly assigned a value to each possible two-digit code. We chose the interval for these values to be [-0.5, 0.5], meaning subcategory 00 would get -0.5, category 50 would get 0, and category 99 would get 0.5. This interval was chosen so that each dimension of this embedding would have the same mean and standard deviation as each dimension of the main category. As a result, they also have, on average, the same distance per dimension. This means that each position of each part of the embedding should contribute to the total distance with the same weight.

The most important part was to assign lengths to the vector of each part in a way that would encode their importance. The main category, the most important part, got a vector of length 28, the subcategory part got length 7, and finally, the details got length 3.

A showcase of the resulting embedding can be seen in Fig. 7.1, where each part is highlighted by a different color and all values are rounded to two decimals.



```
[-0.17, 0.09, -0.1 , -0.41, 0.2,  -0., -0.35, -0.31,
 -0.35, 0.06,  0.03, 0.05, 0.11,  0.19, -0.26, -0.28,
  0.03, 0.23, -0.33, -0.24, 0.48, -0.45, -0.06, -0.03,
 -0.08, 0.15, 0.22, -0.19, -0.03, -0.03, -0.03, -0.03,
 -0.03, -0.03, -0.03, -0.2 -0.2 , -0.2 ]
```

Figure 7.1: Showcase of resulting embedding of specific diagnosis (rounded to two decimal places).

7.1.3 Drug embedding

The embedding for drugs was performed similarly to the diagnosis embedding: each level was embedded separately, and the final embedding was created by concatenating them. For drug codes, each level was embedded using a random vector sampled from a uniform distribution over the interval [-0.5, 0.5]. We chose random vectors because none of the ATC levels contain internal subgroupings analogous to the hierarchical subgroups in diagnosis codes (see Sec. 5.1.2). To encode the importance of each level, we again varied the vector lengths, with higher levels (e.g., anatomical groups) assigned shorter vectors. The specific vector lengths for each level are provided in Tab. 7.1, resulting in a total embedding length identical to the diagnosis embedding. If a code is incomplete (i.e., missing higher levels), the missing parts are replaced with zero vectors, which act as neutral elements in the embedding space.

Level	Length
1	21
2	9
3	5
4	2
5	1

Table 7.1: Lengths of random vectors assigned to each information level of ATC code.

With this embedding, we should obtain codes whose similarity is more dependent on whether the lower, more important levels match than the higher ones.

7.1.4 Medical procedure embedding

Embedding of medical procedures was straightforward since we used an already trained model.

As discussed in Sec. 5.1.4, for the LLM we chose the LaBSE model. It is a model developed by Google to encode text into high-dimensional vectors. This model was trained on 109 languages, including Slovak. Using this model was straightforward, as we just had to input the complete description of the procedure to receive a 768-dimensional dense encoding of it. After computing all embeddings, we performed principal component analysis (PCA) to reduce the dimensionality of this embedding while maintaining most of the variance, or in other words, most of the information stored inside it.

We also tried a different approach using a Word2vec model trained specifically for the Slovak language. More specifically, we used the word2vec-sk model made by the company Essential Data [2]. This model was trained on a corpus containing around 110 million words. We chose the version of the model trained using the CBOW approach. Since this model is trained to embed words and not text, we first split the description of the procedure into words and lemmatized those words, in other words, changed them into their base form. Then, using the Word2vec model, we embedded each word into a dense 200-dimensional vector separately and finally created the description embedding as an average of the embeddings of all words in it.

We expected that the LaBSE model would produce better results compared to standard text embedding models trained solely on the Slovak language, since the LaBSE model is trained by comparing embeddings not only to similar sentences in Slovak but also to their translations in other languages. This could mitigate the relatively small amount of Slovak language data compared to other, more commonly used languages. Additionally, this model could recognize domain-specific words, in our case medical terms, which are often left in a foreign language and would most likely not be found in a Slovak-only corpus.

Finally, we create the record embedding by concatenating all four parts. Since we have two separate datasets, one for prescribed drugs and one for medical procedures, we always have only three out of four pieces of information available for each record. Since timestamp and diagnosis are always available, we substitute the missing part with a vector of zeros of appropriate length, which is the most neutral embedding since we centered both medical procedure and drug prescription embeddings around zero.

7.2 Prediction of record cost

As discussed in Sec. 5.4, we used an MLP to predict the cost of a record. However, since our goal is to predict the future cost category of a patient and not the exact cost amount, we decided to also predict the cost category of the record instead of its precise cost.

To assess how to split the interval of possible costs into sub-intervals corresponding to each category, we visualized the costs of records from our training dataset in a histogram. The resulting histogram can be seen in Fig. 7.2, where the y-axis is linear and the x-axis is logarithmic, with the size of the bins also increasing logarithmically. From this, we can see that most records have costs between 0.1€ and 200€, so that is where we want most of the cost categories to be. We decided to merge all costs under one euro into a single category since, even though they are numerous, they have very little influence on the total cost of a patient in a year. Then, we split the interval between 1 and 200 into 6 categories with increasing width. After that, we grouped a few outliers between 200 and 500 into one category, and finally, we gave all outliers above 500 their own category. This gave us 9 categories, which are summarized in Tab. 7.2.

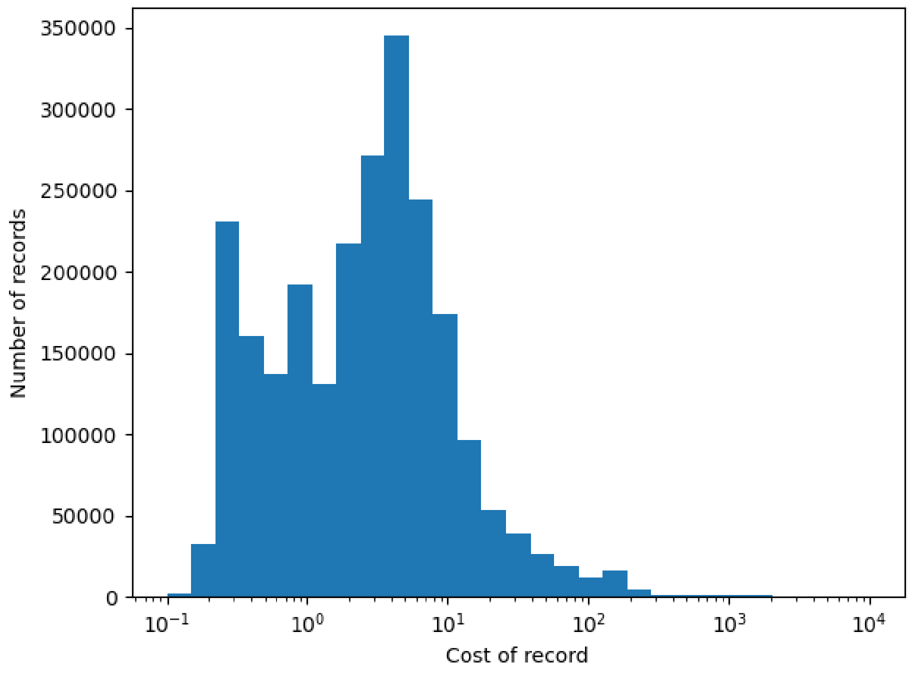


Figure 7.2: Histogram showing distribution of cost of records in training dataset.

Category	Interval
1	[0,1)
2	[1,5)
3	[5,10)
4	[10,20)
5	[20,50)
6	[50,100)
7	[100,200)
8	[200,500)
9	[500,∞)

Table 7.2: Intervals of record cost for each category.

The model itself consists of an input layer of size 196 (the final size of our embedding), followed by multiple linear layers with non-linear activation functions in between. This leads to a final layer of size 9 (matching the number of cost categories). The outputs of this final linear layer are passed through a softmax function to transform the raw values into probabilities for each category.

We optimized multiple parameters of the network itself, as well as key training hyperparameters. From the perspective of the model architecture, we tested varying numbers of layers (model depth), their sizes, and the non-linear functions between them. For training, we experimented with three loss functions: mean squared error loss, cross-entropy loss, and negative log likelihood loss. As the optimizer, we chose Adaptive Moment Estimation (Adam) [24], which adaptively adjusts learning rates during training and is considered an industry standard.

Parameter selection was conducted by training multiple models on a smaller subset of the dataset locally. The final model was then trained with the chosen hyperparameters on the complete dataset using a server. All models were trained using a batch approach to limit the amount of data loaded at once, while avoiding gradient computation based solely on the loss from a single input (in this case, a single patient record).

7.3 Prediction of future records

For the task of predicting future patient records based on their history, we decided to use two models: LSTM and Transformer. We briefly experimented with simpler models, such as basic RNN and GRU, but in both cases, we immediately observed significantly worse results. Therefore, we chose to omit them from our results entirely and focus on the more complex models.

In both tested models, we optimized the model depth, which refers to the number of hidden layers, and the dropout rate, which is the percentage of neurons randomly deactivated (set to zero) at each training step. This technique allows other neurons to learn patterns from those that are dropped out, helping to reduce overfitting and improve generalization. For the LSTM model, we also optimized the size of the hidden layer, and for the decoder-only Transformer, we optimized the number of transformer heads.

Since our embedding of a patient record consists of multiple parts of varying lengths, we modified the loss function to account for these differences, ensuring each part contributes equally to the total loss. We based this custom loss function on the mean squared error (MSE), defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y_i})^2. \tag{7.1}$$

Here, n is the number of dimensions, Y_i is the target value at the i-th dimension, and $\hat{Y_i}$ is the value predicted by the model at that dimension. However, this approach assigns more importance to parts with more dimensions. For example, if a vector has 30 dimensions divided into two parts (20 and 10 dimensions), the larger part would naturally receive twice the importance due to its size. To address this, we computed the MSE loss for each part and then calculated the mean of these losses to obtain the final loss. We refer to this modified loss function as the subpart weighted MSE, as it adjusts the weights of each dimension to give equal importance to each subpart of the vector. The formula for this loss is:

$$SubpartWeightedMSE = \frac{1}{p}\sum_{j=1}^{p}(\frac{1}{l_j}\sum_{i=1}^{l_j}(Y_{s_j+i} - \hat{Y_{s_j+i}})^2). \tag{7.2}$$

In this equation, we have a few additional attributes: p represents the number of subparts, l_j is the number of dimensions of the j-th subpart, and s_j is the index of the first dimension of the j-th subpart. For the optimizer, we used the same one as in the cost of record prediction model, which is Adam.

When we began trying to train these models, we encountered memory limitations, so we decided to limit the context window, meaning the number of past records seen by the model. The whole process of data preparation for training consists of this sequence of steps:

1. Get batch of patients.
2. For each patient in patch:
   (a) Get all records for patient.
   (b) Order records of patient.
   (c) Using moving window of our maximal context size, that moves by one record at the time, this generate list of all windows for patient.
   (d) Generate the input list by removing the last window and the target list by removing the first window.
3. Collect inputs and targets for all patients in batch.

In step 2(b), we primarily order by date in ascending order, and if we have multiple records with the same date, we secondarily order them based on the internal ID value of diagnosis, medical procedure, and drug, in this order. We do this to make sure the ordering is coherent throughout all training inputs.

By generating the input and target lists as explained in step 2(c), we use the i-th window as input and expect the $(i + 1)$-st as output. We did this to increase the amount of information the model can learn by asking it to predict from incomplete context. This means that if we have a context of size n, we do not only want to predict the $(n + 1)$-st record but also, for all k such that $0 < k < n$, we want to predict the $(k + 1)$-st record based on the first k records from the context. Both of our tested models support this behavior. In the case of the LSTM model, it is integrated directly in the model backbone provided by the `Pytorch` library since we are not using the bidirectional version of the model [5], and to achieve this in the Decoder-only transformer model, we used causal masking inside the self-attention layers, which forbids queries from seeing keys from the future based on their perspective.

Similarly to cost prediction models, these models were also first trained on a smaller subset of data to assess the best values for the hyperparameters being optimized, and the best one was then retrained on the complete dataset on the server. Another similarity is that we used batch training for these models as well. However, in this case, since transforming patient records into lists of windows causes them to increase in memory size, we use two types of batches: the first is the patient batch, which is a group of patients whose records are transformed together, and then the training batch, which is the usual type of batch, meaning the number of windows that go into the model at once.

Chapter 8

Research

This chapter is dedicated to determining whether the embeddings possess the desired properties and to finding the most suitable parameters for which the prediction models achieve the lowest loss and highest accuracy.

8.1 Embedding of patient

Once again, the first sub-task in predicting a patient's future costs is to embed each patient record into a numerical vector that can be interpreted by a neural network. Our approach to this process is detailed in Sec. 7.1. Here, we will examine whether the proposed solution possesses the desired property regarding the similarity between two embeddings.

8.1.1 Diagnosis embedding

We need to confirm that closely related diagnoses receive embeddings with smaller distances-indicating higher similarity-compared to less related diagnoses. To verify that our embeddings exhibit these desired properties, we computed the similarity between embeddings of multiple codes. As our similarity function, we used the simple multiplicative inverse of the Euclidean distance. The results are shown in Tab. 8.1. The highest similarity was observed between codes G47.30 and G40.09, which is expected since they belong to the same main category and have very similar subcategories. The second highest similarity was between H40.09 and H18.80, the only other pair belonging to the same main category. This confirms that the main category has the greatest impact, as this similarity is significantly higher than that between G40.09 and H40.09, which differ only in the main category.

Another confirmation that the embeddings function as intended comes from clustering analysis. Specifically, we expect that if we group embeddings into as many clusters

Code A	Code B	Similarity
G47.30	G40.09	2.77
G47.30	H40.09	0.53
G47.30	H18.80	0.46
G40.09	H40.09	0.54
G40.09	H18.80	0.45
H40.09	H18.80	0.84

Table 8.1: Similarities of embedding of multiple chosen MKCH-10 codes

as there are main categories, each cluster should contain mostly, if not exclusively, diagnoses from a single main category. To test this, we used K-means clustering with 26 clusters, corresponding to the number of different main diagnosis categories, and obtained the following results:

Cluster ID	Cluster size	Frequency of first level values
0	654	C: 654,
1	2092	M: 2092,
2	766	Y: 766,
3	543	S: 543,
4	786	Z: 786,
5	1100	T: 1100,
6	1067	X: 1067,
7	620	H: 496, U: 124,
8	752	S: 752,
9	1770	M: 1770,
10	739	Q: 739,
11	584	D: 584,
12	507	F: 507,
13	958	W: 958,
14	556	E: 556,
15	491	A: 491,
16	553	O: 553,
17	627	K: 627,
18	872	V: 872,
19	521	G: 521,
20	463	L: 463,
21	420	R: 420,
22	465	B: 465,
23	717	J: 319, P: 398,
24	568	I: 568,
25	535	N: 535,

Table 8.2: Size of clusters and frequencies of first level diagnosis codes in them.

In Tab. 8.2, we observe that almost every cluster consists of diagnoses from only a single main category. The only immediately visible issues occur in clusters 7 and 23, where two main categories were assigned to the same cluster. This overlap is likely due to their smaller size relative to other categories and the fact that the largest categories (M and S) were split into two clusters.

8.1.2 Drug embedding

To confirm that the desired property of similarity is satisfied also for drug embeddings, we performed a similar check as we did with diagnosis codes. We computed the similarity of four chosen ATC codes and verified if our theory holds. To compute similarity, we again used the multiplicative inverse of Euclidean distance. We selected C01EB15, C01CA04, C10AA07, and J01CA04. We expected the first two to be most similar since they match on the first levels, then the first two compared to the third should have slightly lower similarity since they match on only the first level, and finally, we expected that all three would be least similar to the fourth one since the first level is different, even though the second and fourth match on all other levels except the first. The results are shown in Tab. 8.3. We can see that the results met our expectations, with the highest similarity between the first two and the lowest between any of the first three and the fourth, even in the case of the second and fourth that match on all other levels except the first.

Code A	Code B	Similarity
C01EB15	C01CA04	1.24
C01EB15	C10AA07	0.54
C01EB15	J01CA04	0.45
C01CA04	C10AA07	0.64
C01CA04	J01CA04	0.48
C10AA07	J01CA04	0.38

Table 8.3: Similarities of embedding of multiple chosen ATC codes

Similarly to the diagnosis codes, we can test whether our embeddings primarily cluster according to the first level of the code. Since there are 14 main anatomical or pharmacological groups, as shown in Fig. 5.2, we used the K-means algorithm with 14 clusters and obtained the following results:

Based on the results shown in Tab. 8.4, we conclude that the embeddings generally function as intended. The clusters predominantly consist of embeddings with identical first-level values. Exceptions likely arise from the uneven distribution of drugs across categories. Smaller categories with fewer drugs, such as P, S, D, H, and G, were occasionally assigned to the same cluster as larger categories. Conversely, the largest categories, such as N, C, and V, were split into multiple clusters due to their size.

Cluster ID	Cluster size	Frequency of first level values
0	8645	C: 8645,
1	19132	N: 19132,
2	9322	L: 8657, S: 665,
3	11734	A: 11734,
4	7159	B: 7159,
5	9526	V: 9526,
6	4681	R: 4681,
7	14732	C: 14732,
8	7237	V: 7237,
9	4031	M: 3987, P: 44,
10	3599	G: 3599,
11	2367	N: 2367,
12	9032	D: 1266, G: 897, H: 1136, J: 5733,
13	6093	N: 6075, P: 18,

Table 8.4: Size of clusters and frequencies of first level drug codes in them.

These test results confirm that the embeddings operate as designed overall.

8.1.3 Medical procedure embedding

As stated in previous chapters, we tried two different approaches, both using pre-trained models. After embedding medical procedure descriptions with both methods, we quickly found that many procedures did not receive any embedding from the Word2vec model. More specifically, 731 out of 7,329 procedures, almost 10%, were not embedded. Even among the procedures that did receive some embedding, there were many words that were not embedded and therefore did not contribute to the resulting embedding. This was mostly caused by the limitations of the Word2vec model, which can only embed words that are in its dictionary, meaning those available in its training corpus. It seems that many professional medical terms, such as 'polycystické' or 'cytokín', were not present in the corpus. A similar issue could potentially have occurred when using the LaBSE model, however, we found no case where the LaBSE model was unable to embed an entire procedure. Also, thanks to most professional medical terms being similar across multiple languages, and the fact that the LaBSE model was trained on a much larger corpus consisting of 109 distinct languages, it is highly probable that many more terms were successfully embedded, creating a much better embedding of the whole description.

Due to the significant issues with embeddings produced by the Word2vec model, we proceeded exclusively with the LaBSE model, which successfully embedded all procedure descriptions. This model generated 768-dimensional embeddings. To densify the information while retaining most of its content, we applied PCA. We specifically targeted the first few dimensions that would collectively explain over 90% of variance-a common proxy for preserved information in PCA applications. The algorithm yielded a 119-dimensional embedding that captured 90.4% of the original embedding's variance.

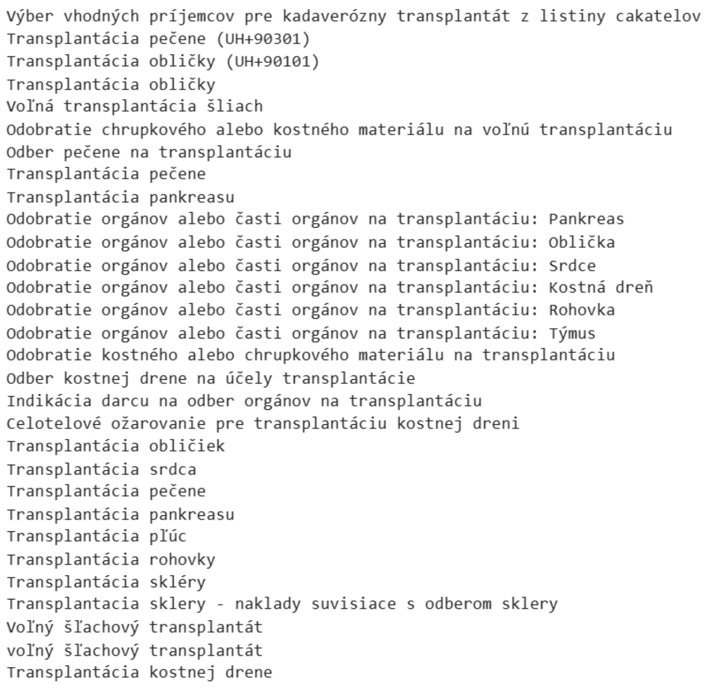
To validate whether these embeddings effectively encoded procedural semantics, we replicated our drug and diagnosis validation approach using K-means clustering. However, unlike those cases where we had hierarchical code levels for validation, we lacked analogous categorical metadata for procedures. We therefore manually inspected cluster contents to evaluate whether grouped procedures shared meaningful clinical or operational relationships.

Given the 7 329 procedures, we chose K (the number of clusters) to be 365, aiming for approximately 20 procedures per cluster on average. This decision was largely arbitrary. Upon inspecting multiple clusters, we identified many meaningful groupings such as cluster 215, whose procedures are listed in Fig. 8.1, which exclusively contains transplantation-related interventions. However, in some instances, clusters contained a mixture of seemingly unrelated procedures, such as those in cluster 45 shown in Fig. 8.2), which combines the evaluation of final reports with investigations of pharmacokinetics and papillosphincterotomy. Fortunately, it appears that the content of clusters like these is not entirely random but rather consists of multiple subgroups, suggesting that the embedding functions as intended and that we may have simply clustered procedures into too few groups.

8.2 Prediction of record cost

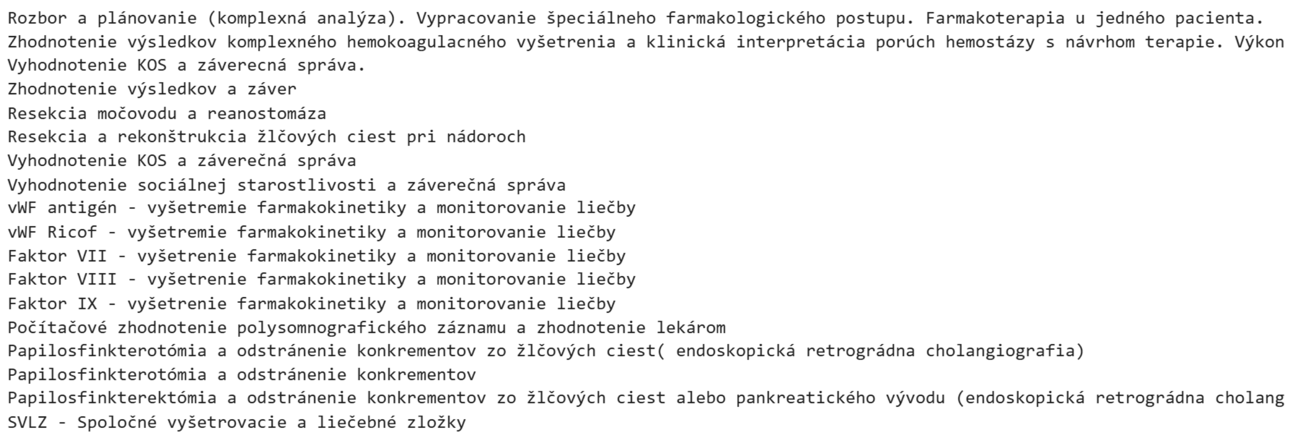
In the case of predicting the record cost category, we planned to use an MLP model. All models we tested have in common that, as input, they take a 196-dimensional vector, which is the dimensionality of our embedding. After that, there are multiple hidden linear layers with non-linear functions between them, with the final layer outputting a 9-dimensional vector, which corresponds to the number of cost categories, followed by a softmax function to transform this vector into probabilities for each category. For the purpose of loss computation, we compared the correct category to the raw result from the model, while for accuracy computation, we first transformed this result into a one-hot vector based on the maximal value in the resulting vector before comparing it to the correct category.

Each model that was trained to assess the best parameters was trained on a subset



```
Výber vhodných príjemcov pre kadaverózny transplantát z listiny cakatelov
Transplantácia pečene (UH+90301)
Transplantácia obličky (UH+90101)
Transplantácia obličky
Voľná transplantácia šliach
Odobratie chrupkového alebo kostného materiálu na voľnú transplantáciu
Odber pečene na transplantáciu
Transplantácia pečene
Transplantácia pankreasu
Odobratie orgánov alebo časti orgánov na transplantáciu: Pankreas
Odobratie orgánov alebo časti orgánov na transplantáciu: Oblička
Odobratie orgánov alebo časti orgánov na transplantáciu: Srdce
Odobratie orgánov alebo časti orgánov na transplantáciu: Kostná dreň
Odobratie orgánov alebo časti orgánov na transplantáciu: Rohovka
Odobratie orgánov alebo časti orgánov na transplantáciu: Týmus
Odobratie kostného alebo chrupkového materiálu na transplantáciu
Odber kostnej drene na účely transplantácie
Indikácia darcu na odber orgánov na transplantáciu
Celotelové ožarovanie pre transplantáciu kostnej dreni
Transplantácia obličiek
Transplantácia srdca
Transplantácia pečene
Transplantácia pankreasu
Transplantácia pľúc
Transplantácia rohovky
Transplantácia skléry
Transplantacia sklery - naklady suvisiace s odberom sklery
Voľný šľachový transplantát
voľný šľachový transplantát
Transplantácia kostnej drene
```

Figure 8.1: Medical procedures grouped in cluster number 215.



```
Rozbor a plánovanie (komplexná analýza). Vypracovanie špeciálneho farmakologického postupu. Farmakoterapia u jedného pacienta.
Zhodnotenie výsledkov komplexného hemokoagulacného vyšetrenia a klinická interpretácia porúch hemostázy s návrhom terapie. Výkon
Vyhodnotenie KOS a záverecná správa.
Zhodnotenie výsledkov a záver
Resekcia močovodu a reanostomáza
Resekcia a rekonštrukcia žlčových ciest pri nádoroch
Vyhodnotenie KOS a záverečná správa
Vyhodnotenie sociálnej starostlivosti a záverečná správa
vWF antigén - vyšetremie farmakokinetiky a monitorovanie liečby
vWF Ricof - vyšetremie farmakokinetiky a monitorovanie liečby
Faktor VII - vyšetrenie farmakokinetiky a monitorovanie liečby
Faktor VIII - vyšetrenie farmakokinetiky a monitorovanie liečby
Faktor IX - vyšetrenie farmakokinetiky a monitorovanie liečby
Počítačové zhodnotenie polysomnografického záznamu a zhodnotenie lekárom
Papilosfinkterotómia a odstránenie konkrementov zo žlčových ciest( endoskopická retrográdna cholangiografia)
Papilosfinkterotómia a odstránenie konkrementov
Papilosfinkterektómia a odstránenie konkrementov zo žlčových ciest alebo pankreatického vývodu (endoskopická retrográdna cholang
SVLZ - Spoločné vyšetrovacie a liečebné zložky
```

Figure 8.2: Medical procedures grouped in cluster number 45.

of the dataset containing 800 patients for training and 200 for validation. In terms of the number of records, the training dataset consisted of 2 132 436 records, while the validation set contained 533 110 records.

As mentioned in Sec. 7.2, we primarily focused on assessing the optimal depth, layer sizes, and non-linear activation functions between those layers, but we also tested multiple different loss functions. Given the large number of possible depths and combinations of sizes and activation functions, we approached this systematically. For the number of layers (depth), we tested models with depths denoted as 0, 1, 3, and 6. A depth of 0 means the model contains a single layer to transform the input vector into a category vector, followed by a softmax function; this served as a baseline to see if the model could learn anything meaningful from the data. Depths of 1, 3, and 7 indicate that there are 1, 3, or 7 additional layers (respectively) between the input vector and the output layer, each with their own activation functions. We chose these values to represent shallow, moderately deep, and relatively deep models.

For each non-zero additional depth, we tried multiple configurations to determine if we would obtain significantly different results. We tested three configuration types from the perspective of layer sizes:

- Gradually decreasing layer size from input to output.
- Initially increasing the size from input to allow the model to learn more patterns, then decreasing it to the output size.
- Maintaining the input size until the very last layer, which decreases to the output size.

For each of these layer size configurations, we tested two non-linear activation function configurations:

- Using only Gaussian Error Linear Units (GELU) between each layer
- A random combination of different activation functions selected from a predefined list

We selected activation functions for models with random combinations from the following list:

- Sigmoid function (Sigmoid)
- Hyperbolic tangent function (Tanh)
- Rectified linear unit function (ReLU)
- Leaky Rectified Linear Unit function (LeakyReLU)
- Gaussian Error Linear Units function (GELU)
- Sigmoid Linear Unit function (SiLU)

We selected these functions to include a mix of more traditional options, such as the Sigmoid function, and more modern variants, like the Sigmoid Linear Unit function (SiLU). This approach resulted in a total of 19 different models, derived from multiplying three layer size configurations by two activation function configurations by three non-zero additional depths, plus a base model with zero additional depth. We conducted this testing three times using three different loss functions, yielding a total of 67 models. The resulting accuracies are compiled in Tab. 8.5.

The first loss function we tested was the mean square error (MSE) loss, which, as the name suggests, computes the average of the squared differences between the model output and the target. When examining the results for this loss function in Tab. 8.5, we observe that adding layers improves model accuracy but only up to a certain point. A model with only a single layer followed by a softmax function achieved the lowest accuracy, barely exceeding 60%, while nearly all other models surpassed 70% accuracy. However, as we deepen the model further, we reach diminishing returns, with almost no improvement once the model has three or more layers, and accuracy slowly approaches 80%. We also observe no signs of overfitting, as the differences between training and validation accuracy were minimal in all cases. From the perspective of layer size architecture, the best results were generally achieved by models that first expanded the dimensionality before reducing it to the number of categories.

The second loss function we tested was cross-entropy loss, which is theoretically better suited for classification tasks like ours. The results show an interesting pattern: very deep models performed similarly to or even worse than shallower ones, while models with intermediate depths achieved the best performance. This could potentially stem from issues like vanishing gradients. From the perspective of layer sizes and activation functions, models using this loss exhibited behavior similar to those trained with MSE loss. Overall, the accuracy of these models was almost always lower compared to models using MSE loss.

The final loss function we evaluated was negative log-likelihood (NLL) loss, which, like cross-entropy loss, is designed for classification problems. The results closely mirrored those of MSE loss across all metrics, with only minor differences: slightly lower accuracy in shallower models. However, for the deepest models tested, we observed minimal to no measurable difference in performance.

Based on these results, we decided to use the model from row 17 in Tab. 8.5 for training on the complete dataset. This model is an 8-layered model (7 additional layers plus a final layer with a softmax activation function) featuring an initially expanding layer size architecture, random activation functions, and MSE loss, which achieved the highest accuracy in our testing. Since the layer sizes, activation functions, and depth were chosen somewhat arbitrarily, we conducted additional testing to fine-tune these parameters further. However, we did not find a setup that yielded tangibly higher accuracy, so we decided to proceed with the current configuration.

Depth	Layer sizes	Activation functions	Mean square error		Cross entropy		Negative log likelihood	
			Test accuracy	Validation accuracy	Test accuracy	Validation accuracy	Test accuracy	Validation accuracy
0	[]	[]	63.2%	63.0%	63.5%	63.3%	62.9%	62.6%
1	[98]	[GELU]	74.5%	74.3%	72.8%	72.6%	73.2%	73.0%
		[Tanh]	71.3%	71.0%	70.9%	70.6%	71.1%	70.9%
	[392]	[GELU]	76.6%	76.3%	74.2%	74.0%	75.5%	75.2%
		[Sigmoid]	70.8%	70.6%	69.9%	69.6%	70.2%	70.0%
	[196]	[GELU]	75.9%	75.6%	74.1%	73.8%	74.1%	73.8%
		[LeakyReLU]	76.0%	75.8%	75.5%	75.3%	74.1%	74.0%
3	[98, 48, 24]	[GELU, GELU, GELU]	74.4%	74.2%	72.0%	71.8%	72.2%	72.0%
		[Sigmoid, ReLU, Tanh]	69.7%	69.5%	69.7%	69.4%	70.3%	70.1%
	[392, 196, 98]	[GELU, GELU, GELU]	77.7%	77.5%	75.2%	75.0%	75.5%	75.3%
		[SiLU, ReLU, GELU]	77.4%	77.2%	74.8%	74.5%	75.2%	75.0%
	[196, 196, 196]	[GELU, GELU, GELU]	77.2%	77.0%	74.7%	74.3%	75.0%	74.7%
		[ReLU, Sigmoid, SiLU]	76.7%	76.5%	73.9%	73.7%	75.0%	74.8%
7	[142, 104, 72, 48, 30, 18, 12]	[GELU, GELU, GELU, GELU, GELU, GELU, GELU]	77.1%	76,9%	72.0%	71.7%	76.5%	76.2%
		[SiLU, Sigmoid, GELU, Tanh, ReLU, Sigmoid, LeakyReLU]	75.1%	75.0%	70.3%	70.1%	75.1%	74.9%
	[588, 294, 147, 49, 98, 36, 18]	[GELU, GELU, GELU, GELU, GELU, GELU, GELU]	77.6%	77.4%	71.2%	71.0%	77.4%	77.2%
		[SiLU, GELU, Sigmoid, GELU, SiLU, GELU, LeakyReLU]	78.3%	78.0%	72.0%	71.8%	76.9%	76.7%
	[196, 196, 196, 196, 196, 196, 196]	[GELU, GELU, GELU, GELU, GELU, GELU, GELU]	77.3%	77.1%	72.8%	72.6%	77.4%	77.2%
		[Sigmoid, GELU, ReLU, SiLU, LeakyReLU, GELU, SiLU]	77.1%	76.9%	72.8%	72.6%	77.2%	77.0%

Table 8.5: Accuracies for various MLP models using different loss functions.

Before moving to the main training, we compared the MLP approach to two other types of classifiers. The first was a Gradient Boosting Classification Tree model, for which we trained trees with maximum depths of 1, 5, 10, and 20. The second classifier we tried was a Ridge regression model, where we tested multiple values for the parameter alpha, which denotes regularization strength. We used values of $10^{-10}, 10^{-5}, 0.1, 1, 10,$ and 10^3.

The results of the Gradient Boosting model are shown in Tab. 8.6. We observe an initial increase in accuracy; however, beyond a depth of 10, no further improvements occur. The best-performing model appears to be the one with a maximum depth of 10, achieving approximately 71% accuracy. This is significantly lower than the 78%

Maximum depth	Train accuracy	Validation accuracy
1	51.5%	51.4%
5	69.0%	68.8%
10	71.5%	71.4%
20	71.2%	71.0%

Table 8.6: Accuracies of Gradient Boosting models with different maximal depths of tree.

accuracy of our top MLP model.

Alpha	Train accuracy	Validation accuracy
10^{-10}	67.1%	66.9%
10^{-5}	67.0%	66.8%
10^{-1}	67.0%	66.8%
10^{0}	67.0%	66.8%
10^{1}	66.9%	66.7%
10^{3}	66.8%	66.7%

Table 8.7: Accuracies of Ridge models with different regularization strength.

The results of the Ridge model testing are shown in Tab. 8.7, where we can see that different regularization strengths have no tangible effect on model performance. There appears to be a very small decrease in accuracy with increased regularization strength, but it is not significant. All models achieve an accuracy of around 67%, which is higher than the most basic MLP or Gradient Boosting models, but is easily outperformed by their deeper versions.

We can see that neither of the models used for comparison was able to outperform our MLP model, so we saw no reason to use any of these models instead. Therefore, we proceeded with our best MLP model, as assessed earlier, for the main training. This model achieved a training accuracy of 78.3% and a validation accuracy of 78.0%. The training loss for this model was 0.0346, and the validation loss was 0.0349.

8.3 Prediction of future records

As stated in Sec. 7.3, we tested two types of models: standard RNNs and a Decoder-only Transformer. For both architectures, we evaluated multiple hyperparameter combinations to identify the most effective configuration.

We trained each model on the same subset of data, which consisted of 100 patients, with a total of 270 351 records, training each model for 5 epochs. We then validated each model on data from 20 patients, which were different from those used in training, and which had 54 429 records in total. These numbers are significantly lower than in the assessment of parameters for the model to predict the cost of a record, because the training of these models was significantly more complex in terms of both computational and memory requirements.

Firstly, we needed to assess which standard RNN model to test, specifically whether a basic Elman RNN would suffice for our task or if using a more complex model like GRU or LSTM would yield tangible improvements. To do this, we tested all three models under the same conditions. The first setup we evaluated consisted of 6 layers with a width of 196 and a 20% dropout rate for regularization. With this setup, we obtained the results shown in Tab. 8.8, where we observed that the LSTM model achieved the lowest loss on both training and validation data. The Elman RNN had the second-lowest training loss, while the GRU had the second-lowest validation loss. Based on these results, we concluded that the LSTM model provided tangible improvements.

Model	Train loss	Validation loss
Elman RNN	0.5757	0.6689
GRU	0.6278	0.6517
LSTM	0.5156	0.6328

Table 8.8: Comparison of RNN models with 6 layers of width 196 and 20% dropout rate.

To ensure these results were not a fluke, we tested another setup. This time, we used a significantly larger model with 12 layers (double the depth) and a width of 784 (quadruple the original width), while retaining the 20% dropout rate. The results from this setup are shown in Tab. 8.9. Here, we observe that while the GRU model appears marginally better based on training loss, the LSTM still retains a slight edge in validation loss over both other models. We conclude that although performance differences diminish with increased model size, the LSTM maintains a small advantage compared to simpler architectures.

Model	Train loss	Validation loss
Elman RNN	0.6107	0.6404
GRU	0.5467	0.6329
LSTM	0.5780	0.6268

Table 8.9: Comparison of RNN models with 12 layers of width 784 and 20% dropout rate.

Based on the results of these two tested configurations, we decided to use the LSTM model for further testing.

8.3.1 LSTM

For the LSTM, we were interested in finding the best combination of the number and size of hidden LSTM layers. Additionally, we wanted to determine whether adjusting the dropout rate would improve the model.

As for depth, we chose three values for initial testing: 3, 6, and 12. These values represented a relatively shallow model, a slightly deeper one, and a comparably significantly deeper one. For the width of these layers, we chose 196 (matching the embedding size), then doubled and quadrupled that size to 392 and 784, respectively. In both cases, we were interested in whether increasing the model size in their corresponding dimensions would bring a sizable improvement in output quality. These two hyperparameters gave us nine combinations to test. In each test, we set the dropout rate to 20%. Testing of different dropout rates was conducted afterward on the best model from this phase.

The results of these tests are shown in Tab. 8.10. From the perspective of model width, we observe that shallower models (3 and 6 layers) exhibit a decrease in training loss as width increases. However, this is accompanied by an increase in validation loss, indicating potential overtraining in wider configurations. For deeper models (12 layers), there is a slight decrease in training loss, while validation loss remains mostly consistent across all widths.

Number of layers	Width of layer	Train loss	Validation loss
3	196	0.4921	0.6389
	392	0.4472	0.6517
	784	0.4114	0.6535
6	196	0.5156	0.6326
	392	0.4786	0.6452
	784	0.4285	0.6493
12	196	0.5983	0.6278
	392	0.5822	0.6292
	784	0.5780	0.6268

Table 8.10: Test and validation loss for different number of layers and width of layer in LSTM model.

When analyzing model performance by depth, we observe similar behavior regardless of width. Counter-intuitively, increasing depth correlates with higher training loss, which may suggest training issues such as insufficient regularization, vanishing gradients, or suboptimal learning rates. Conversely, the slight decrease in validation loss implies that deeper models generalize better.

For dropout rate testing, we selected the 12-layer LSTM model (which significantly outperformed shallower models in validation loss) with a layer width of 196. We chose this width as it showed marginally better results in shallow models and comparable performance to wider models in deeper architectures.

In Tab. 8.11, we see that the optimal dropout rate appears to be 20%, which resulted in a validation loss of 0.6278. We consider this model to be the best LSTM model we have.

8.3.2 Decoder-only Transformer

In the case of the Transformer model, when trying to find the most suitable model, we focused on three hyperparameters: two directly influencing the model (the number of Transformer layers and the number of heads) and one influencing training (the dropout rate).

Dropout rate	Train loss	Validation loss
10%	0.5936	0.6351
15%	0.5964	0.6362
20%	0.5983	0.6278
25%	0.5915	0.6342
30%	0.6257	0.6550

Table 8.11: Test and validation loss for different dropout rate in LSTM model with 12 layers of width 196.

The hyperparameter setting process was very similar to that for the LSTM. First, we tried to assess the best values for the first two hyperparameters that influence model structure with the dropout rate set to 20%. Once we obtained the best results, we tested a couple of dropout rate values on that specific model. For the number of layers, we tested the same model depths as for the LSTM: a shallow model with only 3 layers, a slightly deeper model with 6 layers, and a relatively deep model with 12 layers to evaluate whether deepening the model would have a positive effect on the results. In the case of the number of heads hyperparameter, we were constrained by the fact that this number must be a divisor of the number of dimensions in the input embedding, as the model splits this embedding equally into the heads. Since our embedding has 196 dimensions, with a prime factorization of $2^2 \cdot 7^2$, we chose three values to test: 7, 14, and 49. This allowed us to see if the model would benefit more from a larger head size or a higher number of heads. These two hyperparameter values gave us nine combinations to test.

If we examine the performance of the models shown in Tab. 8.12, we observe that for shallower models (3 and 6 layers), differences in head counts yield negligible results. The best value appears to be 14 heads in both cases, though this might be coincidental due to the minimal performance gaps. For the deeper 12-layer model, increasing the number of heads reduces training loss (improving fit to training data) but increases validation loss (worsening generalization).

Regarding model depth, we see behavior opposite to the LSTM results: deeper Transformer models exhibit significantly lower training loss but higher validation loss, likely due to overfitting.

Number of layers	Number of heads	Train loss	Validation loss
3	7	0.5057	0.6276
	14	0.5035	0.6266
	49	0.5103	0.6297
6	7	0.4737	0.6416
	14	0.4714	0.6398
	49	0.4716	0.6439
12	7	0.4624	0.6537
	14	0.4301	0.6704
	49	0.4119	0.6729

Table 8.12: Test and validation loss for different number of layers and number of heads in Transformer model.

Given these signs of overtraining in deeper models, we selected shallow 3-layer architectures for dropout rate testing. Specifically, we chose the model with 14 heads, which achieved the best training and validation loss among all 3-layer models.

Dropout rate	Train loss	Validation loss
10%	0.4840	0.6433
15%	0.4889	0.6372
20%	0.5035	0.6266
25%	0.5105	0.6232
30%	0.5200	0.6212
35%	0.5225	0.6214
40%	0.5172	0.6226

Table 8.13: Test and validation loss for different dropout rates in the Transformer model with 3 layers and 14 heads.

In the results of dropout rate training shown in Tab. 8.13, we observe that increasing the dropout rate beyond 20% yields a tangible improvement in validation loss, although training loss increases, but only up to a certain point, beyond which no measurable improvement is seen (specifically, beyond 30%). Based on these results, we decided to select as the best Decoder-only Transformer model one with 3 layers, 14 heads, and a 30% dropout rate, which achieved a validation loss of 0.6212.

With this, we ended up with two models to choose from: an LSTM model with a validation loss of 0.6278 and a Decoder-only Transformer with a validation loss of 0.6212. We chose the latter because it not only had a slightly lower validation loss but also a lower training loss of 0.5200 compared to the LSTM model's training loss of 0.5983.

Chapter 9

Results

In this chapter, we will report and discuss the training and testing of the models we identified as the best performing on the training subset of data in Sec. 8.2 and Sec. 8.3, now trained on the complete training dataset.

9.1 Training of final models

Both MLP and Decoder-only Transformer models were trained on a dataset consisting of 156 020 patients (approximately 90% of all patients in our complete dataset). Each model was trained for 10 epochs.

For the MLP model, we settled on a batch size of 512 and an initial learning rate of 0.0004. After the final epoch, the model achieved a training loss of 0.0303 and a training accuracy of 80.2%, indicating a slight improvement over the model trained on the subset of data. However, the improvement is so minor that it suggests either reaching the upper limit of extractable information from the data or an unidentified flaw in our approach.

For the Decoder-only Transformer model, we used a lookback value of 20, meaning the model always processed the last 20 patient records to predict the 21st record. Due to the increased input size, we used a significantly smaller batch size of 64 compared to the MLP model. The initial learning rate was set to 0.0025. Training with these parameters resulted in a final training loss of 0.4005, which is significantly lower than the 0.5200 training loss of the model trained on the subset.

9.2 Validation of trained models

Validation of the models was performed using a dataset containing records of 16 335 patients, accounting for over 9% of our total patient population.

In the case of the MLP model for cost prediction, we obtained a validation loss of 0.0305 and an accuracy of 80.1%. Both loss and accuracy improved compared to the model trained on a subset of the data, which had a loss of 0.0349 and an accuracy of 78%. However, these differences are relatively small. Additionally, the validation loss and accuracy are on par with the training metrics, indicating that the model does not suffer from overfitting.

In our application, we ultimately won't use the cost categories of predicted records as they are but will aggregate them into a single category per patient. If the model occasionally predicts a category that is off by one in either direction, the final results will be affected only slightly, as there is a high likelihood that these errors might offset each other. Therefore, if we consider the percentage of cases where the model predicted either the precise category or one off, we achieve an adjusted accuracy of 96.8% (and 99.2% if we allow the model to be off by two categories).

Another interesting statistic to examine is the model's accuracy in each cost category to ensure it can predict correctly across all categories, not just those with the majority of the data. Here, accuracy is defined as the ratio of records successfully assigned to a category relative to all records that should belong to that category. The results of this check can be seen in Tab. 9.1, where we observe that accuracy declines with the frequency of categories in the dataset. However, despite the low frequency of categories 5 to 7, the model still achieves over 50% accuracy, and only two categories (8 and 9) have accuracy below 50%, which together form a very small part of the dataset (less than 0.4%).

For the Decoder-only Transformer model trained to predict future records, the final validation loss was 0.4740. This is significantly better than the validation loss of 0.6212 for the same model trained on the subset of data and matches or exceeds the training errors of all other model configurations tested during hyperparameter tuning.

Category	Dataset frequency	Validation accuracy
1	34.7%	87.5%
2	37.6%	84.4%
3	15.0%	62.5%
4	7.0%	54.8%
5	3.3%	53.7%
6	1.1%	50.8%
7	0.9%	58.1%
8	0.2%	35.3%
9	0.2%	38.2%

Table 9.1: Accuracies of record cost prediction model in each possible prediction category.

9.3 Testing of patient future cost prediction

The last 1,000 patients, who were not used in either training or validation, were employed to test whether the combination of trained models could succeed in our primary task: predicting the cost of a patient for the next year.

Since the cost for an entire year can be significantly larger than the cost for a single record, we decided to adjust the intervals for categories accordingly. The intervals for patient costs can be seen in Tab. 9.2, where we adopted an approach in which the threshold for the next category is approximately double that of the previous one.

The complete workflow for computing a patient's future cost would look like this:

1. Load patient data.
2. Embed patient records.
3. Compute the number of future records.
4. Predict the given number of future records.
5. Predict the cost of future records.
6. Transform the sum of future record costs into the patient's future cost category.

Category	Interval
1	[0,100)
2	[100,200)
3	[200,500)
4	[500,1000)
5	[1000,2000)
6	[2000,5000)
7	[5000,10000)
8	[10000,20000)
9	[20000,∞)

Table 9.2: Intervals of patient cost for each category.

We very briefly tried, instead of computing the number of future records beforehand, letting the Transformer predict future records until the timestamp surpassed one year after the last current patient record, however, this approach was too inconsistent, so we decided to omit it entirely.

Since we aim to test our model, we modify this workflow slightly. After loading patient data, records from the last year are separated from the rest and are not seen by the model. Using the remaining records, the model predicts costs based on the predefined workflow. Afterwards, the cost is extracted from the separated records, and the real cost category of the patient for that year is compared to the model's predictions to assess accuracy or proximity.

After running testing for all 1,000 patients, we obtained the results shown in Tab. 9.3, where we distinguish three result types:

- Great: The portion of patients for whom the model predicted the cost category accurately.
- Good: Patients for whom the model predicted a category either one lower or higher than the true value.
- Wrong: Patients for whom the model was off by more than one category.

From these results, we see that if we consider success as only precise category predictions, our model achieved 39.9% accuracy. However, if we allow the model to be off by one category, considering results correct if they are in the ballpark, this accuracy increases to 84.4%.

Result type	count	Fraction
Great	399	39.9%
Good	445	44.5%
Wrong	156	15.6%

Table 9.3: Results of future patient cost model split by types of results.

In 60.1% of cases, the model did not predict the precise correct category. We can examine what fraction of these involved overestimation (where the model predicted a higher category than the actual one) versus underestimation (the opposite). These results are shown in Tab. 9.4, where we observe that the model underestimates in far more cases than it overestimates. This behavior might be partially explained by the lower accuracy of the record cost prediction model for high-cost categories. If this component underestimates individual record costs, it would propagate to the complete model, causing systematic underestimation of the patient's future total cost.

Error type	count	Fraction
Overestimate	237	39.4%
Underestimate	364	60.6%

Table 9.4: Assessment of overestimating and underestimating in imperfect results.

In general, we can conclude that our model can predict a patient's future cost to some extent, however, the results are still suboptimal.

Conclusion

This thesis set out to address the challenge of predicting the future costs for patients within the Slovak healthcare system, using underutilized medical data available. The primary objective was to develop a machine learning framework capable of forecasting patient expenses for the following year, using historical records of medical procedures and drug prescriptions.

To achieve this, the work was divided into several key tasks: embedding patient records into meaningful numerical vectors, predicting future medical events, estimating the costs of these events, and aggregating these predictions to forecast annual patient expenses. A variety of models were explored, including multilayer perceptrons (MLP), recurrent neural networks (RNNs) such as LSTM, and decoder-only Transformer architectures. Special attention was devoted to the design of embeddings for diagnoses, drugs, and procedures, ensuring that similar medical events would have similar representations, thereby improving the models' predictive capabilities.

In the embedding tasks, the desired results were achieved, especially for medical procedures. Based on our evaluations, the resulting embeddings can be used for clustering procedures into groups, a capability for which no practical solution currently exists. While the approach did not yield perfect results, we believe that with further refinement, it could become suitable for production use.

The results for predicting patient future cost categories showed that the approach works reasonably well. The model managed to get the exact cost category right in 39.9% of cases, and was within one category in 84.4% of cases.

However, the analysis also revealed limitations. The model tended to underestimate costs, particularly for patients with records in higher-cost categories. This is likely due to class imbalance and the inherent difficulty of predicting rare, high-expense events.

Despite these challenges, the thesis demonstrates that it is possible to use routinely collected healthcare data to forecast future patient costs with decent precision. The framework developed here could help healthcare providers and policymakers with planning resources, identifying high-risk patients earlier, and managing care more efficiently.

In summary, this thesis demonstrates that advanced machine learning methods can improve predictive analytics in healthcare. There's definitely room for improvement, especially as more and better data becomes available, but we believe this work is a step towards more personalized and data-driven healthcare in Slovakia and possibly elsewhere.

Bibliography

[1] Anatomical Therapeutic Chemical (ATC) Classification — who.int. `https://www.who.int/tools/atc-ddd-toolkit/atc-classification`. [Accessed 25-09-2024].

[2] GitHub - essential-data/word2vec-sk: Vector representations of Slovak words trained using word2vec — github.com. `https://github.com/essential-data/word2vec-sk`. [Accessed 04-03-2025].

[3] GitHub - essential-data/word2vec-sk: Vector representations of Slovak words trained using word2vec — github.com. `https://github.com/essential-data/word2vec-sk`. [Accessed 19-10-2024].

[4] Google | LaBSE | Kaggle — kaggle.com. `https://www.kaggle.com/models/google/labse/tensorFlow2/labse/1?tfhub-redirect=true`. [Accessed 18-10-2024].

[5] LSTM — PyTorch 2.6 documentation — pytorch.org. `https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html`. [Accessed 23-03-2025].

[6] Medzinárodná klasifikácia chorôb - MKCH-10 — nczisk.sk. `https://www.nczisk.sk/Standardy-v-zdravotnictve/Pages/Medzinarodna-klasifikacia-chorob-MKCH-10.aspx`. [Accessed 16-09-2024].

[7] sentence-transformers/LaBSE · Hugging Face — huggingface.co. `https://huggingface.co/sentence-transformers/LaBSE`. [Accessed 19-10-2024].

[8] Sivanand Achanta, Rambabu Banoth, Ayushi Pandey, Anandaswarup Vadapalli, and Suryakanth V Gangashetty. Contextual representation using recurrent neural network hidden state for statistical parametric speech synthesis. In *SSW*, pages 172–177, 2016.

[9] Adrien Barbaresi. Simplemma, January 2023.

[10] Andreas Berzel, Gillian Z Heller, and Walter Zucchini. Estimating the number of visits to the doctor. *Australian & New Zealand Journal of Statistics*, 48(2):213–224, 2006.

[11] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[12] Vicent Caballer-Tarazona, Natividad Guadalajara-Olmeda, and David Vivas-Consuelo. Predicting healthcare expenditure by multimorbidity groups. *Health Policy*, 123(4):427–434, 2019.

[13] CDC. ICD-10-CM — cdc.gov. `https://www.cdc.gov/nchs/icd/icd-10-cm/index.html`. [Accessed 16-09-2024].

[14] Centers for Medicare Medicaid Services and National Center for Health Statistics. Icd-10-cm official guidelines for coding and reporting fy 2022. `https://www.cms.gov/files/document/fy-2022-icd-10-cm-coding-guidelines-updated-02012022.pdf`, 2022. Updated April 1, 2022. Accessed April 30, 2025.

[15] Yuriy Chechulin, Amir Nazerian, Saad Rais, and Kamil Malikov. Predicting patients with high risk of becoming high-cost healthcare users in ontario (canada). *Healthcare Policy*, 9(3):68, 2014.

[16] Edward Choi, Cao Xiao, Walter Stewart, and Jimeng Sun. Mime: Multilevel medical embedding of electronic health records for predictive healthcare. *Advances in neural information processing systems*, 31, 2018.

[17] Alexis Conneau and Guillaume Lample. Cross-lingual language model pretraining. *Advances in neural information processing systems*, 32, 2019.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[20] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[21] Fangxiaoyu Feng, Yinfei Yang, Daniel Cer, Naveen Arivazhagan, and Wei Wang. Language-agnostic bert sentence embedding. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 878–891, 2022.

[22] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[23] Miao Jin, Qinzhuo Liao, Shirish Patil, Abdulazeez Abdulraheem, Dhafer Al-Shehri, and Guenther Glatz. Hyperparameter tuning of artificial neural networks for well production estimation considering the uncertainty in initialized parameters. *ACS omega*, 7(28):24145–24156, 2022.

[24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[25] Waldemar Lopez. *VECTOR REPRESENTATION OF INTERNET DOMAIN NAMES USING WORD EMBEDDING TECHNIQUES*. PhD thesis, 11 2019.

[26] Elizabeth C Lorenzi, Stephanie L Brown, Zhifei Sun, and Katherine Heller. Predictive hierarchical clustering: Learning clusters of cpt codes for improving surgical outcomes. In *Machine Learning for Healthcare Conference*, pages 231–242. PMLR, 2017.

[27] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[28] Nada Mimouni, Jean-Claude Moissinac, and Anh Tuan Vu. Domain specific knowledge graph embedding for analogical link discovery. 06 2020.

[29] Riccardo Miotto, Li Li, and Joel T Dudley. Deep learning to predict patient future diseases from the electronic health records. In *Advances in Information Retrieval: 38th European Conference on IR Research, ECIR 2016, Padua, Italy, March 20–23, 2016. Proceedings 38*, pages 768–774. Springer, 2016.

[30] Mohammad Amin Morid, Kensaku Kawamoto, Travis Ault, Josette Dorius, and Samir Abdelrahman. Supervised learning methods for predicting healthcare costs: systematic literature review and empirical evaluation. In *AMIA annual symposium proceedings*, volume 2017, page 1312, 2018.

[31] Mohammad Amin Morid, Olivia R Liu Sheng, Kensaku Kawamoto, Travis Ault, Josette Dorius, and Samir Abdelrahman. Healthcare cost prediction: Leveraging fine-grain temporal patterns. *Journal of biomedical informatics*, 91:103113, 2019.

[32] Online. In: Chatgpt vezia 4. *Available at: OpenAI, URL*, Task:.

[33] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[36] Radim Rehurek and Petr Sojka. Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, 3(2), 2011.

[37] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.

[38] Ján Poprac Zuzana Ďurčíková Ján Popovič Boris Leštianský Stanislav Žiaran, Kvetoslava Bernátová. *Praktická príručka pre používanie pravidiel kódovania v systéme SK-DRG v súlade s Výnosom MZ SR č. 09467/2015*. Úrad pre dohľad nad zdravotnou starostlivosťou, 2016. Dostupné online: `https://www.udzs-sk.sk/documents/14214/24311/03112016_Praktick%C3%A1+pr%C3%ADru%C4%8Dka+pre+pou%C5%BE%C3%ADvanie+pravidiel+k%C3%B3dovania+v+syst%C3%A9me+SKDRG+v+s%C3%BAlade+s+V%C3%BDnosom+MZ+SR+%C4%8D+094672015+_+kb+kody+mkch.pdf`.

[39] Yi Sun, Yu Zheng, Chao Hao, and Hangping Qiu. Nsp-bert: A prompt-based few-shot learner through an original pre-training task–next sentence prediction. *arXiv preprint arXiv:2109.03564*, 2021.

[40] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.